PROTOTYPES AND PRODUCTION RULESt
AN APPROACH TO KNOWLEDGE REPRESENTATION FOR HYPOTHESIS FORMATION

Jan S. Aikins
Heuristic Programming Project, Department of Computer Science
Stanford University, Stanford, CA 94305

A system called CENTAUR has been implemented to interpret data derived from pulmonary function tests using a knowledge representation that combines the advantages of both production rules and frames. The system uses a hypothesis-directed approach to problem solving, in which hypotheses are suggested by the Initial data, further Information is acquired, and then more specific hypotheses are selected. The hypotheses are represented as PROTOTYPES, frame-like data structures each of which characterizes some pulmonary disease. The prototypes guide the invocation of the production rules and focus the search for new information. Some of the advantages afforded by representing knowledge as both prototypes and rules are also presented.

1 Introduction

Much of Artificial Intelligence research has focused on determining the appropriate knowledge representations to use in order to achieve high performance from knowledge-based systems. The principal Artificial Intelligence theme being explored in this present research" is that there are many advantages to a system that uses both frame-like structures and production rules to perform problem-solving tasks in knowledge-intensive domains

In order to test this theme, a knowledge representation was designed using a combination of frames and production rules. The frames are called Prototypes because they represent stereotypical situations which can be used as a basis for comparison to the actual situation given by the data.[22] The domain chosen was that of pulmonary physiology. The task was to interpret a set of pulmonary function test results, producing a set of interpretation statements and a diagnosis of pulmonary disease In the patient. A system called CENTAUR has been written to perform this task using prototypes that characterize the typical features of each pulmonary disease. Each feature is called a Component of the prototype. Associated with each component are production rules used to infer a value for the component. These production rules are a form of procedural attachment with a constrained, stylized syntax that makes them easier to examine than general procedures. This constrained syntax leads to other advantages, such as ease of acquisition and modifiability as discussed in [2]. The prototypes focus the search for new information by guiding the invocation of the production rules and eliciting the most relevant information from the user. These prototypes are linked together in a network in which the links specify the relationships between the prototypes.

This research developed out of the MYCIN project [5], which uses a knowledge base of production rules to perform infectious disease consultations. Initially, a MYCIN-like production rule system called PUFF [4] was written to perform pulmonary function test interpretations. Problems with the production rule formalism In PUFF and similar rule-based systems, such as a need to focus the search for new information and the desire to represent characteristic patterns of disease, motivated the creation of a prototype-directed system.

2 The CENTAUR System

CENTAUR produces an interpretation of data and a pulmonary diagnosis based on a set of pulmonary function test results. The inputs to the system are the pulmonary function test results and a set of patient data including the patient's name, sex, age, and a referral diagnosis. The output consists of both a set of interpretation statements that serve to explain or comment on the pulmonary function test results and a final diagnosis of pulmonary disease in the patient.

CENTAUR uses a hypothesis-directed approach to problem solving where the possible hypotheses are represented by the prototypes. The goal of the system Is to confirm some of the prototypes as matching the data in an actual case. The final set of confirmed prototypes is the system's solution for classifying the data in that case. The prototypes represent the various pulmonary diseases, their degrees and subtypes, with the result that the set of confirmed prototypes represents the diagnosis of pulmonary disease in the patient.

In the example below, the prototype representing a pulmonary function consultation itself, the PULMONARY-DISEASE prototype, has been selected as the initial Current Prototype, the system's best hypothesis about how to classify the data in the case. The Initial data is requested and the user responses (preceded by a double asterisk **) are recorded. Data entered in the system suggests or "triggers" disease prototypes. The triggered prototypes are placed on a Hypothesis List and ordered according to how closely they match the data. The system attempts to fill in values for the components of the current prototype, which may cause rules to be invoked; or, if no rules are associated with the component, the system will ask the user for the value When all of the prototype components have values, a decision is made by the system as to whether the given data values match those expected for the prototype. Another prototype is then selected as the Current Prototype, and the process repeats. The system moves through the prototype network confirming or disproving indicated disease prototypes. Matching data and prototypes continues until each piece of data has been accounted for by some confirmed prototype or until the system has concluded that it cannot account for any more of the data.

* This work was supported by the Advanced Research' Projects Agency under contract MDA 903-77-C-0322. Computer facilities were provided by the SUMEX-AIM facility at Stanford University under NIH grant RR-00785. The author is sponsored by the Xerox Corporation under the direction of the Xerox Palo Alto Research Center.

" The term prototype has been given the same meaning by other researchers, for example in KRL [I], a prototype is a special kind of unit representing a hypothetical individual that Is the typical member of a class.

3 CENTAUR Example

The following is an example of an interpretation of a set of pulmonary function test results for one pulmonary patient. Comments are in italics.

```
*CENTAUR*        14-Jan-79  13:54:07

        . . . . . .PATIENT-7. . . . . .
    1) Patient's identifying number:  ** 7446
    Z) referral diagnosis:  ** ASTHMA
    3) FEV1/FVC ratio:  ** 40
       [Trigger for OAD and CM 900]
```

(Prototype OAD Is Metered by the value 40 for the FEVIIFVC ratio. The Certainty Measure (CM) indicates on a numerical scale the degree of certainty with which the prototype is indicated by the data.)

```
    4) TIC observed/predicted:  •* 139
    5) FVC/FVC-predicted:  ** 81
       [Trigger for NORMAL and CM 500]
```

(The questioning continues and other prototypes are triggered by the data values.)

```
    . . .
    MoreSpecific Prototypes chosen: NORMAL OAD
```

(Although there are five possible, more specific disease prototypes for the PULMONARY-DISEASE prototype, only the two that were triggered by the initial data are selected as possibilities to pursue. These prototypes are filled in with the data values that are already known in the case.)

```
    !Surprise Value! 261 for RV in NORMAL, CM: 700
    !Surprise Value! 139 for TLC 1n NORMAL, CM: 400
    . . .
```

(Any data values that are not consistent with the values expected for that disease prototype are noted as Surprise Values, and the CM for that prototype is lowered. Two of the data values that are not consistent with the NORMAL pulmonary function prototype are shown here.)

```
    Hypothesis List: (OAO 990) (NORMAL -699)
```

(The Hypothesis List of triggered prototypes is then ordered according to the CM of the prototypes and a new Current Prototype, OAD, is chosen.)

```
    Components of OAO to trace:  F25 D-RV/TLC
```

(In order to instantiate the OAD prototype, two more components must have values. These are then asked of the user if there are no rules that can be used to deduce their values. The OAD prototype is confirmed as matching the data in this case. Control information associated with the prototype specifies that the Degree of OAD should be determined next, followed by the Subtype of OAD.)

```
    Confirmed: ASTHMA SEVERE-OAO OAO
```

(Eventually SEVERE-OAD and ASTHMA are also confirmed. Data values that can be accounted for by one of the confirmed prototypes are marked. If there are data values remaining that cannot be accounted for by the confirmed prototypes, the system will attempt to determine if there are multiple diseases in the patient. Refinement Rules associated with the confirmed prototypes are executed to further refine the diagnosis and conclusions which are then printed.)

```
    Conclusions:
    Smoking probably exacerbates the severity
          of the patient's airway obstruction.
    Good response to bronchodilators 1s consistent
          with an asthmatic condition.
    . . t
    Pulmonary Function Diagnosis:
    Severe Obstructive Airways Disease.
    Asthmatic type.
```

4 Prototypes and Components

Following frame terminology, each prototype contains SLOTS of information associated with it. One of these is the slot COMPONENTS that lists the substantive characteristics of the prototype. Each component may, in turn, have slots of information associated with it In the OAD prototype in Figure 4.1, there are components for many of the pulmonary function tests that are useful in characterizing a patient with OAD For example, the TOTAL LUNG CAPACITY of a patient with OAD is typically higher than that of a person with normal pulmonary function. Thus there is a component, TOTAL LUNG CAPACITY, with a range of PLAUSIBLE VALUES that are characteristic of a person with OAD.

Some control information is represented explicitly in slots associated with the prototype. These slots contain a set of one or more clauses that express some action to be taken by the system in order to instantiate the prototype (CONTROL slot), upon confirmation of the prototype (IF-CONFIRMED slot), in the event that a prototype is disproved (IF-DISPROVED slot), and in a clean-up stage after the system processing has been completed (ACTION slot).

PROTOTYPE	Obstructive Airways Disease
GENERAL SLOTS	
—Bookkeeping Information	Author: Aikins Date: 27-OCT-78 Source: Dr. Fallat
—Pointers to other prototypes (link prototype)	Pointers: (degree MILD-OAD)... (subtype ASTHMA)...
—English phrases	Hypothesis: There is an interpretation of OAD."
CONTROL SLOTS	
Control	If-Confirmed:
If-Confirmed	Deduce degree of OAD
If-Disproved	Deduce subtype of OAD
Action	Action: Deduce OAD findings Print OAD findings
COMPONENTS	TOTAL LUNG CAPACITY
Plausible Values	Plausible Values: > 100
Default Value	Importance: 4
Possible Error Values	
Rules	REVERSIBILITY
Importance of value to this prototype	Rules: 19, 21, 22, 25 Importance: 1

FIGURE.4.1 A sample prototype with possible slots on the left and values for OAD on the right.

4.1 Production Rules

The CENTAUR knowledge base also includes sets of production rules. Many of the production rules are classified as INFERENCE RULES, rules used to infer information in the domain. They refer to values for components in their premise clauses and make conclusions about values of components in their action clauses. An example of one of the Inference Rules is given in Figure 4.2. The RULES slot associated with a component contains a list of all Inference Rules that make a conclusion about that component. These may be applied when a value is needed for the component.

If: 1) A: The mmf/mmf-pred 1$ less than 20, and
B: The fvc/fvc-pred 1s greater than 80, or
2) A: The mmf/mmf-pred 1s less than 15, and
B: The fvc/fvc-pred 1s less than 80
Then: 1) There 1s evidonce that the degree of OAD is *severe,* and
Z) One of the OAD findings 1s:
Low mid-exp1ratory flow 1s consistent with severe airway obstruction.

FICURE 4 2 A Sample Production Rule-English Version

5 Control Structure

The control information used by CENTAUR is contained either in slots associated with the individual prototypes or in a simple interpreter. Control strategies specific to an Individual prototype are represented in slots associated with that prototype, with more general system control being expressed in the interpreter.

The control structure can be broken into three stages: a hypothesis-formation stage in which data values are acquired and an attempt is made to match prototypes to data, resulting in a list of confirmed prototypes; a refinement stage in which a set of REFINEMENT RULES are applied to the list of confirmed prototypes to "debug" this list and further refine the recommendations that will be made; and a final "clean-up" stage in which, for example, findings associated with the prototype are printed.

6 Advantages of the Prototype-Directed Approach

The use of this approach for the pulmonary function interpretation task, as compared to the purely rule-based approach used in PUFF, results in two categories of advantages: those dealing with the knowledge base representation Itself and those dealing with the system's reasoning and performance. Advantages in knowledge representation occur partly because some knowledge previously represented in rules is now represented more clearly in prototypes. For example, the prototypes explicitly represent control information formerly represented In the PUFF inference rules. In the PUFF system, there are rules whose purpose it is to guide computation by controlling the invocation of other rules. This feature can be very confusing to the medical experts since they do not know which rules are those intended to represent descriptive medical expertise and which rules are those serving a necessary computational function. New knowledge is also being represented in prototypes; for example, plausible ranges of values for each of the pulmonary function tests for each disease can be listed, as well as the relative importance of each measurement in a particular disease prototype. Advantages in system reasoning and performance, that is, the questions that are asked and the order in which Information is acquired, include the following:

(A) Consultation flow more closely follows physician's reasoning. The process of medical problem solving has been discussed by many researchers (e.g., [3]) and it is widely felt that a sequence of suggesting hypotheses, acquiring further information, and then revising the hypotheses, as is used in CENTAUR, is, in fact, the problem-solving process used by most physicians. Thus CENTAUR offers increased conceptual clarity, in that the user can understand what the program is doing, and this factor leads to other advantages, for example, the system can offer the user a more intelligible explanation of its performance during the consultation.

(B) Questions are asked in a reasonable order. In a rule-based system such as PUFF, questions are asked of the user as rules are invoked containing clauses referring to information not yet known. The expert can control the order in which the questions are asked only by writing rules to enforce some order. As has been discussed, this procedure results in a potentially confusing rule base where some rules guide computation. In the prototype-directed system, the expert can specify the order in which information is acquired for each prototype in the control slot.

(C) Only relevant questions are asked. Another advantage of CENTAUR over PUFF is that only those hypotheses suggested by the initial data are explored. For example, if the Total Lung Capacity (TLC) for the patient is 70, then CENTAUR would begin exploring the possibility of Restrictive Lung Disease (RLD) because a low TLC would trigger the RLD prototype. (A low TLC is consistent with a hypothesis of RLD; a high TLC is consistent with OAD.) In the PUFr program, the first disease tried is always OAD, so the PUFF program would begin asking questions dealing with OAD. These questions would seem irrelevant considering the data, and, Indeed, if there were no data to indicate OAD, such questions would not be asked by CENTAUR.

7 Summary

CENTAUR was designed in response to problems that occurred while using a purely rule-based system. By changing the knowledge representation to include prototypes as well as production rules, new knowledge was represented. Further, knowledge that had been represented rather awkwardly In rules was represented more clearly in prototypes. The production rules were retained as a stylized form of procedural attachment that could be easily examined or modified. By altering the control structure so that a best-fit matching process of prototypes to data produced a current best hypothesis to guide further search, a more focused consultation resulted which more closely followed the way physicians reason. Control knowledge was explicitly labeled and made prototype-specific so that control of the consultation was adapted to the current best hypothesis. In summary, the prototype-directed system achieved better reasoning and performance than the rule-based system. In addition, although representing knowledge in production rules alone did not seem adequate for this task, the ability to represent knowledge In prototypes as well did provide the needed flexibility.

References

[1] D. C. Bobrow and T. Winograd, An Overview of KRL, a Knowledge Representation Language. *Cognitive Science* /(I).

[2] R. Davis and J. King, An Overview of Production Systems. In E. W. Elcock and D. Michie (Eds.), *Machine Intelligence 8,* New York: Wiley & Sons, 1977. Pp. 300-332.

[3] A. S. Elstein, L. S. Shulman, and S. A. Sprafka, *Medical Problem Solving—An Analysis of Clinical Reasoning.* Harvard University Press, Cambridge, Mass., 1978.

[4] J. C. Kunz, R. J. Fallat, et. a!., *A Physiological Rule Based System for Interpreting Pulmonary Function Test Results.* HPP-78-19, C. S. Dept., Stanford University, Dec. 1978.

[5] E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN.* New York. American-Elsevier, 1976.

A BASIC MODEL FOR LEARNING SYSTEMS

Kiyoshi Akama
Dept. Control Engineering
Tokyo Institute of Technology
2-12-1, O-okayama, Meguro-ku,
Tokyo 152, Japan

Atsunobu Ichikava
Dept. Systems Science
Tokyo Institute of Technology
2-12-1, O-okayama, Meguro-ku,
Tokyo 152, Japan

A learning system (LS/0) is designed and implemented on a computer. The LS/0 exchanges information with its environment through three kinds of message strings. Given a question string, the LS/0 produces the response string, then the answer string is shown to the LS/0 as the correct response string. The LS/0 iterates such interactions infinitely. The most important feature of the LS/0 is that it tries to produce the correct response to the unexperienced questions. To do this, the LS/0 organizes the previously acquired information and generates the hypothesis or the knowledge structure. When the response fails to meet the answer, the LS/0 renews the hypothesis or reorganizes the knowledge structure to explain the answer. In the LS/0, the network-like structure called label net plays an important role to represent the knowledge structure.

1. INTRODUCTION

The inductive process must Ue realized by the learning system so that it can improve the hypothesis or the knowledge structure as it acquires the new instances from the environment. The total information that the learning system must store increases quite rapidly in the course of the learning process, and the intelligence of the system depends mostly upon the knowledge structure which is constructed from this large and unorganized set of information. The problem to be solved in the design of the learning system is, therefore, how to construct the representation space, the set of possible knowledge structures, and how to realize *the* algorithm to create,reorganize and utilize the knowledge structure. In order to solve this problem, the learning system LS/0 is designed and implemented on a computer. *The* aim of this paper is to explain the outline of the LS/0. The details of the LS/0 and the basic theory of the learning systems will be reported in the subsequent papers.

	QUESTION	ANSWER
1	DOG-?	DOG-DOG
2	CAT-?	CAT-CAT
3	LION-?	LION-LION
4	L(12,4)-?	12
5	L(34,45)-?	34
6	L(876,6)«?	876
7	R(34,67)-?	67
8	WHATISCAT	ITISANIMAL

Fig. 1 An environment EV_1

2. THE LEARNING SITUATION

We shall consider the sequence of question-answer pairs. This is called an environment (EV).

EV - [$(Qv,A_1),(Q_2,A_2),(Q_3,A_3)$, ...]

where Qt and A_t are strings on the alphabet E, the set of alphanumeric characters except "#", and are called the question and the answer, respectively. Fig. 1 shows a typical example of the environment. We shall denote the environment by EV_1 hereinafter.

The interaction is an information exchange between the learning system and the environment. It consists of three phases: Q, R and A. In the phase Q, the environment gives a question string Q_t to the system. In the phase R, the system must reply a response string R_t to the question Q_t. In the phase A, the environment shows the system an answer string A_t as the correct responce to the question Qt. An interaction at time t can be represented by (Q_t, R_t, A_t).

The system infinitely iterates the interactions with the environment. The infinite sequence of interactions is called the question-response-answer process (Q-R-A process) or the learning process (LP).

LP- [$(Q_l,R_l,A_1),(Q_2,R_2,A_2),(Q_3,R_3,A_3)$, ...]

Fig. 2 shows an example of the learning process,

which is the sequence of interactions between the LS/0 and the EV_1, and is called LP_1 hereinafter.

The interaction is said to be successful when the response and the answer strings are the same. The system is evaluated by the rate of successful interactions in the Q-R-A process.

3. THE LEARNING SYSTEM

A system which operates under the situation described in section 2 is designed in this paper. This is called LS/0 and is implemented on a computer as a FORTRAN program. The system LS/0 is composed of three elements: the execution element, the learning element and the memory element. The execution element receives a question string and generates a response string by utilizing the knowledge acquired so far in the memory element. The learning element makes specific changes in the knowledge in the memory element to improve the future responses. The memory element stores the learned knowledge, which is often called the state of the system.

The learned knowledge in the memory element of the LS/0 is mainly represented by the network-like structure called the label net. The examples of the label net is shown in Fig. 3 (l)-(8) The label net is reviewd briefly in what follows. The details were reported in [1]. The label net can represent much wider variety of structures than the Fig. 3 shows.

The label net consists of finite vertices and finite labels. Each vertex corresponds to a set of strings and each label represents the relation between the set of strings. Strictly speaking, the label net is a set of equations with the variables of sets of strings.

	QUESTION	RESPONSE	ANSWER
1	DOG=?	???	DOG=DOG
2	CAT=?	???	CAT=CAT
3	LION=?	LION=LION	LION=LION
4	L(12,4)=?	L(12,4)=L(12,4)	12
5	L(34,45)=?	L(34,45)=L(34,45)	34
6	L(876,6)=?	876	876
7	R(34,67)=?	R(34,67)=R(34,67)	67
8	WHATISCAT	???	ITISANIMAL
.	.	.	.
.	.	.	.

Fig. 2 The learning process LP_1 between the LS/0 and the EV_1

The meaning of the label net is well explained by the following example. The label between VI and V2 in Fig. 3 (4) means that the set of VI contains the string "a=?#a=a" if the set of V2 contains the string "a". The two labels pointing toward V2 in Fig. 3 (4) means that the set of V2 contains "DOG" and "CAT". These relations determine the set of VI, S(V1), and the set of V2, S(V2).

S(V1) - { DOG-?//DOG-DOG, CAT-?//CAT-CAT }
S(V2) - { DOG, CAT }

The knowledge stored in the lable net is utilized during the course of the learning process through the vertex VI. when there exists a string "a#B" in the set of VI, the structure implies that the string "3" may be the correct response to the question string "a".

4. THE LEARNING PROCESS OF THE LS/0

In order to show how the LS/0 interacts with the environment and how tt improves its knowledge structure, the learning process LP_1 in Fig. 2 and the state change of the LS/0 during the learning process are described in the following.

The system LS/0 starts with the initial state shown in Fig. 3(1).

(1-Q). . . . .DOG*?

The first question is given to the LS/0 by the EV_1. The question is "DOG-?".

(1-R).....???

The state (1) in Fig. 3 indicates the LS/0 has no knowledge on the EV_1. The LS/0 was designed to reply "???" when it fails to find any strings to the given question.

(1-A). . . . .DOG-DOG

The string "DOG-DOG" is given to the LS/0 as the answer. After this interaction, the LS/0 changes the state from (1) to (2) in Fig. 3. The state (2) means that "DOG-DOG" is the correct response to "DOG-?".

(2-Q). . . . .CAT-?
(2-R) ???

The clue which the LS/0 has at this point is only the knowledge (2). The LS/0 replies "???" again.

(2-A). . . . .CAT-CAT

The interaction at time 2 causes the state change from (2) to (3). The structure (3) stores information gained from the interactions at time 1 and 2 separately. The learning element of the LS/0 further tries to organize the structure and gains the state (4). The vertex V2 was newly generated. The learning element assumes that the set of V2 might be equal to the set £*, the set of all strings of any length Including zero

length one. This changes the state from (4) to (5).

(3-Q). . . . .LION-?

(3-R). . . . .LION-LION

The state (5) enables the LS/O to reply the string "LION-LION".

(3-A). . . . .LION-LION

The response is found to be correct. The state changes to (6).

(4-Q). . . . .L(12,4)=?

(4-R). . . . .L(12,4)-L(12,4)

This response is generated by the execution element utilizing the structure (6).

(4-A). 12

The answer string at time 4 shows the correct response is not "L(12,4)-L(12,4)" but "12". The LS/O stores this information in (7).

(5-Q). . . . .L(34,45)=?

(5-R). . . . .L(34,45)=L(34,45)

(5-A). 34

After storing the new information to (7) by adding the labels "L(34,45)'?#34" and "L(34,45)" to VI and V4, respectively,the LS/O tries to reorganize it and constructs the structure (8).

(6-Q). . . . .L(876,6)-?

(6-R). 876

The LS/O utilizes the structure (8) and generates the response "876".

(6-A). 876

This interaction is found to be successful. The information gained at time 6 is stored by adding the same label "876#6" to both V5 and V6.

(7-Q). . . . .R(34,67)«?

(7-R). . . . .R(34,67)=R(34,67)

(7-A). 67

The answer string of the EV_1 again forces the LS/O to reorganize its knowledge structure. The learning process between the LS/O and the EV_1 continues infinitely in this manner.

REFERENCE

[1] Akama, K. and Ichikawa, A. "Representation of Set of Symbol Strings for Learning Systems." Trans. Institute of Electronics and Comunioation Engineers of Japan. J61-D No.9 (1978)

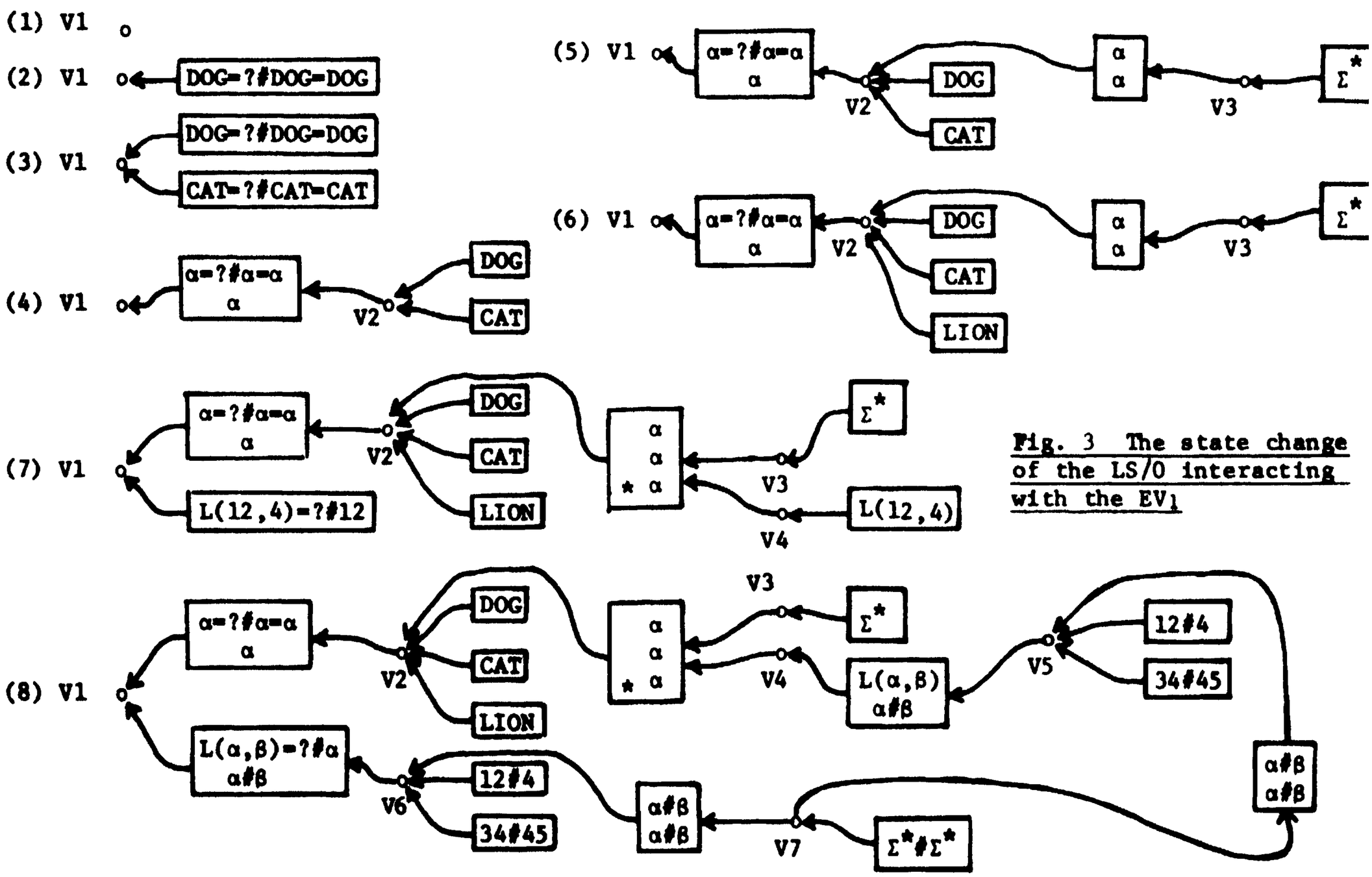


Fig. 3 The state change of the LS/0 interacting with the EV_1

TOWARDS UNDERSTANDING COLOR OCULAR FUNDUS IMAGES

Koichiro Akita

Hideki Kuga

Central Research Laboratory
Mitsubishi Electric Corporation
80 Nakano, Minamishimizu,
Amagasaki, Japan 661

We have studied some fundamental problems on the way of understanding color ocular fundus images by computer. They are the extraction of blood vessels from the retinal background and labeling them arteries and veins. We analyze the chromatic characteristics of color ocular fundus images to get the signal level knowledge on them which is used in the stages of extracting blood vessels and most reliable labeling of initial line segments. We propose a dynamic threshold selection scheme for binarization of gray images that have an amount of shadings or uneveness of gray values. In the labeling stage we use a deterministic procedure which takes into account the physical level knowledge on blood vessels. Some other methods are also discussed.

1. INTRODUCTION

Color ocular fundus images are used in the mass diagnosis of adult diseases such as hypertension and diabetes. Since the photographing system of them is simple and cheap, if the automatic understanding system of them is realized, the mass health management by them together with chest X-ray images will be much more popularized. There are a few examples of computer analysis of ocular fundus images [1],[2],[3],[A],[5].

For color ocular fundus photographs, as far as we know, there has been only one example of computer analysis. It is the pattern recognition of color fundus images by Yokouchi et al. [A],[5], They investigated the possibility of automatic diagnosis of arterio-venous crossing phenomena. But their method is interactive in the sense that they indicate the cross points manually before the computer analysis. The change of venous caliber before and after its crossing was measured along its course and the minimum value of each normalized caliber measurement was utilized to grade the crossing signs. The researches mentioned above stay in the phase of analysis. The objective of our research is the understanding of color ocular fundus images by computer, which is inevitable for the automatic mass diagnosis of hypertension and diabetes. The theme in this paper is to locate blood vessels and identify arteries and vains automatically. This process is fundamental for the analyses of not only arterio-venous crossing phenomena but also other symptoms of adult diseases [9].

We have to make clear the levels of knowledge [10] that can be used in each stage of the process.

For the first step we analyze the chromatic characteristics of color ocular fundus images. We get the signal level knowledge on them which is utilized in the stages of extracting blood vessels and most reliable labeling of initial line segments.

We propose a dynamic thresholding scheme for the extraction of blood vessels. It is very effective against images which have shadings or unevenness of light intensities. Thirdly a deterministic labeling algorithm of blood vessels is proposed in which the physical level knowledge on them is implemented. The lables of some vessel segments from which the labeling starts are determined by the signal level knowledge and some positional constraints.

This work has been supported by Pattern Information Processing System Project of the Agency of Industrial Science, and Technology of the Ministry of International Trade and Industry of Japan.

2. CHARACTERISTICS OF COLOR FUNDUS IMAGES

The first step of understanding color ocular fundus images is the analysis of their photographic and chromatic properties. They are taken on the 35 mm color reversal films by the fundus camera. It requires some skill to take the fundus photos of good quality. Since the state of fundus is individually different, they have some variation of color tone. More specifically, there are two main causes of the variation—the photographic system and the individual. The former involves the spectral characteristics of color films, the condition of photo processing, the optical property of the fundus camera and the amount of light. It is comparatively easy to prescribe these conditions and they do not have so much influence on changing color tone. The latter comes from the age, the size of pupil and other conditions of a person [1]. It changes color tone of photos remarkably. Therefore, it is very important to select parameters that are not influenced by each photograph, in other words we should use relative (not absolute) color information.

2-1. Digitization of Film Images

In our experiment we used a high-speed drum scanner which had the Wratten No.25, No. 47B, and No.58 filters for tricolor decomposition. The photo density was quantized to 256 levels.

The color fundus images on films were scanned by the drum scanner with the sampling pitch of 50 urn, and digitized to 255 x 256 pixels (i.e. 12.8mm x 12.8mm on film).

2-2. Chromatic Analysis of Digitized Fundus Images

A fundus image consists of four portions; Retina, Disc, Artery, and Vein. We plotted the chromatic information of the digitized data in the sample areas of each portion onto the UCS color coordinate system (u, v, V).

The distribution in u-v plane shows that it is parallel to the u-axis, which means the chromatic information of this sample fundus image is approximately represented by the u-coordinate. Furthermore the distribution of retina is roughly separated from that of blood vessels. But the distributions of artery and vein can't be distinguished.

From the distribution of those data in V-u coordinates we see that arteries cant' be completely separated from veins also in this case.

Certainly some confirmed artery or vein pixels are determined by (u, v, V) values.

The analysis tells us that the labeling of blood vessels requires not only signal level knowledge mentioned above, but also physical level knowledge which will be introduced in section 4.

3. A DYNAMIC THRESHOLD SELECTION FOR EXTRACTING BLOOD VESSELS

The extraction or distinction of blood vessels from the retinal background is a milestone of understanding fundus images.

Color ocular fundus images contain an amount of shading or unevenness of light intensity. So the ordinary global threshold selection methods fail to extract blood vessels. Dynamic threshold selection schemes are desirable in such circumstances. Chow and Kaneko's method [6] is a well-known one which has been applied to the extraction of blood vessels in fluorescence fundus images with some modification by some researchers [3] with fairly good results.

The contrasts of fluorescence fundus images are much higher than those of color fundus images. The shortcomings of this method when applied to the extraction of low contrast blood vessels are as follows. The thresholds of all subimages cannot necessarily be obtained and the distribution of them is non-uniform in the image because of the irregular pattern of blood vessels. Therefore, the error of interpolating the threshold at each pixel is often bigger than the contrast between blood vessels and the retinal background, resulting in noisy binary images. We abandoned applying Chow and Kaneko's method to color fundus images because of their low contrasts and unevenness of light intensity. We have developed our own method.

3-1. A Dynamic Threshold Selection Scheme

The gray image is splitted into a set of blocks (subimages) as in the case of Chow and Kaneko's method to eliminate the influence of shading. In each block it is examined whether the boundary of blood vessels and the background is included or not by taking the derivative and binarizing it with variable thresholds. When boundaries are detected in a block, the histrogram of pixels only around boundaries is made to determine the threshold for this block. If a block does not include any boundary, It is

classified into the background. The size of a block which is given in advance is large enough to contain full width of a vein or an artery. The flow of this scheme is the following.

(1) Split the image into N blocks. For each block, do the next steps.
(2) Apply the Laplacian operator to the block. The result will be called the Laplacian subimage.
(3) Binarize the Laplacian subimage by varying the threshold and check whether there is a connected component whose size satisfies some conditions. If not, then set the flag of this block to 0 representing that it belongs to the background. If it exists, the flag is set to 1 showing that this block includes some parts of vessels.
(4) For each block whose flag is equal to 1, the histogram of pixels only around boundaries is made to determine the threshold by Otsu's method [7] and the block is binarized. In the binary block the size of each connected component is measured and if it is less than a given value, the component is eliminated as a noise.

We should add some words on step (3). When the Laplacian sublmage is binarized, the threshold is decremented first from the given upper value, until the size of maximum connected component satisfies a certain condition. If any line segment which corresponds to a part of a boundary exists, then the threshold is incremented next from the given lower value until the size of maximum connected component satisfies another condition. This procedure is important in order not to miss low contrast boundaries which also exist in the block.

For the conditions mentioned above we can set some constraints on the size and the shape of the extracted boundary in the subimage.

A. LABELING BLOOD VESSELS

The second stage of fundus image understanding is the labeling of blood vessels. There are two kind of vessels; arteries and veins. Our aim here is to locate them, which leads to the detection of crossing points and to the measurement of arterio-venous crossing phenomena such as tapering, banking, Salus' sign, S-shaped bend and parallel-Gunn [9].

The physical level knowledge on blood vessels that can be used at this stage is as follows.

(i) Arteries (Veins) do not intersect each other. In other words arteries (veins) intersect only veins (arteries).
(ii) Arteries (Veins) branch off only from arteries (veins).

In order to use this knowledge positively we have adopted the following procedure for locating arteries and veins.

(1) Make the thinned image from the binary vessel image.
(2) Detect characteristic points in the thinned image and make the line segments list.
(3) Label each line segment, namely determine whether it is a part of an artery or a vein.

4-1. Skeleton Patterns of Binary Blood Vessels

The state of blood vessels such as branching, crossing and meandering is simply represented by that of their medial axises or skeletons. The direct results of thinning binary blood vessel images contain small circles, isolated points and prickles that act as noises. We first eliminate them and start with thinned images. In order to analyze blood vessel networks we detect characteristic points and list up line segments. We define three kinds of characteristic points: end point, branch point and cross point. A line segment is defined as a segment of medial axis whose head and tail are characteristic points. Therefore, there are nine situations of a line segment. Several parameters on a characteristic point and a line segment are measured and are written into the characteristic points list (CP-LIST) and the line segments list (LS-LIST), respectively. These lists are very useful to activate the physical level knowledge mentioned above.

* Characteristic Points List (CP-LIST)

It includes several parameters of each characteristic point. They are the (x, y) coordinate, the characteristic index which represents whether the point is an end Pt. or a branch Pt. or a cross Pt., the numbers of the line segments which meet at this point, and so forth.

* Line Segments List (LS-LIST)

The parameters of each line segment in the LS-LIST are the characteristic point numbers of the head Pt. and the tail Pt., the length, the directions of this line segment at the head Pt. and the tail Pt., the chain code and others.

4-2. Labeling Algorithm

The goal of labeling is to decide that each line segment belongs to an artery or a vein or the retinal background. The labeling starts with some line segments that have most reliable initial labels determined by the signal level knowledge.

4-2-1. Initial Labeling

The gray values of arteries are smaller than those of veins in a broad sense. If we limit the area in the V-image and set two thresholds 8L, GU in the gray levels, we can expect with certainty that the pixels whose gray values are less than 8L belong to arteries and the pixels whose gray values are greater than 0U belong to veins. 9U and 0L may be settled by some clustering approaches. The most reliable line segments with which the labeling starts are determined as follows. On every line segment of the skeleton pattern in the certain limitted area the number of pixels with a certain label (or no label) is counted and divided by the line length. There are three ratios, PA, PV, and PN which represent the probabilities that the line segment is a part of artery, vein and unknown, respectively. Hence PA + Pv + PN = 1.0. If a line segment has the probability of PA or Py higher than a given value, it is chosen as one of the most reliable line segments.

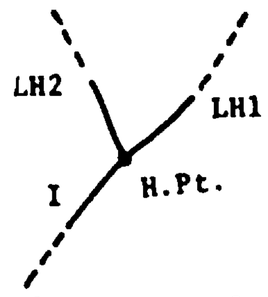
4-2-2. Labeling Procedure

The labeling starts from the line segments which have the most reliable initial labels. If a non-labeled line segment is connected to other labeled line segments, it will be labeled as follows using the physical level knowledge mentioned before. The procedure also takes into account the natural directions of blood streams.

Since the full program structure of the procedure is lengthy to be described here, only the essence of it is presented.

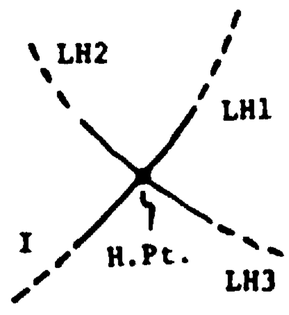
At the head point (H.Pt.) and the tail point (T.Pt.) of a non-labeled and non-isolated line segment I, the next steps are taken.

1) When the H.Pt. is a branching Pt., let the other two line segments be LH1 and LH2.



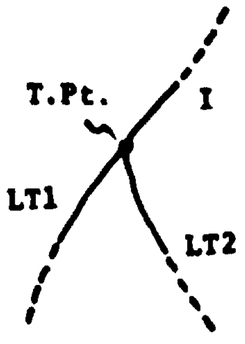
① Label(LH1) = Label(LH2) ≠ C ≠ 0 → Label(I) = C, go to 3).
② Label(LH1) ≠ 0, Label(LH2) ≠ 0 and Label(LH1) ≠ Label(LH2) → Select the line segment whose angle between I is nearest to 180° and call it LH1. Let Label(I) = Label(LH1), and Conflict-Flag(I) is incremented by one. Go to 3).
③ Label(LH1) ≠ 0 or Label(LH2) ≠ 0 → Label(I) = Label(LH1) or Label(LH2), go to 3).
④ Otherwise, go to 3).

2) When the H.Pt. is a crossing Pt., let the other three line segments be LH1, LH2 and LH3.



① Choose the line segment whose angle between I is nearest to 180° and call it LH1. If Label(LH1) ≠ 0, then let Label(I) = Label(LH1) and go to 3). Otherwise do the next steps.
② Label(LH2) = Label(LH3) = artery (vein) → Label(I) = vein (artery), go to 3). Label(LH2) ≠ 0, Label(LH3) ≠ 0 and Label(LH2) ≠ Label(LH3) → Conflict Flag(I) is incremented by one and no label is given to I. Go to 3).
③ Label(LH2) = artery (vein) or Label(LH3) = artery (vein) → Label(I) = vein (artery), go to 3).
④ Otherwise go to 3).

3) When the T.Pt. is a branching Pt., let the other two line segments be LT1 and LT2.



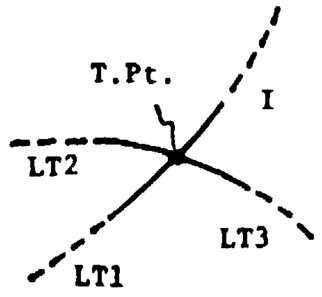
① Label(LT1) = Label(LT2) = C ≠ 0 → If Label(I) = 0, then let Label(I) = C. If Label(I) ≠ 0 and ≠ C, Conflict Flag(I) is incremented. Exit.
② Label(LT1) ≠ 0, Label(LT2) ≠ 0 and Label(LT1) ≠ Label(LT2) → If Label(I) = 0, then choose the line

segment whose angle between I is nearest to 180° and call it LT1. Label(I) = Label(LTI). Exit.
If Label(I) ≠ 0, increment Conflict Flag(I) by one and exit.

③ Label(LTI) ≠ 0 or Label(LT2) ≠ 0
+ If Label(I) = 0, then let Label(I) = Label(LTI) or Label(LT2), and exit.
If Label(I) ≠ 0 and ≠ Label(LTI) and ≠ Label(LT2), increment Conflict Flag (I) by one and exit.

④ Otherwise, exit.

4) When the T.Pt. is a crossing Pt., let the other three line segments be LTI, LT2, and LT3.



① Choose the line segment whose angle between I is nearest to 180° and call it LTI.
If Label(LTI) ≠ 0, and Label(I) « 0, then Label(I) = Label(LTI) and exit.
If Label(LTI) ≠ 0 and Label(I) ≠ 0 and ≠ Label(LTI), increment Conflict Flag(I) by one and exit. Otherwise do the next steps.

② Label(LT2) = Label(LT3) = artery (vein)
-> If Label(I) = 0, then let Label(I) = vein (artery).
If Label(I) = artery (vein), increment Conflict Flag by one. Exit.

③ Label(LT2) ≠ 0, Label(LT3) ≠ 0 and Label(LT2) ≠ Label(LT3)
-> Increment Conflict Flag(I) by one and exit.

④ Label(LT2) » artery (vein) or Label(LT3) ■ artery (vein)
-. If Label(I) - 0, then let Label(I) = vein (artery).
If Label(I) ≠ 0 and Label(I) = artery (vein), increment Conflict Flag by one. Exit.

The labeling procedure terminates when the number of non-labeled line segments does not decrease anymore by iteration.

The isolated line segments which have no reliable initial labels are left to be non-labeled. They need another analysis. The conflicts are caused by artifactitious line segments or by disappearance of true blood vessel line segments. Since the binary blood vessels before thinning have widths at cross points, artifacts are made after thinning. In other words a cross point before thinning is splitted into two branch points after thinning. Another example of artifacts is a bridge caused by noise. The disappearance of a true vessel line segment has the same effect as a bridge. We need more details of the image in order to distinguish the disappearance from noise. From this analysis we know that the conflict flag gives us very important information on crossings, noises and disappearance of true vessel segments.

Another useful physical level knowledge which was not taken into account in our experiment is that there should be no isolated blood vessel pieces except concealments. This tells us that the interpolation of line segments could be done before labeling.

5. DISCUSSION

Here we consider some additional methods that might be useful for our problems.

1) On the extraction of blood vessels:
Another approach is to apply line detection or ridge following algorithms which are often used to find linear features in LANDSAT images. We have applied one of them to our case and got some promising results.

2) On the labeling of blood vessels:
The relaxation labeling [8] might be applied to our case. Since the pixelwise relaxation spends a great amount of time, we should use it in the stage of line segment labeling. We have to define neighbor relationships of line segments, compatibility functions, and weights. The arithmetic updating rule may be used.

3) From the view point of diagnosis:
In case of the analysis of arterial-venous crossing phenomena, some specified retinal areas are inspected. The area around the fovea which includes only very thin vessels and the area within some radius of the disc are outside the inspection. This will go a long way in overcoming the difficulty of noise elimination.

On the other hand there are important phenomena on arteries which might be neglected or erased by careless noise elimination. They are light streaks on arteries (copper wire arteries) which serves much for the diagnosis of arteriosclerosis but often acts as obstacles to labeling. This problem was not dealt with in our experiment.

6. CONCLUSIONS

The difficulty of understanding ooior fundus images lies on the fact that the image qualities are not so good as those of other medical images and the variety of colors and blood vessel shapes is very large. There is still a long distance to get the full automatic understanding of color ocular fundus images. We did not discussed the automatic diagnosis of arterio-venous crossing phenomena. In that process we will have to implement semantic level knowledge on symptoms of them into diagnostic algorithms.

ACKNOWLEDGEMENTS

The authors wish to thank Dr.s A.Tojo and Y.Shirai of the Electro Technical Laboratory for their helpful advice. Thanks are also due to Dr. S.Nakahara and Mr. S.Ikebata for their hospitality and encouragement.

REFERENCES

[1] B. H. McCormick et al., "Image Processing in Television ophthalmoscopy", in Digital Processing of Biomedical Images, edited by K. Preston, Jr. and M. Onoe, University of Tokyo Press (1976).

[2] S. Tamura etal., "Analysis of Fluorescence Fundus Angiographies", National Convention of I. P. S. J., PP. 673-674 (in Japanese) (1977).

[3] Y. Tomioka et al., "Assembling of Eye Fundus Photographs", Proc. I. P. S. J. (in Japanese), Vol. 19, No. 2, PP. 135-144 (1978).

[4] T. Yokouchi et al., "Fundus Pattern Recognition—(No. 1) Pattern Recognition on Crossing Phenomena of Artery and Vein in Fundus by Extraction of Contour Lines of Blood Vessels", Jour, of Med. Elec. & Bio. Eng. (in Japanese), Vol. 12, No. 3, PP. 9-16 (1974).

[5] T. Yokouchi et al., "Pattern Recognition of Color Fundus Photographs—(No. 2) Automatic Diagnosis of Arterio-Venoue Crossing Phenomena Utilizing Color Information", Jour. of Med. Elec. & Bio. Eng. (in Japanese), Vol. 13, No. 5, PP. 16-22 (1975).

[6] C.K. Chow and T. Kaneko, "Boundary Detection of Radio-Graphic Image by a Threshold Methods", Proc. IFIP Conf. Ljubliana, TA-7, P. 130 (1971).

[7] N. Otsu, "Discriminant and least squares threshold selection", Proc. 4th IJCPR, PP. 592-596 (1978).

[8] A. Rosenfeld et al., "Scene Labeling by Relaxation Operations", IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-6, PP. 420-433 (1976).

[9] H. Kabayama, Atlas of Fundus Photography. (in Japanese) Tokyo, Homeido (1975).

[10] T. Kanade, "Region Segmentation : Signal vs. Semantics", Proc. 4th IJCPR, PP. 95-105 (1978).

An example of color ocular fundus image

left : u-component

right: V-component

THEORY OF SELF-ORGANIZING NERVE NETS WITH SPECIAL REFERENCE TO ASSOCIATION AND CONCEPT FORMATION

Shun-ichi Amari

Department of Mathematical Engineering
University of Tokyo
Tokyo, 113 Japan

The present paper proposes a mathematical theory of self-organizing nerve nets, which is applicable to various types of supervised and unsupervised learning, such as learning decision, concept formation, association, etc. Given an environmental information source, a neural system automatically forms a number of separate routines to process the signals in it. This kind of unsupervised self-organization underlies commonly in formation of categories, feature extractors, and content addressable memories. This problem is analyzed mathematically, as well as models of topographic organization of nerve fields and of associative memories, by the proposed method.

1. INTRODUCTION

The brain needs a long period of evolution to get the present information processing manner. Although artificial intelligence is recently investigated rather independently of the behavior of nerve nets, it is plausible that artificial and natural intelligence have some common logic of realizing intelligence at a higher abstract level. They have something to suggest to each other. Neural systems seem, especially, to yield good models of parallel and distributed information processing with self-organizing capabilities of adapting themselves to the environmental information structure.

The present paper proposes a unified mathematical treatment of neural self-organization applicable to wide classes of learning with and without teacher, by summarizing the author's recent researches [1-5], We also present a model which elucidates the mechanism of unsupervised and automatic formation of various routines for processing the signals in the environment [cf. 4, 6, 7], and analyze the model by the proposed method. Recent developments of neurophysiology have revealed the importance of such self-organization. We also touch upon the models of topographic organization of nerve fields [3, 8] as well as of associative memories [9. 10].

2. A GENERAL THEORY OF SYNAPTIC MODIFICATION

We consider a simple mathematical model of a neuron, which receives a vector input signal $x = (x_1, \ldots, x_n)$ from its environment S, and emits one output signal z. These signals in general take analog values between 0 and 1, representing the normalized pulse emission rates (they may take on 0 and 1). Let us denote by $w = (w_1, \ldots, w_n)$ the synaptic efficiencies or weights of the inputs. The input-output relation is given by

$$z = f(\underline{w} \cdot \underline{x} - h) \qquad (1)$$

where h is a threshold, f is a monotonically increasing function transforming the average membrane potential $u = w.x$ into the output pulse emission rate z (Fig.l).

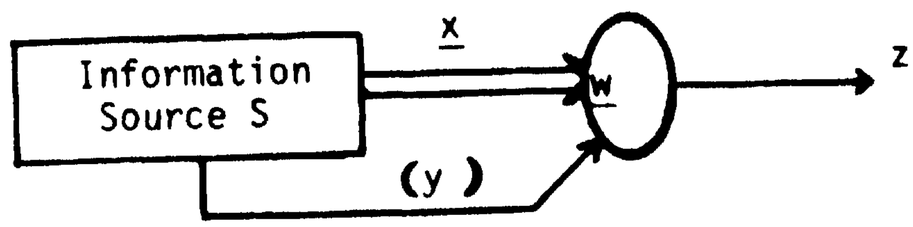


Fig.1 Model of neuron

The neuron self-organizes and adapts to its environment S, by modifying w based on an input time sequence x(t) from S. In some cases, the so called teacher signal y(t) is associated with x.(t). Hence we regard S as an erqodic information source producing (x(t), y(t)), where teacher signal y(t) is absent in the case of unsupervised learning. We consider a simple environment S represented by a probability (density) p(x, y). It chooses a pair (X, y) of signals with the probability p(x, y), each time independently, and outputs it for a fixed time duration, and then choose another, repeating this process.

We propose the following general rule of synapse modification, w being changed based on x(t) as

$$T\dot{w}(t) = - w(t) + cr(t)x(t) \qquad (2)$$

where T is a large time constant, "•" denotes the time derivative d/dt, c is a constant, and r is the reinforcement or learning signal which the neuron produces depending on w, x and y (when it exists) at that time, r(t) = r[w(t), x(t), y(t)]. This is a generalization of Hebbian type learning, which is obtained by putting $r = z$, Perceptron type learning is obtained by r = y - f(w-x_ - h). Most of the neural learning rules proposed so far are obtained by chosing appropriate signals r.

By replacing $r\underline{x}$ in (2) by its average, we obtain

$$\tau\dot{\underline{w}} = -\underline{w} + c\langle r(\underline{w}, \underline{x}, y)\rangle_S, \qquad (3)$$

where $\langle\ \rangle_S$ denotes the average with respect to $p(\underline{x}, y)$ of S, where $\underline{w}$ is considered as constant. This is not a random equation. Since S is ergodic, we can expect that the real behavior of w(t) of (2) is closely approximated by that of (3). This really is proved by the method of stochastic approximation or by the theorem of Geman [11]. Hence, we consider the average learning equation (3) as the fundamental of neural self-organization.

When r depends on $\underline{w}$ and $\underline{x}$ through $\underline{w}\cdot\underline{x}$ only, $r = r(\underline{w}\cdot\underline{x}, y)$, (3) is rewritten in the form

$$\dot{\underline{w}} = -\partial L(\underline{w})/\partial\underline{w} \qquad (4)$$

where

$$L(\underline{w}) = \underline{w}^2/2 - c\langle\int_0^{\underline{w}\cdot\underline{x}} r(u, y)du\rangle_S. \qquad (5)$$

We call L(w) the potential of learning under S. The synaptic weight w converges to one of the minimum of L. In most supervised cases, L has only one minimum, w converging to it. However, in most unsupervised cases, L has a number of minima corresponding to various aspects of S, and w converges to one of them. However, when a pool of neurons receive common inputs from S, every minimum of L is occupied by some neurons in the pool, so that the pool of neurons adapts as a whole to the environment S.

3. FORMATION OF CATEGORY DETECTING CELLS

Information processing routines are formed in the brain to be compatible with the environment by learning. This kind of learning without teacher underlies commonly in the problem of concept formation, formation of feature detectors, etc. Recent developments of neuroohysiology have revealed that the feature detecting cells are really formed depending on the visual experiences of an animal. We propose a simple model, and analyze its capabilities.

The model consists of a set of neurons receiving a common input signal x from the environment S = (p(x)}. They also receive an inhibitory

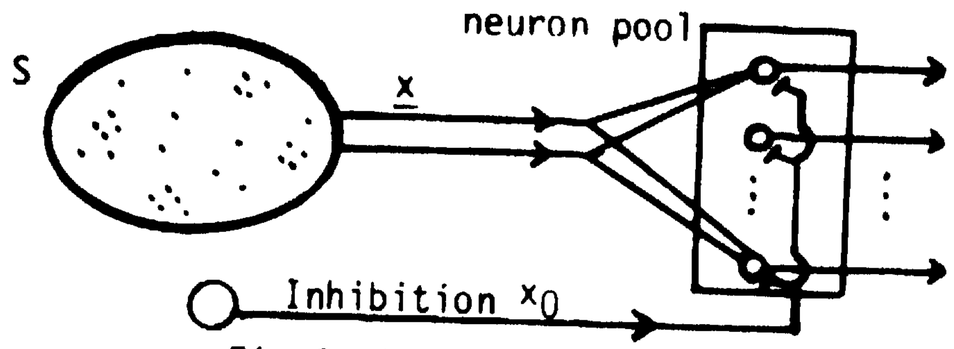


Fig.2 Category formation

input of constant intensity x_0 (Fig.2). Let us take one neuron, and let w be the modifiable synaptic weights of the neuron. Let $-W_0$ be the modifiable synaptic weight for the inhibitory input X_0.

We assume the following learning rule that the learning signal r is equal to 1 when the neuron is excited (z > 0) and 0 otherwise (z = 0), i.e.,

$$r = 1(\underline{w}\cdot\underline{x} - w_0x_0), \qquad (6)$$

wherel(u) is 1 when u > 0 and 0 otherwise, and we pnt h = 0. The average learning equation is

$$\dot{\underline{w}} = -\underline{w} + c\langle 1(\underline{w}\cdot\underline{x} - w_0x_0)\underline{x}\rangle_S, \qquad (7)$$

$$\dot{w} = -w_0 + c'x_0\langle 1(\underline{w}\cdot\underline{x} - w_0x_0)\rangle_S. \qquad (8)$$

The equations are in general multi-stable, having a number of equilibria.

Let us consider a subset A of the signals in S. A *nerve* cell is said to be a detector or a representative cell of A, when it is excited by any signal in A but not excited by any others in S. We say that a subset A forms a category under S, when a detector of A is obtained as a stable equilibrium by self-organization. In other words, a subset A is a category, when a set of weights $\underline{w}^A, w_0^A$ satisfying

$$\underline{w}^A\cdot\underline{x} - w_0^Ax_0 > 0$$

for all $\underline{x} \in A$ but not for any $\underline{x} \in S - A$, is obtained as stable equilibrium of (7) and (8).

Many categories are formed by learninq under S. When the neuron pool includes many neurons of different initial weights, they differentiate automatically to be detectors (representatives or processors) of these categories. Mutual interaction among the neurons of lateral inhibition type prevents a large number of neurons becoming the detector of one and the same category. The problem is to know which subsets become categories under S. We give a n.a.s.c. for a subset A to be cateqory under S. Let x^A be the average of signals in A. $x^A = \int p(\underline{x})\underline{x}d\underline{x}$. and $\lambda = c'x_0^2 c$.

Let $g_A(X) = xA.x$ be the inner product ,

Theorem. A necessary and sufficient condition that A is a category under S is

$$\min_{\underline{x}\in A} g_A(\underline{x}) > \lambda > \max_{\underline{y}\in S-A} g_A(\underline{y}).$$

The theorem shows that a cluster of signals of an adequate diameter forms a cateqory, where the diameter is specified by the parameter x. In other words, λ specifies the resolution of a category, where λ can be controlled by changing XQ. This yields a primitive mechanism of formation of information processing routines by learning.

4. TOPOGRAPHIC ORGANIZATION

Topographic organization is found in many parts of nerve fields in the brain. For example, the continuous projection from the retina to the striate cortex is known as the retinotopy. It is known that, when part of the retina (or striate cortex, or both) is removed, the regeneration of connections takes place such that the topological map is completed between the remaining parts. This suggests that topographic organization is formed based, at least partly, on the self-organizing ability of nerve cells.

Let us consider a simple model consisting of a presynaptic nerve field X and postsynaptic nerve field Y connected by modifiable synapses. Stimuli are supplied from the environmental information source S to the presynaptic field X, exciting neurons of X, and then those of Y. The self-organization takes place based on these excitations. In this case, S brings the topological information of X, in such a way that two nearby neurons of X are frequently stimulated at the same time. When the neurons in Y have fixed mutual connections of lateral-inhibition type and when they have additional modifiable inhibitory synapses, we can prove that continuous connections from X to Y (topographic organization) is formed by Hebbian type unsupervised learning. Moreover, we can prove that some microstructures are also formed in some cases, as is shown in the columnar structures in the cerebrum. The fundamental equations of the model are much complicated, because we should solve the field equations of neural excitation together with the average learning equations. We can solve them.

5. PRIMITIVE MODEL OF ASSOCIATION

Let us consider a pool of neurons, which receive a common input signal x_ and output a vector signal z (the i-th neuron emitting the i-the component z_i). We consider such an environment S that consists of a finite number of signal pairs (X_J, Y_j), where the i-th component Yji of y_j acts as the teacher signal to the i-th neuron. The problem is to modify the synaptic weights $\underline{w}_i$ of the i-th neuron (or synaptic weight matrix $W = (\underline{w}_i)$) such that, receiving an input $\underline{x}_j$, the neuron pool outputs the associated signal $\underline{y}_j$. In this case, one many say that the net recalls $\underline{y}_j$ from the associated $\underline{x}_j$, and that the associated pairs $(\underline{x}_j, \underline{y}_j)$'s are memorized in the synaptic weights W in a distributed and superimposed manner [10].

The correlation learninq [10] is obtained, if each neuron self-organizes with r = y. In this case, W converges to $W = \Sigma\, \underline{y}_j T\underline{x}_j$, T denoting the transposition. This learninq works well only when $\underline{x}_j$'s are mutually orthogonal. The synaptic weights converges to $W = \Sigma\, \underline{y}_j\underline{x}_j^*$ by orthogonal learning [2] with $r = y - \underline{W}\cdot\underline{x}$, where $\{\underline{x}_j^*\}$ is the dual orthogonal system of $\{\underline{x}_j\}$ (i.e., $\underline{x}_j^*\cdot\underline{x}_i = \delta_{ji}$), and associative recall can be done even when $\underline{x}_j$'s are not orthogonal. We can also treat the dynamic recall of signal sequences $\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_t$ [9].

REFERENCES

[1] S. Amari "A Method of Statistical Neurodynamics". Kybernetik, 14 (1974) 201-215

[2] S. Amari "Neural Theory of Association and Concept-formation". Biol. Cybernetics, 26 (1977) 175-185

[3] S. Amari "Topographic Organization of Nerve Fields". to appear

[4] S. Amari and A. Takeuchi "Mathematical Theory of Formation of Category Detecting Nerve Cells". Biol. Cybernetics, 29 (1978) 127-136.

[5] A. Amari and M.A. Arbib "Competition and Cooperation in Neural Nets". In Systems Neuroscience (J. Netzler ed.), Academic Press, (1977) 119-165

[6] C. von der Malsburg "Self-organization of Orientation Sensitive Cells in the Striate Cortex". Kybernetik, 14, (1973) 85-100

[7] K. Fukushima "Cognitron: A Self-organizing Multilayered Neural Network". Biol. Cybernetics, 20 (1975) 121-136

[8] A. Takeuchi and S. Amari; to appear

[9] S. Amari "Learning Patterns and Pattern Sequences by Self-organizing Nets of Threshold Elements". IEEE Trans., C-21 (1972) 1197-1206

[10] T. Kohonen Associative Memory. Springer-Verlag, 1977

[11] S. Geman "Some Averaging and Stability Results for Random Differential Equations". SIAM J. Math. 36 (1979) 86-105

A LEARNING SYSTEM AND ITS PSYCHOLOGICAL IMPLICATIONS

John R. Anderson
Department of Psychology
Carnogio-Mftllon University
Pittsburgh, PA 15213

Paul *1* Kline
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

ACT la a computer simulation program that uses a propositional network to represent knowledge of facts and a set of productions (condition - action rules) to represent knowledge of procedures. There are currently four different mechanisms by which ACT can make additions and modifications to its set of productions: designation, strengthening, generalization, and discrimination. Designation refers to the ability of productions to call for the creation of new productions. Strengthening a production involves adjusting the amount of system reaourcea that will be allocated to its processing. Finally, generalization and discrimination refer to complementary processes that produce better performance by either extending or restricting the range of situations in which a production will apply. Theae learning mechanisms are uaed to simulate experiments on prototype formation. ACT successfully accounts for the effects of distance of instances from a central tendency, frequency of individual instances, and the family resemblance structure of categories.

Introduction

ACT is a theory that combines ideas from cognitive psychology and artificial intelligence. It is both a performance theory and a learning theory. As a performance theory ACT is concerned with the factors that determine how quickly, how reliably, and in what ways humans can perform a cognitive task. As a learning theory it it concerned with how the knowledge needed to perform such a task is acquired and properly integrated.

The behavioral domains we have tried to model with the performance aystem include memory, inference, and language. This work is described in Anderson (1976), Anderson, Kline, and Lewis (1977), Anderson and Kline (1977). Our learning research has concentrated in the past on language acquisition, but more recently we have also become interested in modelling learning of high school geometry and learning of LISP by programming novices. Preliminary reports of this work are Anderson, ine, and Beasley (1977, 1979a, 1979b). This paper focuses on how the ACT theory would apply to prototype formation. There is a considerable body of empirical research on this topic in cognitive psychology.

The ACT system falls at the intersection between cognitive psychology and artificial intelligence. Many of the concepts described in this paper can already be found in the fields of knowledge representation and knowledge acquisition but there are some new concepts developed out of detailed consideration of human behavior. The ACT system architecture provides a novel synthesis of these individual concepts. With these concepts and architecture we are able to account in detail for some important psychological phenomena. Thus, we hope that this paper will provide researchers in Al with a few new concepts but mainly a new perspective on familiar concepts and their possible combination. We also hope that the paper will indicate to researchers in cognitive psychology new potentials for some of the conceots in artificial intelligence.

The ACT System

In ACT knowledge is divided into two categories: declarative and procedural. The declarative knowledge is represented in a propositional network, which is similar to other semantic network representations (Ouillian, 1969; Anderson and Bower, 1973 Norman and Rumelhart, 1975; Shapiro, 1971; Simmons, 1973). While the network aspects of this representation are important for such ACT processes as spreading activation, they are not important to the current learning discussion. For present purposes we will consider ACT's declarative knowledge as a set of assertions or propositions and ignore the technical aspects of its network representation.

ACT represents its procedural knowledge as a set of productions. The ACT production system can be *seen* as a considerable extension and modification of the production systems developed at Carnegie-Mellon (Newell, 1972, 1973; Rychener and Newell, 1978). A production is a condition-action rule. The condition is an abstract specification of a set of propositions in the network. To be matched to the condition, the propositions must be *active.* ACT's activation mechanism is designed such that the only active propositions are those that have recently been added to the data base or that are closely associated to such propositions. Propositions are added to the data base either through input from the environment or through the execution of productions. Thus, this activation system gives ACT the property of being responsive to changes in its environment or in its internal state.

ACT's basic control structure is an iteration through successive *eyeles,* where each cycle censists of a production selection phase followed by an execution phase. On each evete an APPLVLIST is computed which is a probabilistically defined subset of the productions whose conditions have all their constants active in the data base. The probability that a production will be placed on the APPLYLIST depends on the *strength (s)* of that production relative to the sum *(S)* of the strengths of all the productions whose conditions mention active elements; that is. this probability vanes with *s/S.* Discussion of the process of assigning a strength to a production will be postponed until a later section; all that needs to be said here is that this strength reflects just how successful past applications of this production have been. Thus one component of the production-selection phase consists of choosing those productions which are the most likely to apply successfully.

II Learning In ACT

ACT can learn both by adding propositions to its data base and by adding productions. It can also learn by modifying the strengths of its propositions and productions. We will concentrate here on the learning that involves productions. Production learning tend* to involve the more significant events of cognitive restructuring. It is also through production learning that ACT accounts for prototype formation. Productions can be added to the data base in one of two ways. They can be added by detiberate designation as in the encoding of

instructions or they can be added through the spontaneous restructuring of productions in response to experience. Our focus in this paper will be on spontaneous restructuring. The designation process plays only a small role in our modelling of the prototype formation literature.

Generalization

A very important component of the spontaneous learning process involves generalization, ACT uses an algorithm which finds the maximal common generalization of two productions in the sense developed by Vere (e.g. 1977) and Hayes-Roth (e.g. 1976). The basic idea behind this generalization technique is to compare *pairs* of similar productions P_1 and P_2 *and* to form new productions P_3 which have the following characteristics:

(a) P_3 will apply in the circumstances that either P_1 or P_2 did and possibly new circumstances)
(b) P_3 will have the same effect as P_1 or P_2 in the circumstances that P_1 or P_2 applied
(c) There is no production P_4 that satisfies (a) and (b) and only applies in a subset of the circumstances that P_3 does.

The maximal common generalisation is not always unique; in which case ACT selects in a random fashion. These generalizations are formed basically by deleting clauses in the conditions of P. and P_2 and by replacing constants by variables.

We have a number of heuristics to help direct the generalization process, but obviously have no general solution to the NP-complete partial matching problem embedded in the process of generalization (see Hayes-Roth, 1977). Most of the examples we run into *seem* to be computationally tractable. Examples can be designed that push the computational limits of the system but they do not *seem* particularly easy generalization problems for humans either. We have been concerned in ACT with developing (a) principles for deciding when to attempt forming generalizations in a realistically large system with very many productions; and (b) principles of caution for integrating generalizations (there is always a danger of overgeneralization) into the operation of the system.

An example will help to illustrate ACT'S automatic generalization mechanism. Figure 1 illustrates the stimuli studied by subjects in Experiments 3 and 4 of Franks *and* Bransford (1971). We will *assume* that subjects designate productions to recognize each stimulus. So for the first stimulus item subjects would designate the following production:

P1: IF a *triangle* is to the right of a *circle*
and a *square is* to the right of a *heart*
and tha first pair la *above* the second pair
THEN this is *an instance* of tha study material

For the third stimulus the following production would be designated:

P2: IF a *circle is* to the right of *a triangle*
and • *square* Is to tha right of a *heart*
and the first *pair is above* tha second pair
THEN this is *an* Instance of the study material

From these two productions a generalization can be formed that captures what these two productions have in common. This involves deleting terms on which the two productions differ and replacing these terms by local variables. Thus, we have the following generalization.

P3: IF a *LVshape) is* to the right of *a LVshape2*
and a *squaro* Is to the right of a *heart*
and the first pair is *above* the second pair
THEN this is at instance of the study material

The local variables in mis generalization aie -prefaced by LV. This generalization can be thought of as an attempt on ACT's part to arrive at a more general characterization of the study material. Nolo that ACT's generalization mechanism *needs* only two *examples* to propose a generalization. This generalization does not replace the original two but rather co-exists with them as an alternate means of characterizing the stimulus set. Which production will actually produce the response depends on the strength mechanism that we will describe shortly.

Restrictions are needed on how *many* elements can be deleted *in* making a generalization. Consider. ACT's representation for the sixth stimulus from the Franks and Bransford set:

P4: It- a *circle is* to the right of a *triangle*
and a *heart* is to the right of a *blank*
and the first pair is *above* the second pair
THEN this It an Instance of the stimulus Material

If we allowed this stimulus to be generalized with stimulus 1 (P1) we would get a production that would accept any array of four geometric objects as an instance of the study material. While it is conceivable that any such array may be an experimental stimulus, this seems like too strong a generalization to make just on the basis of these two examples. Therefore, a limit is placed on the *proportion* of constants that can be replaced by variables in forming a generalization. Of the two productions from which the generalization is formed, the one with the least number of constants is used as a reference. The number of constants in the generalization can bo no fewer than half this quantity. The terms that ACT considers constants are italicized. There are five constants in productions P1, *P2,* and P3. Production P3 is an acceptable generalization from P1 and P2 because it only involves replacement of two of the constants. P1 and P4 are prevented from forming a generalization because this would *require* replacing four of the five constants with variables.

Even with this restriction on the proportion of constants deleted it is likely that unacceptably *many* generalizations will be formed. A realistic simulation of an adult human's entire procedural knowledge would requiro hundreds of thousands of ACT productions. Under these circumstances it seems infeasible to attempt to generalize all possible pairs of productions. ACT only attempts to form generalizations when a new production has been designated. Also, generalizations *are* attempted only for pairings of newly-designated productions with the productions on the APPLYLIST. Since a production is on the APPLYLIST only if the constants it references are active and it has met a strength criterion, this implies that attempts to generalize will be restricted to productions that are relevant to the current context and which have enough strength to indicate a history of past success.

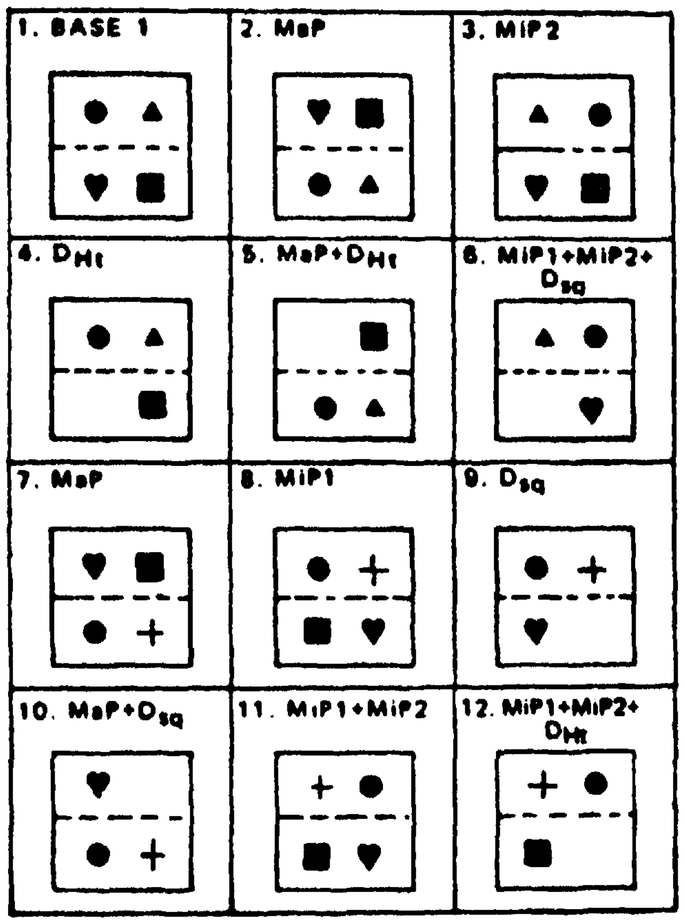


Figure 1. Examples of material from the Franks and Bransford study.

The actual syntax of ACT productions is considerably more complex than this. We present them in a pseudo-English to facilitate readability.

Discrimination

Even with mese restrictions placed on it, ACT'S generalization mechanisms will produce productions that are overgeneralizations of the desired production. However, given our goal of a psychologically realistic simulation, such overgeneralizations on ACT'S part are actually desirable since it can be shown that people make similar overgeneralizations. For example, children learning language (and, it appears, adults learning a second language--see Bailey, Madder, and Krashen, 1974) overgeneralize morphemic rules (Brown, 1973). Thus a child will generate *mans, glved,* etc. ACT will do the same.

Use of a discrimination mechanism seems relatively unique in the knowledge acquisition literature--Winston's (1972) use of "near misses" being the only related mechamsm we are aware of. The use of discrimination is motivated by the fact that any system that tries to improve itself by making generalizations must, occasionally, make overgeneralizations. Discrimination provides an automatic means for "debugging" such *errors.* To correct overgeneralizations ACT must create more discriminate productions.

The discrimination mechanism will only attempt to discriminate an overgeneral production when it has both a correct and an incorrect application of that production to compare. (Productions place new propositions into the data base and emit observable responses; either of these actions can be declared incorrect by a human observer or by ACT Itself. In the absence of such a declaration an action is considered correct.) Basically, this algorithm remembers and compares the variable bindings in the correct and incorrect applications. It attempts to find a variable that had different bindings in these two applications. In forming the discrimination, restrictions are placed on that variable to prevent the match that led to the unsuccessful application while still permitting the match that led to the successful application. In the simulations of schema abstraction that will be discussed, a new production was formed from the old production simply by replacing the variable by the constant it was bound to during the successful application. in some of our other work (e.g. Andorson, Kline, & Beasley, 1979a) the variable was kept but an additional clause was added to restrict what the variable could match.

As an example of the discrimination process, we will consider a categorization experiment from Medim and 5chaffer (1978). We will focus on two instances they presented from category A. One was *two largo rod trltnglos* and the other was *two large bluo circles.* From these two examples, ACT would designate the following categorization productions.-

P6t IF a stimulus has two large red triangles
THEN it it in category P
P7: IF a stimulus has two large blua circlet
THEN It is In catenary ft

From these two ACT would form the following generalization:

P8» IF a stimulus has two larq« LVcolor LVshapes
THEN It is in Category R

However, this turned out to be an overgeneralization. To be in category A the stimulus had to be either red or a circle or both. Thus, the counter-example was presented of two large blue triangles which was a stimulus in category B. Generalization PS misapplied in this circumstance. By notina what distinguished the circumstances of correct applications of generalization P8 from the circumstances of incorrect application, both of the following productions would eventually be formed by the discrimination mechanism. These productions will always produce correct classifications.

PS: IF a stimulus has two large red LVshapes
THEN it is in category R
P10: IF a stimulus has two large LVcolor circlet
THEN it It In cateqory. R

These produetiom were formed from P8 by replacing one of its variables (either IVcolor or IVshapes) by the binding that variable had during a successful application -i.e. an application to a stimulus that was actually from category A. (As an aside, these two productions illustrate how ACT can encode disjunctive concepts by the use of multiple productions).

Production Strength and Specificity

When a new production is created by the designation process there is no assurance that its condition is really the best characterization of the circumstances in which its action is appropriate. For this reason, generalization and discrimination processes exist to give ACT the opportunity to evaluate alternative conditions for this action. It is the responsibility of ACT's strength mechanisms to perform the evaluation of these alternative productions.

Through experience with the ACT system we have created a set of parameters which, although somewhat arbitrary, appe*ar* to yield human-like performance. T'ee first time a production is created (by designation, generalization, or discrimination) it is given a strength of .1. Should that production be recreated its strength is incremented by .05. Furthermore, a production has its strength incremented by .025 every time it applies or a production consistent with it applies. (One production is considered consistent with another if its condition is more general and its action is identical.) Finally, whenever a production receives negative feedback, its strength is reduced by a factor of 1/4. Productions consistent with the misapplied production also have their strength reduced by a factor of 1/4. Since a multiplicative adjustment produces a greater change in strength than an additive adjustment, a "punishment" is more effective than a "reinforcement". Since increments and decrements in strength propagate to more general productions and since qeneral productions can have a number of more specific productions consistent with them, they tend to reflect quite rapidly and sensitively the weight of experience.

Recall that a production's strength determines the probability that it will apply. If s is the strength of a production and S the total strength of all active productions, the probability of that production being chosen on a cycle for application is *1-o* where o is a parameter currently set at 15. If a production passes this probabilistic hurdle it is placed on the APPLVUST. of course, if a production is not applied one cycle and the circumstances do not change, it can apply on a later cycle. Thus, strength affects both the latency and reliability of production application.

While selection rules based on strength can make some of the required choices among competing productions, it is clear that strength cannot be the sole criterion for conflict resolution. For example, people reliably generate irregular plurals (e.g., *mon)* under circumstances in which the "add s" rule for regular plurals is presumably also applicable. This reliable performance is obtained despite the fact that the productions responsible for generating regular plurals are applied much more frequently than those for irregulars and therefore should bo much stronger. ACT has a specificity ordering on its productions which makes it possible to deal with exceptions to strong rules: If two productions both have enough strength to be placed on the APPLVLIST but the condition of one of them is a more specific version of the condition of the other (i.e. satisfied in a subset of the situations that satisfy the other), then the more specific production will apply and will block out the more general production. This principle accounts for the execution of a production generating an irregular plural since its condition presumably contains all of the requirements for generating the regular plural and must, in addition, make reference to the specific noun to be pluralized. This special case principal can be traced back to Waterman (1970) and variants of it can be found in many current production systems (e.g. Rychener ft Newell, 1978).

The precedence of exceptions over much stronger general rules does r>ot imply that exceptions always apply. In order to benefit from the specificity-ordering principle exceptions must first have achieved the amount of strength necessary to be placed on the APPLYLIST. This property of the ACT model is consistent with the fact that words with irregular inflections have high frequencies of occurrence.

Product ion strength is another important way in which ACT differs from moat other computer-based learning systems (e.g., Anderson, 1978; Vere, 1977; Hayes-Roth & McDermott, 1976; Sussman, 1975;

Winston, 1970; Waterman, 1970. The learning of all these systems has an all-or-none character that ACT would share if creating new productions were its only learning mechanism. The strength mechanisms modulate the all-or-none character of production creation enabling ACT to cope with the kind of world that people have to cope with--a world where data is not perfectly reliable and contingencies change in such a way that even being as cautious as possible it it certain that occasional errors will be made.

III Applications to Prototype Formation

There is a growing literature concerned with the process by which humans form concepts by detecting regularities among stimuli (e.g., Franks & Bransford, 1971; Hayes-Roth & Hayes-Roth, 1977; Newmann, 1974; Posner & Keole, 1970; Reed, 1972; Reitman & Bower, 1973; Rosch 8. Menvis, 1975). This literature is often referred to rs studying *protrtype formation,* but for various reasons we also refer to it as studying s*chema abstraction.*

In the remainder of this paper we describe how ACT's automatic learning mechanism can be used to model schema abstraction. In outline, this application is as follows: For each instance presented, ACT designates a production that recognizes and/or categorizes that instance alone. Generalizations occur through the comparison of pairs of these productions. If feedback about the correctness of these generalizations is provided then the discrimination process can be evoked. Our working definition of a concept will be this set of designations, generalizations, and discriminations.

Franks and Bransford: Illustration of Basic Phenomena

We have already introduced (gure 1) the material used by Franks and Bransford in one of their experiments on schema abstraction. Subjects studied the 12 picture* in Figure 1 twice and then were transferred to a recognition phase in which they rated test pictures according to whether they had been studied or not. The test pictures could be classified according to how many transformations separated them from the central tendency of the study stimuli. There were test stimuli 0, 1, 2, or 3 transformations from the study stimuli and some "non-cases" which were still further removed. Some of the test figures were actually studied and some were not. Franks and Bransford report that confidence ratings forecognition generally decreased with the number of transformations from the base and was lowest for the non-cases.

To simulate the Franks and Bransford experiment we ran ACT through propositional encodings of the items in the study set twice, designating a recognition production for each stimulus it saw. Then at test ACT was again presented with a prepositional encoding of each stimulus and the production which applied to this encoding (if any) was noted. Sufficient generalization had occurred so that most of the stimuli were recognized by at least one of the productions.

Since most experiments in this literature report data on subjects confidence in their judgments, a critical question was how to map the production selected onto a confidence rating. We assumed that ACT's confidence would be a function of the number of consfants in the stimulus (and therefore an inverse function of the number of variables). This procedure for assigning confidence will be used throughout this paper. This is a reasonable procedure tor assigning confidence, since the more constants in the recognizing production the closer it is to an encoding of an actual tost item. In the extreme, if the stimulus is recognized by a production with no variables the subject can be sure that the item was studied since a non-vahabilized production is an encoding of a study dem.

To obtain predictions for this experiment we ran a *series* of ACT simulations. Altogether we obtained fifty ratings for each test stimulus and the data we report will be based on averages of these fifty ratings.

O-transformation test stimuli were given a mean rating of 1.66 (i.e. mean number of constants in matching productions)] the one-transformation test stimuli were rated 1.24; the two-transformation stimuli were rated 1.11; the three-transformation stimuli were rated 1.13; and the non-cases were rated .65. This corresponds to Franks and Bransford's report of an overall correlation between closeness to base and rating. (They do not report the actual ratings.) Neumann (1971) performed a replication of Franks and Bransford and he did report mean ratings for each of the five categories of test stimuli. The confidences assigned to the stimuli by his subjects corresponds exactly in rank ordering to the results obtained from ACT.

This experiment serves to illustrate that ACT can account for one of the basic phenomena of schema abstraction -- namely that confidence falls off with distance of the stimuli from the central tendency (prototype) of the category. The generalizations ACT forms represent what various stimuli will have in common. Therefore, there will be more generalizations formed that match central stimuli than ones that match peripheral stimuli. it *is* this greater density of generalization that give central stimuli this advantage.

Hayes-Roth and Hayes-Roth: Variation of Instance Frequency

Hayes-Roth *and* Hayes-Roth (1977) report a study, one function of which was to obtain data relevant to the issue of memory for instances. They presented subjects with three-attribute descriptions of more than one hundred people. One attribute was age and could have values 30, 40. 50, and r>0. Another was education and could have values junior high, high srhool, trade school, and college. The third was marital status which could have values single, married, divorced, and widowed. The descriptions also included proper names and hobbies but this information was not critical. Thus, a subject might hear the description "John Doe, 30 years old. Junior high education, single, plays chess." Subjects' task was to learn to classify these individuals as members of club 1, members of club 2, or neither club.

We will assume that for each individual encountered, subjects designated a production mapping that individual's features into a prediction about club membership. So, for instance, a subject might form the following production:

If a person is forty years old
and ha has gone to high school
and ha it single
Then he is a member of club 1

Hayes-Roth and Hayes-Roth varied the freguency with which various instances were studied. Some instances were presented 10 times while others only once. A study trial consisted of first presenting the subject with an instance, asking him to classify it, and then providing feedback as to which club the instance came from. Some instances received equivocal feedback in the sense that half the time the feedback for those instances specified club 1 and half the time club 2.

After studying a set of stimuli, subjects were shown a critical test set of 28 stimuli. Subjects were first asked to categorize each of the stimuli and then they were asked to decide whether each of the stimuli had been studied or not. The recognition judgment was assigned a confidence from 1-5 as was the categorization Judgment.

This exporiment was simulated with the same parameter settings as the Franks and Bransford experiment. The one significant difference was that ACT was given feedback about the correctness of its classifications. This meant that productions would not simply *increase* in strength with every application, but rather would either increase or decrease in strength depending on their success in classification. Providing feedback also meant that it was possible for ACT to compare variable-bindings on successful applications in order to prooduce more discriminating versions of its overgeneral productions.

As in the Franks and Bransford experiment, confidence was based on

the number of constants in tho production that recognized the stimulus. In this experiment that number would vary from 1 to 3. A value of 0 was assigned if no production was evoked to categorize the stimulus. The categorization scores for a test stimulus were calculated by weighting negatively the confidences of incorrect classifications, weighting positively tho confidences of correct classifications, and ignoring the confidences of classifications to the neither-club category. Figure 2 presents the actual and predicted recognition and classification confidences for the soven categories of test stimuli. The confidence scale used by subjects and the match scale used by ACT have been adjusted to cover approximately equal ranges. As can be seen the ACT predictions closely correspond to the data obtained by Hayes-Roth and Hayes-Roth.

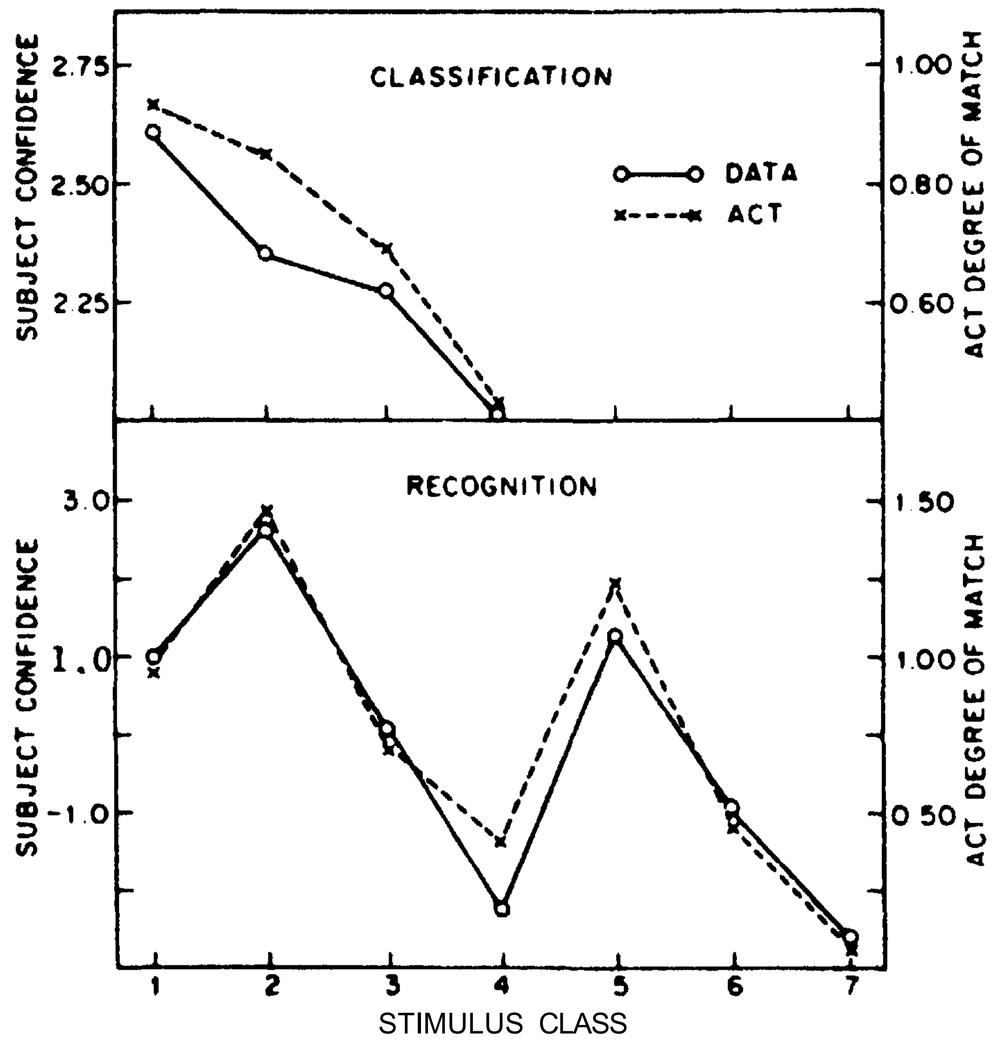


Figure 2, Comparison of ACT's predictions with the results from Hayes-Roth and Hayes-Roth.

Class 1 in Figure *2* is formed from two prototypes which, in fact, were never studied. However, they received the highest categorization rating and a relatively high recognition rating indicating that subjects havo extracted the central tendencies of the categories. Class 2 items, while not as similar to the prototypes as Class 3 items, had categorization ratings second only to the prototypes themselves. Since each item in Class 2 was studied ten limes while items in Class 3 were only studied once, it appears that frequency of exposure has an effect on categorization that can compensate for lack of similarity to the prototypes. A comparison of categorization ratings for Class 3 versus Class 4 holds frequency of exposure constant and the effect of similarity to prototypes re-emerges: Class 3 is closer to the prototypes and receives higher ratings.

Items in the last three classes were neutral with regard to category membership so that correct categorization is undefined and only recognition confidences can be reported. Whereas the data reviewed earlier from Classes 1-4 showed that frequency of exposure affects categorization ratings when we might have expected it only to affect recognition ratings; the data from Classes 5-7 shows that similarity to prototypes affects recognition when we might have expected it only to affect categorization. Class 5 has the same exposure as Class 2 so only its distance from the prototypes can explain its lower recognition ratings. Similarly, no instances of either Class 6 or Class 7 was ever studied by subjects, but Class C received higher recognition ratings and this must be due to the fact that, although very far from the prototypes. Class 6 is not as far from them as is Class 7.

In summary, this experiment serves to illustrate how ACT correctly models the combined effects of frequency of exposure and closeness to prototype. Stimuli that are frequently studied will have fairly strong specific productions to recognize them. Stimuli which are close to the prototype will have many generalizations that can recognize them.

Comparison of ACT with Other Prototype Theories

There are three basic types of theories for prototype formation. One type proposes that subjects form a single characterization of the central tendency of the category. A frequent suggestion is that they distinguish a particular instance (it need not be one they have actually seen) as the prototype for the concept. Other instances are members of the category to the extent that they are similar to this prototype. This class of theories would include Franks and Bransford (1971), Bransford and Franks (197?), Ro.sch and Mervis (1975), Posner and Keele (1968), and Reed (1972). In order to account for the effects of instance frequency demonstrated by Hayes-Roth and Hayes-Rotr the prototypes would have to be auqmented by some memory for the individual instances studied. However, it is much more difficult for prototype theories to accomodate the recent results of Medin and Schaffer (1978) that indicate that subjects are sensitive to similarities among individual instances. On the other hand ACT is able to simulate this data (Anderson, Kbne, & Beasley, 1979b).

A second class of theories (e.g. Medin & Schaffer, 1978) are those that propose subjects store individual instances only, and make their category judgments on the basis of the similarity between the test instance and the stored instances. In a certain sense, any results that can be accounted for by a theory that says that subjects store abstractions can also be accounted for by a theory that says subjects only store instances. The instance theory could always be made to go through a test process equivalent to calculating an abstraction from the stored instances and making a judgment on the basis of the abstraction. However, a difficulty for the instance theory is that subjects frequently report having abstract characterizations or prototypes (e.g., Reed, 1972).

The third class of theories proposes that subjects store co-occurrence information about feature combinations ACT »s an instance of such a theory as are those proposed by Reitman and Bower (1973), Hayes-Roth and Hayes-Roth (1977), and one aspect of Neumann's (1974) model. These models can potentially store all subsets of feature combinations. Thus, they store instances *as* a special case. The Hayes-Roth and Hayes-Roth experiment showed this model has advantages over many versions of the instance-only or prototype models.

It is very difficult to find empirical predictions that distinguish ACT from the various other feature-set theories. Perhaps it would be best to regard them as equivalent given the current state of our knowledge and simply conclude that subjects respond in terms of feature-sets. However, there are a number of reasons for preferring ACT's version of the feature-set theories. First, it is a fully specified process model. It is often difficult to see in *any* detail how some of the feature-set theories apply to particular paradigms or produce particular results.

Second, ACT has a reasonably efficient way of storing feature-sets. It only stores those subsets of properties and features that have arisen because of generalization or discrimination rather than attempting to store all possible subsets of features from all observed instances. While there are empirical consequences of these different ways of storing feature-sets, the differences are so subtle that existing experiments havo failed to test them. However, if there is very little difference in behavior, that would seem to be all the more reason to prefer the more efficient storage requirements of ACT.

Third, it needs to be emphasized that the ACT learning mechanisms were not fashioned to account for schema abstraction. Rather they were designed in light of more general considerations about the nature of the rules that need to be acquired And the information typically available to acquisition mechanisms in real world situations. We were particularly concerned that our mechanisms should be capable of dealing with language acquisition and acquisition of rules for making Inferences and predictions about one's environment. The mechanisms were designed to both be robust (in being able to deal with many different rules in many different situations) and to be efficient. Their success in accounting for schema abstraction represents an independent confirmation of the general learning theory.

Before concluding, we would like to discuss one characteristic of feature-set models which may seem unappealing on first encounter. This is the fact thnt they store so many different characterizations of the category. ACT *may* not be so bad as some of the other theories, yet having a set of productions for recognizing instances of a category still seoms far less economical than having a single prototype. However, two *remarks need* to be made here. The first remark is that the complexity of the representation of a category depends on one's production system implementation. This can be quite complex if each production is represented separately. However, the representation can be quite economical in a data-flow system like Forgy's (1977). *Here* a single discrimination net is constructed to encode the conditions of all productions. This allows overlapping productions to share the same net tests in encoding their conditions.

A second remark is that natural categories defy economical representation. This has been stressed in discussions of their family resemblance structure by Wittqenstem (o.g. Wittgenstein, 1953) and more recently by Rosch (c.g. Rosch 8. Mervis, 1975). The important fact about many natural categories (e.g., games, dogs) is that there is no set of features that define the category nor is there a prototypical instance that functions as a standard to which all other category members must be compared. On the other hand, these categories dc not seem to be unstructured; they are not merely a list of instances. It is interesting to note here that Dreyfus (1972) has claimed that computer implementations are not capable of representing the family resemblance structure of categories. ACT shows Just the opposite to be true.

IV References

[1] Anderson, J.R. *Language, memory and thought.* Hillsdale, N.J.: Lawrence Erlbaum Associates, 1976.

[2] Anderson, J. R. Induction of Augmented Transition Networks. *Cognitive Science,* 1977, *1,* 125-157.

[3] Anderson, J.R. ft Bower, G.H. *Human Associative Memory.* Washington: Winston ft Sons, 1973.

[4] Anderson, J.R.. Kline, P.J., ft Beasley, CM. Complex learning processes. In R.E.Snow, P.A. Fedenco, ft W.E. Montague (Eds.), *Aptitude, Learning, and Instructlont Cognitive Process Analyses,* 1079a, in press.

[5] Anderson, J.R., Kline, P.J., and Beasley, CM. A general learning theory *and* its application to Schema Abstraction. In G.H. Bower (Ed.) *The Psychology of Learning and Motivation,* Vol. 13, 1979b. in press.

[6] Anderson, J.R., Kline, P.J., ft Beasley, CM. A theory of the acquisition of cognitive skills. ONR Technical Report 77-1, Yale University, 1977.

[7] Anderson, J.R., Kline, P.J., ft Lewis, C. A production system model for language processing. In P. Carpenter ft M. Just (Eds.), *Cognitive Processes In Comprehension.* Hillsdale, N.J.: Lawrence Erlbaum Associates, 1977.

[8] Bailey, N., Madden, 0., ft Krashen, S.D. Is there a "natural sequence" in adult second language learning. English Language Institute and Linguistics Department, Queens College and the Graduate Center, City University of New York, 1974.

[9] Brown, R. A *first language.* Cambridge, Mass.: Harvard University Press, 1973.

[10] Bransford, J.D. ft Franks, J.J. The abstraction of linguistic ideas. *Cognitive Psychology,* 1971, 2, 331-350.

[11] Dreyfus, H. L. *What Computers Can't Do.* New York: Harper and Row, 1972.

[12] Forgy, C. L. A production system monitor for parallel computers. Technical Report, Computer Science Department, Carnegie-Mellon University, 1977.

[13] Franks, J.J. ft Bransford, J.D. Abstraction of visual patterns. *Journal of Experimental Psychology,* 1971, 90, 65-74.

[14] Hayes-Roth, B. ft Hayes-Roth, F. Concept learning and the recognition and classification of exemplars. *Journal of Verbal Learning and Verbal Itehavlor,* 1977, *16,* 321-338.

[15] Hayes-Roth, F. The Role of Partial and Best Matches in Knowledge Systems, Rand Corporation, P-5802, 1977.

[16] Hayes-Roth,F. ft McDormott, J. Learning structured patterns from examples. Proceedings of the Third International Joint Conference on Pattern Recognition, 1976, 419-423.

[17] Medm, D.L. ft Schaffer, M.M. A context theory of classification learning. *Psychological Review,* 1978, in *press.*

[18] Neumann, P.O. An attribute frequency model for the abstraction of prototypes. *Memory and Cognition,* 1974, 2, 241-248.

[19] Neumann, P.G. Visual prototype formulation with discontinuous representation of dimensions of variability. *Memory and Cognition,* 1977, 5, 187-197.

[20] Newell, A. A theoretical exploration of mechanisms for coding the stimulus. In A.W. Melton ft E. Martin (Eds.), *Coding processes In human memory,* Washington, D.C: Winston, 1972.

[21] Newell, A. Production Systems: Models of control structures. In W.G. Chase (Ed.), *Visual Information Processing.* New York: Academic *Press,* 1973.

[22] Norman, D.A. ft Rumelhart, D.E. *Explorations In cognition. San* Francisco: Freeman, 1975.

[23] Posner, M.I. ft Keele, S.W. Retention of abstract ideas. *Journal of Experimental Psychology,* 1970, *63,* 304-308.

[24] Quilian, M.R. The teachable language comprehender. *Communications of the ACM,* 1969, *12,* 459-476.

[25] Reed, S. Pattern recognition and categorization. *Cognitive* Paychology, 1972, 3, 382-407.

[26] Redman, J.S. ft Bower, G.H. Structure and later recognition of exemplars of concepts. *Cognftlve Psychology,* 1973, 4, 194-206.

[27] Rosch, E. ft Mervis, C.B. Family resemblances: Studies in the internal structure of categories. *Cognitive Psychology,* 1975, 7, 573-605.

[28] Rychener, M.D. ft Newell, A. An instructible production system: Basic design issues. In D.A. Waterman ft F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems,* New York, N.Y.: Academic Press, 1978.

[29] Schapiro, S.C. A net structure for semantic information storage, deduction, and retrieval. Proceedings of the Second International Joint Conference on Artificial Intelligence, 1971, 512-523.

[30] Simmons, R.F. Semantic networks: Their computation and use for understanding English sentences. In R.C Schank and K.M. Colby (Eds.), *Computer models of thought and language.* San Francisco: Freeman, 1973.

[31] Sussman, G.J. *A Computer Model of Skill Acquisition.* New York, N.Y.: American Elsevier Publishing Co., 1975.

[32] Vere, S A. Induction of relational productions in the presence of background information. Proceedings of Fifth International Joint Conference on Artificial Intelligence, Boston, Mass., 1977, 349-355.

[33] Waterman, D.A. Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence,* 1970, *1,* 121-170.

[34] Winston, P.H. Learning structural descriptions from examples. MIT Artificial Intelligence Laboratory Project AI-TR-231, 1970.

[35] Wittgenstein, L. *Philosophical Investigations.* New York: MacmillAn. 1953.

KNOWLEDGE-BASED PROBLEM SOLVING BY A LABELLED PRODUCTION SYSTEM

Yuichiro Anzai, Naoto Ishibashi*, Yuichi Mitsuya and Shoji Ura

Department of Administration Engineering
Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama 223
Japan

The paper summarizes a system that understands, solves, and learns how to solve geometry problems. The system is represented homogeneously as production rules of Labelled Production System. A brief introduction to the system architecture is given, followed by an example of how the system works. Then the paper discusses briefly some theoretical problems focused upon in design and implementation of the system.

1. INTRODUCTION

Recent development of cognitive science has revealed that problem solving is a total activity of understanding, solving, and learning how to solve problems, controlled on a unified knowledge base. This paper summarizes an effort toward building a system that encompasses such a broad range of knowledge-based problem solving, taking elementary geometry as a task domain.

Our fundamental assumption is that knowledge acquisition exerts a considerable influence on problem-solving behavior. This leads to our emphasis on post-solution analysis for acquisition of new knowledge from the solution process. It also leads to a homogeneous representation of the system as production rules of Labelled Production System (LPS): knowledge acquisition on a well-structured knowledge base of geometry requires a representation which has capability to represent structured knowledge and high learnability at the same time. Also we believe that all information necessary in solving a given problem is not always available when the problem was understood. Some information might be necessarily inferred through subgoals, or popped up by demon-like procedures during the solution process. So interaction of subgoal-oriented and pattern-directed processing is one of our concerns in designing a knowledge-based problem solver.

The above assumptions and thus motivations make our system different from traditional geometry theorem provers (e.g., [6]). Our main concern is to theorize problem solving from a broader viewpoint. The current version of LPS is written in INTERLISP, and running on DEC-20. The version is only for an experimental use, but all the features described here are already implemented in it.

2. SYSTEM ARCHITECTURE

The system comprises four components: the Natural Language Understander, Problem Solver, Post-solution Analyzer, and factual knowledge about geometry. Understander transforms a (Japanese) problem sentence to a set of semantic primitives such as geometric objects and relations, and goals for the proof. Solver works on this set toward solving the problem. It is able to try drawing additional line segments by using a partial-match algorithm and knowledge acquired through solving similar problems. Analyzer undertakes Solver's output, and tries to create possibly helpful productions. First Analyzer generates a hypothesis of a proposition. If the proposition was tested successfully by Solver, Analyzer adds it to the knowledge base. Interaction of subgoal-oriented and pattern-directed processes is realized by ACTIVATE as seen later.

The above process proceeds by utilizing factual knowledge about geometry. All the four components of the system are represented homogeneously as production rules of LPS. LPS is, in short, a production system in which knowledge elements are *labelled.* Although global control of LPS is basically similar to 'recognize-act cycle[1] type production systems [4], its labelled structure greatly facilitates dealing with structured knowledge and reorganization of it. For example, labels such as CATEGORY and A-PART-OF can be used for inference, similarly to slot-names in frame systems [2]. Also we can use labels for referring to a knowledge element in another production directly without using working memory (WM). See [1] for the details of LPS.

* Presently at SONY Corporation.

3. KNOWLEDGE-BASED PROBLEM SOLVING: AN EXAMPLE

An example outlines the system's behavior most explicitly. Fig.1 shows a simple geometry problem. First of all, Understander transforms the sentence to a set of semantic primitives, which are shown in Fig.2.

Problem: Senbunno chuutenwo toori, koreni suichokuna chokusenjouno tenwa senbunno ryoutankara toukyorini aru. (A point on the line through the midpoint of a line segment and perpendicular to it is equally distant from the endpoints of the line segment.)

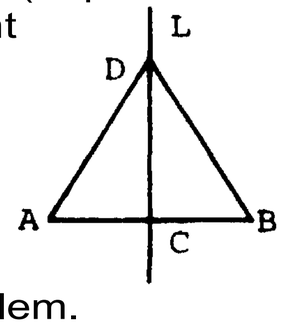


Fig. 1 Example problem.

((PGOAL(LEQUAL(LS D A)(LS D B)))(LS D A)(LS D B) (EP A (LS A B)) (EP B (LS A B)) (PERP (LS A B) (LS C D)) (ON (LS C D) (LINE L)) (LS C D) (ON D (LINE D) (PT D) (PERP (LS A B) (LINE L)) (ON C (LINE D) (LINE L) (ON C (LS A B)) (LEQUAL(LS A C) (LS C B)) (LS A C) (LS C B) (MP C (LS A B)) (PT C) (PT A) (PT B) (LS A B) (READNIL) (READ-START))

Fig. 2 Understander's output for the example problem. (For simplicity, 'notation is slightly different from the original output.)

Evoked by the goal (PGOAL(LEOUAL(LS D A)(LS D B))), meaning to prove equality of some quantity (length here) of the same type of objects (line segments here), Solver tries to ACTIVATE the pattern (CONGRUENCE) to WM. ACTIVATE collects productions related to (CONGRUENCE) by chaining productions backwards, and executes production-firing only on the collected subset of productions. One of collected productions, TRIANGLE, pops up two new triangles, (TRIANGLE D A C) and (TRIANGLE D B C). Then these triangles make the production CONGRUENCE executed, and (CONGRUENCE) deposited into WM. Note that the two triangles did not exist in WM when Solver started to work (Fig.2). They were necessarily recognized in the ACTIVATE function mode. Solver needs to recognize some more relations to attain the goal. It is the end of the problem-solving process when the geometric relation, (LEOUAL(LS D A)(LS D B)), PGOAL's argument, is placed in WM

The current version of Analyzer intends to construct new propositions from Solver's output by means of three different ways. First, note that Solver concludes not only the conclusion for the problem, but also other two relations generated by a side-effect: (1) line segments DA and DB have equal length, (2) angles ADC and CDB have equal degrees, and (3) angles DAC and DBC have equal degrees. So Analyzer builds three separate productions, 'if conditions of the problem hold, then (k) is true,' for k=I,2 and 3.

Productions generated in this manner can deal only with problems that have essentially the *same* conditions as the original problem. Analyzer's second way of knowledge acquisition is to build a production: for a given problem X, 'if *most of* geometric objects, relations and goals generated by Solver for solving the example problem matched WM for the problem X, then try to apply to solve X geometric concepts and propositions used in solving the example problem.' 'Most of implies that, if this production can be executed, X is similar to the example problem if not totally the same. Hence conditions of this production should match partially.

Analyzer's last way of restructuring Solver's output is to use task-dependent heuristics to extract some small amount of mutually related objects and relations, and construct a 'speculative' hypothesis. Fig.3 shows an example of such hypothesis. It needs to be tested by Solver, which goes successfully in this case, and the production in Fig.3 is registered as a new production. Testing the hypothesis by Solver, in this case, happens to involve drawing an additional line segment. Analyzer searches for a production already acquired and similar to the hypothesis, and here draws the segment which connects D with C, the midpoint of AB, in Fig.I.

Analyzer builds up several productions consecutively in the above manners. It is the end of the system's problem-solving process: after Analyzer worked out, the system waits for another problem, which is to be fed first to the understander.

```
(G0900
#A0901  (PGOAL(LEQUAL(ANGLE $A0906 $A0909 $A0907)
                      (ANGLE $A0906 $A0907 $A0909)))
#A0902  (LS $A0906 $A0907)
#A0903  (LS $A0906 $A0909)
#A0904  (LS $A0909 $A0907)
#A0905  (LEQUAL(LS $A0906 $A0909)
               (LS $A0906 $A0907))
ACT  (RESULT(LEQUAL(ANGLE $A0906 $A0909 $A0907)
                    (ANGLE $A0906 $A0907 $A0909)))
     (LEQUAL(ANGLE $A0906 $A0909 $A0907)
            (ANGLE $A0906 $A0907 $A0909))
CATEGORY (TRIAL))
```

Fig. 3 A hypothesis production created by Analyzer.

4. ISSUES IN KNOWLEDGE-BASED PROBLEM SOLVING

4.1 Partial-match and drawing additional line segments

Some work on geometry problem solving is concerned with drawing additional line segments [13] It is a tempting task for AI because it is a good example of generating new appropriate representations.

'Drawing additional line segments, • in our case, is a process of finding a once-solved similar problem, and generating a set of new geometric objects and relations that reduces the difference of the once-solved and currently-attacked problems. Selecting a similar problem corresponds to deciding a best-match between the current WM and condition sides of some learned productions. We have tried to avoid the curse of computational explosion inherent in partial-match by introducing a few heuristics: only geometric objects and relations are taken into account, and an object which is A-PART-OF another object is neglected. This kind of semantic categorization and inference uses the labelled structure of LPS, and saves a fair amount of computation time.

4.2 Interaction of subgoal-oriented and pattern-directed processing

Psychological research [5] tells us that elementary geometry problem solving involves organization of subgoals. This basic problem-solving structure is represented in our system by the function ACTIVATE. ACTIVATE tries to achieve a specified subgoal by organizing related productions as exemplified in the last section.

ACTIVATE is a powerful function for our purpose because knowledge of geometry is well-structured enough to activate reasonable subsets of productions. However, popping up objects or relations in a pattern-directed manner also plays a key role in solving problems, as it might generate useful but formerly unrecognized objects or relations. But since bottom-up processing may easily blow up computationally, efficient embedding of some interactive structure of top-down and bottom-up processes is highly desirable in any knowledge-based problem solver. Using ACTIVATE actions within pattern-evoked productions is a realization of this general framework.

4.3 Structuring new productions from solution experience

To build a hypothesis production as shown in Fig.3, Analyzer must select suitable elements in WM for creating conditions and actions. The mechanism for gathering them is analogous to the heuristics incorporated in the algorithm for drawing auxiliary line segments, but slightly more complex. Analyzer picks up one geometric relation arbitrarily, and makes it as the action in the hypothesis. Then Analyzer collects objects and relations related (in a certain sense) to the action, and assumes them as conditions. The mechanism makes use of LPS's labelled structure.

Note that the example in Fig.3 is equivalent to: the two base angles of an isosceles triangle is congruent. The system succeeded in generating the production in Fig.3 *without* solving this problem. We believe that the effort of Analyzer deverves doing, as this kind of self-organizing performance is the heart of intelligent tasks.

5. CONCLUSION

We have chosen geometry as the task domain mainly because it satisfies our general motivations stated in Section 1, and involves issues described in Section 4. We believe that those motivations and issues are general and central in any knowledge-based problem solving task. LPS is a general-purpose representational system, and our research presented here is a working study toward a general theory of knowledge-based problem solving. Though some detailed part of the used techniques are task-specific, and we have yet little experience on LPS in other domains, our work has achieved a step toward a computational unification of understanding, solving and learning within a general framework of problem solving.

REFERENCES

[1] Anzai,Y., Ishibashi,N., Mitsuya,Y. & Ura,S. "Knowledge-based Problem Solving by a Labelled Production System." Keio Engineering Report, Faculty of Engineering, Keio Univ., to appear.

[2] Bobrow,D.G., Winograd,T. et al. "Experience with KRL-O: One Cycle of a Knowledge Representation Language." In Proc. IJCAI-77. MIT, Cambridge, Mass., August, 1977, pp.213-233.

13] Elcock,E.W. "Representation of Knowledge in a Geometry Machine." In E.W.Elcock & D.Michie (eds.), Machine Intelligence 8, 11-29, Ellis Horwood, 1977.

[4] Forgy,C. & McDermott J. "The 0PS2 Reference Manual." Dept. of Computer Science, Carnegie-Mellon Univ., 1978.

[5] Greeno,J.G. "Processes of Understanding in Problem Solving." In N.J.Castellan et al.(eds.), Cognitive Theory *2,* 43-83,Lawrence Frlbaum,1977.

[6] Nevins,A.J. "Plane Geometry Theorem Proving Using Forward Chaining." Artif.Intell. 6 (1975) 1-23.

INTERACTIVE GRAPH PRODUCTION SYSTEM

Kyota Aoki
Department of Information Science
Faculty of Engineering
Utsunomiya University
Utsunomiya 321-31
JAPAN

Kokichi Tanaka
Department of Information and Computer Sciences
Faculty of Engineering Science
Osaka University
Toyonaka 560
JAPAN

In this paper we propose a new production system called Interactive Graph Production System (IGPS) which represents situations which have interactions. An IGPS is constructed by two labelled directed graphs and two sets of production rules which control interactions and changing of situations. First, IGPS is defined. Then we show examples of IGPS: three coin problem and monkey and banana problem, for making clear structure and move of an IGPS. Next execution of IGPS is discussed. IGPS interpreter must have some functions for efficient execution of IGPS. And for making productions efficiently, we need graph production editting system which enables us to handle graph productions in graphical forms.

1. INTRODUCTION

Eversince production systems (PS) were first proposed by Post [7] as a general computational mechanism, the methodology has heen a great deal of development and has been applied to a diverse collection of problems. A production system may be viewed as consisting three components: a set of rules, a data base, and an interpreter for the rules. In the simplest design, a rule is an ordered pair of symbol strings with a left and right hand sides, the set of rules has a predetermind total ordering and the data base is simply a collection of symbols.

Throughout much of the work reported, there appear to be two major views of PSs, *as* characterized on one hand by the psychological modelling efforts (PSG, PAS II, VIS, etc.) [5, 6] and on the other by the performance-oriented, knowledge-based expert systems (e.g. MYCIN, DENDRAL) [2, 3], For the psychological modellers, production rules offer a clear, formal, and powerful way of expressing basic symbol processing acts, which form the primitives of information processing psychology. For the designer of knowledge-based system, production rules offer a representation of knowledge that is relatively easily accessed and modified, making it quite useful for systems designed for incremental apploaches to competence.

Now we have trend to apply PSs to more and more complex problems. Those problems need more complex knowledge-bases. For instance, the DENDRAL system uses a literal pattern match, but its patterns are graphs representing chemical classes. For expressing complex situations graphs are better than collections of literals for human understandability. In many cases graphs are used for describing situations, and we see many usage of graphs for explanations while collections of assertions are used for an internal representation of a system. So we want to construct a PS which treats not symbol strings or collections of assertions, but graphs.

In complex problems we find two situations interacting each other. For instance, in a problem solving for controlling a robot we find a robot and its environment interacting each other. We think that it Is better than a description by a single situation which represents the robot and its environments, to describe the world by the set of two sub-situations: one represents the robot and the other does its environment; and changing of the world by interactive changing of two sub-situations. So we want to construct a PS which can express systems that interact each other also.

In this paper we will propose a new graph production system that has interactions called an Interactive Graph Production System (IGPS). An IGPS represents a situation by a set of two labelled directed graphs and changing of situations by rewriting rules of graphs. So we will construct an IGPS based on the graph grammar system which is discussed in [1], In this paper we will first describe definitions of an

IGPS, next show some examples of IGPSs for making clear the method of descriptions and moves, and then discuss execution of an IGPS.

2. INTERACTIVE GRAPH PRODUCTION SYSTEM

In this section we will describe definitions of an IGPS. An IGPS is constructed by two labelled directed graphs and two sets of production rules which control interactions and changing of situations.

2.1 Situations of an IGPS

A situation of an IGPS is represented by a tuple,

$$(O_0,$$

where o_0 and O_1 are sub-situations described by labelled directed graphs. More formally, a sub-situation *0.* is represented by a 3-tuple,

$$(N_i , L_i , E_i),$$

where N_i is a set of nodes, L_i is a function: N_i -> a set of labels, and E_i is a set of edges and is included into $N_i \times N_i$.

[Example 1] Here we show an example of a sub-situation.

Let $N_i = \{1, 2, 3\}$,

$L_i(1) =$ he ,

$L_i(2) =$ is ,

$L_i(3) =$ diligent ,

and $E_i = \{ (1, 2), (2,3) \}$.

Then the sub-situation (N_i, L_i, E_i) is the graph: he → is → diligent .

2.2 Structure of IGPS

An IGPS is a 7-tuple:

$$s = (c, V, R, i_0, P_1, P_1),$$

where *C* is a set of labels of sub-situations, *V* is a set of variables whose ranges are subsets of C, *R* is a function: $V \rightarrow 2^r$ which defines the ranges of variables, t_0 and i_1 are two finite initial sub-situations whose sets of labels are C, and P_0 and P_1 are sets of productions which are applied to two sub-situations: O_0 and a , respectively.

In this paper we will describe elements of *C* by strings of lower case letters and elements of *V* by strings of upper case letters.

2.3 Variables

An element of *V* is *V.* whose range is $R(v_i) \in C$.

A variable *V.* can have a value of an element of $R(v_i)$.

[Example 2] Here we show an example of a variable.

Let $C = \{$ station, school, boy, girl $\}$,

$V \ni$ PLACE ,

and, R(PLACE) ■ { station, school }.

Then the value of 'PLACE' can be 'station' or 'school', but it can not be 'boy' nor 'girl'.

2.A Productions

We describe a production of P_0 or P, by the form:

$$(G_1) \quad G_2 \Xi> G_3 ,$$

where G_1 is a labelled directed graph whose $/N_1/$ is a non-negative integer and the set of labels is COW, G_2 is a labelled directed graph whose $/N_2|$ is a natural number and the set of labels is CUW, and G_3 is a labelled directed graph whose set of labels is CUVU{null} and satisfies the conditions listed below.

$$N_3 \supset N_2 \quad \text{and} \quad E_3 \supset E_2 ,$$

and any $n_3 \in N_3 : L_3(n_3) \in V$;

there exists n_2 or n_1 :

$$n_2 \in N_2 \text{ and } L_2(n_2) = L_3(n_3),$$

and/or $n_1 \in N_1$ and $L_1(n_1) = L_3(n_3)$.

At an application of a production each variable has one value. When two labelled directed graphs are compared, a variable *V* and a constant *c* are compatible if the value of *v* is c.

[Example 3] We show an example of a production in Fig. 1. In this example *C, V* and *R* of the IGPS are same as those in Example 2.

2.5 Effect of Application of Production

Let a situation of an IGPS be (σ_0, σ_1), and let a production p..: $(G_1)\ G_2 \Xi> G_3$; be included in P. (i=0, 1). Then the production p.. may be applied, if Condition 1 and 2 are satisfied.

[Condition 1] Let a be $O_{1\text{-}2}$., and let G be G_1, $_{1\text{-}i'}$ then Condition 1' is satisfied. If G is an empty graph, then this condition is satisfied.

[Condition 1']

Let $O = (N, L, E)$,

then there exists $N' \subset N$: $(N', L, E \cap N' \times N')$ and G are compatible.

When Condition 1 is satisfied, the production p.. can be applied if Condition 2 is satisfied.

[Condition 2) Let G_2' be a graph which is made by rewriting labels of G_2 which are elements of V and used in G_1 as the value of the variable. Let o be o_i and G be G_2' then Condition 1' is satisfied.

If a production satisfying Condition 1 and 2 is applied, then a sub-graph of O_i matching G_2 is rewritten as G_3'. Here G_3' is a graph made by rewriting labels which are elements of V as the value of the label.

In an IGPS, production has a structure described in 2.4, therefore deletion of nodes or edges is not avairable. But in an IGPS the special label 'null' expresses that the node will be not rewritten nor referred. Hence an IGPS interpreter can delete a node whose label is 'null' and edges which connect those nodes, there are no defferences in moves of the IGPS.

[Example 4] We show an example of a production and its application in Fig, 2 In this example,

C = { monkey, at, placel, place2, box, move },

V = { PLACEX, PLACEY },

and R(PLACEX) = R(PLACEY) = { placel, place2 }.

When a situation of the IGPS is (σ_0^0, σ_1^0) of

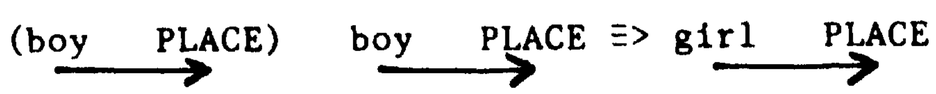


Fig. 1 An example of a production of IGPS.

Fig. 2, if production-1 of Fig. 2 is an element of P_0 of the IGPS, then in production-1 the value of PLACEX is palce2 and the value of PLACEY is placel, and the production can be applied to σ_0^0. If the production is applied to σ_0^0, then a situation of the IGPS becomes to be (σ_0^1, σ_1^0) of Fig. 2.

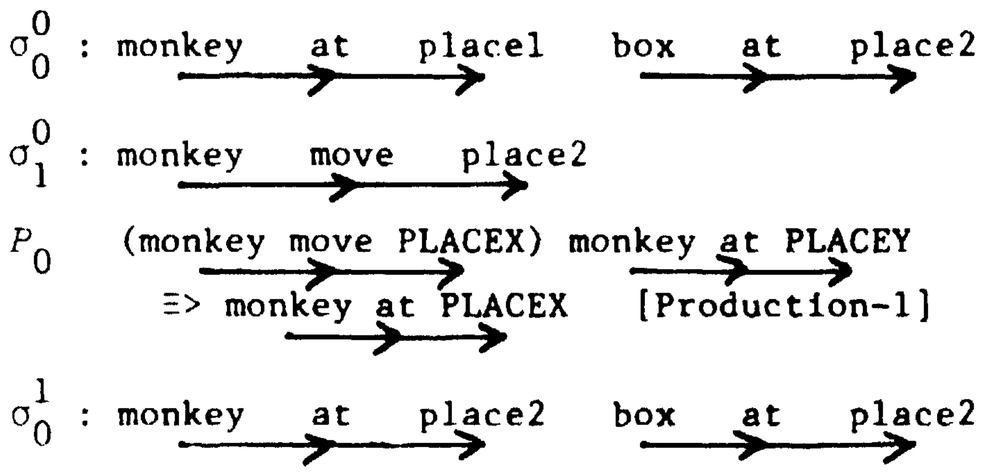


Fig. 2 Example of a production and its application.

2.6 Moves of an IGPS

One move of an IGPS is constructed by two sub-moves. When a situation of an IGPS is (σ_0^0, σ_1^0), one sub-move is an application of an element of P_0 to σ_0^0, or preservation o σ_0^0 n e n no element of P_0 can not be applied to σ_0^0. Let the result be (σ_0^1, σ_1^0). Next one sub-move is an application of an element of P_1 to o , or preservation o f σ_1^0 e n no elements of P. can not be applied to σ_1^0. Let the result be (σ_0^1, σ_1^1).

3. SOME EXAMPLES OF IGPS

In this section we show some examples of IGPS for making clear the structure of an IGPS and the moves of an IGPS.

3.1 Three Coin Problem

Here we show the three coin problem of Jackson [4].

Problem: Given three coins initially HHT (i. e., head, head, tail), in exactly three moves

make all coins show the same face. A move consists of flipping a coin over.

We show below elements of an IGPS which describes the three coin problem and chunks of knowledge for solving the problem.

C = { start, cont, flip, end, coinl, coin2, coin3, inc, countO, countl, count2, count3, head, tail, op, counter } ,

V = { COINX, COINY, COINZ, CONT, COUNT, COUNTALL, COUNTNEXT, STATE, STATEX, STATEY } ,

R(COINX) = R(COINY) = R(COINZ)
= { coinl, coin2, coin3 } ,

R(CONT) = { cont, end },

R(STATE) = R(STATEX) = R(STATEY)
- { head, tail } ,

R(COUNTALL) = { countO, countl, count2, count3 } ,

R(COUNTNEXT) = { countl, count2, count3 } ,

R(COUNT) ■ { countO, countl } ,

i_0 - start -► cont ,

i_1 = inc -> countO -► countl coinl -> head
inc -► countl -► count2 coin2 -► head
inc -► count2 -► count3 coin3 -> tail
head <- op -► tail counter -> countO

P_0 and P_1 are shown in Fig. 3 and Fig. A, respectively.

(COINX → STATE COINY → STATE, counter → COUNT) cont ⇒ flip → cont ↓ COINX (I)

(COINX → STATE COINY → STATE, COINZ counter → count2) cont ⇒ flip → end ↓ COINZ (II)

Fig. 3 P_0 of the IGPS which represents the three coin problem.

(flip -> COINX, CONT) COINX -> STATEX STATEX <- op -> STATEY counter-> COUNTALL inc +COUNTALL -> COUNTNEXT

⇒ COINX -► STATEY STATEX <- op -> STATEY counters COUNTNEXT inc -> COUNTALL -> COUNTNEXT

Fig. 4 P_1 of the IGPS which represents the three coin problem.

In this IGPS the sub-situation o_1 which is initially i_1 expresses a situation of coins, and the sub-situation o_0 which is initially i_0 expresses a situation of a process of solving. Productions of P_0 generate moves which fit in the situation with referring a sub-situation O_1 by getting knowledges of coins' situation and how many times the move is with G_1 . A production of P_1 expresses the move which is generated by P_0. In o_1 "inc-> countO-► countl" expresses the move of a counter. And "head <-op-> tail" expresses that 'head' and 'tail' are opposite faces of each other.

In Table 1, we show changes of a sub-situation O_0 of the IGPS. And in Table 2, changes of a sub-situation O_1 of the IGPS are shown. In Table 2 we omit part of sub-situation O_1 which do not change for making short.

Table 1 Process of changing of o_0 .

applied production	O_0 which is the result of application of the production
I	start → flip → cont ↓ coinl
I	start → flip → flip → cont ↓ ↓ coinl coinl
II	start → flip → flip → flip → end ↓ ↓ ↓ coinl coinl coin3

Table 2 Process of changing of σ_1.

value of σ_1 'COINX'	which is the result of application of the production		
coinl	coinl → tail	coin2 → head	coin3 → tail
coinl	coinl → head	coin2 → head	coin3 → tail
coin3	coinl → head	coin2 → head	coin3 → head

3.2 Monkey/banana Problem

The familiar monkey and banana problem is formulated as an IGPS. In three coin problem, IGPS always generates correct answers, but in the present case IGPS expresses a process of monkey's trial and error process. Here, monkey

does not want to do a move which can not be carried out. And if monkey can take banana, he must take it. We show elements of the IGPS below.

C = { mky, box, ban, placel, place2, place3, at, over, on, has, move, push, climb, take, start, cont, cont', end } ,

V - { PLACEX, PLACEY, DO, DOX } ,

R(PLACEX) - R(PLACEY) = { placel, place2, place3 } ,

R(D0) - { climb, take, move, start } ,

R(DOX) = { push, move } .

P_0 and P_1 of the IGPS are shown in Fig. 5 and Fig. 6, respectively. The initial sub-situation

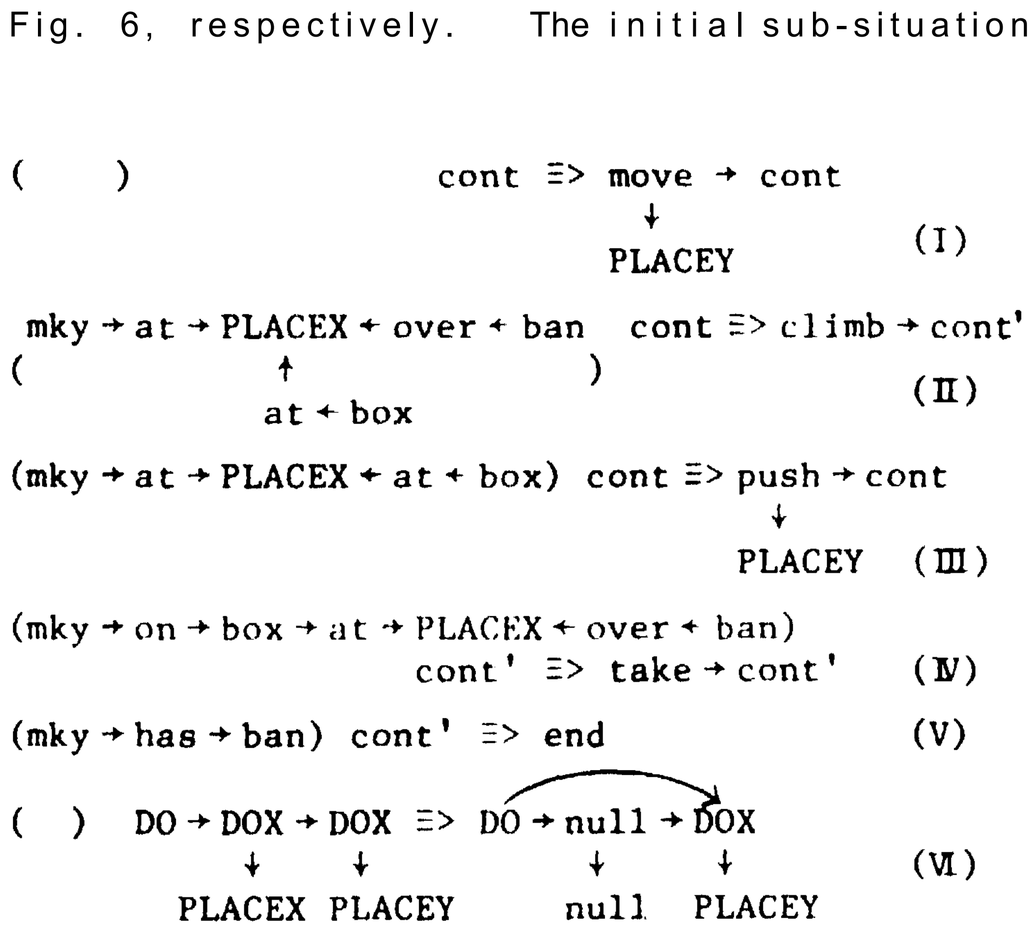


Fig. 5 P_0 of the IGPS which represents the monkey and banana problem.

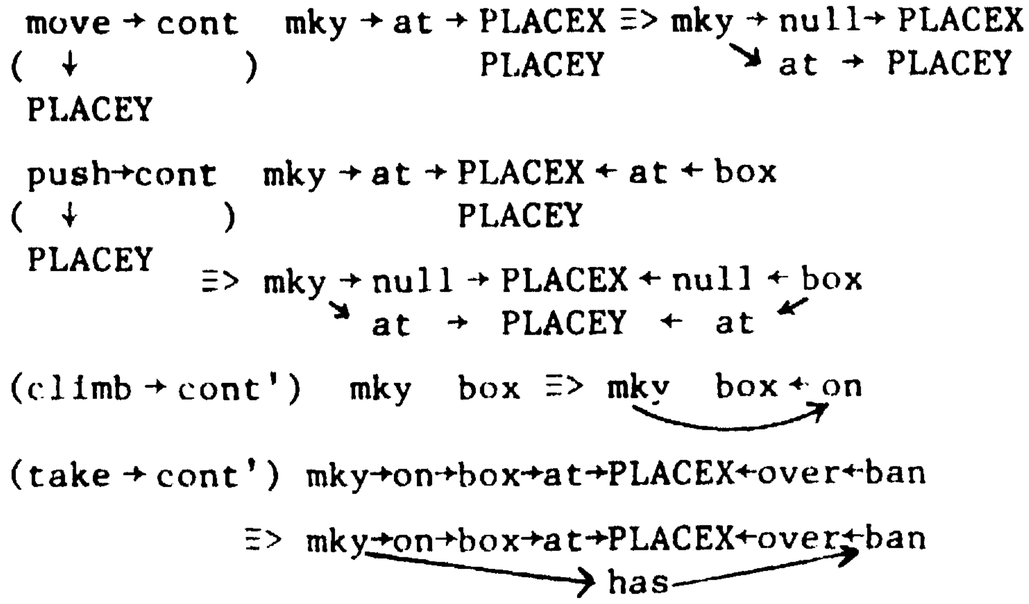


Fig. 6 P_1 of the IGPS which represents the monkey and banana problem.

i_0 is shown in Table 3, and i_1 is shown in Fig. 7

A process of monkey's trial and error are shown in Table 3.

Table 3 Process of changing of σ_0.

applied production	σ_0 which is the result of application of the production
i_0	start → cont
I	start → move → cont ↓ place3
I	start → move → move → cont ↓ ↓ place3 place2
VI (I)	start → null → move → cont ↓ ↓ null place2
III (I)	start → move → push → cont ↓ ↓ place2 placel
III (I)	start → move → push → push → cont ↓ ↓ ↓ place2 placel place3
VI (I, III)	start → move → null → push → cont ↓ ↓ ↓ place2 null place3

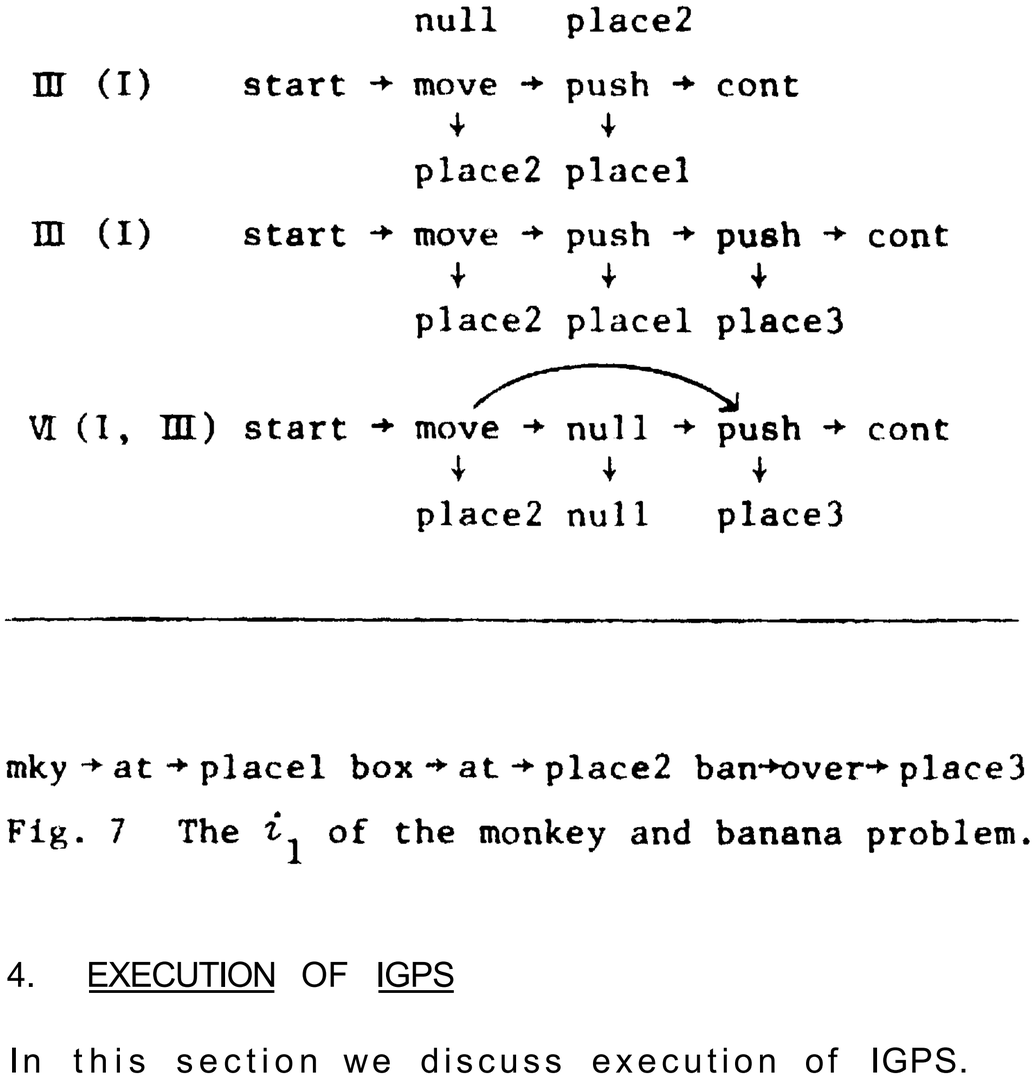


Fig. 7 The i_1 of the monkey and banana problem.

4. EXECUTION OF IGPS

In this section we discuss execution of IGPS. The core of execution of IGPS is an IGPS interpreter, which receives an IGPS and execute the IGPS. And we need a production editting system for easy description of productions. The production editting system enables us to edit productions on a graphic display unit and generate sets of productions.

4.1 IGPS Interpreter

We implement an IGPS interpreter which executes an IGPS originally in accordance with the definitions of IGPS. But the IGPS interpreter has some functions for making processing fast and description of productions easy.

The IGPS interpreter has two modes except a basic mode. In the basic mode the IGPS interpreter applies one production at one time according to the definition of the IGPS. In expanded mode 1. the IGPS interpreter tries to apply productions which are permitted to apply at one time. And in expanded mode 2, the IGPS interpreter tries to apply all productions to all sub-graphs at one time while productions can be applied. In those expanded modes, the number of derisions on whether a production may be applied or not decreases, so processing time at one application of a production decreases.

In the definition of IGPS we did not define how to select a production, so we must now decide how to select a production. In most of production systems, for instance RPS [8], productions are ordered and the first production which matches the data-base is applied. But if we use rule-ordering for selection of a production, we can not describe the monkey and banana problem as in 3.2. Therefore the IGPS interpreter enables us to decide a selection method of a production among three methods. The first method is the conventional rule-ordering. The second one enables us to specify a priority of productions at each move. And the last one enables us to specify an algorithm which decides a priority of productions at each move.

4.2 Production Editting System

A production of IGPS is constructed by three tuple of labelled directed graphs. For editting productions efficiently, we need some functions, which enables us to define a set of constant *C,* a set of variables *V* and a range function R, and to input labelled directed graphs, and to check inputted productions. Input of labelled directed graphs can be done by inputting a label of each node and a tuple of a head and a tail of each edge using punched cards, but we can not inspect graphs efficiently using a list of a label of each node and a list of a tuple of a head and a tail of each edge. So the production editting system must enable us to edit productions: three tuple of labelled directed graphs, using graphical desciprion. Therefore the production editting system uses a graphic display unit for display of productions. And the production editting system enables us to edit productions interactively.

5. CONCLUSION

In this paper, we have proposed a new production system: IGPS. IGPS is a formal system which expresses a structure and moves of a system which has interaction. And we have shown examples of an IGPS for making clear descriptive power and moves of IGPS.

In the field of artificial intelligence, we treat a diverse collection of problems. Some of them have Interaction in their situations. For expressing these problems we need a production system which can express interaction. Of course conventional production systems can express a system which has interaction. But simplicity, modularity and other favourable PS's features are lost. IGPS can express a system which has interaction while increasing favourable PS's features. And further using labelled directed graphs, IGPS can express very complex situations.

ACKNOWLEDGEMENTS

The authors appreciate many helpful conversatio with several graduate students and members of Prof. Tanaka's Lab. of Osaka University.

REFERENCES

[1] Aoki, K. and Tanaka, K./'The ability of the Interactive Systems of Context-free Form" Trans TECE, Vol. 61-D, No. 8, pp 525-532, 1978 (in Japanese).

[2] Davis, R., Buchanan, B.G. and Shorliffe, E. H.,"Production rules as a representation for a knowledge-based consultation program," AI Memo 226, Computer Science Dept., Stanford University, 1975.

[3] Feigenbaum, E.A., et. al., "On Generality and Problem Solving - a case study involving the DENDRAL program" Machine Intelligence 6, Edinburgh University Press, pp 165-190, 1971.

[4] Jackson, P.C., Introduction to Artificial Intelligence, Petrocelli, New York, 1974.

[5] Moran, T.P., The Symbolic Hypothesis: A Production System Model, Computer Science Department, Carnegie-Mellon University, 1973.

[6] Newell, A./'Production Systems: Models of Control Structures', Visual Information Processing, Academic Press, pp 463-526, 1973.

[7] Post, E.,"Formal Reductions of the General Combinatorial Problem" Am. Jnl. Math., Vol. 65, pp 197-263, 1943.

[8] Vere, S.A.,"Relational Production Systems" Artificial Intelligence, Vol. 8, No. 1, pp 47-68, 1977.

META-KNOWLEDGE AND COGNITION

Avron Barr
Heuristic Programming Project
Computer Science Department
Stanford University
Stanford, CA 94306

In AI knowledge representation schemes, structures that describe other structures *are* said to represent "meta-knowledge," or knowledge about other knowledge. After describing some studies of human behavior that demonstrate people's ability to reason about what they know and about how they reason, we review the use of explicit meta-knowledge in aeveral rescent AI systems. The concept of meta-level knowledge *captures* Intrinsic, commonplace properties of human cognition that are central to an underatanding of knowledge and intelligence.

In the last few years, several AI researchers have proposed the use of "meta-level" knowledge representation structures for a variety of tasks. In AI, the phrase "meta-knowledge" generally refers to data-structures In a knowledge representation scheme that "represent" knowledge about other other data-structures, as opposed to representing knowiege about "things In the world." For example, a rule in the knowledge base of an expert medical diagnosis system might be annotated with a meta-level description of the rule's history (e.g., which expert entered it or last modified it) or a description of its relation to other rules in the database.

The use of meta-knowledge of this type in AI systems like TEIRESIAS (Davis, 1076) is a key breakthrough In the design of "knowledge-based" intelligent systems. Meta-level knowledge has been used in these systems primarily In the Implementation of "introspective" processes: *acquisition* of new knowledge from human experts and *explanation* of the system's reasoning to *users.* The usefulness of meta-level descriptions for these and other functions has prompted proposals for their Incorporation in several new general-purpose representation schemes, like KRL, as described below.

But there Is more to meta-knowledge than its typical characterization in AI captures. In human experience, meta-level knowledge and reasoning are an integral part of common, everyday cognitive activity. For example, consider the well-known "tip-of-the-tongue" phenomenon;

> You run Into someone you have met once before, and you can't remember his name. You remember very well your first meeting at a New Year's Eve *party* In Oakland, and that he is the brother-in-law of your wife's boss. Then you remember that he has a foreign-sounding name. It rhymes with spaghetti...

You could use all of this knowledge In trying to recollect his name. You would certainly say that you "know his name," even though you can't recall it: You "know that you know It" — knowledge *about* what you know. It Is this *intuitive* knowledge about what we know, and also about how we use what we know, that is the most compelling reason for viewing meta-knowledge as having a central role In human cognition. After exploring some psychological studies which indicate the nature and extent of the role of meta-knowedge in human memory, reasoning, and understanding, we will examine tho recent use of explicit representations of meta-knowledge In several recent AI systems. We will argue, In conclusion, that the apparent difference in character between human meta-cognltive activity and the use of mets-level representations in AI Is an important Indication of the difference between "representation" and "knowiege" that should be explored further.

Meta-knowledge in Human Cognition

The psychological studies reviewed here deal with human memory, plausible reasoning, and cognitive development. The conceptual framework offered by "meta-knowledge" is essential to understanding these results: It will be argued that much remembering and reasoning Is best described as meta-level activity, that at the core of these mental processes people use knowledge about their own cognitive ability, style and experience, and about the extent, origin, and certainty of their knowledge.

It Is Important to keep in mind that the cognitive behaviors described here *are* not the results of trick questions or contrived experimental situations. The phenomena that are described are an Intrinsic part of human cognition, from remembering to everyday inference making.

Meta-memory: Knowing What You Know

The experience of meta-knowledge by humans Is addressed directly in a paper by Kolers *and* Palef (1076). They point out that the "knowing not" phenomenon is a very common characteristic of human cognition; this Is simply where people often know rapidly and reliably that they *do not know* something. Furthermore, these researchers point out that this trait is not easily captured by current "searching" models of memory.

Meta-knowledge and Inference

Allan Collins and his colleagues have for some time been studying "reasoning from incomplete knowledge," that Is, what one can conclude from the fact that one doesn't know something. Since the prerequisite to this kind of reasoning is *awareness of not knowing some fact,* these inferences relate directly to the "knowing not" studies, and to meta-knowledge. For example, In the "lack of

knowledge" Inference, the fact that you would know some fact if It were true, but you don't remember it, leads you to believe that It Just isn't true. Gentner and Collins (1976) suggest two factors used in making the lack-of-knowledge inference. First, one estimates the (relative) "importance" of the fact; the more important It is, the more the fact that you don't remember it implies that It "ain't so." Second, one's own expertise in the topic area is estimated—the more one knows about the area, the more likely not remembering something implies it isn't true. Collins (1978) stresses that much of human plausible reasoning Is based on meta-level reasoning about what one knows, and what one would know if some fact were true.

The Development of Meta-cognition

John Flavell has been studying for some time the way children develop Increasingly accurate "feelings" about their performance on cognitive tasks involving learning, remembering, and understanding. In a recent paper, Flavell (1979) discusses these results in terms of *metacognitive knowledge* and *metacognitive experience.* Metacognitive knowledge Is knowledge about people as cognitive systems, about the cognitive tasks they face, and about the strategies they employ to accomplish them. Metacognitive experiences are the realizations about some aspect of a cognitive enterprise, particularly how well it's going. For example, a person may "feel" that he has memorized a list completely, or that he doesn't understand some Instructions.

The point of these studies for the current discussion Is that they indicate that the knowledge we have about our knowledge and memory Is not simply that we "know that we know X" or that we "believe that adults know what *2+2 i*s." The meta-level knowledge that appears to be useful to cognitive processes like learning, remembering and understanding, or what Flavell (1979) calls generally *metacognition,* covers the full range of "knowing" about other knowledge.

The Phenomenology of Mete-knowledge

The psychological phenomena reviewed here Illustrate the pervasive role of meta-level activity in human cognition. The studies by Kolers and Palef deal with some fundamental properties of human remembering, namely, that one often seems to "know" rapidly and reliably that one doesn't know something, without going through any sort of "memory-scanning" process. In other words, people seem to have "at their fingertips" an Idea of the *extent* of their knowledge in somo kinds of tasks. Recent research on memory has extended this notion, viewing recall as a problem-solving activity that uses knowledge or descriptions of memories just as other problem-solving tasks use domain knowledge (see Williams, 1977, and Norman and Bobrow, 1979).

CollIns's studies indicate that this kind of meta-knowledge may be an Integral part of much of people's everyday reasoning. Flavell's work shows that meta-cognition develops gradually in children and that certain meta-cognitive tasks, like estimating how hard a problem will be or knowing when one has understood directions, are performed surprisingly poorly by young children. Once again, what Is developing here is not the child's knowledge of the world, but his understanding of what he knows and doesn't know, and of his own cognitive performance. What does It mean to "represent" this kind of knowledge?

The Representation of Mota-knowlodge

The AI systems reviewed below all allow the *explicit* declaration of meta-level data-structures in their representation schemes. In other words, the representation formalisms allow encoding of data-structures that "describe" other data-structures. The Issues discussed here have come *up* in many, maybe all, AI systems and are relevant to all representation schemes: predicate calculus, production rules, conceptual dependency nets, semantic nets, procedures, frames, etc. The particular systems described here have attempted to use explicitly represented meta-knowledge.

TEIRESIAS

The use of meta-knowledge evolved naturally in systems developed In what might be called the "Transfer of Expertise" paradigm (Barr, Bennett, and Ciancey, 1979). Systems like DENDRAL and MYCIN perform a complex task by using a database acquired from the human experts who are good at the task. The need to give these systems meta-knowledge, knowledge about their own structure and about what they know, developed naturally as part of the effort to endow them with some *introspective* capabilities, In particular, facilities for automating the acquisition of new knowledge from humans, for doing automatic bookkeeping on the database, and for explaining the system's decisions and reasoning strategies to humans. A prototype system for Incorporating such capabilities into AI programs, called TEIRESIAS, was designed by Randy Davis, and led him directly to the development of techniques for the explicit representation of meta-knowledge (see Davis, 1976, and Davis & Buchanan, 1977).

TEIRESIAS's various meta-knowledge representation structures are all encoded and used differently within the system, each having its own set of data structures and Interpreting procedures. However, the important point Is that all of these structures arose out of the effort to Implement some new, introspective abilities in the system which were needed to facilitate transfer of expertise interactions with humans.

KRL and KRS

The best known of the new frame-oriented representation languages is KRL, being developed at Xerox PARC. The explicit representation of meta-knowledge was already an espoused feature of the first Implementation effort, KRL-0 (Bobrow and Winograd, 1977). Each slot of a unit, or frame, could be tagged with certain *features,* selected from a set of predefined meta-level characteristics, which were used In Inheritance and matching.

Besides the *feature* tags, use of an entire description to describe another description, i.e., as a meta-description, was proposed in KRL-O, but was not thought out further until work on the KRL-1 implementation. Recent work on KRL has strongly Influenced a theory of the formal semantics of representation languages proposed by Brian Smith at MIT (B. Smith, 1976). Smith's formalism, called KRS, Includes meta-level descriptions, called *layers,* as one of Its basic characterizations of a representation. The most Important aspect of Smith's model of meta-level descriptions is that, unlike the ad hoc character of TEIRESIAS's meta-level knowledge, KRS offers a unified conception of the role of meta-level atructurea In which the various layers share the same syntax end Interpreting process.

FOL

Its explicit representation of its own reasoning mechanisms makes the POL proof-checking system (Weyhrauch, 1079) of Interest in this discussion: FOL makes direct *use* of meta-knowledge in a first-order logic representation scheme, based on the idea of *simulation structures* which are used to establish the semantics of expressions. The key Idea is that, since the FOL proof checker Is itself a program, composed of data structures, It Is the natural simulation structure to "attach" to a theory of language/simulation-structure pairs, a theory of reasoning. In such a theory one could reason about (prove theorems about) any particular language/simulation-structure pair by using general theorems about L/SS pairs, *meta-theorems.* Although FOL is a very powerful proof-constructing/checking system in its own right, it is uniquely of Interest in this discussion because of the neat way that meta-level reasoning fits into the formalism.

The State of the Art

Explicit declarations of the form of the system's representation schemes were necessary In TEIRESIAS to implement "introspective" capabilities like *explanation.* The form of the meta-level representations In TEIRESIAS was ad hoc, but their use was clear. On the other hand, the representation of meta-knowledge as *feature tags* in KRL-0 was more uniform, but the ideas about how to use this knowledge were incomplete, involving rather vague ideas of Inheritance and matching. Both KRS and FOL have elegant ideas about how to represent meta-level knowledge In their representation schemes, but have not actually specified yet how this kind of knowledge is to be used.

Meta-knowledge and Computation

The first observation that must be made is that there Is a qualitative difference between human meta-cognitive capabilities, like "knowing not," "meta-memory," and the "lack-of-knowledge Inference," and the current uses of meta-level representational structures in Al systems. In particular, viewing meta-knowledge as additional "facts" *or* "rules" which *describe* the object-level knowledge does not completely capture its essential characteristics.

Typical (pre-meta-knowledge) Al programs can achieve expert performance in their domain and yet be unable to answer questions like "Why did you do this?" or "How do you know that?"--questions that a human expert would naturally be able to answer. It was an attempt to implement these very abilities In TEIRESIAS that led Randy Davis directly to the use of meta-knowledge. Humans acquire, as a natural, integral part of their development and training, knowledge about their own reasoning processes as well as knowledge about what they know. These psychological studies of meta-cognitive behavior are Important because they deal with commonplace human cognitive abilities, like "knowing not" and "meta-memory," that are *Very* difficult to understand In terms of "storage and retrieval" models of memory (see Barr, 1077, and Restle, 1974). This indicates that there are aspects of "knowing" and "remembering" that have so far remained unexplored in Al research--we have only begun to examine the full fabric of behavior that is the reason we *ascribe* knowledge and Intelligence to people.

References

Barr, A. *Meta-knowledge and Memory.* Unpublished Working Paper HPP-77-37, Computer Science Department, Stanford, November, 1977.

Barr, A., Bennett, J., & Clancey, W. *Transfer of Expertise: A Theme for A! Research.* Unpublished Working Paper HPP-79-11, Computer Science Department, Stanford, March, 1079.

Bobrow, D., & Winograd, T. An overview of KRL, a knowledge representation language. *Cognitive Science,* 1977. 1, 1-48.

Collins, A. Fragments of a Theory of Human Plausible Reasoning. TINLAP-2, 1978, 194-201.

Davis, R. Applications of meta-level knowledge to the construction, maIntainance, *and use* of large knowledge bases. Memo AIM-283, Stanford Al Lab, 1978.

Davis, R. and Buchanan, B. *Meta-level knowledge: Overview and Applications.* Proceedings of IJCAI-6, MIT, Cambridge, 1977.

Flavell, J. Metacognition and Cognitive Monitoring: A New Area for Cognitive-Developmental Inquiry. Unpublished manuscript, Department of Psychology, Stanford University, 1979.

Gentner, D., and Collins, A. *Knowing about Knowing: Effects of Meta-knowledge on Inference,* Unpublished manuscript, Bolt Beranek and Newman inc., Cambridge, Mass., 1978.

Kolers, Paul and Sandra Palef. *Knowing not.* Memory and Cognition, 1976, 4:6, 663-668.

Norman, D. A., & Bobrow, D. G. Descriptions: An Intermediate Stage In Memory Retrieval. To *Appear* In Cognitive Psychology.

Restle, F. *Critique of Pure Memory. i*n Solso, R. (Ed.), Theories in Cognitive Psychology; The Loyola Symposium. Potomlc, Maryland: Lawrence Erlbaum Associates, 1974, 203-217.

Smith, Brian. *Levels, Layers, and Planes: The Framework of a System of Knowledge Representation Semantics.* Unpublished Draft, MIT, Cambridge, February, 1978.

Weyhrauch, Richard. *Prolegomena to a theory of mechanized formal reasoning.* Unpublished working paper, Stanford Al Lab, January, 1979.

Williams, M. Some Ovservations on the process of retrieval from *Very* long term memory. (Doctoral dissertation, Department of Psychology, University of California, Sandiego, 1977). Dissertation Abstracts International, 1977.

KNOWLEDGE ENGINEERING IN NUCLEAR PHYSICS

David R. Barstow
Department of Computer Science
Yale University
P.O. Box 2158 Yale Station
New Haven, CT 06520

Gamma ray activation spectra are used by nuclear physicists to identify the elemental composition of unknown substances. Neutron bombardment causes some of the atoms of the sample to change into unstable isotopes, which then decay, emitting gamma radiation at characteristic energies and intensities. By identifying the unstable isotopes, the composition of the original substance can be determined. Since the performance of such analysis relies on large amounts of various kinds of knowledge, the task seems appropriate for the techniques of knowledge engineering. An experimental system, GAMMA, has been developed, based on the generate-and-test paradigm. GAMMA's performance has been good enough that it is currently in use by practicing nuclear physicists.

1. INTRODUCTION

Gamma ray activation spectra are produced by bombarding a sample with neutrons, thereby producing unstable isotopes which begin to decay. While decaying, these isotopes emit gamma rays at characteristic energies and intensities. By measuring the gamma rays emitted by the sample after bombardment, the unstable isotopes (and from these, the elements of the original sample) can be identified. Fig. 1 shows the spectrum produced after bombarding a sample with neutrons. Some of the peaks are labeled by their energies. This is considered to be a high resolution spectrum, since the energies can be identified quite accurately. Fig. 1 also shows the isotopic interpretation made by a nuclear physicist. The peaks are labeled by the isotope which emitted those gamma rays during decay. The elemental interpretation attributes the Na-24 to sodium, the Cl-38 and S-37 to chlorine, the K-40 and Th to natural radiation, and the Ba-137 to an impurity.

The task of interpreting gamma ray activation spectra requires considerable knowledge about nuclear physics, suggesting that the techniques of knowledge engineering may be usefully applied. Some of this knowledge has been codified into a machine-usable data base, and an experimental program, called GAMMA, has been implemented for using this data base within a generate-and-test paradigm. Preliminary results with GAMMA have been good enough that even tne experimental version has already been used for cross-checking analyses by several practicing nuclear physicists.

The work reported herein was accomplished by the author while serving as a consultant to the Computer Intelligence Program at Schlumberger-Doll Research Center. A detailed version of this paper is available from: Dr. W. Frawley, Schlumberger-Doll Research Center, Old Quarry Road, Ridgefield, CT 06877.

2. KNOWLEDGE BASE FOR GAMMA RAY ANALYSIS

The process that produces gamma ray spectra can be seen at six different levels as follows:

(1) elements in original sample
(2) isotopes in original sample
(3) isotopes after bombardment
(1) decays
(5) emissions during decay
(6) detections during decay

The relationships between the levels can be understood in terms of the different kinds of knowledge that play a role. The relationship between levels (1) and (2) involves the relative concentrations of the naturally occurring isotopes of each element. The relationship between levels (2) and (3) is more complex. Under neutron bombardment, a given isotope can go through any of four basic transitions, and the frequency with which the different transitions may occur depends on the particular isotopes involved, on the device used to produce

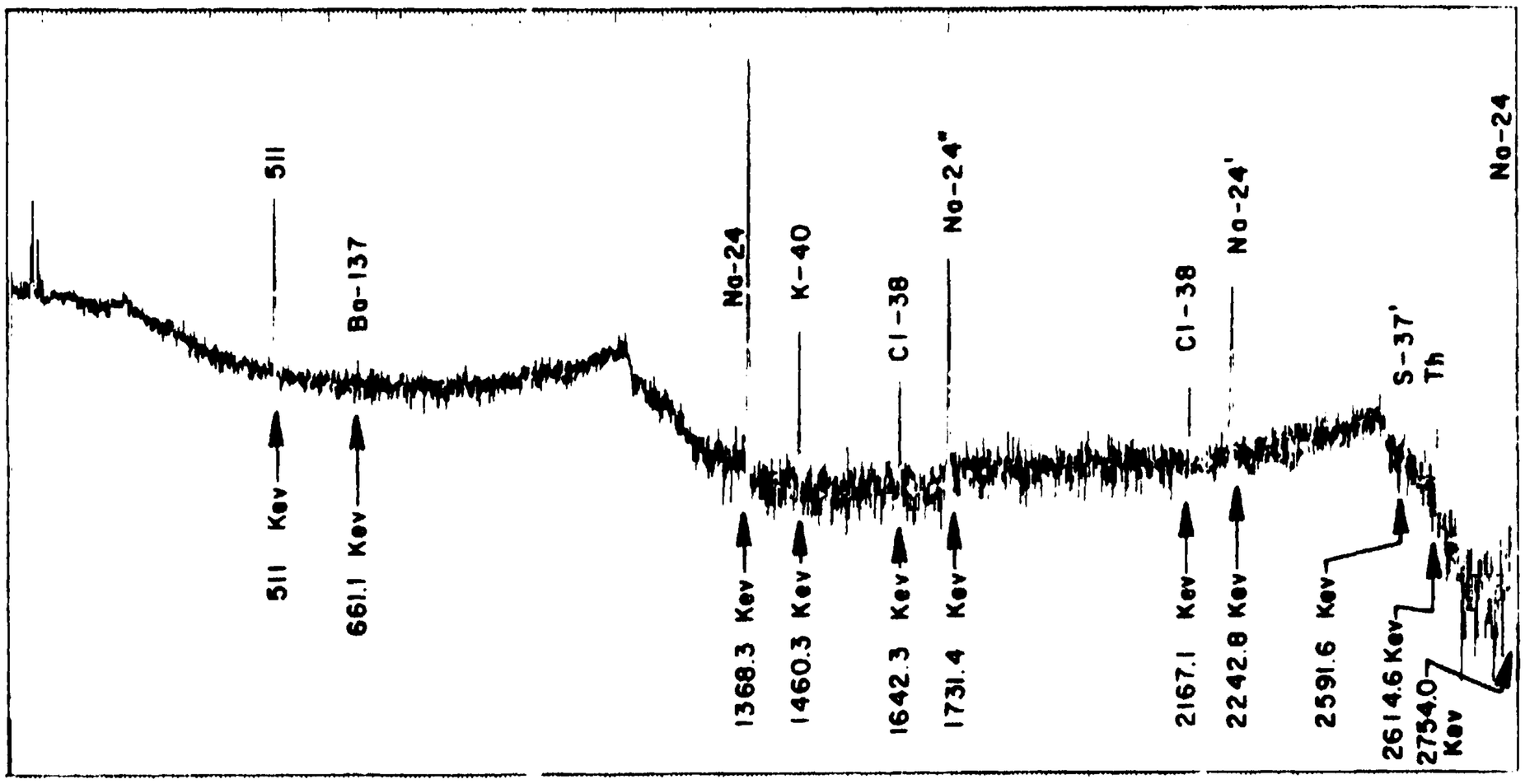


Figure 1.

the neutrons,and on the length of time during which the sample is exposed to neutrons. The principal factors relating levels (3) and (4) are the half lives of the unstable isotopes and the time during which the detector is exposed to the sample after bombardment. The relationship between levels (4) and (5) is relatively simple: every unstable isotope has a characteristic set of gamma rays (with relative intensities) emitted during decay. The relationship between levels (5) and (6) depends on the detector: any given detector will detect some fraction of the emitted gamma rays. This detector efficiency may depend on the energy of the gamma rays. A second factor, the presence of escape peaks (e.g., those labeled Na-24' and Na-24" in Fig. 1), also depends on the energy of the main peak.

Much of the knowledge discussed in this section is available to nuclear physicists in the form of books and articles; we have converted the data from one of these [1] into a machine-usable LISP data base.

3. PARADIGMS FOR KNOWLEDGE ENGINEERING

Based on the discussion of the preceding section, several implementation paradigms can be imagined. Generate-and-test could be based on a list of elements. For each element, one could progress from level (1) to level (6), predicting each level from the previous one. The predicted level (6) peaks could then be matched against the peaks of the spectrum to accept or reject the original element. Backward-chaining would involve progressing through the levels in the same order, but in some cases it might be unnecessary to go all the way to level (6) for all subgoals, because rejection of one subgoal makes the others irrelevant. In addition, the list of elements would only be implicit in the list of rules relating levels (1) and (2). In a forward-chaining (or data-driven) paradigm, one could work upwards from the spectral peaks at level (6) to level (1), using data (or hypotheses) at each level as evidence for hypotheses at the next. Finally, one could imagine some kind of mixed-chaining, using individual peaks in the spectrum to suggest candidates to be evaluated further. (This is roughly the strategy used by nuclear physicists.) We plan eventually to experiment with all of these paradigms.

4. GAMMA

Our first experimental system, GAMMA, uses the generate-and-test paradigm. GAMMA has routines to predict from hypotheses at one level to hypotheses at the next level. Chaining together predictions from a single element at level (1) to level (6) gives a predicted pattern of peaks, which can then be matched against the peaks of the spectrum to determine whether or not the element was present in the original sample. (Note that this technique only works when the hypotheses (i.e., elements) at the top level can be considered relatively independently.)

Prediction from level (1) to level (2) is quite simple, since the relative abundances of the naturally-occurring isotopes of an element are stored in GAMMA'S data base. Prediction from level (2) to leve (4) is collapsed into one step, from isotopes in the sample to decays of unstable isotopes after neutron bombardment. We

are currently using an analytically determined formula which incorporates an estimated cross-section for each possible transition, the half-life of the resulting isotope, and the three relevant time periods (bombardment, detection, and the delay between them). Prediction from level (M) to level (5), from decays of single isotopes to emitted gamma rays, is quite simple, since for most isotopes the characteristic gamma ray energies and intensities have been measured and this list is stored in GAMMA'S data base. The final prediction step, from level (5) to level (6), is based on two experimentally determined functions relating detector efficiency to a peak's energy.

Due to noise and error, the predicted peaks may not occur in the spectrum at precisely the predicted energies and intensities. To determine whether or not a predicted set of peaks actually occurs, GAMMA currently uses a simple rating scheme. The match value of a set of peaks is the average of the match values of the individual peaks, weighted by intensity of the predicted peak (i.e., predicted peaks of high intensity are more important). For single peaks, there are two cases. (1) If there is no spectrum peak within a window of the predicted peak, and the predicted intensity is larger than the background intensity, a negative rating (based on the ratio of the intensities) is assigned. (2) If there is a spectrum peak within a window, the rating incorporates an energy rating (based on the difference between the predicted and detected energies) and an intensity rating (based on the ratio of the intensities and whether the detected is less than the predicted intensity).

This scheme is intended to make use of both positive and negative evidence, allowing for the imprecision of the prediction and detection processes and for the background radiation. While it is based largely on guesses by nuclear physicists, we have found it to be quite satisfactory. In fact, the difference between the presence and absence of an unstable isotope in such high resolution spectra seems to be great enough that almost any "reasonable" matching procedure would work well, as long as it does not assume too much precision on the part of the predictor and detector.

Let us reconsider the spectrum in Fig. 1 from GAMMA'S viewpoint. The statistically significant peak energies and intensities are computed by a preprocessor developed by the human experts. For each element, GAMMA predicts a set of peak energies and intensities, and then matches these against the spectrum peak list. The elements are then sorted by their match values. From this list, the spectrum peaks are labeled by all elements which matched the peak and whose match rating was above a certain threshold. The resulting interpretation is substantially the same as that of the human nuclear physicists given earlier. The only differences are that GAMMA found no elemental scuce for the Ba-137 peak (which the physicist attributed to an impurity), and that GAMMA labeled the Th peak as Po-212 (which is one isotope in the Thorium natural radiation chain). For the curious, the substance being analyzed is a sample of Alaskan salt water.

5. DISCUSSION

We selected the generate-and-test paradigm for our first implementation because it is relatively simple to implement. In fact, the major effort involved in building GAMMA was the conversion of the data base from human to machine-usable form. This first version has been fairly successful, and is, in fact, being used as an aid by nuclear physicists. While they are not yet ready to blindly accept GAMMA's interpretations, they have found GAMMA to be useful in cross-checking their own analyses. The fact that this version succeeded fairly well is due primarily to several features of the domain: an exhaustive generator is available, the prediction process is fairly simple, and the high resolution of the spectra permits the use of a relatively imprecise matching process.

In the future, we plan to pursue two directions. First, we hope to improve the current version of GAMMA by incorporating the use of multiple spectra (which would permit more use of half-life information) and re-calibration on the basis of an initial interpretation. Second, we plan to experiment with several other paradigms, including backward-chaining, forward-chaining, and mixed-chaining. In so doing, we hope to learn more about the relative merits of these paradigms, and what domain characteristics are important in selecting one over another.

ACKNOWLEDGEMENTS

Those at SDRC have been of invaluable assistance: nuclear physicists A. Becker and J. Schweitzer; W. Frawley, H. Austin and P. Pruchnik of Computer Intelligence; K. O'Brien and P. Martinich of the office staff. K. Pitman of MIT-LCS designed and builehe user interface for GAMMA. R. Davis of MIT-AI and M. Stefik of SU provided helpful comments on earlier drafts.

REFERENCES

[1] Erdtmann, G. Neutron__Activation__Tables, Verlag Chemie, New York, 1976.

THE ROLES OF KNOWLEDGE AND DEDUCTION IN PROGRAM SYNTHESIS

David R. Barstow
Department of Computer Science
Yale University
P.O. Box 2158 Yale Station
New Haven, CT 06520

In the earliest attempts to apply artificial intelligence techniques to program synthesis, deduction (that is, the use of a general purpose mechanism such as a theorem prover) played a central role. Some recent systems have relied almost exclusively on knowledge about programming in particular domains, with no significant role for deduction. Even in such knowledge-based systems, however, there seems to be an important role for deduction in testing the applicability conditions of specific programming rules. This auxiliary role for deduction can be seen clearly in a hypothetical synthesis of a breadth-first enumeration algorithm. The hypothetical synthesis also demonstrates the utility of detailed programming knowledge for dealing with non-algorithmic specifications.

1. INTRODUCTION

In the earliest attempts to apply artificial intelligence techniques to the programming problem, deduction (i.e., the use of a relatively general purpose deductive mechanism such as a theorem prover) played a central role. The Heuristic Compiler [9], based on the GPS formalism, used means-end analysis to determine a sequence of operators that achieved the desired result. Green [4] and Waldinger [10] developed systems that used resolution theorem provers to construct programs. First, a proof was developed for a theorem derived from the input/output specifications of the program. Then a program was constructed directly from the theorem's proof. The only knowledge (i.e., specific detailed facts, rules, or procedures) available to these early systems was knowledge about the target language. In the Heuristic Compiler, this was encoded in the operator difference tables. The systems of Green and Waldinger had axioms describing various target language constructs. Later, Manna and Waldinger [6] showed the importance of knowledge about the domain, in addition to knowledge about the target language. Their recent work [7] has also emphasized the utility of knowledge about certain general aspects of programming, such as the introduction of conditionals and recursive calls, but still within a deductive framework.

In contrast, PECOS [2, 3] represented an application of a different artificial intelligence paradigm, knowledge engineering, to the programming problem. PECOS had a knowledge base of about four hundred specific rules (or facts) about many aspects of symbolic programming, including such abstract concepts as sorting, enumeration, collections, and mappings, and such concrete concepts as loops, lists, arrays, property lists, and record structures. PECOS could apply these rules to the task of implementing abstract algorithms. PECOS was able to implement algorithms in a variety of domains, including symbolic programming (simple classification and concept formation algorithms), graph theory (a reachability algorithm), and simple number theory (a prime number algorithm). PECOS's knowledge of alternative implementation techniques enabled the construction of a variety of implementations for each algorithm.

Although deduction (in the sense described above) played no significant role in PECOS, there is a rather important role that deduction can and should play in knowledge-based automatic programming systems. This role can be seen by considering a relatively simple rule:

> If you know that an object is larger than (or equal to) every member of an ordered sequence, the object may be added to the sequence by inserting it at the back.

This material is based upon work supported by the National Science Foundation under Grant No. MCS78-03827. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

This rule is something that human programmers probably know, and seems reasonable to include in an automatic programming system. The question is: How should the condition in the rule be tested? There are many and varied situations in which the condition holds. One simple case is when the insertion routine for a selection sort is being written; another will be seen in the next section. Since there are so many situations in which the condition holds, the right way to test it seems to be to pass it off to some mechanism that can test whether or not it holds for the program being constructed that is, to a relatively general deductive mechanism that can test and prove properties about programs (and partially developed ones). Thus, we see that deduction plays an important (if auxiliary) role when retrieving and applying this rule, and it is this role that will be the focus of the remainder of this dicussion. This is not to say that other roles for deduction are unimportant for automatic programming, but merely that this role is especially important for knowledge-based automatic programming systems, and is one that has received relatively little emphasis in automatic programming research. (The one notable exception is the Programmer's Apprentice [8].)

In the next section, a hypothetical synthesis of a graph theory algorithm will be presented. The synthesis consists of a sequence of reasoning steps, each suggesting a fact or rule that seems useful for an automatic programming system to know. Several of the steps depend on some condition holding for the program under construction. These are the conditions that seem appropriate tasks for a deductive system, and are indicated by the word "PROOF" in the synthesis. Note that the synthesis is driven primarily by the programming knowledge, rather than by the deductive system. The synthesis also demonstrates the utility of programming knowledge for non-algorithmic program specifications. (Recall that programs were specified to PECOS as abstract algorithms.) Finally, it should also be noted that the problem considered in the next section is beyond the capabilities of any current automatic programming system.

2. HYPOIHETICAL SYntHESIS

2.1 Specification

> Write a program that inputs the root R of a tree with a known successor function CHILDREN(N) and constructs a sequence of all (and only) the nodes of the tree, such that if the distance from R to N1 is less than the distance from R to N2, then N1 precedes N2 in the sequence (where the distance from X to Y, denoted by D(X,Y), is the number of arcs between X and Y; thus, D(R,X) is the depth of X).

This specification is basically a non-algorithmic description of a breadth-first enumeration of the nodes in a tree. One way for an automatic programming system to deal with this specification would be to recognize that a breadth-first enumeration is the desired program, and to use specific knowledge about breadth-first enumerations (perhaps even a template) to produce a program. In this hypothetical synthesis, we will consider the more interesting case in which the basic algorithm must actually be developed.

2.2 Overview

The synthesis process goes through four major stages. First, the task is broken into two parts, a "producer" that generates the nodes in the desired order, and a "consumer" that builds a sequence from the nodes generated by the producer. The second stage involves constructing the consumer, which simplifies into a simple concatenation operation. The third stage, in which the producer is built, is relatively complex but eventually a simple queue mechanism is developed, in which nodes are generated by taking them from the front and the children of each generated node are added at the back. Finally, the two processes are combined together into a simple WHILE loop.

2.3 The Synthesis

(1) Since the input is a tree and the output is a sequence whose elements are the nodes of the tree, the central task is one of using objects from one structure (the set of nodes) to build another structure (the sequence of nodes). Such a task can be implemented by building two processes, a producer and a consumer. The producer produces the objects one at a time and the consumer consumes them. In this case, the producer must produce all of the nodes of the tree and the consumer must build the desired sequence from them.

(2) When objects are being transferred one at a time, there is some order in which the objects are being transferred. This temporal order will be referred to as the "transfer order." There are many possible transfer orders. When the task of the consumer is to build a sequence satisfying some ordering constraint, one possibility is to force the transfer order to

satisfy the same constraint. In this case, the constraint is:

if D(R,X)<D(R,Y) then transfer X before Y

(3) The consumer is a process that accepts objects one at a time and builds a sequence S from them. The build process is one of adding an individual object to S. After each addition, S must satisfy a particular constraint:

if D(R,X)<D(R,Y) then X precedes Y in S

(4) Initially, S must be empty. Thus, the initialization step of the process is the following:

S := <>

where <> denotes the empty sequence.

(5) For sequences, the addition process involves finding a suitable position to add the new object, followed by inserting it there. In this case, since P follows every element in S according to the ordering constraint (proof given below), the back is always a suitable position. Thus, the addition step (the body of the consumer) is:

P := receive from producer
insert P at the back of S

PROOF: P follows every element in S

The ordering constraint on S is:

if D(R,X)<D(R,Y) then X precedes Y in S

The order in which the consumer receives the objects is:

if D(R,XXD(R,Y) then X is received before Y

Thus, if X must precede P in S, X is received before P. Thus, when P is received, any object that must precede P in S has already been received. Therefore, P follows every element in S. QED

(6) The consumer is finished:

Initialization;
S := <>
Body:
P :s receive from producer
S :r S . <P>

where "." signifies concatenation of sequences.

(7) The producer is a process that produces objects one at a time, guaranteeing that each object is produced once and only once. The set of objects to be produced is the set of nodes of the tree. The order in which they are produced (the "production order") must satisfy the following constraint:

if D(R,X)<D(R,Y) then produce X before Y

(8) In a tree, the set of nodes is the set of objects that are reachable from the root of the tree by following zero or more arcs. How should this set be constructed? The obvious way is to follow all arcs from the root as far as possible.

(9) At any point in the process of following all arcs, there will be certain nodes that have been found and certain that have not. Of those that have been found, some will have had their successors looked at and others will not. Thus, at each point, the state of knowledge about the nodes can be described as a mapping, which we will call MARK:

Domain: nodes

Range: {"CHECKED"."UNCHECKED"."UNKNOWN")

with the following semantics:

MARK[X]="CHECKED"
iff X is known to be reachable
and X's successors have been checked
MARK[x]="UNCHECKED"
iff X is known to be reachable
and X's successors have not been checked
MARK[x]="UNKNOWN"
iff X is not known to be reachable

(10) Initially, all that is known to be reachable is the root, and it's successors haven't been checked; everything else is not known to be reachable:

MARK[X]="UNCHECKED" iff X is the root
MARK[X]="UNKNOWN" iff X is not the root

(11) Each step in the process consists of picking one node that is known to be reachable but whose successors haven't been checked, and to check its successors (as well as marking it as being checked). If a successor is "UNKNOWN" then change it to "UNCHECKED". But if it is "CHECKED" or "UNCHECKED", then don't do anything with it, since it has already been found. Thus, the body of the process is:

```
X := any node narked "UNCHECKED"
change MARK[X] from "UNCHECKED" to "CHECKED"
for all successors, Y, of X:
     if MARK[Y] = "UNKNOWN"
         then change MARK[Y] from "UNKNOWN"
                                 to "UNCHECKED"
```

(12) When no "UNCHECKED" nodes remain, all of the arcs have been followed, and everything marked "CHECKED" is reachable. Hence, the termination test is:

```
is MARK-1["UNCHECKED"] empty
```

where MARK-1 denotes the inverse image under the MARK mapping. The set of reachable nodes is:

```
MARK-1["CHECKED"]
```

(13) We now have a way to determine the set of nodes to be produced. In order to actually construct a producer, we need a way of enumerating them. One way would be to construct the set and then enumerate the objects in the set. In this case, however, we can produce them as we go along. We can do this since each node is marked "CHECKED" only once.

PROOF: a node is marked "CHECKED" only once

Initially no nodes are marked "CHECKED", and there are only two changes that happen in the mapping:

```
change MARK[X] from "UNCHECKED" to "CHECKED"
change MARK[X] from "UNKNOWN" to "UNCHECKED"
```

Thus, once a node is marked "CHECKED", it's marking is never changed. QED

(14) But this technique of producing them as they are checked will work only if the X's are picked to satisfy any ordering constraints on the producer. So the producer body is now the following:

```
X := any node in MARK-1["UNCHECKED"]
change MARK[X] from "UNCHECKED" to "CHECKED"
send X to consumer
for all successors, Y, of X:
     if MARK[Y] = "UNKNOWN"
         then change MARK[Y] from "UNKNOWN"
                                 to "UNCHECKED"
```

where the "any" operation must satisfy the following constraint:

```
if D(R,X)<D(R,Z) then X is selected before Z
   (even if X or Z is not marked "UNCHECKED")
```

(15) The constraint can be satisfied if the node X selected by the "any" operation satisfies the following property:

```
D(R,X)<D(R,Z) for any Z marked "UNCHECKED"
     or marked "UNKNOWN" and reachable from R
```

(16) If the X selected is the closest of those nodes marked "UNCHECKED" or "UNKNOWN", then the property will be satisfied. This can only be done if one of the nodes marked "UNCHECKED" satisfies the property. If there is such a node, that node must be the closest of the nodes marked "UNCHECKED". However, how do we know that such an X is also closer than those nodes marked "UNKNOWN"?

PROOF: if X is the closest "UNCHECKED" node and Z is "UNKNOWN" and reachable from R, then D(R,X)<D(R,Z)

Since Z is reachable from R, there is some path from R to Z. Then Z is reachable from some node marked "UNCHECKED", call it W. This is true since either (i) R is "UNCHECKED" and W can be R; or (ii) the path from R to Z begins with some nonzero number of "CHECKED" nodes and the next node (which is the desired W) cannot be "UNKNOWN", since all of a node's "UNKNOWN" successors are changed to "UNCHECKED" as the node is changed from "UNCHECKED" to "CHECKED". Since 0<D(W,Z), and since, in a tree, there is only one path from R to Z or W, D(R,W)<D(R,Z). Since W is "UNCHECKED", D(R,X)<D(R,W). Hence, D(R,X)<D(R,Z) . QED

(Note that a proof not dependent on the graph being a tree would have to use the fact that the closest node was being taken each time.)

(17) The producer's action has been simplified:

```
X :r closest node marked "UNCHECKED"
change MARK[X] from "UNCHECKED" to "CHECKED"
send X to consumer
for all successors, Y, of X:
     if MARK[Y] = "UNKNOWN"
         then change MARK[Y] from "UNKNOWN"
                                 to "UNCHECKED"
```

(18) Part of the body involves a test of whether a node is "UNKNOWN". We will now see that this test must necessarily return "TRUE", and hence can be omitted from the action of the producer:

```
X := closest node marked "UNCHECKED"
change MARK[X] from "UNCHECKED" to "CHECKED"
send X to consumer
for all successors, Y, of X:
     change MARK[Y] from "UNKNOWN"
                         to "UNCHECKED"
```

PROOF: every successor of X is "UNKNOWN"

Every successor of X was initially "UNKNOWN", since every node (except the root, which is not a successor of any node), was initially "UNKNOWN". The mark of an "UNKNOWN" node is changed in only one place, when the node is the successor of some other node. Thus, if a successor of X is not "UNKNOWN", it must be the successor of some node other than X, which is impossible because the graph is a tree. Hence, every successor of X is "UNKNOWN". QED

(19) At this point, notice that the only marking that is really used is the "UNCHECKED" marking. Thus, the mapping can be simplified somewhat by getting rid of references to the other two possibilities (i.e., making them implicit instead of explicit):

```
Initial state:
  MARK[R]="UNCHECKED"
Each step:
  X := closest node in MARK-1["UNCHECKED"]
  undo MARK[X]
  send X to consumer
  for all successors, Y, of X:
      MARK[Y] :r "UNCHECKED"
```

(Note: if MARK[X] is not "UNCHECKED", we will consider it to be undefined.)

(20) The MARK mapping can be inverted (call the inverted mapping MARK'), so we have:

```
Initial state:
  MARK'["UNCHECKED"] = {R}
Each step:
  X := closest node in MARK'["UNCHECKED"]
  remove X from MARK'["UNCHECKED"]
  send X to consumer
  for all successors, Y, of X:
      add Y to MARK'["UNCHECKED"]
```

(21) The MARK' mapping can be represented as a record structure, with one field for each domain element. And since there is only one domain element of concern, "UNCHECKED", there is only one field, so we might as well just use the set itself, and call it Q. Thus, the producer is:

```
Initial state:
  Q = {R}
Termination test:
  is Q empty
Each step:
  X := closest node in Q
  remove X from Q
  send X to consumer
  for all successors, Y, of X:
      add Y to Q
```

(22) If Q is represented as a sequence, ordered by distance from the root, the "closest" node will be the first element:

```
X := first element of Q
```

(23) In a sequence, a removal operation normally requires searching for the location in which X is stored, but since X was taken as the first element of Q, we can simply remove the object stored in the first location:

```
remove first element of Q
```

(24) In a sequence, an addition operation normally requires searching for a suitable position to insert the new element, but in this case we can show that the back is a suitable position, since Y follows every element in Q according to the ordering constraint.

```
insert Y at back of Q
```

With the following lemma, the necessary proof is fairly simple.

LEMMA: for some X in Q,
D(R,X)<D(R,Z)1D(R,X)+1 for all 2 in Q

This is clearly true initially, since Q = {R}. Suppose it is true at some point. There are only two changes in Q that can be made during one step:

(a) X is removed from Q
(b) Y is added to Q

(a) After X is removed, either Q is empty (in which case adding Y restores the desired property) or there is some new closest node, call it W. We know:

D(R,X)iD(R,W)iD(R,Z) for all Z in Q

so substitution (into the inequality that held before X was removed) gives us:

D(R,W)iD(R,Z)<D(R,W)+1 for all Z in Q

(b) After Y is added, since D(R,Y)=D(R,X)+1,

D(R,Z)iD(R,X)+1=D(R,Y)<D(R,WM for all Z in Q

Thus, by induction we have:

for some X in Q,
D(R,X)iD(R,Z)<D(R,X)+1 for all Z in Q

QED

We are now ready to prove the condition necessary for adding Y, a successor of X, at the back of Q.

PROOF: Y follows every element of Q

The ordering constraint on Q is:

if D(R,X)<D(R,Z) then X precedes Z in Q

So what we wish to show is:

if D(R,X)<D(R,Z) for all Z in Q,
and Y is a successor of X,
then D(R,Z)<D(R,Y) for all Z in Q.

Since the successors of X are all the same distance from X (and hence from the root), their relative order is unimportant; thus, showing the above is sufficient to enable us to insert them one at a time at the back.

The lemma indicates that there is some X in Q such that everything else in Q is at least as far from the root as X, but not further away than 1 arc. Therefore, this property must hold for the closest node in Q. Since D(R,Y) = D(R,X)+1, if Y is a successor of X, we know that D(R,Z)<D(R,Y) for all Z in Q. Thus, Y follows every element of Q. QED

(25) The producer is finally finished:

```
Initialization:
  Q := <R>
Termination test:
  is Q empty
Body:
  X := first element of Q
  remove first element from Q
  send X to consumer
  for all successors, Y, of X:
      insert Y at the back of Q
```

(26) One simple way to combine a producer and a consumer is to use a WHILE loop:

```
S := <>
Q := <R>
while Q is not empty do
  X := first element of Q
  remove first element from Q
  insert X at back of S
  for all Y in CHILDREN(X) do
      insert Y at back of Q
```

From this point on, rules about simple symbolic programming (such as those in PECOS) could produce the final implementation. The interesting aspects involve representing the sequences S and Q (each is probably best represented as a linked list with a special pointer to the last cell), and the "for all" construct (which depends on the representation of the set returned by CHILDREN(X)).

3. TOWARD AN IMPLEMENTATION

As noted earlier, the hypothetical synthesis presented above is beyond the capabilities of any current automatic programming system. Nonetheless, it seems a reasonable goal for the near future. We are currently attempting to codify programming knowledge about elementary graph algorithms, and breadth-first enumeration is one of our target programs (along with such graph problems as minimum-cost spanning tree, reachability, and connectedness) [1]. The preliminary design of our system (called PKX, for Programming Knowledge experiment) reflects the roles of knowledge and deduction discussed here. In designing and implementing PKX, several issues seem particularly important.

One of these is simply: What knowledge about graph algorithms should be represented? Much programming knowledge is currently available only informally (or even subconsciously) in the heads of programmers, and it must be explicated to a rather detailed level if it is to be usable by a machine. The hypothetical synthesis above suggests some of the necessary knowledge. For example, step (9), which introduced the MARK mapping, is based on knowing a particular scheme for saving the state of an enumeration of the nodes in a graph. But this knowledge must still be reduced to some kind of rule form, and organized into a usable data base. In trying to use this knowledge, two other issues arise. PECOS's refinement paradigm was fairly successful and convenient for exploration, primarily because of the rarity of dead-ends. In PKX, especially when dealing with non-algorithmic specifications, this is unlikely to be the case, so we expect more search to be involved. (And the cost of search is higher because of the cost of the deductive system proving some rule applicability conditions.)

Another kind of search involves selecting the "best" implementation from among the successful paths. In order to control both kinds of search, we plan to make extensive use of plausibility and preference rules, such as those of LIBRA [5].

As far as the deductive system is concerned, if we take the five proofs in the above synthesis as representative, the complexities of the deductions seem to be generally comparable to those of state-of-the-art theorem provers. The one exception is that of step (24), which involved a lemma in addition to the main proof. In order to experiment with axiomatizations of the relevant concepts, we have built an experimental resolution theorem prover, and were able to construct proofs for steps (5) and (16). The proofs for steps (13) and (18) both involve changes made to a data structure, a topic that will require more effort to axiomatize adequately. Of course, various properties of graphs and trees, as well as operations on them, will also have to be axiomatized. Perhaps the most important conclusion from our experiment with a resolution theorem prover was that it was too undirected for our purposes, so we plan to design and build a more special-purpose deductive system for use with PKX.

4. CONCLUSIONS

In order for future automatic programming systems to be truly useful when applied to real-world tasks, they will necessarily have to deal with complex programs at a rather detailed level. It seems to me that this will only be possible if these systems have effective access to knowledge about what we today consider to be the task of programming. While some of this knowledge certainly involves general strategies, such as the conditional and recursion introduction rules of DEDALUS, much of the knowledge also involves rather specific detailed facts about programming in particular domains, such as PECOS's rules about symbolic programming. (This may, in fact, merely be another restatement of the trade-off between generality and power.) The first attempts to build such programming knowledge into an automatic programming system involved applying the knowledge to algorithmic specifications, in which the user described the program abstractly and the system produced a concrete implementation. The hypothetical synthesis of a breadth-first enumeration program suggests that specific detailed knowledge about programming can also be of significant value for non-algorithraically specified programs. It also illustrates an important role to be played by deduction in such knowledge-based automatic programming systems: as a mechanism for answering particular queries about the program being constructed, as part of the process of testing the applicability of particular rules. This auxiliary role for deduction seems an important one for future development.

ACKNOWLEDGEMENTS

Much of this work resulted from discussions during seminars in Knowledge-based Automatic Programming held at Yale in Fall 1977 and Spring 1979. Mark Zbikowski built the resolution theorem prover. Chuck Rich, Howie Shrobe, Dick Waters, and Richard Waldinger provided helpful comments on an earlier draft of the paper.

REFERENCES

[1] Barstow, D.R. "Codification of programming knowledge: graph algorithms," Yale University, Department of Computer Science, TR 149, December 1978.

[2] Barstow, D.R. "An experiment in knowledge-based automatic programming," Artificial Intelligence, (to appear) 1979.

13] Barstow, D.R. Knowledge-baaed Program Construction, Elsevier North Holland, 1979.

[4] Green, C.C. "The application of theorem proving to question-answering systems," Stanford University, Computer Science Department, AIM-96, August 1969.

[5] Kant, E. "A knowledge-based approach to using efficiency estimation in program synthesis," Sixth International Conference on Artificial Intelligence, (to appear) 1979.

[6] Manna, Z. and Waldinger, R. "Knowledge and reasoning in program synthesis," Artificial

[7] Manna, Z. and Waldinger, R. "Synthesis: dreams => programs," SRI International Technical Note 156, 1977.

[8] Rich, C. and Shrobe, H.E. "Initial report on a LISP programmers apprentice," IEEE Transactions on Software Engineering. 4, November 1978, 456-467.

[9] Simon, H.A. "Experiments with a heuristic compiler," Journal of; the. ACM, 10, April 1963, 493-506.

[10] Waldinger, R. and Lee, R.C.T. "A step toward automatic program writing," International

Washington, D.C., 1969, 241-252.

PROPERTY DRIVEN DATA BASES

Jean Paul BARTHES, Michel VAYSSADE, and Monika MIACZYNSKA*
Department of Applied Mathematics and Computer Science
University of Technology of Compiegne
60206 Compiegne Cedex, France

In this paper we suggest how, focusing on the concept of property, one can use a very small number of operators (functions) to organize a factual data base whose information structures are constructed as simple association lists. The approach was used to design and implement a data base management system, operating in a small-core/slow-secondary-storage environnment, as a part of a low cost multimicroprocessor robot system.

1. INTRODUCTION

The A.I. community has developped efficient techniques for in core handling of symbolic knowledge representation. On the other hand commercial data bases use extensively secondary storage, but for various reasons suffer from very rigid data structures. Both sides have proposed abstract representation models with the hope to find an efficient solution [1,2,3,] However difficulties still subsist when one has to actually implement the models, in particular when secondary storage is involved. Our recent work aimed at building a low cost multimicroprocessor robot system led us to address this problem in a small-core/large-but-slow secondary storage environnment. Our objective was to implement a data base using a 16 bit microprocessor. We then reconsidered the simple property value representation which proved to be surprisingly powerful. Indeed it is possible to build easily a data base management system as an interpretor of a set of properties, each being described in turn as a property - value list (introducing readily a metalevel of description). Then, while processing data, the presence of any given property in the data stream can trigger associated functions, which renders the overall approach somewhat analogous to a production rule system. Hence the name of Property Driven Data Base.

The main characteristics of our approach are

- no distinction between physical and logical models, which avoids the so-called "data independence" problem plaguing most commercial systems *[6]* .

- no a priori information (or object) classes ; each entity stands for itself. It is represented as a property list and possesses an internal name. The representation is very similar to the one found in Abrial [I} .

- operators imbedding semantic meaning are attached to properties.

- relational information is implemented at property level.

- access to the data base is done through specific entry points unlike in Mc Dermott [4].

- privacy and user classes are dealt with using the notion of model and related access paths.

- access paths are computed dynamically.

*This work was supported in part by CNRS under ATP granted by the Committee for Industrial Robotics and by the University of Technology of Compiegne.

2. DATA BASE MODEL

2.1 Static View of the Data Structure

It is built from 3 distinct ensembles, E, the entity set or set of objects, P, the property set or set of attributes, and V, the value set. It must be noticed that we do not consider any "relation set". Relations an implemented via the property set which accordingly is partitioned into 2 subsets, T, the terminal property set, and S, the structural property set, with P - T u S. To each entity e. of E is associated a subset $P_i = T_i U S_i$ of P. To each property tj of Ti we associated a subset Vij of V. To every property sj of Si we associate a subset Eij of E. Hence tj describes local attributes of ei, whereas sj gives the connections of ei with other entities (equivalently the relations). Furthermore for each entity IPiI-ISiI ♦ ITiI

may be different and depends solely on what is known about e. (|X| denotes the cardinality of X).

Elementary Operations: Three basic operators are needed:

π associating each entity to itscorresponding properties: π : E -> P (where P denotes the power set of P)

- T yielding the values associated to a terminal property t. of e. (a restricted case of the LISP function GET): τ : E x T -> V

γ yielding all entities linked to e. (a different use of GET : γ : E x S -> E

Additional Definitions:

Successor: An entity e_k is a "successor" of a given entity e. if e_k e y(s,e.).

Predecessor: A "predecessor" e_k of e. is an entity for which e. is a successor.

Access and Entry Port: Every entity having no predecessor is called an access of the data base.

The data base, G, will be said to be normalized if all accesses are made the successor of a single entity e which thus becomes the only access of G.

However e could be linked to other entities in G. All successors of e will be called entry-ports, abbreviated ports.

2.2 Exploring Mechanisms

Information can be obtained incrementally from any given location in the data base without any a priori knowledge of its organization, simply using operators π, τ, γ. In particular from a statical point of view it is possible to obtain full information solely from the elements (e , t, τ, γ) e being the data base access. This possibility should be reserved to the data base manager, normal users being excluded.

2.3 Modifying the Data Base

Deleting Information: Deleting values or links at the entity level is straightforward. However deleting an entity requires updating of all its predecessors.

Adding Information: is a much more interesting operation. Indeed the overall structure of the data base must remain consistent. It is convenient to associate to each terminal property t. of P a screening operator ξ_i such that for any external value W:

$$\xi_j(W) = \begin{cases} W & \text{if W is consistent with } V_{ij} \\ \phi & \text{otherwise} \end{cases}$$

In practice the corresponding function carries on such tasks as validity checks and external/internal format translation. This permits to associate a specific data type to any given property, and to support easily heterogeneous collections of data.

Retrieving Information : Let us consider a subset E. od E, a property p. of P and an external set of values W s t e n t with P., i.e. such that $\xi_{.}(W) \neq \phi$. One can define one or more selection operators w_i. associated with p., extrac ting from E. all elements satisfying agiven criteriunm w..(E.,W) = EcE.(u). is a mapping $w_{i.}$: E x W->E^J). The involved criterium can be represented by a Boolean function $\beta_{.j}$, e.g.

$$E = \{e_i \mid \beta_j(V_{ij},W)\} \quad \text{if } p_j \in T_i$$

The selection operator w_i is similar to Lindgreen's detection operator *[3]* but is not constrained to "HIS" (Homogeneous Information Classes). More will be said concerning its implementation.

2.4 Accessing Structures

Map and Models: A large data base especially in A.I. type of applications may contain many chunks of information stored in a very disorderly fashion. Whenever secondary storage is used it is necessary to minimize the number of acces ses. We introduce to this purpose the notion of map and models. In addition to increasing access efficiency they permit to solve neatly some of the problems related to privacy and user's view of the data base. An interesting feature of such models is the possibility to construct them a posteriori using pattern recog nition techniques. In order to define them we need some additional concepts.

Following Property: Property P will be said to be following a property P. associated with entity e., if e is a successor of e. and $P_n \in \pi(e)$.

We can define mapping r_a such that *V* : S -> P and extend it over the set of properties Γ^{*}_{a} : P + P.

Map: It is the graph (P,Γ^{*}_{a}) =connecting properties.

Mode 1: A model is any subgraph of the map. A model implements a user's view of the data base.

Exploring Using Criteria: Let E. a subset of E be named a position. Let X be an operator which when applied to E_i will yield a subset Eq deri-

ved from a given property p. and a given model M^n: $X(E_i, P_k, M^n) = E_q$ where $x: E \times P \times M \rightarrow E$

E is the set of entities that can be reached from E. by using some existing path in M leading to a property p_k. Access paths are dynamically computed in a model.

Clearly some entities in the data base having property p_k will not be reached, even from e, if there are no existing paths in the given model M. Furthermore if, instead of M, one considers subgraphs of M", access to entities will be further restricted. For example if E. is a set of entities each one having property p., one can easily obtain all entities having both properties p. and p_k by applying $x(E_i, P_k, \{p_k\})$• An analogous result can be obtained from any position E_i using $x(X(E_i, P_j, M^n), P_k, \{P_k\}$

Associative Access: It is then possible to combine x with previously defined selection operators. Using a screening operator (or filter) w_i. associated with a property p. one can distinguish 2 ways of filtering information:

- cumulative filtering $E \blacksquare w_i (w_k) (E., W_k), W.)$ yielding all entities from position E. satisfying 2 (or possibly more) criteria associated with one (or more) properties.

- repetitive filtering $E = w_i (x(E., p., M), W.)$ yielding all entities that can be reached from position E. according to the model M and that satisfy criteria implemented by w.. Obviously such filtering can be applied recursively.

Classes of Users, each having various access privileges, can be defined by means of domains: $D = (M_U, P_U, Z_U)$ including a set of models M_U, a set or properties P_U, associated filters and screening operators Z_u.

3. IMPLEMENTATION AND DISCUSSION

An implementation was done at the UTC on a PDP 11. The host language is a shallow binding LISP dialect, functions were developed allowing to store and to access directly variable length character strings identified by variable length keys, sequential access was added to the LISP I/O subsystem for reading bulk data and for producing printed forms. The volume of object code limited us to 5000 list cells and 1250 atom hash cells. This realizes in effect a small processing window over the data base held in secondary storage. An additional primitive for reclaiming atom space had to be developed for dealing with the clogging of the hash table.

Data is structured as association lists. Entities linked by structural properties are reversely chained. Ports are implemented via the hash table. Operators are implemented as LISP functions

A present restriction in our approach is the lack of possible deduction when the search fails. However the general structure makes it rather straight-forward to introduce "advising functions" at the property level. Furthermore screening operators could be implemented as microprograms to be loaded into "stream processors" to take full advantage of the recent developpments in secondary storage architecture [6]. On the other hand, although path access leads to iterative retrieval of entity subsets, it offers a solution to the privacy problem and to the implementation of specific user's view of the data base.

REFERENCES

[I] Abrial, J.R. "Data Semantics", in Data Base Management, ed. J.W. Klimbie and K.L. Koffeman, North Holland, 1974.

[2] Codd, E.F. "A Relational Model of Data for Large shared Data Banks", CACM 13:6 (1970) 377

[3] Lindgreen, P. "Basic Operations on Information as a Basis for Data Base Design", in Information Processing, North Holland, 1974.

[4] Mc Dermott, D.V. "Very Large Planner-Type Data Base", MIT - AI Memo 306, 1974.

[5] Date, C.J. " An introduction to Data Base Systems" Addison-Wesl y, 1977.

[6] Computer Issue of March 1979.

SACON: A KNOWLEDGEBASED CONSULTANT FOR STRUCTURAL ANALYSIS

James S. Bennett and Robert S. Engelmore
Heuristic Programming Project, Computer Science Department
Stanford University, Stanford, CA 94305

This paper presents an application of artificial intelligence methods to the engineering domain of structural analysis We have developed and partially implemented an "automated consultant" called SACON (Structural Analysis CONsultant), using the EMYCIN system as its framework. SACON advises engineers in the use of a targe, general-purpose structural analysis program. The structure of the knowledge base, including the major concepts used and inferences drawn by the consultant, is presented. We conclude by making some observations in light of this application about the EMYCIN system as a representational vehicle.

I Introduction

We describe an application of artificial intelligence methods to structural analysis, in particular, the development and (partial) implementation of an "automated consultant" to advise engineers in the use of a general-purpose structural analysis program*. The analysis program numerically simulates the behavior of a physical structure subjected to various mechanical loading conditions The automated consultant, called SACON (Structural Analysis CONsultant), was constructed using the EMYCIN system, the domain-independent version of the MYCIN program [I] Originally developed to advise physicians in the diagnosis and treatment of Infectious diseases, the domain-specific medical knowledge in MYCIN is represented as production rules, and is kept independent of the "inference engine" that uses the rules. By substituting structural engineering knowledge for the medical knowledge, the program was converted easily horn the domain of infectious diseases to the domain of structural analysis

The purpose of a SACON consultation is to provide advice to a structural engineer regarding the use of a structural analysis program called MARC [21 The MARC program uses finite-element analysis techniques to simulate the mechanical behavior of objects. The engineer typically knows *what* he wants the MARC program to do--e.g., examine the behavior of a specific structure under expected loading conditions—but does not know *how* the simulation program should be set up to do it. The MARC program offers a large (and, to the novice, bewildering) choice of analysis methods, material properties, and geometries that may be used to model the structure of interest From these options the user must learn to select an appropriate subset that will simulate the correct physical behavior, preserve the desired accuracy, and minimize the (typically large) computational cost. A year of experience with the program is the typical time required to learn how to use all of MARC's options proficiently. The goal of the automated consultant is to bridge the "What-to-How" gap, by recommending an analysis strategy. This advice can then be used to direct the MARC user in the choice of specific input data—e.g., numerical methods and material properties Typical structures that can be analyzed by both SACON and MARC include aircraft wings, reactor pressure vessels, rocket motor casings, bridges, and buildings.

This research was supported by the Defense Advanced Research Projects Agency (ARPA Order No 2494 Contract No DAHCI5-73-C-0435) and the Air Force Flight Dynamics Laboratory. RSE's present address: DARPA.

In this report we describe the general structure of the structural analysis knowledge base, Including the major concepts used and the inferences drawn by the consultant. We conclude by making some observations, in light of the SACON knowledge base, about the EMYCIN system as a representational vehicle and about the process of acquiring knowledge for rule-based systems. Further details of this application can be found in [3].

2 The SACON Knowledge Base

The objective of a SACON consultation is to identify an analysis strategy for a particular structural analysis problem. The engineer can then implement this strategy, using the MARC program, to simulate the behavior of the structure. This section introduces the mathematical and physical concepts used by the consultant when characterizing the structure and recommending an analysis strategy.

An analysis strategy consists of a number of analysis classes and their associated analysis recommendations Analysis classes characterize the complexity of modelling the structure and the ability to analyze the material behaviors of the structure. Currently, 38 analysis classes are considered; among them, *Nonlinear geometry crack growth, Bifurcation, Material instability, Inelastic stiffness degradation, Linear analysis,* and *No analysis.* The analysis recommendations advise the engineer on specific features of the MARC program that should be activated when performing the actual structural analysis The example consultation concludes with 9 such recommendations (see below)

To determine the appropriate analysis strategy, SACON infers the critical material stress and deflection behaviors of a structure under a number of loading conditions. Among the material stress behaviors inferred by SACON are *Yielding collapse, Cracking potential, Fatigue,* and *Material instabilities,* material deflection behaviors are *Excessive deflection, Flexibility changes, Incremental strain failure, Buckling,* and *Load path bifurcation.*

Using SACON, the engineer decomposes the structure into one or more substructures and provides the system the data describing the materials, the general geometries, and the boundary conditions for each of these substructures A substructure is a geometrically contiguous region of the structure composed of a single material such as high-strength aluminum or structural steel and having a specified set of kinematic boundary conditions A structure may be subdivided by the structural engineer in a number of different ways; the decomposition is chosen which best reveals the worst-case material behaviors of the structure.

For each substructure, SACON estimates a numeric total loading from one or more loadings. Each loading applied to a substructure represents one of the typical mechanical forces on the substructure during its working life. These might, for example, include loadings experienced during various maneuvers such as braking, banking, etc., for planes or, for buildings, loadings caused by natural phenomena such as earthquakes or wind-storms Each loading is In turn composed of a number of point or distributed load components.

Civen the descriptions of the component substructures and the descriptions of the loadings applied to each substructure, the consultant estimates stresses and deflections for each substructure using a number of simple algebraic equations. The behaviors of the complete structure are found by determining the sum of the peak relative stress and deflection behaviors of ail the substructures. Eased on these peak responses (essentially the worst-case behaviors exhibited by the structure), its knowledge of available analysis types, and the tolerable analysis error, SACON recommends an analysis strategy. Figure I illustrates the basic inferences drawn by SACON during a consultation.

Analysis Strategy of the Structure
t
Worst-case Stress and Deflection
Behaviors of the Structure
t
Symbolic Stress and Deflection
Behaviors of each Sub-structure
t
Composite Numeric Stress and Deflection
Estimates for each Loading
T
Numeric Stress and Deflection
Magnitudes for each Load Component

Fig. I Inference structure during the consultation.

Knowledge about the structural analysis task is represented in the form of production rules. An example rule, which provides the transition from simple numeric estimates of stress magnitudes to symbolic characterizations of st*ress* behaviors for a substructure, is illustrated below. For details on this representational scheme, see [4j.

RULE050

If: I) The material composing the sub-structure is one of: the metals, and
2) The analysis error (in percent) that is tolerable is between 5 and 30, and
3) The non-dimensional stress of the sub-structure is greater than .9, and
4) The number of cycles the loading is to be applied is between 1000 and 10000
Then: It is definite (1.0) that fatigue is one of the stress behavior phenomena in the sub-structure

The current consultant is able to select from among 36 nonlinear analysis strategies. If nonlinear analysis is not Indicated by low response estimates, the consultant recommends either linear analysis or no analysis. The knowledge base consists of 170 rules and about 140 consultation parameters. A typical consultation (2 substructures, 3 loadings, 3 load components) requires about 25 minutes at an interactive terminal.

To bring the SACON program to its present level of performance, we estimate that two man-months of the experts' time were required to explicate the consultation task and formulate the knowledge base, and about the same amount of time to implement and test the rules. For the first 170 rules in the knowledge base, the average cost of formulating and implementing a rule was about 4 hours The marginal cost for a new rule is about 2 hours The following transcript illustrates SACONs interactions with an engineer and SACON's recommendation of an analysis strategy.

————STRUCTURE-28. . . . . .
1) What is the name of STRUCTURE-28?
** BOEINC 747 WING
2) What is the analysis error (in percent) that is tolerable?
** 10
3) Do you want to examine the responses of the boeing 747 wing, to evaluate its instability, or both?
** EXAMINE-RESPONSES
4) Does the boeing 747 wing have time dependent terms in its equations of equilibrium?
** NO
5) Is heating or cooling of interest in determining structure responses?
** NO
6) Are there any sub-structures?
** YES
————SUB-STRUCTURE-1————
7) What is the name of SUB-STRUCTURE-1?
** TOTAL W1NC
8) What is the material composing the total wing?
** HIGH-STRENGTH-ALUMINIUM
9) What is the length of the total wing?
** 1420 INCHES
10) What is the weight of the total wing?
** 46000 POUNDS
11) Is the total wing solid or thin-walled?
** THIN-WALLED
12) What is the configuration of support for the total wing?
** ONE-SIDE
13) What is the modelling dimensionality of the total wing?
** 2
...
The first loading applied to the total wing will be called:
————LOADING-I . . . . . .
16) What is the name of LOADING-I?
** FLIGHT
17) Enter the number of loading cycles to be applied:
** 20000
The first load component of LOADING-1 will be called:

—. . . . LOAD-COMPONENT-1 . . . . . .
18) What is the distribution of LOAD-COMPONENT-1?
** DISTRIBUTED
19) What is the surface to which it acts normal?
** WIDTH-LENGTH

26) Are the support conditions of the structure nonlinear?
** NO

your structure: General-inelastic

The following are specific analysis recommendations you should follow when performing the structure analysis:

Activate incremental stress - incremental *strain* analysis.
Model nonlinear stress-strain relation of the material.
Solution will be based on a mix of gradient and Newton methods.
Logic to scan peak *stress* at each step and evaluate fatigue integrity should be used.
Logic to scan stresses, smooth, and compare with allowable stresses (with appropriate safety factors) should be used.
Logic to scan deflections, calculate relative values, and compare with code limits, should be called upon.
Cumulative strain damage should be calcuated
Analysis should include two or more load cycles *(if* cyclic) with extrapolation for strain accumulation.
Shakedown extrapolation logic should be used.

3 EMYCIN aS a Knowledge Representation Vehicle

A primary goal of this research was to determine if current "knowledge-engineering" techniques could be usefully applied *in* the development of a computer-based consultant in structural analysis. Specifically, our research was a test of the applicability of the rule-based formalism of the EMYCIN system. As such, we neither explored the *use* of other available consultation systems (e.g., PROSPECTOR, RITA) nor examined with the expert the pros and cons of using the different representation schemes that they provide.

At no time did we find the representation formalism of EMYCIN to be a hindrance to either the formulation of the knowledge by the expert or its eventual implementation in the SACON program In fact, the simplicity of using and explaining both EMYCIN's rule-based formalism and its backward-chaining control structure actually facilitated the rapid development of the knowledge base during the early stages of the consultant's design Moreover, the control structure, like the rule-based formalism, seemed to impose a salutary discipline on the expert as he formulated the knowledge base

The development of SACON represents a major test of the domain-independence of the EMYCIN system. Previous applications using EMYCIN have been primarily medical with the consultations focusing on the diagnosis and prescription of therapy for a patient. Structural analysis, with its emphasis on structures and loadings, allowed us to detect the small number of places where this medical bias had unduly influenced the system design, notably text strings used for prompting and giving advice.

Our expert found that his knowledge was easily cast into the rule-based formalism and that the existing predicate functions and context-tree mechanism provided sufficient expressive power to capture the task of recommending an analysis strategy. The existing interactive facilities for performing explanation, question-answering, and consultation were found to be well developed and were used directly by our application. None of these features required any significant reprogramming and, for the most part, worked without modification.

One major feature of EMYCIN that was not used in this task was the confidence factor mechanism [4]—i.e., the ability to draw inferences with uncertain knowledge. The present consultation strategy and the associated mathematical models were designed to estimate extreme loading conditions, from which SACON concludes the appropriate analysis class. Consequently, by using a "conservative" model, the rules, though inexact, are sufficiently accurate for predicting response bounds with certainty.

Finally, we note that the development of the knowledge base was greatly facilitated when the knowledge engineering team elicited a well-specified consultation goal for the system and then an inference structure such as that depicted in Figure 1. Without this conceptual structure to give direction to the knowledge explication process, a confused and unusable web of facts issued from the expert. We speculate that the value of these organizational structures is not restricted to the production system methodology. They seem to be employed whenever human experts attempt to formalize their knowledge using some representation formalism. Indeed, when difficulties arise in building a usable knowledge base, we expect that the trouble is as likely to be because of a poor choice of inference structure than from the use of a particular representation scheme.

The inference structure *is* a form of *meta-knowledge,* i.e knowledge about the structure and use of the domain expertise. Our experience shows that this meta-knowledge should be elicited and discussed early in the knowledge acquisition process, in order to insure that a sufficient knowledge base is acquired to complete a line of reasoning, and to reduce the time and cost of system development.

Currently, the inference structure is not an explicit part of the program (or of any other expert system of which we are aware), and hence is unavailable for the purposes of explanation, tutoring, or further acquisition of the knowledge base lis critical role in building a successful knowledge engineering application, however, would suggest making such meta-knowledge an explicit part of the consultation system. Following [5], future research on the interactive acquisition of knowledge from experts should benefit from the representation and use of such domain-specific meta-knowledge.

4 Acknowledgements

We wish to acknowledge the contributions of Prof. Robert Melosh, who provided the structural analysis expertise, Dr. Lewis Creary, and Dr. Pedro Marcel William van Melle and Carli Scott provided invaluable technical advice on the care and feeding of the EMYCIN system.

[1] van Melle, W., A Domain-independent Production-rule System for Consultation Systems, these Proceedings

[2] MARC User Information Manual, Available from MARC Analysis Research Corporation, 260 Sheridan Ave., Court House Plaza, Palo Alto, CA. 94306, Revised, September 1976.

(3) Bennett, J.S., Creary, L, Engelmore, R.S., Melosh, R., SACON: A Knowledge-based Consultant for Structural Analysis, Stanford Computer Science Report CS-78-699, December 1978.

[4] Shortliffe, E.H., *Computer-based Medical Consultations: MYCIN,* New York: American Elsevier, 1976.

[5] Davis, R., Applications of Meta-Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases, Ph.D. Thesis in Computer Science, Stanford Computer Science Report CS-76-552, July 1976.

TREE ANALYSIS TECHNIQUES IN TSUMEGO

David B. Benson, Bruce R. Hilditch, J. Denbigh Starkey
Computer Science Department
Washington State University
Pullman, Washington 99164 USA

The tsumego program performs extensive node analysis at the upper tree levels instead of attempting maximum node visitation. The success rate of the program is tested against a published set of problems of known difficulty.

1. INTRODUCTION

We use a tree searching technique of performing a great deal of analysis per node in order to choose superior lines for subsequent play. This idea is thought to be especially valuable for the chosen problem area, tsumego problems. Tsumego problems are restricted-size situations in Go that determine a local, tactical result. The most interesting feature for game-playing is the peculiar shape of the game trees. The number of level 1 branches of the tree is approximately equal to the depth, and the branching factor declines fairly regularly with the search depth (n at level 1, n-1 at level 2, etc. where n is the initial number of possible plays). This implies that high-level pruning is especially effective. Doing considerable analysis per node, as shown by exercising the program on a problem set, provides enough information quickly enough to make good decisions on choosing the line of play.

2. THE GAME

2.1 Go.

The strategy and rules of Go have been concisely and accurately described by Iwamoto [4]• Some aspects of programming the game have been treated by Benson [1] [2], Reitman & Wilcox [5] [6], Ryder [7], Thorp & Walden [9] [10], Zobrist [11], and others but the fundamental strategic and tactical heuristics used in human-quality play are still essentially undiscovered.

2.2 Tsumego.

The main threat used in deterring intrusions into one's rightful territory is the threat to capture the invading stones. It is thus frequently necessary to evaluate small sections of the board to determine whether a group of stones will be safe from capture. This can be accomplished by connecting to another safe group or by gaining 'life' on one's own merit. Since the connection-to-another-group problem usually depends on global game information concerning threats, book problems usually deal with local, tactically-oriented life-or-death situations. These problems generally resolve themselves into two subclasses: first, a 'race-to- capture' where it is necessary to capture the opponent's stones before he succeeds in capturing your own stones. Second, sequences of play that attempt to form the two protected empty intersections (eyes) necessary to be safe from attack. Strategic concerns normally depend on the outcome of the tsumego battle but do not greatly Influence its tactics. Thus we have an important but independent segment of the game which can be treated separately.

3. THE PROGRAM

3.1 The program's purpose

The tsumego program is designed to be one of several service routines callable by a driver. The functions of each of these subprograms (tsumego, semeai, fuseki, chuban, yose) will be closely defined and specific. The driver program has the responsibility of developing and maintaining a coherent strategy for the various phases of the game (opening, midgame, endgame) and of calling the service routines, as necessary, for answers to the queries which it develops. For the tsumego routine, input consists of the board configuration containing the groups in question, a set of control information (who plays first, limits on computation resources, etc.) and a list of desired results i.e., the specified group shall live, die, connect out, or live in sekI. LIfe-in-ko is not consIdered to be a distinct end-point condition, Output consists of a binary result (YES implies that a desired result can be achieved or NO, it cannot) and, if YES, the answer includes a forcing play-tree leading to the result. Other statistics are gathered to allow us to quantitatively measure the suitability of heuristics for different classes of problems.

3.2 The program's method

The heart of the tsumego program is a heuris-

tic algorithm which generates only the 'good' plays. These plays of course are the heuristic's ideas of good plays and human players often take a different view.

Since constructing a 'live' group requires the formation of two simultaneously existing eyes that are both adjacent to the specified group, the program calculates all points which might become an eye. That is, it looks for unoccupied points that have all four of the adjacent points vacant or occupied by friendly stones and at least 3 of the 4 "shoulder" points vacant or occupied by friendly stones. The program then constructs a list of all pairs of these points which might form a pair of connected eyes. Note that if two possible eye points are adjacent then both points cannot simultaneously be eyes and thus would be excluded from the list of pairs of possible eyes. Construction of the list is basically derived from a transitive closure algorithm with additional code meant to discard the inviable pairs. For all pairs which make the list, the program then calculates the plays necessary to complete and connect the two eyes. After a slight, artificial increase in the frequency counts of certain threatening and capturing plays, the program orders the resulting list by the frequency of occurrence of the potential plays. The list thus represents an ordered set of plays that may produce a live group. The plays at the front of the list and therefore those that are considered first by the recursive game player are those which appear more frequently in board configurations featuring live groups. The list is constructed at each level in the game tree since the board configuration changes at each level as new plays are made.

If the program is on 'defense', its play generation task is done. If it is on 'offense', we rely on the well-known Go heuristic: 'my opponent's best play is often my own best play.' This tactic leads to occasional wildly aggressive offense and often to a rather passive defense (typical beginner habits) but it is remarkably effective for several types of tsumego problems. It is remarkably ineffective for several other types.

Another module recognizes 'unconditional life' as proposed by Benson [1]• Unconditional life is a condition where an attacker is allowed an unlimited number of consecutive plays and is still unable to destroy the unconditionally alive group(s). This concept has been extended to also recognize 'unconditional death' where the defender is allowed an unlimited number of plays but is never able to produce a live group. The seki condition is also treated but this is not a static condition in the sense of the previous two ending conditions since a seki group might be sacrificed in return for a threat elsewhere. Additionally, the seki condition (a sort of half-life, half-death) occurs much less frequently than the other two, unconditional, ending conditions.

4. TESTING AND RESULTS

The tsumego problems used in testing our program were taken randomly from a collection of such problems by Davies [3]. The reasons for choosing this set of problems was simple: they come from published material and represent problems that humans solve. They were not composed especially for computer solution; their difficulty is easily rated; and we thought them to be within the capability of a specialized heuristic. Tne problems were modified slightly to restrict their size for economic reasons but we were careful to maintain all possibilities of attack, defense and counter-attack. As soon as the next module, a semeai-player, is incorporated into the program, these modifications can be removed since the program will be self-limiting.

The following two figures illustrate in a simple way the type of Go problem we are treating and the program's resulting search tree.

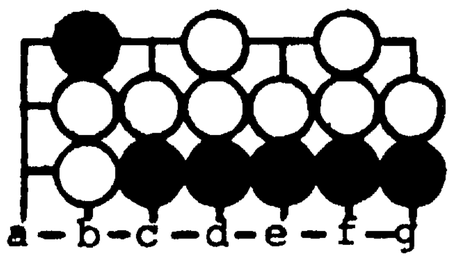


Figure 1: A simple tsumego problem

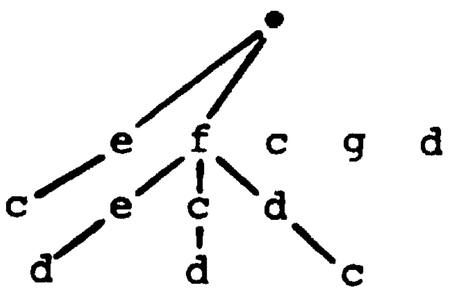


Figure 2: The corresponding computed game tree for Figure 1.

In Figure 2, note the decrease in branching factor as depth increases in the tree search for the sample problem. This could be expected since stones do not move, captures are infrequent and so each play reduces the number of possible continuations. If the pruning is heavy enough and early enough we are able to dispense with immense amounts of lookahead because of the top-heavy nature of

the lull gane tree. Table 1 gives taoular results for the complete set of problems tested.

TABLE 1

Prob. no.	To play	Result	Init. plays	Max. ply	#leaves visited
p15#1	white	yes	5	0	1
p15#1	black	yes	7	0	1
p19#2	black	yes	5	3	4
p19#4	white	no-0	13	9	11
p23#5	white	yes	14	3	7
p23#5	black	yes	11	6	8
p27#3	white	no-1	15	0	1
p27#3	black	no-1	15	0	1
p31#5	white	no-2	23	-	-
p31#5	black	no-2	21	-	-
p35#1	black	yes	13	22	263
p39#5	white	no-2	17	-	-
p39#5	black	yes	18	17	285
p43#2	white	no-1	15	0	1
p43#2	black	no-1	18	0	1
p47#2	white	yes	13	4	9
p47#2	black	no-0	14	9	38
p51#3	white	no-1	21	0	1
p55#1	black	yes	13	11	145
p58#1	white	no-2	19	-	-
p63#1	white	no-1	16	0	1
p67#4	white	no-2	22	-	-
p71#3	black	no-2	21	-	-

Notes:
0: heuristic failed... gave wrong binary result. We are counting this as a wrong answer. Both wrong answers were the result of a bug in the seki recognition routine.
1: program returned a message saying the problem was unsuitable for the eye-stealing heuristic. we are not counting this result as a wrong answer since another local expert routine will presumably eventually be available to take over.
2: program was making progress but exceeded allotted resources. This indicates that the single heuristic is not suitable for this class of problem. Additional local-expert subroutines will have to be included to extend the range of the program's capabilities.

5. CONCLUSIONS

Bushy, top-heavy game trees where the number of future plays is inversely proportional to the tree depth suggest the need for heavy, early pruning. The strategy of performing a full-width search to a pre-set depth (Slate & Atkin [8]) would in fact search a considerable portion of the game tree. In addition, the complexity and diversity of patterns in Go make the development of effective evaluation functions difficult. Chess-type strategies which rely on a slow build-up of information are inappropriate for tsumego.

As an alternative, we provide enough heuristic analysis at high tree levels to provide information necessary for effective play ordering and then let a Dranch-and-bound technique severely prune the tree. This procedure has worked well for our tsumego program (9 correct vs. 2 incorrect). There is considerable pruning as evidenced by realized branching factors (<2 at upper levels) in the tested problems.

To date the program includes an eye-forming/eye-stealing heuristic as the play generator. The heuristic is effective for solving tsumego problems of the type found in the first five chapters of Davies [3]. The program is handicapped when solving other classes of problems, particularly problems involving race-to-capture (semeai) and play-under-the-stones (ishi-no-shida). Its proficiency for tsumego problems might generously be rated as the same as a human player in the 20-25 kyu range. Any increased capability of the program will be measured objectively by attempts to solve published problems.

REFERENCES

[1] Benson,D., 'Life in the Game of Go', Information Sciences, 10:1 (1976) 17-29.
[2] Benson,D.; Evers,P.; Miller,J.; Tackett,M.; Starkey,D.; 'Computerizing the Game of Go', In Proc. Northwest 76: ACM Pacific Regional Symposium, ACM, New York, 1976.
[3] Davies, J. Life and Death, Tokyo: Isni Press, 1975.
[4] Iwamoto, K. Go for Beginners, Tokyo: Ishi Press, 1973.
[5] Reitman, W.; Wilcox, B.; Nado, R., 'Goals and Plans in a Program for Playing Go.' In Proc. of the ACM Annual Conf., ACM, New York, 1974, pps 123-127.
[6] Reitman, W.; Wilcox, B. 'Perception and Representation of Spatial Relations in a Program for Playing Go.' In Proc. of the ACM Annual Conf. ACM, New York, 1975, pps 37-41.
[7] Ryder,J. 'Heuristic Analysis of Large Trees as Generated in the Game of Go.' PhD thesis, Computer Science Dept., Stanford University, 1971.
[8] Slate, D.; Atkin, L. 'Chess 4.5: The Northwestern University Chess Program.' In Chess Skill in Man and Machine, P.Frey,ed., Springer-Verlag, New York, 1977.
[9] Thorp, E.; Walden, W. 'A Computer Assisted Study of Go on MxN Boards', Information Sciences, 4:1 (1972), 1-33.
[10] Thorp, E.; Walden, W. 'A Partial Analysis of Go', Computer Journal, 7:3 (1964), pps 203-207.
[11] Zobrist, A. 'A Model of Visual Organisation for the Game of Go', PhD thesis, Computer Science Dept., university of Wisconsin, 1973.

ON THE CONSTRUCTION OF EVALUATION FUNCTIONS FOR LARGE DOMAINS

Hans Berliner
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pa. 15213

ABSTRACT

We present the SNAC method of encoding Knowledge In a polynomial function. The most common use for such a function r. to evaluate competing alternatives in a problem solving situation. The SNAC method was discovered during research on a program that plays backgammon. It has resulted in highly consistent, skilled performance, and has made adding new knowledge very easy. This has resulted in large performance increments in the backgammon program.

We show how to create sensitive evaluation functions, and how to avoid stability problems in non-linear functions. We also demonstrate two effects, not previously found in the literature: the *suicide construction,* and the *blemish effect.*

I. Introduction

In problem solving there exists a set of states of the domain, a sub-set of these that are terminal states (corresponding to the achievement of some goal of the domain), and a set of operators that transform one state into another. In a given domain, it may be possible to apply any one of dozens of operators to a given state. Further, the current state may require hundreds of consecutive operator applications in order to arrive at a terminal state. Since such a search grows exponentially with depth, generally no path to a terminal state from a given starting state will be found within a reasonable effort. This is especially true in large domains (>10^{14} states).

When it is not reasonable to expect the solving process to search toward a terminal state, we must have recourse to knowledge to lead the way to a goal. The knowledge is in the form of "properties" or "features" that can be used to describe any state of the domain. Each such feature can take on a range of values and thus defines a dimension in a hyper-space of features. A polynomial that is the sum of various functions on these features is used to assign values to nodes and thus locate them in the hyper-space. These values are then used to order the nodes with respect to their goodness (closeness to goals of the domain). The AI literature does not contain much information about how to construct evaluation functions; only the work of Samuel[4, 5] attempts to shed light on how the construction characteristics of the function (rather than the content) bear on the performance of the program using the function.

We have been developing a backgammon program since 1974 [2]. During this time, we ran into many construction prooiems dealing with the lack of context provided by a linear polynomial (as Samuel had indicated), and problems of occasional erratic behavior when certain types of non-linearity are introduced. Recently, we have developed a method which we call SNAC (for Smoothness, Non-linearity, and Application Coefficients) which appears to have remedied all previous problems. We present details of the method below.

II. Non-linearity

Linear functions have difficulty in accurately approximating comple) behavior over a large range. Consider a simple price relationship between oranges and apples. If we assert that *an* orange is twice as valuable as an apple, we may be stating something that is correct *on average.* However, this type of advice would not be satisfactory when there is a great orange glut and a shortage of apples. Thus, a linear function seldom is *sensitive* enough over a large range; e.g. it fails to take into account the relative supply of oranges and apples, and will thus prove to be too constricting at times.

However, a linear function is very well behaved. Arithmetically combining two variables, each of which could have a range of 0:50, produces a resulting range of 0:2500 when multiplication is used. The contribution of such a term to the evaluation could vary widely, causing *stability* problems for the value of the polynomial.

Another type of problem with non-linear functions is the following. Say, we have some advice to the system in the form of S=I*D, where S is suffering, I is the intensity of pain, and D is the duration of pain. The object is to minimize the value of S (as there will be a minus sign in front of this term in the polynomial). This seems to be a well formed piece of advice. People, in general understand that the idea is no reduce both I and D. However, a program that is allowed to manipulate D, when fased with excruciating pain that is difficult to remcse, may well recommend suicide. This usually does not qualify as a solution. However, such advice can be forthcoming even when some other term of the polynomial places a large premium on staying alive. We therefore term a relation, where there exists the potential to manipulate one of the variables in a generally undesirable direction, a *suicide construction.*

Non-linearity is desirable because of the increased sensitivity it provides, while care must be taken to control volatility and avoid the suicide construction.

*This work was sponsored by the Defense Advanced Research Projects Agency (DOD), Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

III. Smoothness

Any function on the set of features in our evaluation hyper-space will define a surface. Let us consider what can result from a lack of smoothness in the surface. If there is a ridge, a discontinuity, or a sudden step in the surface, then this is a place where the values on one side of such a blemish may be quantitatively *very* different from those on the other side. Thus, a *very* small change in the value of some feature could produce a substantial change in the value of the function. When the program has the ability to manipulate such a feature, it will frequently do so to its own detriment. Simply put, the program will act to hurry across such a blemish if it leads to a favorable evaluation, and wait as long as possible to cross it, if it leads to unfavorable consequences. This behavior resembles the horizon effect [1], although caused by a very different set of circumstances. We now name this effect the *blemish effect.* Because of the blemish effect, smoothness is absolutely essential for reliable performance.

The above may seem so obvious that the reader would feel that no one could possibly overlook such a thing. However, this is far from the case. There is very little published on the structure of evaluation polynomials that *are* used in game playing programs. One exception is the work of Samuel [4, 5]. Samuel investigates how to achieve non-linearity without creating an unstable function. His solution, in both cited works is to map a variable with a large range onto a small range. In the earlier work, he creates "binary connective terms" which *are* a reduction to a binary value of a range that was large to start. Clearly, this will cause the blemish effect in the vicinity where the value of the variable changes from 0 to 1. In the later work, large ranges are reduced to ranges of from 3 to 7 in order to fit more easily into a "signature table" of limited size. Again, the blemish effect will occur near the locations where the value changes occur. We conjecture that the reason that Samuel's program did not perform better after learning non-linear functions is that the blemish effect caused it to commit serious errors occasionally. We now are able to observe the blemish effect in the performance of older versions of our backgammon program.

Whenever the coefficient of a non-smooth term is under the control of the program, the blemish effect can occur. Consider a chess, evaluation term that says "in *an* endgame the king should be centrally located, and at other times it should be located near the corners". Let us assume the king is presently in a corner, and the "endgame" is defined as being those positions where there is less than a certain amount of material on the board. The program may then avoid swaps in material in order to consider its king's position as "good" (non-endgame) even though the endgame is imminent. Here a step is created by the coefficient that defines endgame, and MMS acts to the program's detriment. A correct definition of endgame would be a smooth function from early middle game to late endgame. In this way, the degree of endgameness increases with each swap of material.

IV. Application Coefficients

We have indicated how sensitivity can be achieved, and how the blemish effect can be avoided by the use of smoothness. However, there is a major problem in avoiding the creation of terms that are very volatile. Otherwise, the program may be trying to produce some extreme value in the volatile term, because it will outweigh the combined values of all the other terms of the polynomial. This problem stems from the fact that it is very difficult to anticipate the range of values that a term m?y take on over a large domain. This is especially true when functions continue to be changed during program development. Volatility can be avoided by constraining the values that one of the variables can take on. If this were done in a construction such as S=I*D, then the volatility of the term would disappear. Yet the constrained variable would provide more sensitivity to context than a constant coefficient.

There *are* two ways of achieving this: 1) By fixing the value of the variable to be that in the original problem situation., (frozen variable), i.e. not recomputing it for each new node, and 2) By choosing variables that vary very slowly (application coefficients). We have used both methods successfully.

Using the value of a frozen variable gives the problem solving process a global outlook, where all terms using this variable are viewed as they would have been in the original situation. This has the advantage of not letting small variations create too much of an effect, and suppressing volatility for large variations. It has the disadvantage of making the process insensitive to certain kinds of changes. This method is good for functions that require some discrimination to determine the degree to which they apply, but are not required to discriminate minimal changes. This method has been used previously for efficiency reasons, when the cost of recomputing the variable at each new node is high.

The other method is to use *application coefficients.* An application coefficient is a variable that tends to vary slowly with operator applications (moves) due to the nature of the domain. Thus for a set of nodes that *are* a few operator applications apart, the value of the variable will tend to be relatively constrained. This results in a coefficient that will provide sensitivity without volatility. We give examples of typical application coefficients below.

Our typical evaluation polynomial is of the form $V=A_1F_1+A_2F_2+ \cdots A_nF_n$, where the Aj's are application coefficients or frozen variables, and the

Fi,s are functions of features of the domain. We have found that while there are usually many dozens of useful Fj*s in a domain, on the order of six or fewer application coefficients appear to emerge. These are strongly related to ideas such as game phase, ease of winning, readiness for attack, etc.

In chess, typical application coefficients are: 1) What stage the game is in, as denoted by the amount of material on the board, and the degree to which the pieces are still in their original positions; 2) The degree to which the winning side is winning, indicated by (W-V)/W+V), where W is the score of the winning side and V that of the losing side; and 3) the ease of being able to win, which is a function of the number of pawns left for the winning side *and* the ease with which they may be exchanged, and the degree to which the position is blockaded. Similar application coefficients exist in backgammon.

Such application coefficients provide a program with a great deal of context for making decisions. Thus a program that understands the stage of the game in chess, as a function of amount of material on the board, will allow its king to gradually achieve a more central position as the amount of material diminishes. Further, the *suicide construction* can be avoided by using a frozen variable in place of one that can be varied adversely. In the example quoted earlier, the duration of life of the subject becomes frozen, and that value must be used in all functions that could otherwise be subject to the suicide construction.

V. Results

We have three methods of measuring the performance of our backgammon program: 1) Performance on a problem set in an intermediate level instruction book [3], 2) Games against other backgammon programs or earlier versione of our own program, 3) Performance against human opponents.

Our previous best version before using the SNAC method was called BKG 8.0. This program was the result of about 30 man-months of effort. The present version is BKG 9.7, the result of about 8 additional man-months of work. In tests on the problem book, BKG 8.0 achieved a score of 45% based on 74 problems that it could attempt. Without any pre-testing, BKG 9.7 achieved 667 on the full set of 77 problems.

Against the best commercially available backgammon micro-processor, BKG 8.0 achieved 567 of the points, while BKG 9.5 (considerably inferior to BKG 9.7) scored 787 of the points in a set of 100 games. BKG 9.7 is now much too good to test against the micro-processor. Our current tests pit BKG 8.0 vs. BKG 9.7, with BKG 9.7 scoring 647 of the points.

BKG 8.0 played in the Carnegie-Mellon University backgammon championships in spring of 1978 and lost its first two matches thus being eliminated. In May 1979, BKG 9.7 played in a tournament of intermediate players in Portola Valley, California and won its first two matches before losing to the ultimate winner of the tourney in the 3rd round. The competition in the California tournament was somewhat better than that in the earlier tourney.

BKG 8.0 has been available on POP-10 machines for some time and has been regarded as a good game playing program, and by far the best backgammon program around. Yet, as the above results indicate, the SNAC method has resulted in a rather significant improvement *in* the program. In evaluating the above, the reader should bear in mind that in backgammon, chance plays a significant role. Professional backgammon players will tell you that a few percentage points difference in skill is all they need to have an opponent that they can win from consistently. A 607 edge is quite extreme.

The recent improvement of the program was made possible by the SNAC method that made it possible to: 1) Organize existing knowledge so that the functions are sensitive to local conditions (non-linearity) without being subject to significant volatility, 2) Avoid the blemish effect (which used to cause occasional serious errors), and 3) Add new smooth knowledge functions that contribute their part without creating opportunities for new blemish effect situations.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the influence of discussions with David Slate on his understanding of the role of smoothness in evaluation. Suggestions by Allen Newell have greatly benefitted the organization of this paper.

REFERENCES

[1] Berliner, H J., "Some Necessary Conditions for a Master Chess Program", *Proceedings 3rd International Joint Conference on Artificial Intelligence,* pp. 77-85, August 1973.

[2] Berliner, H J., "Experiences in Evaluation with BKG - A Program that Plays Backgammon", *Proceedings of the 5th International Joint Conference on Artificial Intelligence,* IJCAI Press, Computer Science Dept., Carnegie-Mellon Univ., Pittsburgh, Pa. 15213.

[3] Holland, T., "Better Backgammon", Reiss Games, Inc., New York, 1974.

[4] Samuel, A. L, "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development,* Vol. 3, No. 3, 1959, pp. 210-229.

[5] Samuel, A. L, "Some Studies in Machine Learning Using the Game of Checkers, II - Recent Progress", *IBM Journal of Research and Development,* Nov. 1967, pp. 601-617.

LEARNING STRUCTURAL DESCRIPTIONS OF GRAMMAR RULES FROM EXAMPLES

Robert C. Berwick
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

This paper describes a LISP program that can learn English syntactic rules The key idea is that the learning can be made easy, given the right initial computational structure: syntactic knowledge is separated into a fixed mterpicter and a variable set of highly constrained pattern-action grammar rules Only the grammai rules are learned, via induction from example sentences presented to the program. The interpreter is a simplified version of Marcus's parser for English [I], which parses sentences without backup. The currently implemented program acquires about 707. of a simplified core grammar of English What seems to make the induction easy is that the rule structures and their actions are highly constrained: there are only four actions, and they manipulate only very local parts of the parse tree.

1. INTRODUCTION

An important goal of modern linguistic theory is to show how learning a grammar can appear to be so easy, given the poor quality of the data children receive. This paper reports on a currently running LISP program which, by computationally embodying some theories of transformational grammar, can learn syntactic rules in the manner of Winston's blocks world program [21 The program proceeds by examining example sentences to modify its descriptions of grammar rules that make up part of its knowledge about language.

The key idea is that learning syntactic transformations is easy, given the right initial computational structure. This program uses as initial structure a simplified version of Marcus's PARSIFAL [1], a parser for English which is an interpreter for grammar rules of a particularly simple production rule form. The basic operation of the interpreter is taken as fixed, corresponding to an intial set of computational abilities. Only grammar rules are learned.

This research was conducted at the Artificial Intelligence Laboiatory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0643, and in part by NSF Cram MCS77-04828.

Because the rules themselves are so simple, and the operation of the interpreter so constrained, bugs have a diameter-limited location. Further, the parser itself is strictly deterministic; that is, already-built portions of the parse tree are assumed correct, and there is no backup. As shown below, these assumptions are crucial in the operation of the learning algorithm.

More specifically, the Marcus interpreter uses the following data structures: A parse tree, a syntactic representation of the input sentence. The lowest, right-most node of the tree under construction is called the current active node, denoted C. A buffer of three (to five) cells that holds words from the input sentence or as yet not-completely analyzed phrases. Phrase structure rules that are used to turn on and off logically grouped sets of grammar rules (for example, the rule S-->NP+VP would first activate all grammar rules that start sentences, then turn off that group and activate noun phrase rules). The phrase structure control system was designed by Shipman [31 Production rules (also called grammar rules) of the form: IF <*pattern*> THEN <*action*>. Each <*action*> does the actual work of building the parse tree, attaching words or phrases from the buffer onto the parse tree, moving new words into the buffet, and so forth. <*Pattcrns*> determine if the given action is to fire; if the pattern given in a grammar rule matches the pattern in the buffer, the specified <*action*> takes place.(Patterns use common lexical features like *Noun phrase, sinpilar* or *Verb, transitive.)*

The learning program acquires only the patterns and actions of the grammar rules. One of the accomplishments of this research has been to simplify the original Marcus parser to a point where there are only four valid actions: ATTACH first buffer item to C; SWITCH first and second buffer items; INSERT a specific lexical item into the first buffer slot; and INSERT-TRACE into first buffer slot. *{Traces* are not further discussed in this paper but function as in Chomsty's theory; see Fiengo [4] for discussion.)

Learning proceeds by induction on the <*patterns*> and <*actions*>, but with an important constraint: children (and this program) do not appear to receive negative data examples on what is not a sentence (see Brown and Hanlon [$] and Anderson [6] for discussion). On the other hand, children (and this program) do appear to receive reinforcement on what is a semantically meaningful sentence. Therefore, the current program does assume a lexicon and selectional restrictions on the phrase-structure categories (for example, that *Mary* is a noun and can fit under a noun phrase). More advanced versions of the program will probably have to assume a known case-frame representation for the input sentence given (Fillmore [7]), but this has not yet been found to be necessary; a recent result obtained by Wexler [8] proves mathematically that, given a transformational grammar to be learned and only surface sentences as input, a recursive learning procedure for the grammar does *not* exist (this was shown by Cold [9]), but that such a procedure *does* exist if surface sentences are paired with some representation of the underlying meaning of the sentence.

2. THE LEARNINC PROCEDURE

The learning program starts with an interpreter, a lexicon, simple phrase-structure rules, selectional restrictions, but no grammar rules. The program is then given input sentences to parse. If it gets stuck in a parse-- if no current rule patterns match or if all current rules cause selectional enors-- then the program attempts to build a new grammar rule that will apply at that point. It does this by trying each of its possible actions in turn: attach, switch, insert, insert-trace. (This ordering was found empirically.) The first action that succeeds in satisfying the selectional restrictions is saved along with the current machine state (buffer plus current active node) as the pattern; this becomes the new rule. If no possible generated rule has worked, the active phrase structure rule is assumed to be optional. Finally, rules with common actions within a phrase-structure group have their patterns continually generalized via intersection.

3. AN EXAMPLE: AUXILIARY-INVERSION

Suppose that at a certain point the program has all and only the grammar rules necessary to build a parse tree for Mary did hit the ball. The program now gets as input, Did Mary hit the ball? No rule currently known can fire, for in the phrase structure packet S activated at the beginning of a sentence, the only rules have the pattern [=Noun Phrase][=Verb], and the buffer holds the pattern [=Did: auxerb][=Mary: Noun]. A new rule must be written, so the program tries each of its possible rule actions in turn. Attach fails because of selectional restrictions; did can't be attached as a noun phrase. But Switch works, because when the first and second buffer positions are switched, and the buffer now looks like [=mary][=did], an existing rule for parsing declarative sentences can match. The rest of the sentence is parsed as if it were a declarative. Finally, the switch rule is saved along with the current buffer pattern as a trigger for the next case of auxiliary inversion. It is ciucial to notice that the debugging is strictly local: the error is assumed to lie exactly where the error first occurred, and not in some other rule. At most one new rule is added or one old rule modified with each example sentence, a kind of incremental debugging that is analagous to Sussman's *debugging almost right programs* [101 In this regard it is important to point out that Wexler has proved [8] that local debugging is apparently a necessary condition for the learning of a transformational grammar.

The currently implemented LISP version of this procedure has acquired about 702 of a "core-grammar" of English originally developed for the Marcus parser, as well as some new rules; acquired rules include unmarked-order, auxiliary inversion, imperative, simple there-insertion, to-infinmve, do-support, and some passives. On the other hand, rules for parsing the complicated complement structure of English have yet to be learned, nor is it clear how they might be. Future work will consider a straightforward way to learn the phrase structure rules themselves, by generalizing templates of phrase structure rules according to Chomsky's X-bar theory (Jackendoff [11]). The relationship between the local debugging constraints assumed by the learning procedure and those constraints found necessary by Wexler [8] will also be investigated.

ACKNOWLEDGEMENTS

Special thanks are of course due to Mitch Marcus, whose thesis and advice forms the foundation of this work, and to the linguistic expertise of Candy Sidner and Beth Levin.

REFERENCES

[1] Marcus, M., "A Computational Account of Some Constraints on Language," in proceedings of *Theoretical Issues in Natural Language Processing* (July 1978; TINLAP-2), pp. 236-246.

[2] Winston, P., "Learning Structural Descriptions from Examples," in P. Winston, editor, *The Psychology of Computer Vision,* New York: McCraw-Hill, 197$.

[3] Shipman, D., and Marcus, M., "Towards Minimal Data Structures for Parsing," *Proc IJCAI-79,* Tokyo, Japan, 1979.

[4] Fiengo, R., "On Trace Theory," *Linguistic Inquiry* 8:1 (1977), pp. 3S-61.

CS] Brown, R., and Hanlon, C,"Derivational Complexity and Order of Acquisition in Child Speech", in J.R. Hayes, ed., *Cognition and the Development of Language,* New York: John Wiley and Sons, 1970.

[6] Anderson, J. R., "Induction of Augmented Transition Networks", *Cognitive Science,* 1, (1977) pp.I2S-IS7.

[7] Fillmore, C.J., "The Case for Case", in Bach, E., and Harms, R.T., editors, U*niversals in Linguistic Theory,* New York: Holt, Rinehart, and Winston, 1968.

[8] Wexler, K., "Transformational Crammars are Learn able from Data of Degree Less than or Equal to Two," *Social Sciences Working Papers,* 139, School of Social Sciences, University of California, Irvine, 1977.

[9] Cold, E.M., "Language Identification in the Limit," *Information and Control,* 10 (1967) pp. 447-474.

[10] Sussman, C, *A Computational Model of Skill Acquisition,* American Elsevier, 197S.

[I I] Jackendoff, R., X-bor *Syntax: A Study of Phrase Structure,* Cambridge, Mass: MIT Press, 1977.

ON SYNTAX-DIRECTED, SEMANTICS-SUPPORTED PROGRAM SYNTHESIS

Wolfgang Bibel
Institut fur Angewandte Informatik und Formale Beschreibungsverfahren
Universitat Karlsruhe
on leave from Institut fur Informatik
Technische Universitat Munchen

A number of strategies for the synthesis of algorithms from a given input-output specification of a problem are presented which are centered around a few basic principles. It has been verified for more than ten different algorithms that their uniform application in all cases results in a successful deductive synthesis. They include a spanning-tree algorithm, a graph-circuits algorithm, a finding-the-ith-smallest-element algorithm, and a linear (string) pattern-matching algorithm. Two of them are presented here.

INTRODUCTION

This paper is based on the view that (human or mechanical) programming is a deductive process which starts with a more or less detailed specification of the given problem in some representation language and after some search guided by several strategies eventually ends up with a deduction of the program which consists in the application of rules taken from logic, the underlying theory and from a knowledge base for programming skill. It also adopts the view that predicate logic (in a wider sense) sofar still is the most suitable representation language for studying this process, because of its conciseness, naturalness, flexibility, extendibillty - both descriptive problem specification and machine programs can be represented in logic - and in particular because of the well-understood deductive mechanisms for it.

Specifically, the topic of this paper is the search for such a deduction from a logical description of a problem to some equivalent *functional* form (be it in logic or in an Algol-like language). It is no secret that the search-tree for such a deduction is tremendous even for rather simple problems. Yet, it is a common observation that people are able to move in a rather direct way towards a solution even for problems that require knowledge not experienced before. There must be general guidelines or strategies, then.

Unfortunately, (no, fortunately!) our human thinking is not organized such that these strategies could be easily bootstrapped from our behaviour. Therefore the only way to detect general strategies seems to be by a detailed comparative study of different deductions of different problems in order to detect common mechanisms. This has been done by the author for the usual maximum problem and for Hoare's FIND problem resulting in a few principles in the form of strategies [6,9]. The surprising experience *after* this study was that these strategies proved successful for a number of different other problems. They include such interesting algorithms like a linear (string) pattern-matching algorithm [16], a spanning tree algorithm [1], and a graph-circuits algorithm [20].

The strategies center around four basic ideas how to deal with a predicate logic formula describing a given problem: an J-quantifier together with a non-functional specification requires some kind of guessing (see GUESS); two (or more) consecutive 3-quantifiers require the evaluation of the dependency of the respective variables inherent in the specification (see GET-RNV,-SOC,DEPEND); for both requirements the cardinality of certain problem inherent domains and other semantic knowledge provide helpful strategic information (see DOMAIN and CHVAR); the rewriting process can be strongly goal-oriented with several intermediate goals (see GET-G). This uniform and concentrated strategic concept seems to distinguish this approach from other work done in this field like [11-15,19, etc.]. Specifically, no other approach offers both, a uniform scheme based on a few basic principles, and a guidance of the synthesis to such an extent that there is some reason for the

claim that the automatic synthesis of programs like those in this paper seems to become feasible. Of course, the second claim could only be fully verified by a running system (which at present does not exist). But the text has been formulated with such a system in mind and several details have been included in order to support that claim.

In the following the basic strategies are introduced and explained by application to the usual maximum problem. Exactly the same sequence of strategies mechanically produces the deduction of a linear pattern matching algorithm which for reasons of space is presented in a very packed form (see [7] for more details).

BASIC STRATEGIES

Logically, a programming problem has the form $\forall i \exists o(IC(i) \rightarrow OC(i,o))$,i,o,IC,OC abbreviating input, output, input-condition, output-condition, resp. For example, the usual maximum problem in this format reads

1 . $\forall S \exists m(S \neq \emptyset \rightarrow m \in S \wedge S \leq m)$, I y VS 3m MAX(S) - m,

where $S \leq m :\leftrightarrow \forall x(x \in S \rightarrow x \leq m)$.

Neither :m € S nor S ś m (nor both) provide a way how to determine m for given S, not even partially. Therefore, in view of an algorithmic solution some trial-and-error method cannot be avoided. In this simple example we only can try for a hopefully correct m. Apparently it is wise to restrict such attempts somehow, specifically by requiring a subset of the conditions listed in the output-condition. The subset has to be proper since otherwise no reduction of the problem would be possible. Formally, this idea in the case of our example reads

1.2. $\forall S \forall m' \exists m(m' \in S \cdot (S \neq \phi \rightarrow m \in S \wedge S \leq m \wedge (m' \neq m \veebar m' = m)))$, or

1.3. $\forall S \forall m' \exists m(S \leq m' \rightarrow (S \neq \phi \rightarrow m \in S \wedge S \leq m \wedge (m' \neq m \veebar m' = m)))$

The underlying general strategy which is of a purely syntactic nature is the following (with v denoting exclusive disjunction).

GUESS: Transform a problem of the form
Vi3y(IC(i) -*OC(i,y)) to
ViVy" 3y(ds+ (IC(i) +OC(i,y) A (y^y'vy^y[1])))
where domain-specification (ds) is a conjunction of a proper subset of the conjuncts in OC(i,y')

Without any additional support the synthesis would now have to consider in our example two (in general as many as there are proper subsets of clauses in OC(i,y')) different cases, (1.2) and (1.3) possibly leading to different algorithms. In order to reduce the search space, a semantical argument in favour of one of those cases would be desirable. In our example such an argument can easily be seen, favouring (1.2) rather than (1.3) for the case of usual assumptions on the domain of S and its representation. Namely, if we keep in mind that m' has to be chosen to satisfy the domain-specification then apparently this choice can be performed immediately for m' € S while it would require a number of steps for Sim'. Moreover, if S is finite then in the first case the guess also is more likely to be correct than in the second one since $|S| < |\{m' | S \acute{s} m'\}| = \infty$.. These considerations can be generalized in the following strategy which requires semantic knowledge for its application.

For a problem as described in GUESS consider the set P of subsets of those conjuncts in OC(i,y) which specify y. By conjuncting the elements of each of these subsets P becomes a set P of formulas. For each C€P let q (i):= (y|c holds} , and s (i) be the number of steps required to determine an arbitrary y such that C holds. Then for a given weight-factor k and for C,D€P let C< D iff q -s < k-q -s A q -s 4 k-q -s . With this priority-relation wmch captures the considerations illustrated above in a general form, the strategy reads as follows.

DOMAIN: For domain-specification in GUESS choose a minimal C€P w.r.t. < (which has not been considered sofar).

Apparently, all the information necessary to apply DOMAIN has to be provided to the system from outside (interactively or via a knowledge base). Of course, application of DOMAIN to (1.1) yields m'€s as domain-specification, thus excluding (1.3) in a first attempt to synthesize MAX.

In the following we will see that (1.2) implicitly already contains an algorithmic solution for MAX which only has to be unfolded by rewriting the formula in a rather constrained goal oriented way. The general strategic scheme and three of its instances are the following ones.

GET-G (goal-oriented equivalence transformation with goal G): Rewrite a given formula according to domain and goal dependent equivalence transformation rules until goal G is achieved.

GET-DNF: the r;oal is that the resulting formula is in disjunctive normal form (alternative cases).

GET-REC: the goal is to find some kind of recursion; if necessary, generalize the problem

GET-EP: the goal is that predicates become evaluable, at least recursively.

In a program we need alternative cases. Therefore GET-DNF will be applied to the conclusion

in (1.2) which yields

1.4. $m' \in S \rightarrow (S \neq \emptyset \rightarrow m \in S \wedge S \leq m \wedge m \neq m') \dot{\vee} (S \neq \emptyset \cdot m \in S \wedge S \leq m \wedge m = m')$

(often we drop the quantifiers and some parentheses if they are understood, where $\wedge$ binds more than $\dot{\vee}$, both more than $\rightarrow$, etc.)

Next we look for some kind of recursion for the first alternative (in the second one the result is already determined). The formula talks on two objects m,m' and on a set S. Imagine a recursion generator which from a knowledge base produces the different possible recursive forms of problem reduction for these kinds of objects. It is just reduction of cardinality in the present case of a set (but even in general there are only a few known possibilities). A rule generator (knowledge base and/or theorem prover) tries to merge the newly derived information ($m \neq m'$ in 1.4) by rewriting the conjunction of these clauses on the basis of the input-condition and the underlying theory (a search limited by a function in the number of given clauses even in the exhaustive mode). Here this might start with the application of the rule $m' \neq S \rightarrow (m \in S \wedge m \neq m' \leftrightarrow m \in S - m' \wedge m \neq m')$. Whenever in the result there is a match with one of the alternative recursive forms ($m \in S - m'$ in this case), this suggests a substitution possibly leading to a recursive problem reduction. It is explored for the whole problem under the guidance of a theorem pover. Because of

$m' \in S \rightarrow (S \leq m \wedge m \neq m' \leftrightarrow S \setminus m' \leq m \wedge m' \leq m \wedge m \neq m' \leftrightarrow S \setminus m' \leq m \wedge m' < m)$,

this process yields

1.5. $m' \in S \rightarrow (S \setminus m' \neq \emptyset \dot{\vee} S = \{m'\} \rightarrow m \in S \setminus m' \wedge S \setminus m' \leq m \wedge m' < m) \dot{\vee} MAX(S) = m'$

1.6. $m' \in S \rightarrow (S \setminus m' \neq \emptyset \rightarrow m \in S \setminus m' \wedge S \setminus m' \leq m \wedge m' < m) \wedge S \neq \{m'\} \dot{\vee} MAX(S) = m'$

1.7. $m' \in S \rightarrow MAX(S \setminus m') = m \wedge m' < m \wedge S \neq \{m'\} \dot{\vee} MAX(S) = m'$

completing the efforts of GET - REC in this case.

There is still the unpleasant clause $m' < m$ which cannot be executed, not even recursively. This causes GET - EP to rewrite (1.7) which gives the equivalent formula

1.8. $m' \in S \cdot MAX(S \setminus m') = m \wedge m' < MAX(S \setminus m') \wedge S \neq \{m'\} \dot{\vee} MAX(S) = m'$

All these 8 formulas are logically equivalent. The last one can easily (and automatically) be compiled into efficient algorithmic code (see [7] for some more details). Note that the deduction also serves as a correctness proof for the resulting algorithm.

A CHALLENGING PROBLEM

For the general case a few additional strategies are required. But even with these basic ones quite a few non-trivial algorithms can be synthesized by mechanical application of the basic and standard sequence GUESS, DOMAIN, GET-DNF, -REC, -EP. Linear string-pattern-matching is among these algorithms, for the development of which several outstanding researchers in computer science have been awarded with honour.

We denote by $\mathbb{N}_m^n$ the natural numbers k with $m \leq k \leq n$. A string is a function $c_S : \mathbb{N}_1^{|S|} \rightarrow$ <characters>. Further assume fct as a built-in test for functionality, dom as the function which denotes the domain of a function. Then the definition of the pattern-matching problem can be stated as:

2.1. $\forall c_S \forall c_P \exists k (fct\ c_S \wedge dom\ c_S = \mathbb{N}_1^n \wedge fct\ c_P \wedge dom\ c_P = \mathbb{N}_1^m \rightarrow k \in \mathbb{N}_0^{n-m} \wedge \forall j (j \in \mathbb{N}_1^m \rightarrow c_P(j) = c_S(j+k))$

shortly $\forall c_S \forall c_P \exists k (PM(c_S, c_P) = k)$

GUESS, DOMAIN, GET-DNF, -REC yield

2.2. $\forall c_S \forall c_P \forall k' \exists k (k' \in \mathbb{N}_0^{n-m} \rightarrow (IC \rightarrow OC \wedge k = k') \dot{\vee} (IC \rightarrow k \in \mathbb{N}_0^{n-m} \setminus k' \wedge \forall j (j \in \mathbb{N}_1^m \rightarrow c_P(j) = c_S(k+j)) \wedge \exists \bar{j} (\bar{j} \in \mathbb{N}_1^m \wedge c_P(\bar{j}) \neq c_S(k'+\bar{j}))))$

GET-EP recognizes a subproblem determined by $\bar{j}$ which has to be solved prior to the solution of the whole problem. This subproblem can be treated in isolation since there is no clause containing both output-variables k and $\bar{j}$; a solution - the well-known quadratic algorithm - is easily and mechanically obtained with the standard technique of this paper (see [8]).

The linear solution can be obtained only if this treatment in isolation is abandoned or, more precisely, if the range of the $\exists \bar{j}$-quantifier, which determines the subproblem, does include also the remaining clauses in the second alternative in (2.2) (causing a search in general which again is limited by a function in the number of clauses). In this case our subproblem, denoted by DIFF, has the following output-condition (abbreviated by $A(\mathbb{N}_1^m)$)

2.3. $k \in \mathbb{N}_0^{n-m} \setminus k' \wedge c_P(j) = c_S(k+j) \wedge \bar{j} \in \mathbb{N}_1^m \wedge c_P(\bar{j}) \neq c_S(k'+\bar{j})$

GUESS adds a $(\bar{j} \neq \bar{j}' \dot{\vee} \bar{j} = \bar{j}')$-clause where DOMAIN chooses $\bar{j}'$ from $\mathbb{N}_1^m$. The only clauses with which $\bar{j} \neq \bar{j}'$ in the first alternative can be merged are those containing $\bar{j}$ with the following obvious result.

2.4. $A(\mathbb{N}_1^m \setminus \bar{j}') \wedge c_P(\bar{j}') = c_S(k'+\bar{j}') \dot{\vee} A(\mathbb{N}_1^m) \wedge \bar{j} = \bar{j}'$

But this time the GET-REC can be more successfull if it keeps merging the added conjunct with the clauses in the original part, specifically with $c_P(j) = c_S(j+k)$; since in the first alternative for $c_P(j) \neq c_P(\bar{j}')$ we have:

$c_S(j+k) = c_P(j) \neq c_P(\bar{j}') = c_S(k'+\bar{j}') = c_P(j+k'+\bar{j}'-j)$,

i.e. $k \neq k'+\bar{j}'-j$ in this case for any appropriate $k', \bar{j}', j$. Similarly, in the second alternative for $c_P(j) = c_P(\bar{j}')$ and $j \neq \bar{j}'$ we have:

$c_S(j'+k') \neq c_P(\bar{j}') = c_P(j) = c_S(k+j) = c_S(\bar{j}'+k+j-\bar{j}')$

i.e. $k \neq k'+\bar{j}'-j$ also in this case for any appropriate $k', \bar{j}', j$. We obtain the following recur-

sive solution for DIFF in which PK, $P\bar{J}$ denote the sets of possible k- and j-values.

2.5. $D_1 := (k' + \bar{j}' - j \in PK \mid k' \in PK \wedge \bar{j}' \in P\bar{J} \wedge j \in \mathbb{N}_1^m \wedge c_p(j) \neq c_p(\bar{j}')) \wedge$
$D_2 := (k' + \bar{j}' - j \in PK \mid k' \in PK \wedge \bar{j}' \in P\bar{J} \wedge j \in \mathbb{N}_1^m \wedge c_p(j) = c_p(\bar{j}') \wedge j \neq \bar{j}') \wedge$
$\bar{j}' \in P\bar{J} \rightarrow DIFF(P\bar{J}, PK) = DIFF(P\bar{J} \setminus \bar{j}', PK \setminus D_1) \wedge c_p(\bar{j}') = c_s(k' + \bar{j}') \vee$
$DIFF(P\bar{J}, PK) = PK \setminus D_2$

which has an obvious algorithmic interpretation. With that solution (2.2) now reads

2.6. $k' \in PK \rightarrow (IC \rightarrow OC \wedge k = k') \vee (IC \rightarrow OC \wedge k \in DIFF(\mathbb{N}_1^m, PK) \setminus k')$

Comparison with (2.1) gives

2.7. $k' \in PK \rightarrow PM(c_s, c_p, PK) = k' \vee$
$PM(c_s, c_p, PK) = PM(c_s, c_p, DIFF(\mathbb{N}_1^m, PK) \setminus k')$

In algorithmic (loop) form this reads:

2.8.
```
proc pm(c_s, c_p)=k; n ← |dom c_s|; m ← |dom c_p|; PK ← IN_0^(n-m);
    loop |PK|; choose k' such that k'∈PK; PK ← diff(IN_1^m, PK);
    if PK=false then return k'; PK ← PK\k';
    if PK=∅ then return false;
    end;
```

FURTHER STRATEGIES AND ALGORITHMS

For more general cases the following additional strategies are necessary: In order to structure complex problems output-conditions have to be separated into (partly) independent parts (GET-SOC); if possible, output-variables have to be expressed by others in the presence of more than one 3-quantifier (GET-RNV), otherwise, a hierarchical order between the different $\exists$'s has to be determined (CHVAR).

These and the basic strategies applied mechanically (hand-simulation) in the standard sequence yielded a detailed synthesis - as that for the maximum and the pattern-matching algorithm - of the following algorithms: maximum algorithms in different semantic context; binary search in lists; searching for duplicates in lists; testing equality of two lists; partitioning a set w.r.t. a given element; finding the i-th smallest element in a set; determining circuits in a graphs-determining a minimum-cost spanning tree in a graph. Others have been synthesized in detail such as determining the moves of a roboter, etc. This remarkable list suggests that there is something fundamental in our strategies. For further details and all references see

[7] Bibel, W., Syntax-directed, Semantics-supported Program Synthesis, Proceedings 4th Workshop on Automated Decution (W.H.Joyner,jr., ed.,IBM, Yorktown Heights, N.Y.), Austin, Tex., 140-147 (1979).
Improved version available as Report no. 78, University of Karlsruhe (submitted to Artificial Intelligence).

This work has been partially supported by the Deutsche Forschungsgemeinschaft (DFG).
Typescript: G. Weiher

A SYSTEM WHICH SYNTHESIZES ARRAY-MANIPULATING PROGRAMS FROM SPECIFICATIONS

M. BIDOIT, C. GRESSE, G. GUIHO
Laboratoire de Recherche en Informatique
Batiraent n° 490
Universite de Paris-Sud
91405 ORSAY (France)

A 3>stem is presented for constructing array-manipulating programs from given specifications. The system accepts high level specifications and produces recursive programs in an Algol-like language.
Restriction to a specific domain and use of powerful techniques such as first-order matching or generalization make our system very efficient. It has been implemented in LISP on a small computer and has been tested for a large number of examples.
The general techniques we use, are briefly discussed and the synthesis of one program produced is presented.

Key-words Array-manipulating programs - High-level specifications - First order matching - Generalization - Program synthesis.

1. INTRODUCTION

Although program synthesis is a recent development, there are already a great number of approaches (1-7).
Our approach is similar to that of Manna and Waldinger [5] but we restrict ourselves to the synthesis of array-manipulating programs. We think that this restriction will limit backtracking.
The principle of this kind of synthesis is :
We provide the specifications of the problem to be solved in a high-level language (not compilable in a conventional way). The system works as a transformation program in order to obtain a final program written in a compilable language. For this purpose, the system uses heuristics and rewriting rules to eliminate non-compilable terms which occur in the specification language. The synthesis is then achieved by use of matching and generalization. This system has been implemented on a small computer (MITRA 125) in LISP and has been tested for a large number of examples.

2. LANGUAGES

2.1 The specification language

The language level should be high enough to represent the abstract types close to the concepts used by the programmer in his description of the problem. In our restriction to the array type, the constructors and selectors are simple and few. Here are some constructions we use as specification language.
Note that for all the examples the system knows that T is an input array with bounds A and B and that I is a subscript. So input specifications like in the DEDALUS system of Manna and Waldinger are not necessary.

a. Find I between A and B such that $\mathcal{P}$ (I,input variables)

Examplea :
- *Find T between A and B such that T(I)=Z*
- *Find I between A and B such that T(I)≥All T[A.B*

b. Test if $\mathcal{P}$ (input variables)

Examples :
- *Test if Z<All T[A.Bi*
- *Test if for all T between A and B-I Til) < T(T+1)*

c. Construct TAB[BI . .B2] such that TAB=constructor (I, input variables and $\mathcal{P}$ (I, input variables) from T[A. .B]

Example :
- *INSERT (T,A,B,Z).* **We want to insert an integer Z in a sorted array**
- *construct TAB[B1..B2] such that TAB=CONC(T[A..I],Z,T[I+1..B]) and AllT[A..I]≤Z and All T[I+1..B]>Z from T[A..B]*

d. Compute the number of elements T[1]such that $\mathcal{P}$ (I,input variables)

Examp l e :

- *the number of elements T[1] such that* $T[I] \geq Z$.

e. Two of the upper constructions can be combined to make a more complex one

Example :

- *Find I between A and B such that (the number of elements* $T[J]$ *such that* $T[J] \geq T[I]$ *)=K.*

Although we use few constructions, the possibility of combining them makes our program more flexible in the sense that we can use different specifications for solving the same problem.

Example : Finding the maximum element of an array can be specified both by :

- *Find I between A and B such that*
  T[I] .All T[A..B]
- *Find I between A and B such that (the number of T[J) such that T[j]>T[l])=0.*

These different specifications lead to different programs

f. etc...

2.2 The output language

The output language is an Algol-like language and we extensively use recursion and basic operators relevant to array structure.

For the INSERT example, the output will be of the form :

INSERT (T,A, B,Z)=INSERTGEN (T,A,B,Z,A)
INSERTGEN *(T,A,B,Z,Cy*if* OB : TAB<-CONC(T[A..B] ,Z)
ifnotif(T(C)<Z) : TAB<-K:ONC{T[A. .C-I] ,Z,T[C..B])
else INSERTGEN (T,A,B,Z,C+1)
endif

3. REWRITING RULES AND HEURISTICS

The are two kinds of heuristics. Some of then are related to abstract types which are used and some others are related to general programming principles.

3.1 The first class of heuristics deals with the constructors and selectors of abstract types [12], Informally, the heuristics use selectors to achieve a decomposition of the problem and then constructors to solve the original problem. In our implementation, the only type we use is the array type so this family of heuristics is restricted. It corresponds to the case formation rule (see Manna [5] or Weigbreit [8]) :

- If the array is empty, then no operation is necessary
- If not, we try to deal with one particular element of the array (generally the first one) and then look at the rest of the array.

3.2 The second class of heuristics deals with general knowledge on programming. The basic principles are similar to those used in Weigbreit [8].

- Formation of recursive calls.

During the synthesis, the first class of heuristics defines subgoals of the initial problem. If by matching, we notice that a subgoal is an instance of a previous one or an instance of the initial goal, we infer a recursive call. It is then necessary to verify a well-founded set condition.

- Generalization : (Insane heuristic) (Wegbreit [8] , Kodratoff (4)). In the previous process, when matching fails ; it is possible in most cases to generalize the goal used. With this generalization, matching becomes effective and a recursive call can be generated.
- Subsidiary programs : (Manna and Waldinger [5])

In the case formation rule, the inferred conditions can be non-elementary (i.e. it contains some specification language terms). Then a subproblem has to be introduced at this point.

4. TECHNICALS TOOLS

4.1 First order matching

This matching is mainly used for the detection of recursive calls. The algorithm implemented here is from Huet [9]. Note that recursive calls are always terminal.

4.2 Second order matching

A second order matching in necessary, in principle, in order to select the transformation rules to be applied. In fact, in our present implementation we don't nead a 'sophisticated" algorithm and we use a simple key-word detection algorithm similar to pattern-directed invocation.

4.3 Theorem prover

To solve the subgoals induced by the case formation rule it is often necessary to use a theorem prover.

It is surprising that the necessary proofs are often very simple. So we have implemented a very "naive" theorem prover which has been sufficient for all our examples.

5. ONE EXAMPLE OF SYNTHESIS : INSERT

The input array T[A..B] is sorted and we want to insert an integer Z in T such that the output array TAB(B1..B2] remains sorted.

We provide the following input to our system

-INSERT (T,A,B,Z)
-CONSTRUCT TAB[B1..B2] such that
TAB=CONC (T[A..I] ,Z,T[I+I..B)) and All(T[A..I] < Z and All T [I+I..B]>Z fromT[A..B]
-For All I between A and B-I T(I) < T(I+1)

* The first rule, when applied, transforms the problem into a first goal :

(I) (Find I between A and B such that
All T[A..I]<Z and All TII+1..B]>Z
then (construct TAB[BI..B2I such that
TAB = CONC (T[A..I],Z,T(I+1..B))

* At first the system considers only the first part of the goal and transforms in into

(Find I between A and B such that

All T[A..I]<Z) then (verify
All T{I+1..B]>Z) then (construct ...)

The system now applies the decomposition rule to the first sub-goal and generates a segment of the desired final program and three new subgoals.

if A>B then (2)
ifnotif 1[T(A)<Z] then (3)
else (4)

where (2),(3),(4) are the following subgoals.

(2) I<-B then. .. .unchanged . . . .
(3) I<-A-I then....unchanged ....
(4) (Find 1 between A+I and B such that A11T[A+1 ..I]<-Z) then (Verify AIIT[I+I..d>Z) then (Construct TAB[BI..B21 such that TAI,=CONC(T[A..I] ,Z,T[I I..B]))

The system considers the goal (4) : Matching (4) with (1) fails because A can't be matched simultaneously with A+I(Find...) and A(Construct...) So the system introduces a new generalized problem : INSERTGEN(T,A,B,Z,C) with :

INSERT(T,A,B,Z)<=TNSERTGEN(T,A,B,Z,A)

The input for INSERTGEN is :

- (Find I between C and B such that AI IT[C. .B] <Z) then (Verify A11T [I+I..B]>Z) then (Construct TAB[B1..B2] such that TAB=CONC(T [A..I3,Z,T[I+I..EJ))
- For All I between A and B-I T(I)<T(I+1)

* Applying the decomposition rule leads to

if C>B then (2) *ifnotifl(T(C)<Z)* then (3)*else* (4)

(2) I<-B then. ., .unchanged. .. .
(3) I<-C-1 then ....unchanged....
(4) (Find I between C+1 and B such that..)then . . .

* Now matching succeeds because A is matched with A and C with C+I. So (4) is achieved by a recursive call to INSERTGEN(T,A,B,Z,C+1)
* (2) and (3) are rewritten as

(2') (Verify AIIT[B+1..B]>Z) then (Construct TAB[BI..B2I such that TAB=CONC(T[A. .B],Z,TiB+ I.B]))
(3') (Verify A11T[C..B]>Z) then (Construct TAB[B1..B2] such that TAB=CONC(T[A. .C-I] ,Z,TtCB])

Verifications are very easy since in
(2') T[B+1..B]= and any property holds for every element of an empty array
(3') the theorem prover uses the two conditions :
T(C)>Z (introduced by the case formation rule)
T[A..B] is sorted (input specification)

* So (2') and (3') are rewritten as

(2") TAB=CONC(T[A..BI ,Z)
(3") TAB=C0NC(T[A..C-I| ,Z,T[C..B1)

INSERT(T,A,B,Z)«-INSERTGEN(T,A,B,Z,A)
INSERTGEN (T, A, B,Z,C)«*t.f C>B:TAB<-CONC(T [A. .B] ,Z)
ifnotifl{T(C)<;Z) :TAB+C0NC(T(A..C-1] ,Z,T[C..B])
else INSERTGEN(T,A,B,Z,C+1) *endif*

6. CONCLUSION

This system has been implemented on a small computer and works efficiently for all examples listed below (and some others). The maximum time necessary for one example is less than one minute. There are three main reasons for efficiency:

- the system has a thorough knowledge of the domain and so backtracking is strongly reduced
- we use a very powerful first order matching algorithm
- the simplicity of the theorem-prover makes it very efficient.

This system, in our opinion, proves that the Manna and Waldinger approach is correct and can be very powerful in specific domains. This shows that studies in the field of automatic program synthesis can be relevant for practical applications .

Programs constructed :

Representative temples of the programs constructed by our syslea are the following :

• finding the maximum element of an array
- testing if an array is sorted
- testing if a number belongs to an array and finding its subscript
- tasting if a nuaber ie less than every element of an array
- tasting if an array contains duplicates
- testing if every element of an array is less than ovary element of another
- inaerting an element in an array (INSERT)
• reversing an array
- finding an element equal to its subscript
- finding the k^{th} (in increasing order) element of an array
- computing the number of elements greater than a particular one
- tasting if two arrays are identical
- computing the intersection of two arrays
- *find I betwn A and B euoh that AIIT\I. .B}iZ and T(I-1)<Z*
- *find I between A and B suoh that AIIT\A. .fltAIIT'II. .C\ (vh*r* T[A..B) and T'[A..C* are *input array9).*

asjrajusm.B*

(I) Biermaaa, Krishnasvany "Constructing programs from example computation IEEE Trana. on Software Eng., vol. 2. n' 3, Sept. 1976.

(21 Summers P.D. "A methodology for LISP program construction from examples". Journal of ACM 24-1, 1977.

[3] Cuiho C, Jouannaud J.P. "Inference of functions with an interactive system". Machine Intelligence 9. 1977.

[4] Kodratoff Y. "Choi* d'un programme LISP correspondent a un example". Congres AFCET, Reconnaissance des Formes, 1978.

[5] Manna I., Waldinger R. "Synthesis : Dreams-Programs", lechnkal notes 156, SRI. 1977.

(6) libel V., Purbach U., Schreiber J.P. "Strategies for the synthesis of algorithms". AISB/CI Conference on Artificial Intelligence, Hambourg, 1978.

171 Waldinger R. "Constructing programs automatically using theorem proving". Pd.D Thesis. Carnegie Mellon, 1969.

(81 Wegbreit 8. "Coal Directed program trana format ions". EEE Trana on Software Eng. Vol. 2, n' 2, 1976.

(91 Huat C. "L'unification dans les langages I,2,...w". Thcae doDuctrat d'Etat Univarsit* Paris VII, 1976.

(I0| Creussay P. "Contribution a la definition interprlr-'.ive eta l'mnlojouation des lambda langages". These de Doctoral d'Etat, Univ. Paris VII. 1977.

[11] Crease C. at Cuiho C. "Etude comparative des programmes de synthese Je programmes". Congree AFCET Informalique, 1978.

(I2| Cuttag J., Horowits E., Mnsser D.R. 'The design of data types mx'cili.it Research Report. Information Sciences Institute, Marina del Roy, Lalifcmi

A PROVER FOR GENERAL INEQUALITIES

W. W. Bledsoe
Peter Bruell
Department of Mathematics
The University of Texas at Austin
Austin, Texas 78712

Robert Shostak
SRI International
333 Ravenswood Avenue
Menlo Park, California 92025

A variation of an earlier prover (described in Machine Intelligence 8) is used to prove theorems about general inequalities, i.e., first-order logic with equality where the only predicate symbols are <, ≤, and =, and where function symbols are admitted. Transcripts of some proofs are given.

1. INTRODUCTION

This paper, which is an abridgement of [1], describes a program that has been used to prove a number of first-order theorems involving inequalities (and equalities). It is a variation of an earlier natural deduction prover [2], The class of formulas dealt with permits arbitrary quantification and uninterpreted function symbols and can therefore encode predicate calculus. Our intention, however, was not to provide a general-purpose prover for first-order logic but rather to be able to prove naturally arising formulas involving inequalities. We have made use, where possible, of previously developed techniques. Among these are decision procedures for ground Presburger formulas [3-8] and the restriction variable methods of [9]. We emphasize that this prover is completely automatic and that all examples were proved without recourse to human interaction. So far as we are aware, this prover is the first one able to prove formulas of the kind typified by the examples in an automatic way.

2. THE PRINCIPAL PARTS

Because of space limitations we are forced to refer the interested reader to [1] for the descriptions of the algorithms that the program employs. However, we must mention that the heart of the program is the routine IMPLY, which coordinates the search for the proof of a theorem. The guiding principle of this search is the retention of the implication symbol as the main connective of the theorem being proved.

Without going into detail we also note that when the conclusion of a theorem is an inequality, IMPLY causes a function named PROVE-LE to be called. This function in turn calls several subfunctions:

1. LESS-
   A quick routine designed to solve such trivial subgoals as 4 < 5, 6 < «, etc.

2. PROVE-LE-GROUND-CASE
   Called if the conclusion is a ground inequality in an attempt to establish whether this inequality follows from ground inequalities in the hypothesis (stored in a special data structure called the TYPELIST) .

3. RESTRICTION-LE
   Called if the conclusion is a non-ground in equality which can be "solved" by simply placing restrictions on variables which occur in it

4. MATCH-IN-TYPELIST
   Called when the conclusion is a non-ground inequality (which cannot be "solved" by RESTRICTION-LE) in an attempt to "solve" it by finding a match with a ground inequality stored on the TYPELIST.

5. MATCH-LE
   Called if the conclusion is an inequality and none of the above techniques succeeds (or applies) in an attempt to use the hypotheses of the theorem other than those on the TYPELIST.

In the section of examples we use the following abbreviations for these functions:

GLE for PROVE-LE-GROUND-CASE

RLE for RESTRICTION-LE
MTY for MATCH-IN-TYPELIST
MLE for MATCH-LE

Data Base

A data base, DB, is employed which is "dynamic" or "contextual," in that it may change as the proof progresses. It has three parts: A-UNIT, RESTRICTION-LIST (or RL), and TYPELIST (or TY). The A-UNIT is not used here (see [9]).

RESTRICTION-LIST (or RL) is the place where restrictions on variables are recorded. If, for example, an entry

(1) ('int' x (< a) ($\leq$ b))

is present in RL, it means that the variable x (which is to be instantiated, or bound to a value) has not yet been given a definite value but has been restricted to reside in the interval $a < x \leq b$. If later x is bound to a particular value c, then c must satisfy this same constraint, $a < c \leq b$. Other restrictions like $a \leq x \leq b$, $a \leq x < b$, etc. are possible.

TYPELIST (or TY) is the place where ground inequality information is stored. If an entry

(2) ('int' x ($\leq$ a) ($\leq$ b))

is present in TY it means that one of our hypotheses is $a \leq x \leq b$. Notice the fundamental difference between this and (1) of RL. In RL, x is a variable to be instantiated, whereas in (2) it is a constant. RL represents knowledge about the "solution" to their variables (i.e., the bindings for the variables), whereas TY represents given knowledge or hypothesis knowledge that can be used to obtain the solution. Getting a contradiction in TY is desirable in that it completes the proof of the current subgoal, whereas a contradiction in RL is undesirable in that it indicates a failure to find an acceptable solution (binding) for x. An entry like (1) represents a whole interval of acceptable solutions for x (provided that it can be proved that a < b).

Other concepts such as REDUCTIONS, controlled backtracking, and algebraic simplification are used to advantage.

3. EXAMPLES

In the following examples we will depict a call to IMPLY as follows:

(TL) ([TY] $\wedge$ H $\Rightarrow$ C) {RL}

Here the theorem label, TL, a map for keeping track of subgoals, is in the left margin. The TYPELIST, TY, being part of the hypothesis, is shown with H to the left of the arrow implying C. The restriction list, RL, is shown to the right. In the descriptions of proofs of the examples we will often leave out some of the steps when clarity is not impaired. Also, the components TL, TY, H, C, and RL will not all be given at each step.

The theorem label, TL, starts as an empty list ( ) and grows as the depth of the proof increases. It consists of a sequence of symbols representing the actions that have been taken. Some of the symbols used in TL are

1	first branch of an and-split
2	second branch of an and-split
P→	promote
CHECK	check that a proposed binding is consistent with RL

In the following examples all symbols a, b, c, f, g, Z, etc. that are not explicitly quantified will be treated as skolem constants.

Ex. 1. $(f(\ell) < 0 \wedge 0 \leq f(b) \wedge b \leq \ell \rightarrow b < \ell)$

Proof.

( ) $(f(\ell) < 0 \wedge 0 \leq f(b) \wedge b \leq \ell \rightarrow b < \ell)$

(P→) $([f(\ell) < 0 \wedge 0 \leq f(b) \wedge b \leq \ell] \Rightarrow b < \ell)$

PROVE-LE is called, which calls LESS= and RLE, which fail.

PROVE-LE then calls the ground prover GLE which succeeds as follows:

The $\sim(b < \ell) \equiv (\ell \leq b)$ is inserted into TY which becomes

$$[f(\ell) < 0 \wedge 0 \leq f(b) \wedge b \leq \ell \wedge \ell \leq b]$$

CONTRADICTION is called but it finds no contradiction in TY; however, it does find and return the equality unit $(b = \ell)$, which is then applied to TY to obtain

$$[f(\ell) < 0 \wedge 0 \leq f(\ell)]$$

which has a contradiction. Q.E.D.

Ex. 2. $f(\ell) < 0 \wedge 0 \leq f(b) \wedge \ell < c \wedge b \leq \ell \rightarrow$

$$\exists y[\forall z(z \leq b \wedge f(z) \leq 0 \rightarrow z \leq y) \wedge y < \ell])$$

(P→) $([f(\ell) < 0 \wedge 0 \leq f(b) \wedge \ell < c \wedge b \leq \ell]$

$$\Rightarrow (Zy \leq b \wedge f(Zy) \leq 0 \rightarrow Zy \leq y) \wedge y < \ell)$$

Note: y is a variable, Zy is a skolem function

of y, and the other letters represent constants.

(P→ 1) $([f(\ell) < 0 \wedge 0 \leq f(b) \wedge \ell < c \wedge b \leq \ell]$

$\Rightarrow (Zy \leq b \wedge f(Zy) \leq 0 \rightarrow Zy \leq y))$

(P→1 P→) $([f(\ell) < 0 \wedge 0 \leq f(b) \wedge \ell < c \wedge b \leq \ell]$

$\wedge Zy \leq b \wedge f(Zy) \leq 0 \Rightarrow Zy \leq y)$

LESS=, RLE, GLE, and MTY fail.

MLE: $Zy \leq b$: b/y

Note: The substitution b/y has succeeded on subgoal (P-> 1) ; prover will not proceed to subgoal (P-> 2) with y replaced by b.

(P→ 2) $([f(\ell) < 0 \wedge 0 \leq f(b) \wedge \ell < c \wedge b \leq \ell]$

$\Rightarrow b < \ell)$

GLE inserts $\sim(b < \ell)$ into TY which becomes

$[f(\ell) < 0 \wedge 0 \leq f(b) \wedge \ell < c \wedge b \leq \ell \wedge \ell \leq b]$

CONTRADICTION discovers the equality $(b = \ell)$ in TY, getting

$[f(\ell) < 0 \wedge 0 \leq f(\ell) \wedge \ell < c]$

which holds a contradiction. Q.E.D.

Ex. 3. $(a \leq 2 \leq b) \rightarrow \exists x\,(0 \leq x \leq 5 \wedge a \leq x))$

( ) $(a \leq 2 \wedge 2 \leq b \rightarrow (0 \leq x \wedge x \leq 5) \wedge a \leq x)$

(P→) $([a \leq 2 \wedge 2 \leq b] \Rightarrow (0 \leq x \wedge x \leq 5) \wedge a \leq x)$

(P→ 1) ([ ] $\Rightarrow (0 \leq x \wedge x \leq 5))$

(P→ 1 1) ([ ] $\Rightarrow 0 \leq x)$

PROVE-LE is called which calls LESS= which fails.

PROVE-LE then calls RLE which succeeds with

$\{x: 0 \leq x < \infty\}$

(P→ 1 2) ([ ] $\Rightarrow x \leq 5)$

RLE $\{x: 0 \leq x \leq 5\}$

(P→ 2) $([a \leq 2 \wedge 2 \leq b] \Rightarrow a \leq x)$

RLE $\{x: (\max\ 0\ a) \leq x \leq 5\}$

Here RLE has intersected the interval $\{x: a \leq x < \infty\}$ with $\{x: 0 \leq x < 5\}$ to obtain the desired result. However, before it can return this answer it must verify that it is not empty; i.e., that $a \leq 5$.

(P→ 2 CHECK) $([a \leq 2 \wedge 2 \leq b] \Rightarrow a \leq 5)$

PROVE-LE calls GLE, which inserts $\sim(a \leq 5) \equiv$

$(5 < a)$ into TY to get

$[5 < a \leq 2 \wedge 2 \leq b]$

which has a contradiction (by routine CONTRADICTION) . Q.E.D.

The reader is referred to [1] for a description of the proofs, given by the prover, of Examples 4-6 below. LUB, L1, and L2 are lemmas used in these proofs.

LUB: $([\exists u \forall t(t \leq b \wedge f(t) \leq 0 \rightarrow t \leq u)$

$\wedge \exists r(r \leq b \wedge f(r) \leq 0)] \rightarrow$

$[\forall x(x \leq b \wedge f(x) \leq 0 \rightarrow x \leq \ell)$

$\wedge \forall y(\forall z(z \leq b \wedge f(z) \leq 0 \rightarrow z \leq y)$

$\rightarrow \ell \leq y)])$

L1: $\forall x[a \leq x \leq b \wedge 0 < f(x) \rightarrow$

$\exists t(t < x \wedge \forall s(t < s \leq x \rightarrow 0 < f(s)))]$

L2: $\forall x[a \leq x \leq b \wedge f(x) < 0 \rightarrow$

$\exists t(x < t \wedge \forall s(x \leq s < t \rightarrow f(s) < 0))]$

Ex. 4. $(LUB \wedge f(\ell) < 0 \wedge 0 \leq f(b) \wedge$

$\forall s(0 \leq f(s) \wedge \ell \leq s \rightarrow t \leq s) \rightarrow t \leq \ell)$.

Ex. 5. $(LUB \wedge L1 \wedge L2 \wedge a \leq b \wedge f(a) \leq 0$

$\wedge 0 \leq f(b) \rightarrow 0 \leq f(\ell))$.

Ex. 6. $(LUB \wedge L1 \wedge L2 \wedge a \leq b \wedge f(a) \leq 0$

$\wedge 0 \leq f(b) \rightarrow \exists x(f(x) \leq 0 \wedge 0 \leq f(x)))$

4. REMARKS

The authors would be interested in hearing how well other provers are able to handle these examples, especially Ex. 4-6.

One of the reasons for our success here was our use of devices such as an algebraic simplifier to avoid the explicit use of the axioms of the real number system. Our special handling of inequalities also attempted such a saving by avoiding such axioms as the transitivity of $\leq$,

$(x \leq y \wedge y \leq z \rightarrow x \leq z)$, but much more needs to be done if we are to smoothly and economically handle the "low level" calculations associated with proofs in analysis. For example, special techniques are needed for absolute values, especially when they occur in connection with inequalities.

We don't think of this prover as ultimate in any sense. At best it will become a part of a more powerful prover we are now trying to build, which will include many of the features mentioned in [10]. For instance, recent work indicates that the use of automatically generated counterexamples can greatly speed up the proofs of these examples and others like them.

ACKNOWLEDGEMENT

This work was supported in part by National Science Foundation Grants MSC 77-20701 at The University of Texas at Austin and MCF 76-81425 at SRI International.

REFERENCES

[1] Bledsoe, W. W., P. Bruell, and R. Shostak. "A Prover for General Inequalities." Math. Dept. Memo ATP-40A, Univ. of Texas at Austin, February, 1979.

[2] Bledsoe, W. W., and Mabry Tyson. "Typing and Proof by Cases in Program Verification." Machine Intelligence 8 (1977).

[3] Shostak, Robert. "A Practical Decision Procedure for Arithmetic with Function Symbols." JACM, April, 1979.

[4] Nelson, Greg, and Derek Oppen. "A Simplifier Based on Efficient Decision Algorithms." IN Proc. 5th ACM Symp. on Principles of Programming Languages, 1978.

[5] Suzuki, Norihisa. "Verifying Programs by Algebraic and Logical Reduction." In Proc. of Int'l Conf. on Reliable Software, IEEE, 1975, 473-481.

[6] King, J. C. "A Program Verifier." Ph.D. thesis, Carnegie-Mellon University, 1969.

[7] Hodes, Louis. "Solving Problems by Formula Manipulation in Logic and Linear Inequality." In Proc. IJCAI-71, London, 1971, 553-559.

[8] Cooper, D. C. "Programs for Mechanical Program Verification." Machine Intelligence 6 (1971), 43-59.

[9] Bledsoe, W. W. "A Maximal Method for Set Variables in Automatic Theorem Proving. Machine Intelligence 9 (1979).

[10] Bledsoe, W. W. "Non-resolution Theorem Proving." AI Jour. 9 (1977), 1-35.

SYMMETRY ANALYSIS Of TWO-bIMENSIONAL PATTERNS FOR COMPUTER VISION*

Robert C. Bolles
SRI International,
333 Ravenswood Avenue,
Menlo Park, California 94025

A system is described that automatically determines the rotational and mirror symmetries of two-dimensional patterns. The properties of patterns that determine their symmetries are delineated, a representation scheme based on these properties is defined, and algorithms to perform the symmetry analysis are presented. An implementation based on the SRI vision module is described.

1. INTRODUCTION

Two fundamental properties of a two-dimensional pattern are its rotational and mirror symmetries. These properties are important in the recognition and location of patterns, because they can be used to locale key features that determine the orientations of the patterns and differentiate them from those that are similar. These properties are especially important in performing industrial vision tasks, in which symmetrical, or almost symmetrical, parts are common. For example, Figure 1 is a part to be picked up and added to a subassembly. It is two-fold rotationally symmetric (i.e.. its appearance is unchanged by a rotation of 1o0 degrees). If it is occasionally presented upside down, it would be important to know that it is not mirror-symmetric (the relative orientation of the holes with respect to a line joining their centers is different in the mirror image;. Since the pattern is not mirror-symmetric, a vision system could detect upside-down parts and instruct the manipulator to turn them over before adding them to the subassemblies.

Current industrial vision systems, like SRI's vision module [5] and Bausch and Lomb's 0MNIC0N [2], recognize patterns on the basis of such global features as the area and perimeter of a pattern, because statistical pattern recognition techniques can easily model such features. Since these simple vision systems do not use local features, such as holes and corners, or their symmetries, they cannot automatically determine the orientation of the pattern in Figure 1 or distinguish it from its mirror image.

A few artificial intelligence programs have performed symmetry analysis. Evans' program, which worked geometric-analogy problems, tested the primitive figures to see whether they were mirror-symmetric about a horizontal or vertical axis, [3]. Gips' program analyzed two 3-dimensional strings *of* cubes to determine whether they were rotational or mirror

* This work was supported by NSF under grant APR75-13074.

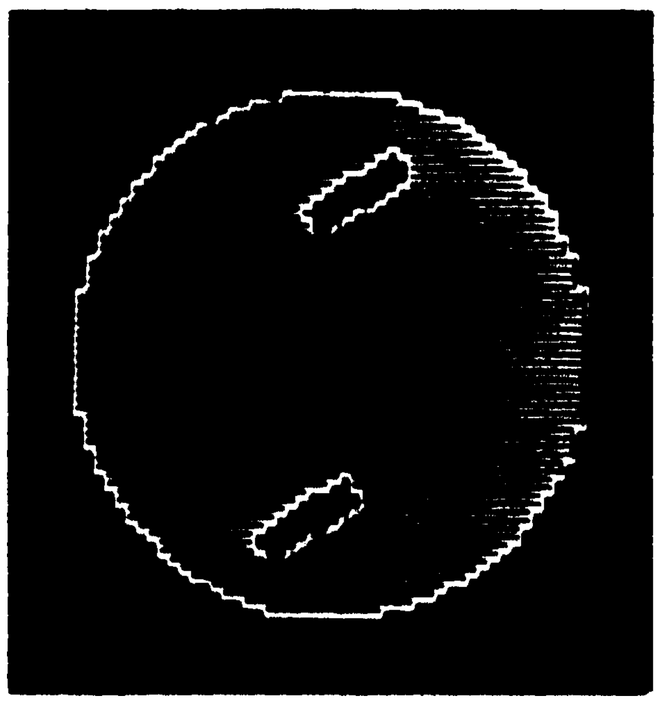


FIGURE 1 PICTURE OF A SAMPLE PART

transformations of each other [4], Perkins' program [7] used a form of correlation to determine whether a pattern was rotationally symmetric. That program, like its counterparts, was not intended tO perform a general-purpose symmetry analysis aimed at understanding local features, structures of local features, and their properties. It is precisely this kind of comprehensive understanding that is the purpose of the research reported in this paper.

In a recent paper Wechsler [8J describes an algorithm that decomposes two-dimensional patterns into mirror-symmetric components. His approach employs the same mathematical descriptions and tests for symmetry as are referred to in this paper, but his goal is to describe regions rather than characterize structures of local features.

2. REPRESENTATION *SCHEME.*

We shall now define a representation scheme based on a set of structural primitives for patterns. The intent Is to identify important constructs, their properties and relationships, and to define a convenient way to describe patterns.

Pattern symmetries are determined by the properties of groups of "similar" features. Features are defined as similar if (1) they have the same description (i.e., type, size, and shape), (2) they are equidistant from the pattern's center of gravity, and (3) they have the same orientation with respect to a line from the pattern's center of gravity to their own. Therefore, if two features are similar, the pattern can be rotated about its center of gravity so that one feature is repositioned at the other feature's original position and orientation. For example, the two holes in Figure 1 are similar.

Groups of similar features are important because they form structures of features that limit the potential number of rotational symmetries for the pattern. In particular, if a pattern is M-fold rotationally symmetric and it contains a structure of features that is N-fold rotationally symmetric, M must be a divisor of N,

The representation scheme describes patterns in terms of these structures of features. Since the features in a structure are similar, the structures can be represented by one example of the feature and a list of angular positions. Individual features can either be simple, such as a corner, or complex, such as the entire pattern in Figure 1. Complex features are described in terms of structures of other

Therefore, the representation scheme patterns as trees in which the nodes . . individual features or structures of features. Figure 2 shows the two types of nodes. The "descriptions" of individual features correspond to the global features used by simple vision systems.

FEATURE	STRUCTURE OF FEATURES
POSITION	POSITION
ORIENTATION	ORIENTATION
ROTATIONAL SYMMETRY	ROTATIONAL SYMMETRY
DESCRIPTION: TYPE	POINTER TO SAMPLE FEATURE
SIZE	NUMBER OF OCCURRENCES
SHAPE	POINTER TO LIST OF ANGULAR POSITIONS
POINTER TO LIST OF STRUCTURES	

FIGURE 2 NODES IN THE REPRESENTATION

Every node in a tree that represents a pattern has a position, orientation, and rotational symmetry. The "leaf" nodes, which correspond to simple (or primitive) features, have inherent values for these properties. The values for the other nodes are functions of their inherent values and the values of the nodes directly beneath them. An important part of the representation scheme is the set of conventions that specify these functional relationships. Unfortunately, these conventions cannot be described for lack of space, but may be found in [6j.

3. £UHMEI£I AfIALLSIS

In this section we briefly describe algorithm for performing the symmetry analyses.

3 .1 Rotational Symmetry

Given a pattern, the rotational symmetry analysis builds a tree to describe it and in the process, determines its rotational symmetry. The algorithm is as follows:

```
RotSymm(PATTERN) =
 | Create a new feature node for the PATTERN
   Fill in the description of the PATTERN
   Set the node's list or structures to empty
   If the PATTERN is complex then
     For each subpattern. call RotSymm(subpat)
     Group together similar subpattems
     For each group
        Create a new structure node
        Add it to the list of structures
        Select a sample feature for the struct.
        Kill in the number of occurrences and
             angular positions of the features
        Delete the unused feature nodes
        Compute the position, symmetry, and
             orientation of the structure
   Compute position, symmetry, and orientation
        of the PATTERN
   Return a pointer to the PATTERN'S node
```

The rotational symmetry of the pattern is the symmetry of the topmost feature node in the tree description.

•2 Mirror Symmetry Analys

iven a tree description of a pattern, the mirror symmetry analysis builds a tree to describe the mirror image of the pattern and, if the two trees are similar, declares the pattern to be mirror-symmetric.

Two short recursive procedures can build the mirror image tree. One produces the mirror image of a feature node, the other produces a mirror image of a structure-of-features node. Since the similarity of two patterns is independent of their orientation, the mirror image of a pattern can be formed by reflecting it about any axis, not necessarily an axis of mirror symmetry. Therefore, it is not necessary to locate an axis of mirror symmetry to determine whether or not a pattern is mirror-symmetric. The mirror image of a feature node can be formed by reflecting its position and orientation about the X-axis and replacing each structure in the list of substructures by its mirror image. The mirror image of a structure can be formed by reflecting its position and orientation about the X-axis, replacing the sample feature by its mirror image and reflecting all the angular positions in the list of occurrences about the X-axis.

The mirror symmetry analysis could be performed without constructing the mirror image tree, but the algorithm would be more complicated.

The experimental system is an extension of the vision system developed by Agin at SRI International [1], which performs a connectivity analysis of binary pictures and produces a tree description of the "blobs" in the picture. (A blob is a connected region of uniform color.) The symmetry analysis converts blob descriptions into the symmetry representation and performs the mirror symmetry analysis. The features it uses are holes, elongations (i.e., axes with the minimum second moments), and minimum and maximum radii. The analysis automatically locates these features, groups them into structures of features, and determines their symmetries. Sample results are shown in Figure 3. The pattern in Figure 3a is not rotationally symmetric, but it is mirror-symmetric. The pattern in Figure 3b is three-fold rotationally symmetric, but not mirror-symmetric. The pattern in Figure 3c is two-fold rotationally symmetric, but not mirror-symmetric.

Since the algorithms analyze digital images, thresholds are required for making some of the decisions. For example, how close to perfect symmetry does a set of features have to be for the program to call it symmetric? The program automatically establishes these thresholds on the basis of the pattern being analyzed and a model of the expected pixel errors. These thresholds, however, can be interactively overridden.

One of the basic goals of this research is to simplify the process of producing programs for recognizing and locating patterns. Ultimately this means the development of a system that can be trained by showing it examples. Such a system would determine the symmetries of a pattern, locate key features, choose the most cost/efiective features to use at run time, and produce a program to perform the task. In contrast thereto, the current program is limited to determining the symmetries and locating some kev features.

REFERENCES

[1] G. J. Agin and R. Duda, "SRI Vision Research for Advanced Automation," Second USA-Japan Computer Conference, Tokyo, Japan (August 1975).

[2] "Bausch and Lomb Omnicon Pattern Analysis System," Analytic Systems Division Brochure, 820 Linden Avenue, Rochester, New York, (1976).

[3] T. G. Evans, "A Program for the Solution of a Class of Geometric-Analogy Intelligence-Test Questions," Semantic Information Processing, pp. 271-353, (MIT Press, Cambridge, Massachusetts 1968).

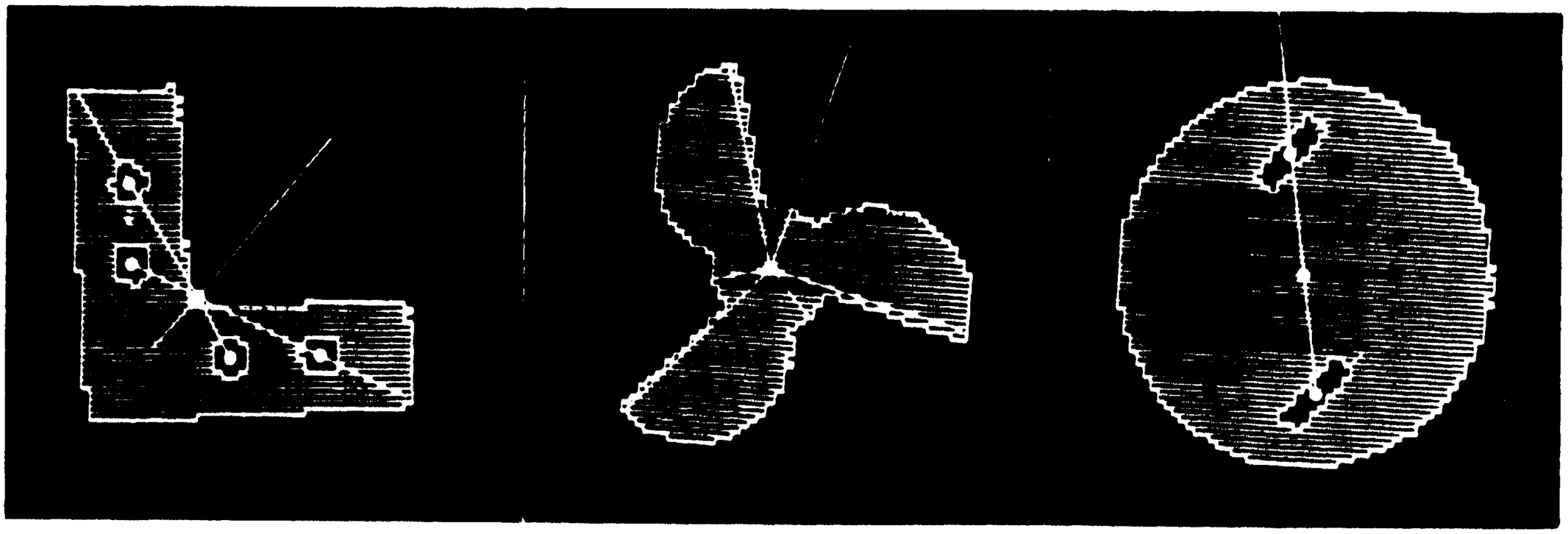
[4] J. Gips, "A Syntax-Directed Program that Performs a Three-dimensional Perceptual Task," Pattern Recognition, Vol 6, pp. 189-199 (1974).

[5] G. Gleason and G. J. Agin, "A Modular Vision System for Sensor-Controlled Manipulation and Inspection," 9th Intl. Symposium on Industrial Robots, Washington, D.C., pp. 57-70 (March 1979).

[6] D. Nitzan, G. Agin, R. Bolles, G. Gleason, J. Hill, D. McGhie, R. Prajoux, A. Sword, and W. Park, "Machine Intelligence Research Applied to Industrial Automation," SRI Report to NSF, SRI Intl., Menlo Park, California (August 1979).

[7] W. A. Perkins, "A Model-based Vision System for Industrial Parts," IEEE Trans. on Computers, Vol. C-27, pp. 126-143 (February 1978).

[8] H. Wechsler, "A Structural Approach to Shape Analysis Using Mirroring Axes," Computer Graphics and Image Processing, Vol. 9, No. 3, pp. 246-266 (March 1979).



(A) (B) (C)

FIGURE 3 RESULTS OF THE SYMMETRY ANALYSIS

AUTOMATIC DETERMINATION OF IMAGE-TO-DATABASE CORRESPONDENCES

R.C. Bolles, L.H. Quam,
M.A. Fischler, H.C. Wolf
SRI International,
Menlo Park, California 94025

New techniques are described for some of the important computations in the automatic determination of image-to-database correspondences. In particular, a technique to predict a region in the image within which a feature is expected to appear, a set of techniques to verify feature matches, and a technique to extend the refinement process to include a new type of match based on linear features, such as roads, are discussed. These techniques are demonstrated in an example in which the system reduces the uncertainties from approximately plus or minus 200 feet on the ground to approximately plus or minus two feet.

1. INTRODUCTION

Computing an image-to-database correspondence Is a general problem occurring in all knowledge-based systems. In most image tasks the correspondence is a projective transformation, which can be modeled as a function of the camera parameters, such as focal length, X, Y, Z, heading, pitch, and roll. If the parameters are known precisely, the model can precisely predict the two-dimensional image coordinates for any three-dimensional database point.

form of the image-to-database
ence problem is to be given good
of the camera parameters and be asked
them. In navigation this refinement
the crucial step that improves the
stimate of the location of the plane.
detection it is used to align two
that the corresponding regions can be

The basic approach we are using to refine a correspondence is to locate known features in the image and use their locations to improve the correspondence. A database contains descriptions of the available features. The predicted viewpoint and viewing conditions are used to choose features to be located. The estimates of the camera parameters are used to predict what the features look like and where they are likely to appear. Feature detection techniques ("operators") are chosen to locate the features and they are applied. Since the operators may not locate their intended features, their results are verified either by locating a larger portion of the features or by checking the relative positions of other features. After a set of features has been found, their locations are used to refine the estimates of the camera parameters. The parameters are refined by searching the parameter space for sets of parameter values that minimize the distances between the predicted locations of features and the locations determined by the operators. If the correspondence is not precise enough, the whole process can be repeated.

* This work was supported by ARPA under contract DAAG29-76-C-0057.

A number of people have worked on parts of this process [1, 5, 4, 6, 7, 8, 10, 11, 12], but mainly for pairs of images that were taken closely in time and from similar viewpoints.

Our research goal is to produce an automatic system to refine correspondences for Images taken from a variety of viewpoints and under a variety of viewing conditions. This variety makes makes It more difficult to locate features, which in turn, increases the need for verification techniques and carefully planned strategies. Which operators should be used for an image taken from this viewpoint and under these conditions? When should the results of one operator be used to reduce the predicted search area for a nearby feature? This type of question becomes more important as features become harder to find.

To build an automatic correspondence refinement system we need to develop new models and techniques for several of the steps in the process. So far we have concentrated on a few of them: the prediction of Image locations for features, the verification of the results of operators, and the computation of refined correspondences. In this paper we will state our assumptions, describe our new techniques, and present an example.

2. ASSUMPTIONS

Our assumptions are summarized in Figure 1. Our experimental system refines correspondences for aerial images of rural areas containing roads [9].

We assume that we have a database of the area on the ground contained in each picture to be analyzed. The database contains the geometry and topology of the features to be used In the refinement process.

We expect the pictures of an area to be from different viewpoints. They may be taken at

5,000 feet or 20,000 feet; they may be *45-degree obliques or vertical pictures. This variety implies that intensity correlation is not always sufficient to locate features.

(1) Aerial images of roads
(2) Perspective images
(3)Repetitive coverage
(5) Variety of sun angles
(5) Variety of viewpoints
(6) Ground resolutions between 20 feet/pixel and 1 foot/pixel
(7) Time of day and day of year image was taken
8) Estimates of camera parameters
9) Small parameter uncertainties
(10) Maximum uncertainty regions on the ground of 200 x 200 feet
(11) Database of roads and other features
(12) Features obscured by clouds, shadows,and terrain features

FIGURE 1 ASSUMPTIONS

Even though the viewpoints may vary widely, we expect to be given good estimates of the camera parameters for each picture. We also expect a measure of the uncertainty associated with each parameter estimate. For example, the HEADING might be estimated to be 75 degrees, plus or minus two degrees. These uncertainties are used to predict the regions in a picture to be searched in order to locate a feature. We will refer to these search regions as "uncertainty regions." The smaller the uncertainties, the smaller the uncertainty regions; the smaller the uncertainty regions, the easier it is to automatically locate the desired features.

Two of our assumptions restrict the range of initial uncertainties about the camera parameter estimates. The first one restricts the combined uncertainties so that they do not imply uncertainty regions on the ground of more than approximately plus or minus 200 feet. The second one restricts the size of each parameter's uncertainty so that it is relatively small. The first assumption, in effect, restricts the sizes of the uncertainty regions that have to be searched to locate a feature. For example, if an image has a resolution of 1 foot/pixel, the largest uncertainty region would then be approximately 400x 400 pixels. The second assumption limits the portion of the parameter space that the optimizer has to search. It also indirectly limits the maximum geometric change in the appearance of a feature.

3. UNCERTAINTY. REGIONS

What region in the picture will have a given probability (e.g., a 95% probability) or containing a point feature from the database? To answer this question, one has to predict the effect on the location in the image of a feature caused by changing the parameter values in accordance with their stated uncertainties. To do that, one needs a model of their uncertainties. The error model we use is that the parameters vary according to a joint normal distribution, which is a reasonable assumption for measurements produced by a device such as an inertial guidance system because each parameters error is a sum of several small errors. For this model the uncertainty regions are ellipses in the image plane. The derivation of this fact can be found in [5].

Having found one feature, one would expect that its location would greatly restrict the possible locations for a nearby feature. This idea leads to a second type of uncertainty region, a relative uncertainty region. In addition to the normal Information used to compute an uncertainty region, a relative uncertainty region is a function of another feature and its location. Given the assumption that the camera parameters vary according to a joint normal distribution, the relative uncertainty regions are also ellipses. A derivation of the mathematical description of a relative uncertainty region can be found in [5].

A relative uncertainty region is used to reduce the amount of work required to locate a second feature after a nearby feature has been found. This is particularly useful when a possible match for a feature is being verified. The logic is as follows: if this is feature A, then feature B should be in a small region over there; if B is not there, then we cannot depend on this being A.

Figure 2 shows the initial uncertainty ellipse and the relative uncertainty ellipse about a point feature. The large ellipse is the uncertainty region predicted from the uncertainties about the camera parameters. The small ellipse is the relative uncertainty region derived from the location of the road marking just above it in the picture.

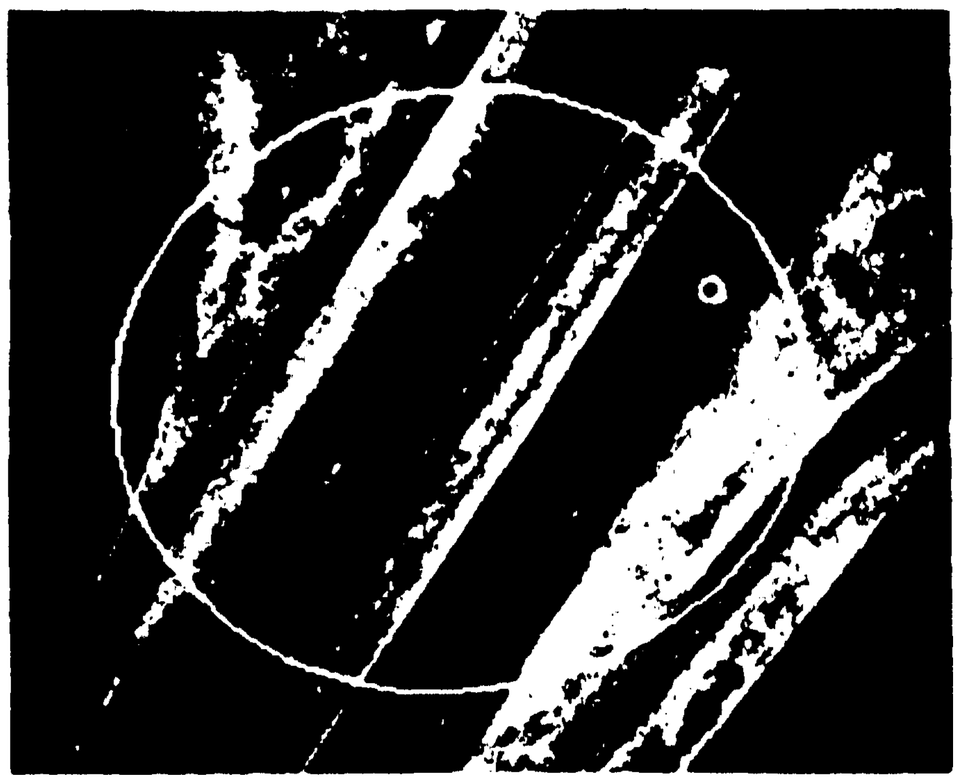


FIGURE 2 UNCERTAINTY REGIONS

4. PONT-Qh-A-LINE MATCHES

Point-to-point matches are the standard feature matches used in correspondence refinement. We have introduced point-on-a-line matches because many of the prominent features in pictures are linear, such as roads in aerial images and surface-intersection edges in blocks-world images. To do so we have implemented operators to find points on roads, developed techniques to verify these matches, and extended the parameter refinement process to use them.

We have Implemented two techniques that locate oints on roads. One is used at low resolution e.g., 20 feet/pixel) when roads appear as lines, and one is used at high resolution

(e.g., 1 foot/pixel) when the internal structure of the road is discernible. The low resolution technique is based on the Duda road operator [2], which is a type of line-finding operator. The high resolution technique is an adaptation of Quara's road tracking operator [13]. which performs a one-dimensional correlation of the expected road cross section to portions of the image.

There is a built-in trade-off between point features and line features: it is easier to find a point on a line than it is to locate a point feature, but less information is gained by doing so. Point-to-point matches produce twice the number of constraints for the refinement process, but they are generally more expensive to find because an area search is required as opposed to a linear search for point-on-a-line matches.

5. CORRESPONDENCE REFINEMENT

The correspondence refinement process (or "optimizer") is similar to Gennery's approach to calibration [11]. It solves the nonlinear problem by iteratively solving linear approximations. For point-to-point matches a three-dimensional point in the world is matched with a two-dimensional point in the image. In that case the optimizer has two residuals per match to use to improve the camera parameter estimates: the X and Y components of the difference between the predicted image of the world point and the point in the image at which the operator located its match. If instead of locating a specific point, an operator locates a point on a line, the optimizer only has one residual to use because the point could be any place along the line. The residual for a point-on-a-line match is the shortest distance from the point to the line. As the optimizer searches for improved camera parameters, the image of the three-dimensional line should get closer to the point located by the operator, but the closest point on the line may slip back and forth along the line.

So far the optimizer has only been extended to handle point-on-a-line matches. It may be useful to extend the optimizer to include other types of matches that involve a point and an analytic curve, e.g., a point-on-an-ellipse match. The main components of such an extension are (1) a procedure *to* compute the minimum distance between a point and the curve and (2) a procedure to compute the partial derivatives of that distance with respect to the camera parameters.

The optimizer could even be extended to arbitrary curves by incorporating a procedure, suoh as chamfering [3],that Cmputes the distance between a point and an arbitrary curve. Unfortunately, such distance computations are generally expensive.

The current implementation of the optimizer is relatively fast. It takes one second on our KL-10 to perform one iteration when 100 residuals are used to refine the estimates. (Recall that each point-to-point match adds two residuals; each point-on-a-1ine match adds one.) Five to ten iterations are normally required to achieve convergence, which is defined to be a state in whioh the parameter adjustments are on the order of .00005 units.

As Gennery points out, the optimizer can be used to filter out "mistakes" by iteratively deleting the match with the largest normalized residual, if it is larger than a threshold. In practice this heuristic has proven to be useful, but it is expensive and does not always produce the intended result. Therefore, it is important to filter out as many mistakes as possible before calling the optimizer. The next section describes some of the ways this filtering or verification can be done.

6. FEATURE VERIFICATION

As mentioned in the last section, it appears to be more cost-effective to filter out mistakes, if at all possible, before applying the optimizer. We have identified four possible methods for performing such filtering:

Operator threshold - Be suspicious of any match for which the operator does not produce a confidence above a certain threshold; e.g., if a two-dimensional correlation operator roduces a correlation of less than .8, gnore its results.

Self support - Be suspicious of any match that cannot be verified by locating a larger portion of the same feature; e.g., if an operator locates a point that is supposed to be on a road but the road tracker cannot extend the match, ignore it.

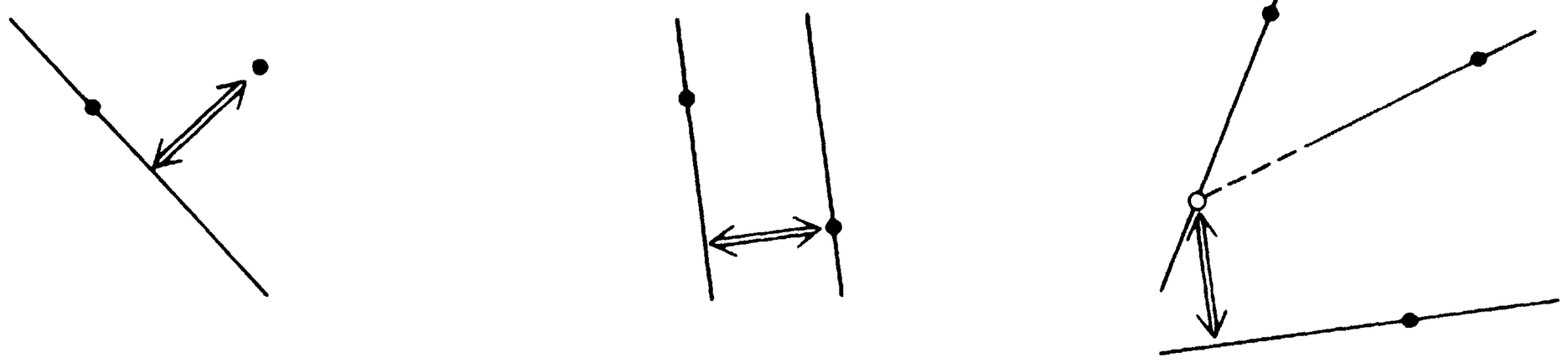
Pairwise support - Be suspicious of a pair of matches if they are not positioned correctly with respect to each other: e.g., if the distance between the matching locations for two road markings is significantly longer than expected, assume that at least one of the matches is incorrect.

Group support - Be suspicious of a set of matches whose locations do not form a predicted structure; e.g., if the matching locations for four features do not form a square, as expected, assume that at least one of the matches is incorrect.

We differentiate between these heuristics because they require different models and techniques.

It is relatively straightforward to apply all of the verification methods to point features. The relative uncertainty regions can be used to determine if two features are mutually consistent. This pairwise consistency can be extended to group consistency through maximal clique techniques [1] or through optimal embedding techniques [8].

Additional care has to be taken to apply the verification techniques to point-on-a-line matches. The Important test is to be able to distinguish pairwise consistent matches from pairwise Inconsistent matches when one or more of the matches is a point-on-a-line match. Figure 3 shows the three significantly different cases. In Figure 3a one of the two matches is a point-to-point match and one is a point-on-a-line match. If the slope of the line is known accurately, the distance between the point and the line can be used to determine if the matches are consistent. Since the uncertainties associated with each camera ?arameter are relatively small, the slope of the ?ine should remain relatively constant. Thus the distance from the point to the line should be relatively constant.



(A) (B) (C)

FIGURE 3 "PAIRWISE" SUPPORT CONFIGURATIONS

In Figure 3b both of the matches are point-on-a-line matches and the lines are essentially parallel. In this case the distance between the lines is sufficient to check the relative positions of the two matches. For example, if an operator is trying to locate both sets of lanes on a freeway, the distance between the two sets of lanes should be within a predetermined range.

If both of the matches are point-on-a-1ine matches and the lines are not parallel, as in Figure 3c, some additional information is needed in order to check their relative consistency. One solution is to intersect the two lines and use that point in conjunction with a third match to check the relative position of all three matches.

7. EXAMPLE

The example task is to refine the image-to-database correspondence for the picture shown in Figure 4using its full resolution of approximately 2 feet/pixel. The initial uncertainties about the camera parameters imply uncertainties in the image of plus or minus 95 pixels, which correspond to approximately plus or minus 190 feet on the ground. The goal is to reduce these uncertainties to approximately plus or minus one pixel, an increase in precision of almost two orders of magnitude.

The database used in this example contains two types of features, linear road segments and road surface markings. Figure 5 shows the features that are available for this site. The lines represent the road segments and the pluses represent the surface markings. The appearance of each road segment is described by a road cross section model. The appearance of a surface marking is described by an image patch from a previous picture of the site.

A fixed strategy has been implemented to use these features to perform the task and demonstrate our new techniques. The basic approach is to locate the linear features first because they are less expensive to find, use them to refine the camera parameters, locate the point features, use them to verify the first refinement, and then perform a second refinement using both the points and the lines.

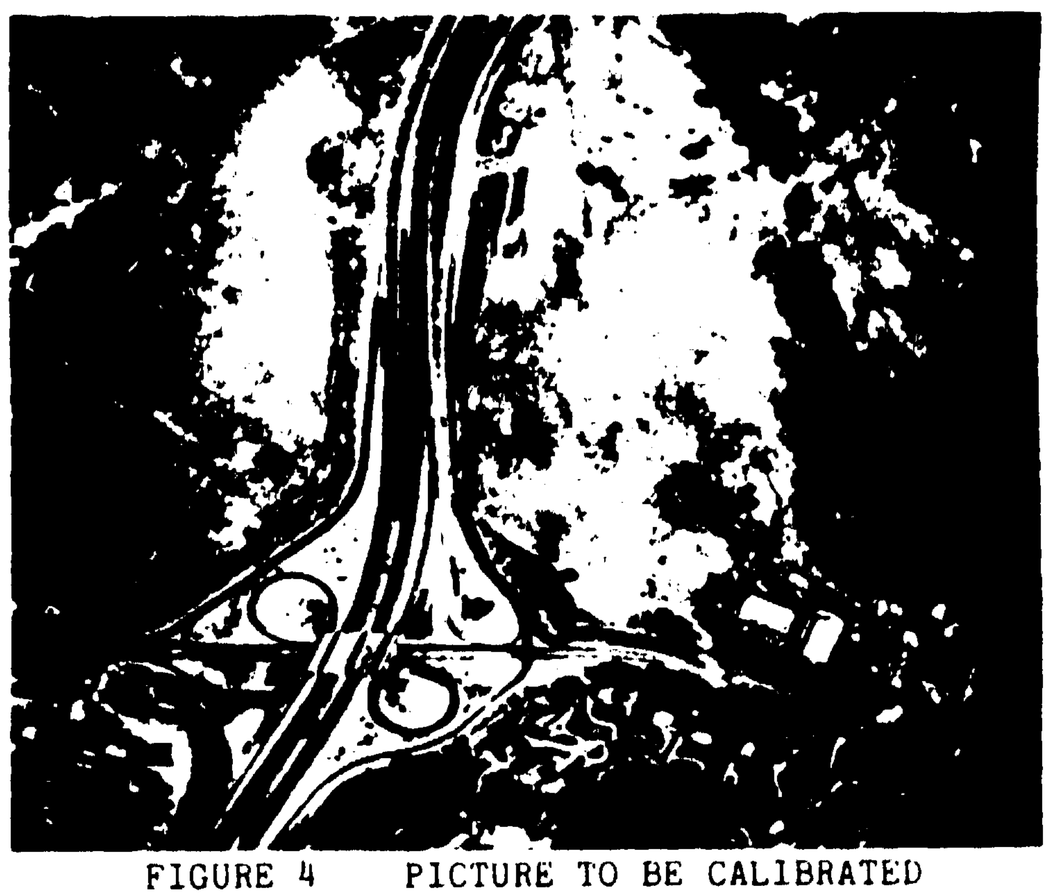


FIGURE 4 PICTURE TO BE CALIBRATED

FIGURE 5 DATABASE FEATURES

Given estimates for the camera parameters, the system predicts the location of the road segments In the new picture. Figure 6 shows these predictions, which are shifted left and down approximately 60 pixels from their actual locations. The estimates of the camera parameters are also used to warp each road cross section to the expected size and orientation of the corresponding road segment. In addition, the estimates of the uncertainties about the camera parameters are used to predict the uncertainty regions about the center points of each linear segment. Figure 6 shows these uncertainty ellipses that have a 95%probability of containing the desired point.

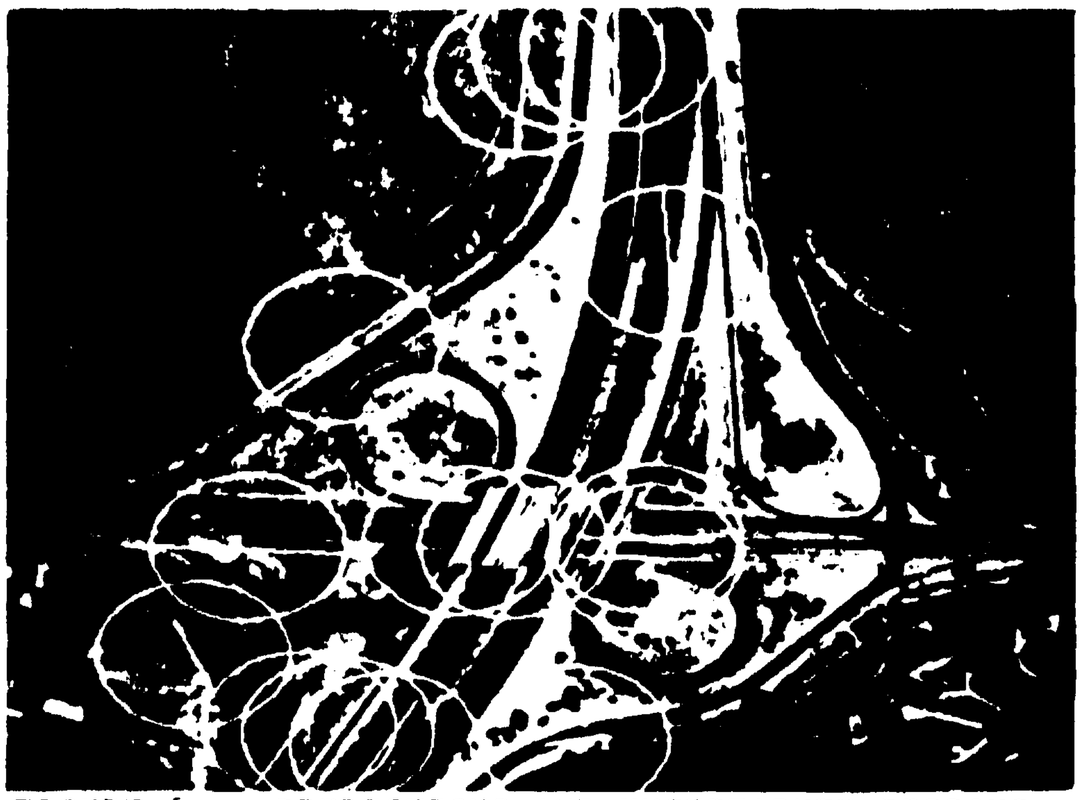


FIGURE 6 PREDICTIONS AND UNCERTAINTY REGIONS

The search strategy for a linear feature is to look along lines perpendicular to the expected location of the feature. The lengths of the search lines are determined by the uncertainty ellipse.

The high-resolution, one-dlmenslonal correlation operator Is applied along the search line to locate points that may be on the desired road. The self-support method is used to verify each candidate point. That is, the road tracker is asked to track the road for a short distance. If it cannot, the point is abandoned.

Self-support is not sufficient to identify some road segments because there are two or three parallel roads that are close to each other and all look alike. Pairwise and group support are used to identify these roads on the basis of preplanned groups of features. For example, Figure 7 shows three sets of lanes, two of which are difficult to tell apart. The relative locations of the three sets of lanes are used to determine the correct matches. The lines perpendicular to the roads indicate the final choice for a consistent set of matches.

Figure 8 shows the results of searching for all of the road segments In the database (shown In Figure 5). Two of the roads were not found because the contrasts were not sufficient to produce matches with the desired confidence. The matches were given to the optimizer along with the initial estimates of the camera parameters and the uncertainties about the estimates; the optimizer produced new estimates

FIGURE 7 EXAMPLE OF GROUP SUPPORT

FIGURE 8 ROAD ACQUISITION RESULTS

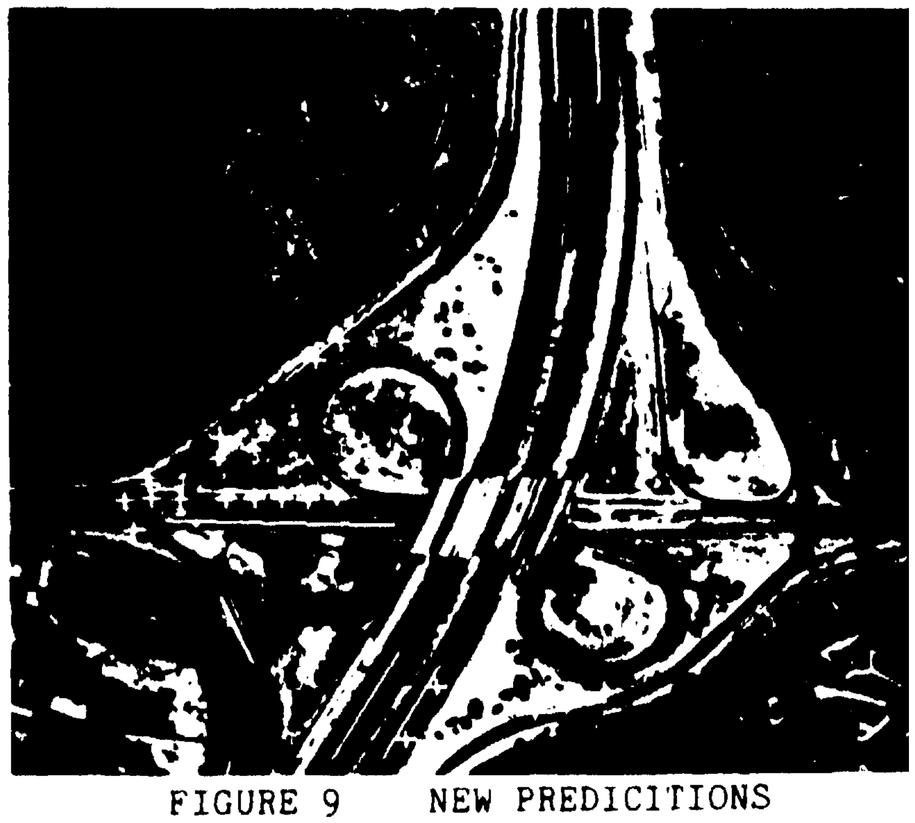


FIGURE 9 NEW PREDICITIONS

for the parameters and new uncertainties. Figure 9 shows the new predictions for the locations of the road segments. The new uncertainties imply uncertainties in the image of approximately plus or minus 1.5 pixels, close to our goal.

To verify the new estimates the surface markings were located. The new estimates were used to predict the locations and appearances of the features; the new uncertainties were used to predict the uncertainty regions; and two-dimensional correlation was used to locate the features. The average difference between the predicted location and the matching location was approximately 1.3 pixels and the largest distance was 1.7 pixels. The final refinement based on both the lines and the points reduced the uncertainties in the image to approximately 1.1 pixels, which is very close to our goal and corresponds to approximately 2.2 feet on the ground.

We have begun to experiment with pictures containing clouds. Figure 10 shows the linear features that the system found in one example.

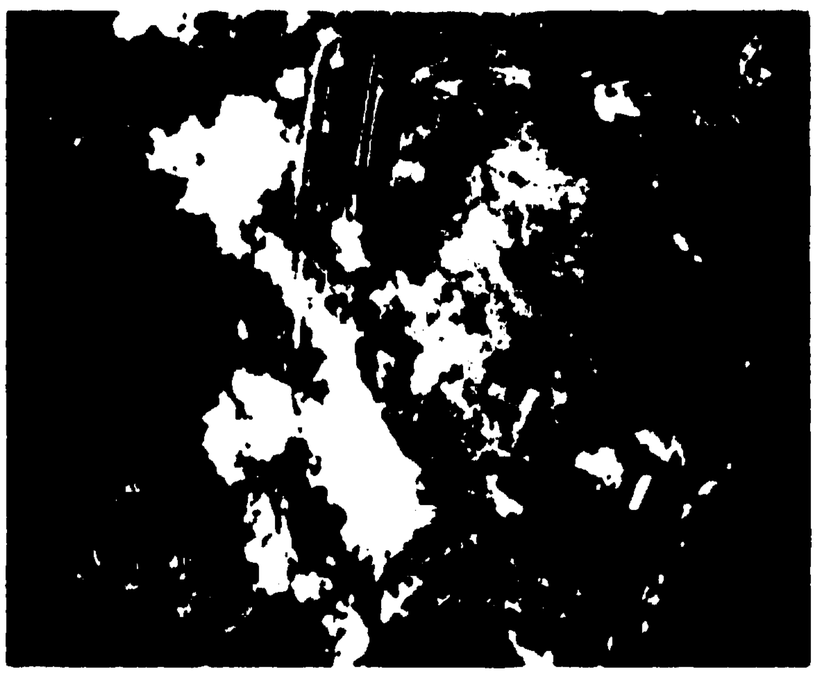


FIGURE 10 ROAD ACQUISITION RESULTS IN CLOUDS

8. CONCLUSION

wee described and demonstrated a set of techniques to perform some of the subtasks required in an automatic system to refine image-to-database correspondences. In particular, we discussed techniques to compute uncertainty regions, techniques to incorporate point-on-a-line matches, and techniques to verify the results of operators. These techniques were combined to form a strategy, which we demonstrated in an example task.

Additional research is required on several other key subtasks required in an automatic system, such as, the selection of features and the tailoring of a strategies to different tasks. Other needs include better feature modeling, better operators to locate features over a wide range of viewing angles and conditions, and an alternative to least-squares optimization.

REFERENCES

[1 A.P. Ambler et al., "A Versatile Computer-Controlled Assembly System," Proc. Third IJCAI. pp. 298-307 (August 1973).

[2] H.G. barrow, "Interactive Aids for Cartography and Photo Interpretation," Semiannual Technical Report, SHI Project 5300, SRI International, Menlo Park, California (November 1976).

[3] H.G. Barrow et al., "Parametric Correspondence and Chamfer Matching: Two New Techniques for Image Matching," Proc. Fifth IJCAI. pp. 659-663 (August 1977).

[4] fi.C. Bolles, "Verification Vision for Programmable Assembly," Proc. Fifth IJCAI. pp. 569-575 (August 1977).

[5] R.C. Bolles et al., "The SRI Road Expert: Image-to-Database Correspondence," Proc. Image Understanding Workshop (November 1978).

[6] B.L. Bullock et al., "Finding Structures iisn Outdoor Scenes," Hughes Research Report 498, (July 1976).

[7] L. Davis, "Shape Matching Using Relaxation Techniques," TR-480, Computer Science Dept., University of Maryland, (September 1976).

[8] M. Fischler and R. Elschlager, "The Representation and Matching of Pictorial structures," IEEE Trans on Computers, No. 22, pp. 67-92 (1973)7

[9] M. Fischler et al., "The SRI Road Expert: An Overview," Proc. image. Understanding Workshop (November 1978;.

[10] T.D. Garvey, "Perceptual Strategies for Purposive Vision," Technical Note 117, SRI Intl., Menlo Park, California (September 1976)

[11] D.B. Gennery, "A Stereo Vision System for an Autonomous Vehicle," Proc. Fifth IJCAI. pp. 576-582 (August 1977).

[12] B.K.P. Horn and B.L. Bachman, "Using Synthetic Images with Surface Models," Comm, of the Vol. 21, No. 11, pp. 914-92U (November 1978).

[13] L.H. Quam, "Road Tracking and Anomaly Detection in Aerial Imagery," Proc. Image Understanding Workshop, pp. 51-55(May 1978).

UNDERSTANDING MEDICAL JARGON AS IF IT WERE A NATURAL LANGUAGE

Alain Bonnet
Heuristic Programming Project, Computer Science Department
Stanford University, Stanford CA 94305

This paper presents BAOBAB-2, a computer program built upon the MYCIN system [II], that is used for understanding medical summaries describing the status of patients. Due to the stereotypic way the physicians present medical problems in these summaries in addition to the constrained nature of medical Jargon, these texts have a very strong structure. BAOBAB-2 takes advantage of this structure by having a model of this organisation as a set of related *schemas* that facilitate the interpretation of these texts. Structures of the schemas and their relation to the surface structure of the text are described. Issues relating to selection and use of these schemas by the program during the interpretation are discussed.

I Introduction

Early works on memory [1] and numerous psychological experiments [5] have suggested that people hearing a story make assumptions that they might revise or refine as more information comes in to confirm or contradict them. Making such assumptions entails building (or retrieving) models of the expected text contents. A corollary of this process is that, if the story adequately fits the model people have in mind, the story will be understood more easily.

Although it is difficult to give a formal definition of what constitutes a coherent text, we have an intuitive notion that sentences that compose it form episodes that are linked by different kinds of relationships, cause-effect, chronological orderings and the like. Medical summaries are highly structured; their structure can be described in terms of topic. An important problem is recognizing the different topics and deciding when a "shift of topic' occurs.

2 Related work and goals

AI research has recently explored strategies to recognize shifts of topic occurring during dialogs or written texts. The two main issues faced in doing so are the necessity to narrow down the space of possible referents of a linguistic object and the risk of combinatorial explosion. Thus, Grosz [6] studied the role of focus in the interpretation of utterances and its relation to domain structure. She used the task structure to resolve definite noun phrases in task-oriented dialogs. Sidner [12] extended this work to determine the use of focusing in the resolution of pronoun references and other kinds of anaphors occurring in dialogs. De Jong (in [10]) has used "scripts" [9] to study another kind of structured texts, such as accident reports.

AII of this work and the present study have a common feature: they do not interpret sentences in isolation, but rather in' the context of an ongoing discourse and hence they use *discourse structure.* BAOBAB-2 " also explores *issues* of (1) what constitutes a model of highly-structured texts and (2) how and when topic shifts occur. The rationale to study schema shifts in the present work are twofold: first, it enables the program to narrow down the space of possible interpretation! of inputs and second, it capitalizes on the coherence of texts, which is mainly a task of detecting anomalies, asking the user to clarify vague pieces of information or disappointed expectations, suggesting omissions.

" This research was supported by the Advanced Research Projects Agency under contract DAHC 15-73-C-0435 and a grant from I'Institut de Recherche pour I'Informatique et i Automatique.

The domain of application is the medical summaries for which processing so far [8] has mainly consisted of filling in formatted grids and has not exhibited any interactive behavior. The program objective is to understand summaries about meningitis cases typed in "medical natural jargon" by a physician, interacting with her or him either to ask questions or to display what it has understood.

The program utilizes a model of what medical summaries typically look like to guide the comprehension. This model consists of a set of related schemas (structures containing domain-specific knowledge which resemble "frames" [7] or "scriptsts" [9]) described below For example, it knows that there is a main character who is the patient This patient presents symptoms He is admitted to the hospital A physician observes signs Some exams are performed, cultures are taken and eventually results are obtained. The physician describes the status of a patient The program uses both its medical knowledge and its model of the usual description of a medical case in order to produce an internal structure usable by MYCIN, which then attempts to make a diagnosis

The program behaves like a clerk or a medical assistant who knows what the physician has to describe and how a malady is ordinarily presented. It reacts to violations of the model, such as a description ignoring symptoms or the mention of a culture that has been drawn but for which no result is ever given. It docs not attempt to use its knowledge to infer any diagnosis but in certain cases can draw inferences that will facilitate MYCIN'S task. It uses these to establish relationships between the concepts stated in order to interpret what is said; for example, it knows that "semi-coma" refers to the state of consciousness of the patient and "hyperthyroidism" to a diagnosis. A potential use of the program is to allow the physician to volunteer information before or during the consultation thereby decreasing her/his frustration at having to wait until asked by the system to mention a crucial symptom.

BAOBAB-2 is comprised of: (a) a parser mapping the surface input into an internal representation; (b) a set of schemas, representing a model of the kind of information it is ready to accept and the range of inferences it will be able to draw; (c) episode-recognition strategies, making it possible to focus on particular pieces of the texts and (d) a generator of English used to display in a non-ambiguous fashion what has been understood This paper will emphasize on the episode-recognition strategies. Detail on the other parts can be found in Bonnet [2] and i3]. These techniques have been successfully implemented using INTERLISP in a program connected with MYCIN'S data base, running on the DEC KA-10 of the medical center at Stanford University.

3 Text and model.

As noted earlier, medical summaries have a stereotypic structure. They can be viewed as a sequence of *episodes* that correspond to phrases, sentences or groups of sentences dealing with a single topic These topics constitute the model and are represented by schemas Understanding the content of an episode leads to building one or several internal clauses referring to the same schema In other words, processing and understanding a text consists of mapping episodes in the text onto schemas that constitute the model. Matching a schema can be "discontinuous", that *is,* two episodes referring to the same schema are not necessarily adjacent in the text (they might be separated by an episode referring to another schema). We will refer to this phenomenon as a temporary schema shift.

A typical scenario is as follows; The medical case is introduced by giving general information such as the date and the reason for admission to the hospital. Then the patient is presented (name, age,...). Symptoms (noted by the patient) and signs (observed by the physician) are described. A physical exam is usually performed and cultures are taken for which results are pending or available. In the latter case, they are given in detail. The structure of such a text can be captured in a sequence of schemas as shown on Figure 2.

```
[$SYMPTOMS
 (SYMP legalvals (CHILL HEADACHE PHOTOPHOBIA....)
       tobefilled T )
 (TEMP expect  (BETWEEN 94 108)
       whenfilled INFERFEVER)
          :
          :
 (STATE-OF-CONSCIOUSNESS expect (ALERT OBTUNDED..)
                         default ALERT)]
```

Figure 1: Part of a schema.

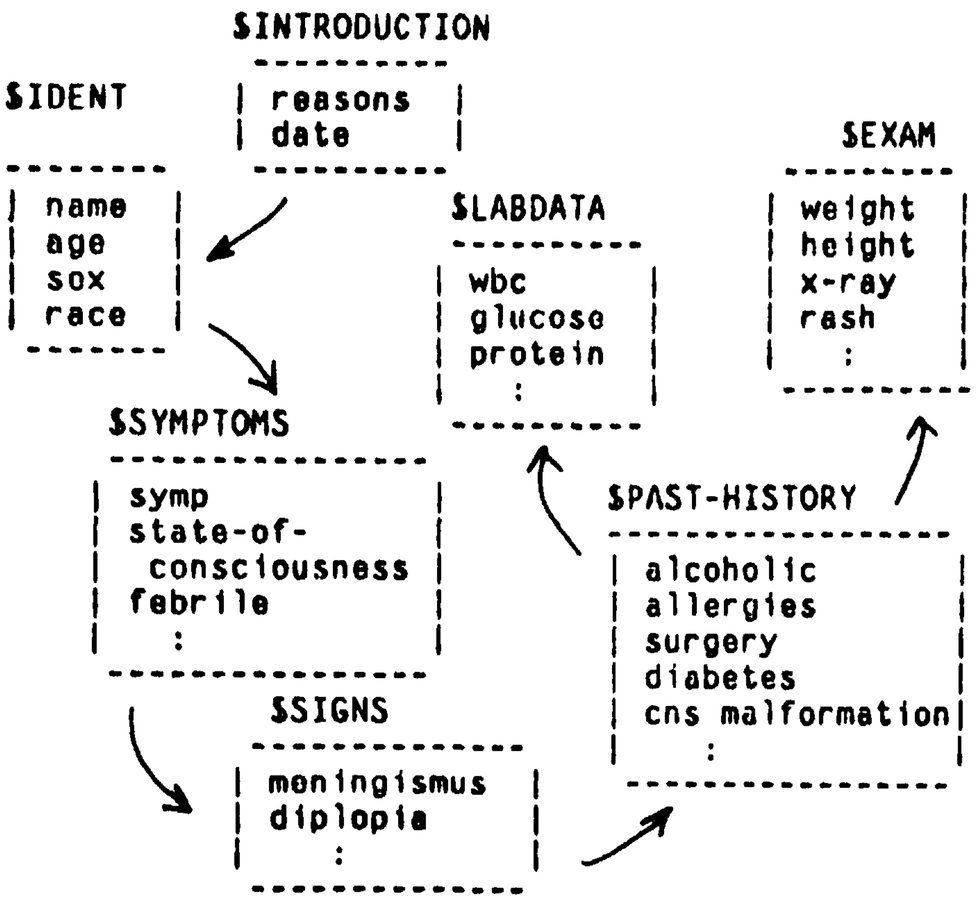


Figure 2: Set of schemas
(Arrows indicate sequencing preferences.)

4 Strategies to detect new episodes.

Top-down vs. bottom-up Sometimes the schema is explicitly announced, as in "Results of the culture". This is a name-driven invocation of the schema Mote often, the instantiation of the schema is content-driven The clues used are : the attributes associated with the schema, their expected values *(if* any), and other concepts that might suggest the frame; for example "skin" is related to "rash" which belongs to the *physical exam* schema. These are indeed very simple indices Research on more sophisticated methods for recognizing the relevant schema, such as discrimination nets, have been suggested by Charniak [4].

Suggest vs. confirm: Sidner [12] makes a distinction between potential and actual shifts of focus, pointing out that the cues suggesting a new frame must be confirmed by a subsequent statement in order to avoid making unnecessary shifts

We handle this phenomenon in a different fashion Instead of waiting for the suggestion to be confirmed, a qualitative distinction is made between the slots of a frame. The ones marked as suggesting but not confirming are regarded as weak clues and will not lead to a shift of focus, whereas the ones marked as confirming (hence suggesting) are sufficiently strong clues to command the shift. This distinction can be illustrated by the following two examples.

(1) The patient was found comatose. She was admitted to the hospital A lumbar puncture was performed. She denied syncope or diplopia...

(2) The patient was found comatose. He was admitted to the hospital. The protein from csf was 58 mg%.... (csf - cerebro spinal fluid)

In example 1, the lumbar puncture weakly suggests that "csf results" are available. In example 2, a detail of csf result (strong clue) is given directlv; in other words, the physician jumps into detail and the topic csf results" is directly confirmed.

Termination conditions and actions. A schema is usually terminated when all of its slots have been filled. This however *is* an ideal situation that does not occur very often More often, the intervention of a new schema implies the current schema becomes out of focus A "terminated-by" slot has been created to define which schemas can explicitly terminate others; for example, the $SYMPTOM schema usually closes the $IDENTIFICATION schema (name, age, sex, race), as it is very unlikely that the speaker will give the sex of the patient in the middle of the description of the symptoms. This is due to the highly-constrained nature of the domain When a schema is terminated, the program infers all the default values of the unfilled slots. It also checks whether the expectations set during the story have been fulfilled These actions can be performed only when a shift has been detected or at the end of the dialogue; otherwise, the program might ask too early about information that the user would indeed give later. In the case where a schema has been exhausted (all its slots filled), an a priori choice is made with regard to the next schema likely to appear. This is performed by using a *preferabh-followcd-by* pointer that, in the absence of a bottom-up (data-driven) trigger for the next schema, decides in a top-down fashion which one is the most probable at a given point.

Schema-grammar links. The grammar *used* to parse the text is divided into specific rules and non-specific rules. Specific grammar rules are associated with particular slots in the schemas and describe the way they could be mentioned at the surface level. Categories used in the rules are things like <patient>, <sign>, <diagnosis>. This link between the grammar and the schemas provides a means to try in priority those grammar rules that are appropriate to the schema in focus. The

notion of grammar rules in focus can be viewed as an extension of Grosz's [6] notion of focusing mentioned above. A schema shift thus leads to re-ordering the specific rules.

5 Conclusion,

The strategies outlined above could be applied to a broad range of structured texts. The approach rests on the assumption that their scenario can be seen as sequences of episodes identifiable by the program, in order to be integrated into appropriate schemas. Therefore, clustering attributes into frame-like structures must make sense in the domain of application The episodes could simultaneously refer to several schemas, that is, the schemas associated could have slots in common. Furthermore, it should be possible to define partial-ordering links between schemas. Therefore, the more structured the text is the less unexpected shifts are necessary, and the easier is the interpretation process

Expert systems usually need some kind of understanding to communicate in natural jargon with various users (expert, consultant, student). The technique described here (breaking clown the knowledge into schemas that should correspond to different pieces of texts, associating semantic grammar rules with the schemas. and using strategies for recognizing episode shifts) should be applicable in such domains.

G Example of a session

The physician types the input in upper case after the double asterisk, BAOBAB-2's responses are in lower case. Generation of what the program has understood is somewhat "heavy" for the main purpose is to be unambiguous. Some passages have been cut off for lack of space. A few comments have been inserted on the right for sake of illustration. Names of schemas start with a 8.

** REASONS FOR ADMISSION: CHILLS HEADACHE MALAISE
(sets up expectations)
Ok, I will oxpoct these symptoms
** DATE Of ADMISSION; DECEMBER 25 1978
My understanding 1s that:
the patient was admitted at the hospital 4 days ago
** PAT IS A 26 YEAR OLD CAUCASIAN FEMALE
My understanding 1s that:
the ago of the patient 1s 26 years
tho race of the patient 1s Caucasian
the sox of tho patient 1s female
the name of tho patient 1s Pat
** ONE DAY PRIOR TO ADMISSION SHE EXPERIENCED CHILLS AND HEADACHE
(shift to $ symptoms)
My understanding 1s that: *(time inferences)*
Chill 1s one of the symptoms that the patient had 5 (lays ago, headache 1s one of the symptoms that the patient had 5 days ago
** ONE DAY AF1ER SHE HAD A MALAISE
My understanding 1s that:
malaise 1s one of the symptoms that the patient had 4 days ago
** A LUMBAR PUNCTURE WAS PERFORMED
My understanding 1s that:
a lumbar puncture has been performed on the patient
** SOCIAL HISTORY: OCCASIONAL ALCOHOL USE
Do you suspect rocent alcoholic history 1n Pat?
(clarification)
** NO
My understanding 1s that:
tho patient Is not an alcoholic
** THE SKIN IS NORMAL
(leads to a clause via inference)
My understanding 1s that: the patient does not have a rash or cutaneous lesions
** NECK: MARKED NUCHAL RIGIDITY
My understanding 1s that:
MenIngismus 1s one of the signs that the patient showed *(clause via inference)*
** CSF RESULTS *(sets up a context)*
Ok.
** WBC 9500 *(understood in context)*
My understanding 1s that:
the WBC from the CSF 1s 9500
etc.
**WBC FROM CBC: 235
My understanding is that:
The white count from the patient's peripheral cbc (in thousands) 1s 235
** THE VALUE OF WBC IS 225
{corrects a mistyping)
Instead of 235?
** YES
My new understanding is that:
The white count from the patient's peripheral cbc (1n thousands) 1s 225
etc.

7 Acknowledgments

I wish to thank James Bennett, Barbara Crosz, Candy Sidner and Steven Rosenberg for their helpful comments on earlier drafts of this paper.

[i] Bartlett F„ Remembering, a study in experimental and social psychology, Cambridge University press, 1932.
[2] Bonnet A., Baobab, a parser for a rule-based system using a semantic grammar, Stanford Computer Science Dept, Report STAN-CS-78-668, September 1978
[3] Bonnet A , Schema-shift strategies for understanding structured texts in Natural Language, Stanford Technical report, 1979.
[4] Charniak E, With a spoon in hand, it must be the eating frame, Proc of Tinlap-2, Urbana-Champaign, Illinois, 1978.
[5] Collins A, Brown J. S., Larkin K., Inference in text understanding, in Spiro, Bruce and Brewer, Theoretical Issues in reading comprehension, Hillsdale, N. J Lawrence Erlbaum Associates, 19/8.
[6] Grosz B, The representation and use of focus in a system for understanding dialogs, Proc. of 5th Ijcai, Cambridge Mass, August 1977.
[7] Minsky M, A framework for representing knowledge, In P. Winston (ed.) The Psychology of Computer vision. New York McGraw-Hill, 1975.
[8] Sager N., Natural language information formatting, the automatic conversion of texts in a stuctured data base, Advances in computers vol.17, 1978.
[9] Schank R, Abelson R., Scripts, plans and knowledge, in Advance papers of the 4th IJCAI, Tbilisi, USSR, 1975, pp 151-158.
[10] Schank, R., Abelson, R., Scripts, plans, goals and understanding, an inquiry into human knowledge structures, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1977.
[II] Shortliffe E., MYCIN, a rule-based computer program for advising physicians regarding antimicrobial therapy selection, Ph D thesis, October 1974
[12) Sidner C, A computational model of co-reference comprehension in English, Ph D Thesis, MIT, Cambridge, Mass, 1979.

LEARNING OF ABSTRACTIONS FROM STRUCTURAL DESCRIPTIONS OF PICTURES

P. BONZON - University of Lausanne - Switzerland

ABSTRACT

The acquisition of concepts induced by structural descriptions of pictures is discussed and a representation scheme is presented which allows the construction of various abstractions based on different points of views and their storage in a simulated associative memory.

INTRODUCTION

In order to introduce the material contained in this paper, we will follow the views and terminology used by Hayes-Roth and McDermott in their recent work /1/, /2/ :

> "Concepts are pattern temptates : events which match a concept are recognized as belonging to the class delimited by that concept. (...) The machine learning problem with which we are concerned can be stated in the following way : given a collection of concepts (...) and a way of describing events in terms of their structure, construct a machine which is able to induce additional concepts (...) from training data. " 1/1/, p. 156)

From there, they identify three distinct problems J

1) the description problem, asking for a "formal scheme for describing complex events which facilitates the generation of abstractions"
2) the comparison problem, i.e. the problem of finding commonalities between two examples of the same concept or abstraction
3) the storage problem, i.e. the problem of storing discovered abstractions in such a way that tney could be used as temptates for recognition of events, and as knowledge representations to be improved by learning, learning being viewed here "as a continual process of knowledge expansion, that is, as the acquisition, in adaption to training experiences, of higher-order, more complex and more elaborate knowledge structures".

We propose in this paper a different solution to each of these problems, our approach being motivated by a rather divergent view on the modalities of knowledge expansion.

TWO POSSIBLE APPROACHES FOR KNOWLEDGE EXPANSION

The approach taken by Hayes-Roth and McDermott is, in their own wording, to improve the precision of the knowledge representations by learning, if new instances of the same concept are provided. More precisely, according to the example they give, they first construct an initial (actually too narrow) concept by listing all common properties of two selected events, and then refine it (actually extend it) by removing some commonalities not found in a third event, and so on : this amount to first retaining all constituents as potential properties of a concept and then gradually discarding some of them. The matching involved in recognition is then a partial matching.

On the other hand, the approach we have taken consists of isolating first basic constituents, and classifying them according to various points of views, thus defining basic concepts in different contexts; a global concept is then constructed by combining basic concepts into more elaborate, higher-order concepts, and so on, until gradually all constituents have been taken into account : the matching involved is thus a complete matching, dissimilarities being removed by means of equivalence classes, or subconcepts.

A consequence of our approach is that learning cannot be considered as an autonomous process, but must necessarily interfere with a complementary tutoring process, providing the frame for

the differentiation of contexts, classification of constituents and the naming of concepts. This will be first illustrated by means of a simple example of a teaching session conducted on our interactive graphic system.

Example :

In a first step, the teacher will be asked to draw simple connected patterns (i.e. basic constituents), and to provide the appropriate names for equivalence classes, thus defining basic concepts by extension:

pattern	concept name
	nose
	eye-mouth
	shape

In a second step, he can draw pictures with more complex structures (i.e. single or multiple events, or instances of higher-order concepts), together with concept names:

picture	higher-order concept name
	face

A definition by generalization of the corresponding concepts will be constructed by substituting basic concept names for occurences of simple connected patterns (first level of abstraction). To verify that these concepts have been learned, the teacher can then draw pictures, asking for recognition:

picture	response
	face
	face
	?? (unrecognized)

The learning process could go on in two different directions :

first by drawing more and more complex pictures, for example introducing body configurations with heads, arms and legs, leading to higher levels of abstraction

secondly by defining alternate basic concepts (i.e. different equivalence classes of the same patterns) for analyzing the same pictures, thus implicitely refering to another context or frame of intentionality and achieving various levels of differential abstraction

e.g.

pattern	concept name
	sad
	happy
	undefined

pictures	differential concept name
	happy face
	sad face

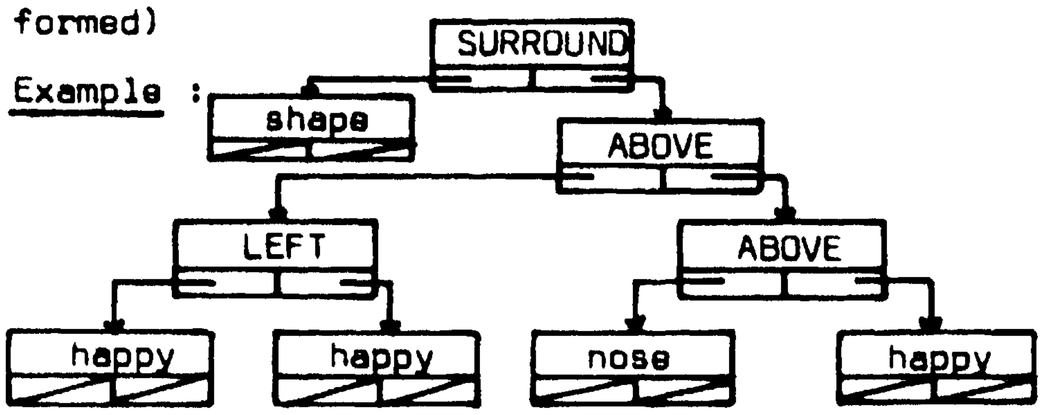
The solutions to the three problems mentionned above will be now reviewed in detail.

THE DESCRIPTION PROBLEM

The interface between the teacher (human operator) and the learning system (computer program) is a graphic tablet together with a display. A fairly complex software analyzes line drawings and identifies simple connected patterns such as those presented in the above examples. This primary capability of the learning system could be viewed as a pure sensorial perception. In order to obtain a satisfactory structural description, a few more primitive perceptual capabilities are required. Our present (rather incomplete) system allows for the determination of the relative position of simple connected patterns using the three binary relations ABOVE, LEFT, and SURROUND.

The structural description resulting from this perception is given in the form of a binary tree, the perceptual tree, where basic concept names have been substituted for any of their instances (i.e. a first level of abstraction has been performed)

Example :



picture perceptual tree

It is noteworthy that this description, in contrast to that in /1/, is not parameterized, the identification of each constituent being taken into account by its position in the binary tree.

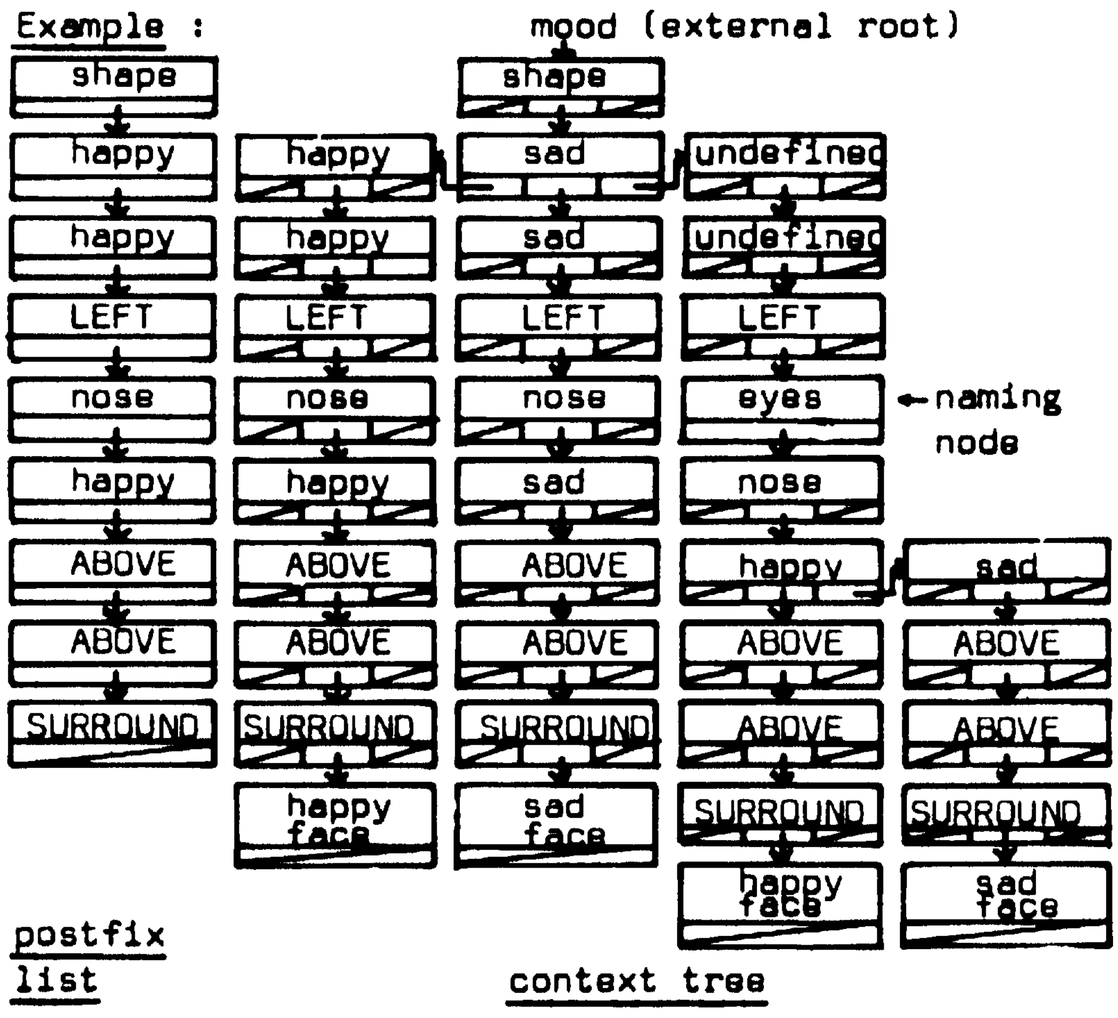
THE STORAGE PROBLEM

The solution to the storage problem, which in Hayes-Roth and McDermott terminology refers to the way of representing abstractions or concepts, will be presented next to allow for an easy

treatment of the comparison problem.

The purpose of classifying constituents Into multiple equivalence classes. Is to allow for the recognition of events in various contexts. A workable Knowledge representation can thus be obtained by a mere reshaping of perceptual trees, grouping multiple concept definitions based on the same point of view into a ternary tree, each such tree representing a context.

More precisely, each perceptual tree is first rewritten as a list in postfix order, and then imbedded in a binary search tree where leading common sublists are factorized (or shared) for economical memory storage and quick access. The appropriate concept name is then attached to the queue of each list in a so-called naming node. The resulting structure forms a context tree. Leading sublists can be used to define subconcepts by inserting naming nodes, which have then to be Jumped over.



THE COMPARISON PROBLEM

With the type of knowledge representation just described, finding commonalities and recognizing events becomes equivalent to retrieving, in context trees, concepts whose definitions match subtrees of perceptual trees.

More precisely, recognition of a event given by its perceptual tree will be conducted by successive reductions : each subtree (of a given perceptual tree) matching a concept's definition will be replaced by the name of that concept) if the perceptual tree eventually reduces to a unique naming node, a global concept will thus be identified.

Knowledge expansion, on the other hand, is achieved by the insertion in postfix order, into appropriate context trees, of reduced, but unrecognized, perceptual trees.

The reduction algorithm itself consists of three imbedded recursive procedures : the most inner procedure is the classical recursive version of binary search with insertion using a sentinel, modified to Jump over naming nodes; the second inner procedure traverses subtrees of perceptual trees in postfix order, each node being in turn searched for at a given level in a context tree: this amounts to attempting a partial match on subtrees) the outer procedure traverses perceptual trees in postfix order, attempting a partial match at each node, i.e. on each subtree.

EXPERIMENTS PERFORMED WITH THE SYSTEM

This system has been used primarly in an experiment involving the learning of Chinese characters /3/. The range of candidates (a subset of 70 characters based on 50 basic patterns have been selected) was imposed by the limited perceptual power of the system.
On a configuration involving a CDC CYBER 73 computer, a character abstraction or recognition takes from less to 0.1 to 0.5 sec depending on the character's complexity.

REFERENCES

/1/ Hayes-Roth, F., and McDermott, J., Knowledge Acquisition from Structural Descriptions, Proc. 5th IJCAI, 1 (1977), 356-382

/2/ Hayes-Roth, F., and McDermott, J., An Interference Matching Technique for Inducing Abstractions, CACM 21,5 (May 1978), 401-410

/3/ Tran Van Nam, Syntactic Pattern Recognition, A General System for Description of Handprinted Text Written on a Graphic Tablet, PH.D Thesis, Swiss Fed. Inst, of Techn, (EPFL Lausanne 1978)

FINDING THE AXIS OF AN EGG

Mike Brady,
department of Computer Science,
University of Essex,
Colchester, CO4 3SQ,
Essex, England.

Marr's *[2]* axial symmetry theorem is extended to cater for situations in which the expansion function has no points of inflexion.

1. INTRODUCTION

Marr's [2] "analysis of occluding contour" (henceforth AOC) argues that we simply cannot avoid interpreting silhouettes as being formed by "generalised cones", where, among the various definitions which have been offered for such surfaces, we refer to the relatively restrictive formulation given by Marr [2, P.467] That being the case, one faces the practical problem of extracting from a silhouette the cone's defining parameters: its straight line axis A, the twice continuously differentiable closed planar curve p(r, 8) which is the cross section function of the cone, and the twice continuously differentiable positive real-valued function h(z) which defines the expansion of the cross-section function p as it is moved along the axis A. Theorem 3 of *[2],*the "axial symmetry theorem" (henceforth AST), embodies the basic technique for extracting the image A* of the axis A. Unfortunately, one of the conditions of AST, necessary for the proof, is that h have at least one concavity. Since the end points of a generalised cone are clearly convex, this amounts to requiring at least one point of inflexion on h, which by lemma 1 of AOC is preserved by an orthogonal projection.

Many generalised cones have no points of inflexion in their expansion function. Fig. 1 shows some commonly occurring examples, (a) an ellipse (and in particular a circle), (b) an egg shape, and (c) a smoothed (right) cylinder. One clearly would not hesitate to assign axes to these figures, save perhaps for the circle. Of course, depending on the particular function h, the assignment of axes may be rather ambiguous (fig. 2), This issue will not be dealt with in this paper.

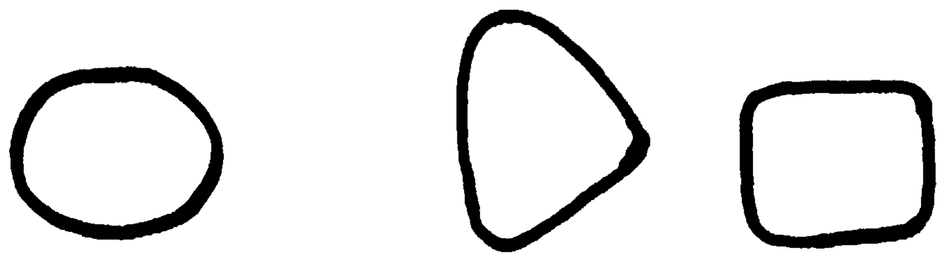
ellipse circle egg shape smoothed cylinder

$$\frac{x^2}{a^2} = \frac{y^2}{b^2} = 1 \qquad x^2 + y^2 = r^2$$

(a) (b) (c)

Generalised cones with no point of inflexion in their expansion function

Fig. 1



ellipse a surprising egg a "cubist" smoothed cylinder

$$\frac{x^2}{a^2} + \frac{y^2}{(0.99a)^2} = 1$$

(a) (b) (c)

Generalised cones whose axis assignment is ambiguous

Fig. 2

An examination of Fig. 1, and many similar examples, suggests a modified form of Marr's AST which relies upon finding zeroes and stationary

points of the curvature of h, and upon matching parts of the contour along which the curvature (as opposed to the value of h) is monotonic. To bring this idea to fruition we need to examine how the concavity clause is used in the proof of the AST.

2. A REFORMULATED AXIAL SYMMETRY THEOREM

The assumed concavity of h is used to guarantee at least two matchable segments of contour, and to force a unique choice of axis. It is noted that a concave (convex) segment of h gives rise to two concave (convex) segments of the contour on either side of the image A^* of the axis, possibly laterally displaced because of the eccentricity of the cone. The preservation of points of inflexion of h, where the curvature of h is zero, is guaranteed by lemma 1 of AOC and is used implicitly.

Suppose we replace "point of inflexion" by "a point where the curvature of h is zero or has a stationary value". Curvature is defined by

$$k = h_2/(1 + h_1^2)$$

and

$$k_1 = h_3(h_1^2 + 1) - 3h_1h_2^2)/(1 + h_1^2)^{3/2}$$

where $h_i = \partial^i h/\partial z^i$. Since curvature depends on the second derivative, its stationary points are only defined if the third derivative exists and its continuous. We now impose this requirement on h, which amounts to strengthening slightly the notion of a "nice" surface in AOC.

Marr's approach depends upon discovering which properties of a planar curve are preserved under a nonsingular linear transformation, in particular under orthogonal projection. Lemma 1 of [2] shows that points of inflexion of the expansion function h are so preserved, and it is straightforward to show that stationary points of curvature are too. Take any planar slice through the generalised cone containing the straight axis A. The definition of the generalised cone guarantees the existence of a continuous 1-1 onto function $H : R \to R^2$ which relates movement along A to the corresponding movement along the contour. Since L is linear, $(LH)_n = L(H_n)$ for all n. In particular, it is crucial for the proof of Lemma 1 of AOC, though not noted there, that $(LH)_n = 0$ iff $H_n = 0$.

Most of the proof of AST carries over also. The bounding contours are defined by $A^{(i)}(z, \theta_i) = h(z) * \rho(\theta_i)$ for $N \leq z \leq S$ and $i = 1, 2$. That is to say there are two contour generators $A^{(i)}$ between any pair of "poles" of the generalised cone. Note that $\partial A^{(i)}/\partial z = \rho(\theta_i) * h_1(z)$, which is the basis of the proof of AST. Moreover, $A_n^{(i)} = \rho(\theta_i) * h_n(z)$ so that a portion of h having negative (positive) curvature produces two such components flanking the image of the axis and preserves their sense. The remainder of the proof of AST carries over and we have

Axial symmetry theorem: Let $\sum = \rho \times h$ be a generalised cone with convex cross section ρ, viewed distantly from its associated viewing plane Π. Then

either the silhouette is circular and may be decomposed along any diameter

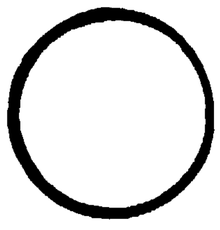
or
(1) the silhouette decomposes into segments by splitting it at points where the curvature is zero or is stationary
(2) the image of the axis establishes an axial symmetry between $\max(2, n - 2)$ contour segments in which the correspondence preserves the monotonicity of the curvature
(3) if c_{i1} and c_{i2} are corresponding segments and if $d(c_{ij})$ denotes the average perpendicular distance from d_{ij} to the image of the axis, then $d(c_{i1})/d(c_{i2})$ is independent of i.

3. REMARKS

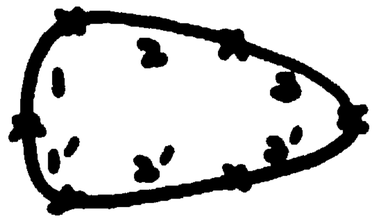
Case 1 of the re-formulated AST is to cater for the case where curvature is constant and non-zero. It is a happy feature of this re-formulation that it explicitly involves curvature, a surface property which plays a vital role in perception ([1], [3], [4]). Since points of inflexion are points where the curvature is zero there are always at least as many sectors generated by the reformulation as in the orginal AST, and there are usually several more besides. This means that there are more points for a program to match and so the extraction of the axis is likely to be more accurate, even in the case of joins, such as Marr [2, fig. 18].

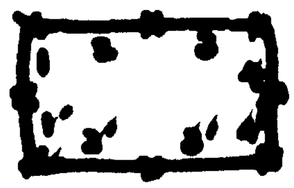
For completeness, the segmentation points and axes are given in Fig. 3 for the shapes in Fig. 1.

(note that matching 1 = 2, 1' = 2' yields the minor axis



any diameter





Segmentations points for the objects in Fig. 1

Fig. 3

ACKNOWLEDGEMENTS

The research reported here was completed while the author was a visiting scientist at the MIT Artificial Intelligence Laboratory, Thanks are due to the many fine scientists at the Laboratory who made the visit so memorable, particularly Patrick Winston and Berthold Horn. Especial thanks are due to David Marr for his help, advice and inspiring example. Thanks are also due to the American referee for his insightful comments which led to substantial improvements in presentation of this paper.

REFERENCES

[I] Attneave, F. "Some informational aspects of visual perception"$_t$ Psych. Review 61, (1954), 183-193.

([2] Marr, D. "Analysis of occluding contour", Proc. R. Soc. Lond. B. 197, (1977), 441-*7fT.* (See also MIT AI Memo 372).

[3] Ullman, S. "Filling the gaps: the shape of subjective contours and a model for their generation", Biol. Cybernetics 25, (1976), 1-6.

[4] Woodham, B. "Reflectance map techniques for analysing surface defects in metal castings", (Ph.D.), (1979), MIT AI-TR-457.

INFERRING THE DIRECTION OF THE SUN FROM INTENSITY VALUES ON A GENERALISED CONE

Mike Brady,
Department of Computer Science,
University of Essex,
Colchester, CO4 3SQ,
Essex, England.

An initial attempt to synthesise some of the ideas of Marr and Horn about surface geometry and lighting is described. In some situations the direction of the sun can be deduced from intensity values on a section of a generalised cone which flanks a bounding contour.

1. INTRODUCTION

A key problem in developing a computational theory of visual perception is to discover the constraints which hold between lighting, and the geometry and photometric characteristics of a surface, which together with the prior assumptions of a viewer enable the interpretation of a two dimensional image as depicting a three-dimensional scene. The work reported here arose from a study of the ever-present problem of determining occlusion relationships in a single image. A companion paper [I] argues firstly that the current stock of ideas about occlusion are extremely limited, and secondly that the best way forward currently seems to be to aim for a synthesis of the work of Marr [3] and Horn *[2]*. Marr [3] argues persuasively that a viewer has strong preconceptions regarding the three dimensional surface which generated a given silhouette in an image, and proves that under a number of precisely stated assumptions these preconceptions amount to requiring that the generating surface be a "generalised cone". Several accounts of generalised cones have appeared in print, and we adopt here the restricted version defined by Marr [3|. A generalised cone results from moving a twice continuously differentiable planar closed curve p(r, 6), called the cross-section function along a straight axis A, subject to expansion according to a twice continuously differentiable positive real-valued function h(z), [2, p.467 and fig. 5]. Horn [2] shows how surface geometry, lighting and the reflectance characteristics of a surface combine to produce intensity values, and points out that in some cases one can deduce surface geometry from intensity shading. Woodham [5] and Ullman (personal communication) have pointed out that although one can make surprisingly detailed assertions about the curvature of a surface from a small region in an image, it is hard to decide whether it is convex or concave. However, this ambiguity is usually resolved if one can see a portion of the bounding countour of an object. This suggests that a good way to begin synthesising the ideas of Marr and Horn is to see what can be deduced when the contour region of a generalised cone is illuminated. This paper describes an initial attempt to develop such an analysis, and shows that in some cases the direction of the sun in an image can be deduced from the intensities near the surface of a generalised cone.

2. THE BASIC APPROACH

Following Marr [2, p.467], "let V be a distant vantage point for the generalised cone Σ such that (i) the image formed from V is an orthogonal projection, and (ii) the rays in the projection all be parallel to the plane of the cross-section of Σ". V is associated with a specific angle ϕ in the polar coordinate system of ρ. Let $\underline{v}$ be a unit vector in the direction of V. The contour generator Γ_θ is the set of points $\{(\rho(\theta) \cdot h(z), \theta, z) | z\}$ and lies in the same section plane Ω_θ as Λ. The normal $\underline{n}$ to the surface Σ at a point on Γ_θ can be separated into two components, namely its projection $\underline{n}_\theta$ onto the section Ω_θ, and its projection $\underline{n}_\rho$ onto the cross-section ρ. Since one can decouple the expansion function h and cross-section function ρ for a generalised cone $\Sigma = h \times \rho$, differentiating along Γ_θ leaves $\underline{n}_\rho$ constant, so that $\Delta\underline{n}$ is equal to $\Delta\underline{n}_\theta$. We shall be interested in

pixels close to the specific contour generator Γ_ϕ, and it is reasonable to suppose for a sufficiently fine spatial image resolution that they lie on a contour generator Γ_θ where θ is close to ϕ.

Let an arbitrary vector direction $\underline{\ell}$ be given Later $\underline{\ell}$ will be specialised to point towards the single distant light source (the sun). Suppose we denote the scalar product $\underline{\ell} \cdot \underline{n}$ by X. Consider a short displacement along Γ_ϕ, say Δs, so that the surface normal changes from $\underline{n}_1$ to $\underline{n}_2$.

$$\Delta X = \underline{\ell} \cdot \underline{n}_1 - \underline{\ell} \cdot n_2$$

$$= \underline{\ell} \cdot \Delta\underline{n}$$

$$= \underline{\ell} \cdot \Delta\underline{n}_\phi$$

since we are differentiating along Γ_ϕ. If we consider a Horn-like model of illumination, X can be the intensity Int = $\Phi(I, E, G)$ [3]. In this way, we hope to extract $\underline{\ell}$, the sun position, from ΔInt/Δs.

3. COORDINATES

In order to flesh out the above proposal, we need to be aware that there is a problem regarding the coordinate systems we use. The "natural" coordinate frame for a generalised cone is one that is local to it: cylindrical coordinates z along the axis, and the girdle angle ϕ about the axis Λ relative to some dorsal zero. The "natural" coordinate frame for a vector field, to facilitate the computation of scalar products, is a global frame, such as a Cartesian system. In fact, Marr and Nishihara [4] faced precisely this difficulty and their methods propose a solution in the form of specifying a space arrow vector ("$spasar") relative to a given axis ("$axis"). In fact, their methods apply to specify a general vector relative to the axis Λ of the generalised cone $\sum$. Given an origin on Λ, they suggest encoding a vector of magnitude s at a point Q on the surface of $\sum$ using six parameters. The first three parameters (z, r, θ) specify the point Q, while the other three (α, β, s) define the vector. Here α is the inclination of the vector to Λ and β the girdle angle of the vector about Λ.

In the cases considered here, Q will always be the position on Γ_ϕ at which the normal is computed, and it will be supressed. Moreover, we are only interested in vector directions and so we dispense with the norm s. Thus two parameters (α, β) are required to specify a vector. We need to compute the scalar product of two such vectors, and the following easily proved lemma achieves this.

Lemma 1 The scalar product of (α_1, β_1) and (α_2, β_2) is given by

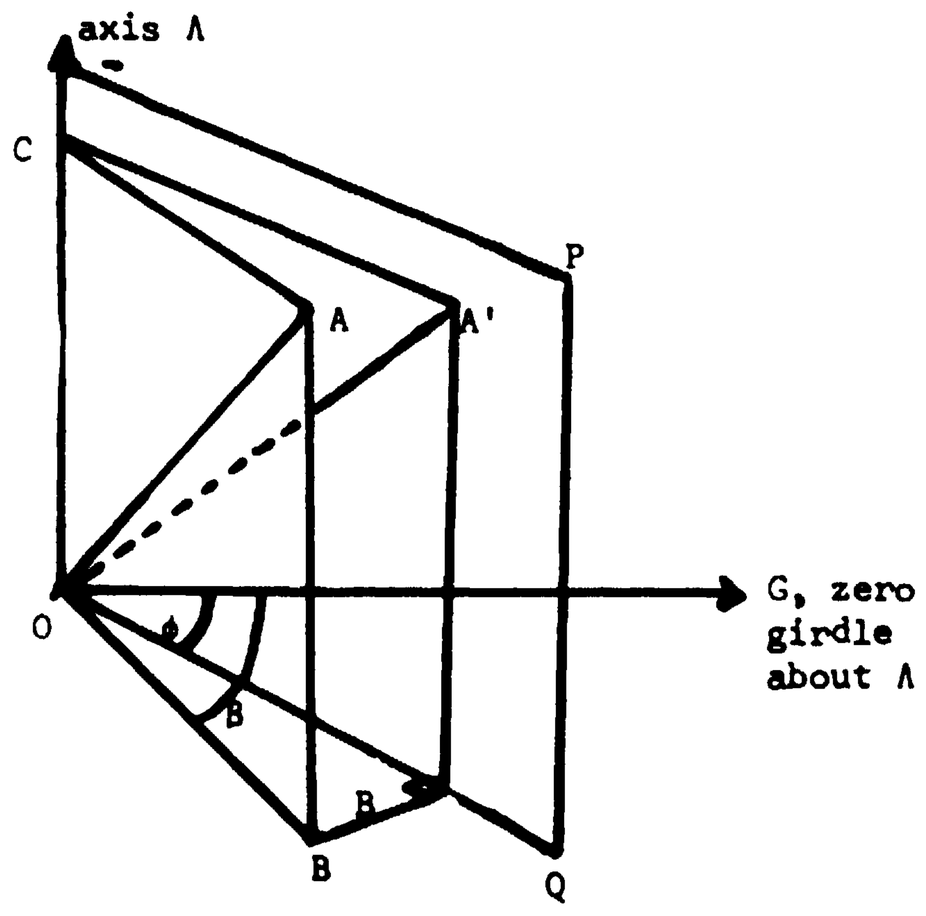
$\cos\alpha_1 \cos\alpha_2 + \sin\alpha_1 \sin\alpha_2 \cos(\beta_1 - \beta_2)$.

Suppose we have a vector (α, β). We shall need to consider its projection onto the section plane Ω_ϕ containing the axis Λ and a vector with girdle angle ϕ.

Lemma 2 The projection on (α, β) on Ω_ϕ has girdle angle ϕ and inclination μ where

$$\tan\mu = \tan\alpha \cos(\beta - \phi)$$

Proof: (refer to Fig. 1).



$B\hat{O}G = \beta$, $A\hat{O}C = \alpha$, $Q\hat{O}G = \phi$.

The projection of $\vec{OA}$ onto the plane OPQ is defined by keeping $OC = |OA|\cos\alpha$ and projecting AB onto A'B' in POQ. Now $|OB'| = |OB|\cos(\beta - \phi)$
$= |OA|\sin\alpha \cos(\beta - \phi)$
Thus the inclination of $\vec{OA'}$ is
$\arctan OB' \div OC = \tan\alpha \cos(\beta - \phi)$

Fig. 1

4. THE NORMALS OF A GENERALISED CONE AND THE MAIN RESULT

Consider the section Ω_ϕ of a generalised cone

(Fig. 2). The contour generator Γ_ϕ is $\{(z, h(z) * \rho(\phi) * \sin\psi_\phi) | z \text{ on axis}\}$ where ψ_ϕ is the eccentricity of the cross radial ρ in the section Ω_ϕ. Clearly, the direction of the tangent is given by

$$\tan\alpha = h'(z) * \rho(\phi) * \sin\psi_\phi$$

so the tangent vector is (α, ϕ). The normal $\underline{n}_\phi$ is given by $(90 - \alpha, \phi)$.

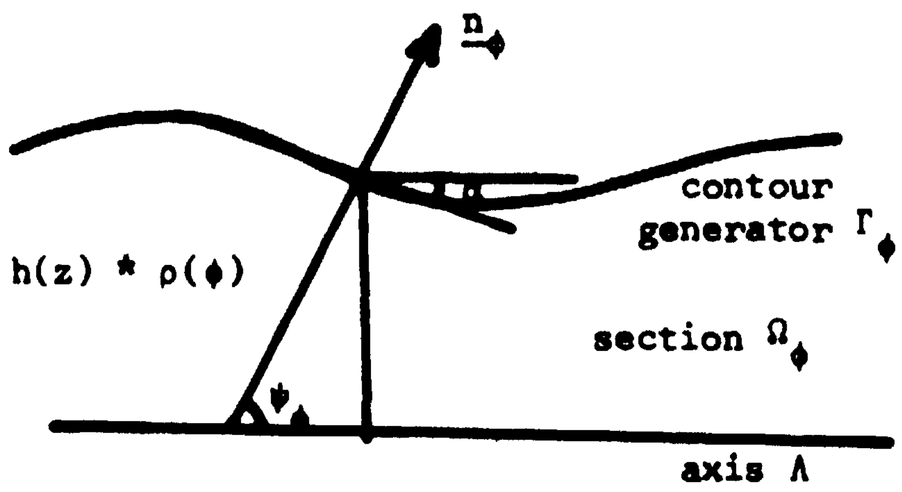


Fig. 4

Consider now an arbitrary vector (λ, μ) and define the scalar product $X = (\lambda, \mu).(90 - \alpha, \phi)$. At a neighbouring point of the contour generator Γ_ϕ, $\alpha' = \alpha + \Delta\alpha$. As in section 2

$$\frac{\Delta X}{\Delta s} = \frac{1}{\Delta s}\{(90 - \alpha, \phi).(\lambda, \mu) - (90 - \alpha', \phi).(\lambda, \mu)\}$$

where s is taken along Γ_ϕ. By lemma 1,

$$\Delta X/\Delta s = \frac{1}{\Delta s}\{\cos(90 - \alpha)\cos\lambda + \sin(90 - \alpha)\sin\lambda\cos(\phi - \mu) - \cos(90 - \alpha')\cos\lambda - \sin(90 - \alpha')\sin\lambda\cos(\phi - \mu)\}$$

$$= \frac{1}{\Delta s}\{\cos\lambda(\sin\alpha - \sin\alpha') - \sin\lambda\cos(\phi - \mu)(\cos\alpha - \cos\alpha')\}$$

recall that

$$\sin\alpha' \simeq \sin\alpha + \cos\alpha\,\Delta\alpha$$

and $\cos\alpha' \simeq \cos\alpha - \sin\alpha\,\Delta\alpha$. Thus

$$\frac{\Delta X}{\Delta s} = \frac{\Delta\alpha}{\Delta s}(\cos\lambda\cos\alpha - \sin\lambda\sin\alpha\cos(\phi - \mu)).$$

Now $\Delta\alpha/\Delta s$ is the curvature of Γ_ϕ and can be measured in the silhouette. We denote it by $\text{curvat}_\phi(z)$. Thus

$$\frac{\Delta X}{\Delta s} = \text{curvat}_\phi(z) * (\cos\lambda\cos\alpha - \sin\lambda\sin\alpha\cos(\phi - \mu)) \qquad (1)$$

We now specialise the vector (λ, μ) used in the above derivation to be the sun vector (λ_s, μ_s). Clearly, there is no change in the phase angle G along Γ_ϕ and so following Horn [2] we have

$$\frac{\partial\text{Intensity}}{\partial s} = \frac{\partial\phi(I, E)}{\partial s} = \frac{\partial\phi(I, E)}{\partial I}, \frac{\partial I}{\partial s} + \frac{\partial\phi(I, E)}{\partial E} \cdot \frac{\partial E}{\partial s}.$$

We can now apply formula (1) deduced above to compute $\frac{\partial I}{\partial s}$ using the sun vector (λ_s, μ_s) and $\partial E/\partial s$ using the view vector $\underline{v}$. This doesn't exploit the fact that the contour Γ_ϕ along which we are differentiating is a bounding contour, however.

There are two ways to incorporate this extra constraint.

1. Take the general formula for the surface normal $\underline{n}$ and use the fact that $\underline{n} \cdot \underline{v}$ is zero, since $\underline{n}$ is orthogonal to $\underline{v}$. This has the disadvantage that the formula defining $\underline{n}$ is rather complex. It can be cleaned up by restricting $\underline{v}$ to line in a plane parallel to ρ as in Marr's [3] first three theorems; but this misses the essence of the idea that it is largely irrelevant as $\Delta\underline{n} = \Delta\underline{n}_\phi$ along Γ_ϕ.

2. Build the assumption $e = \pi/2$ (i.e., E = zero) directly into the function Φ, so that Φ is a function only of I, G. We then get $\Phi(I, \emptyset, G) = R(I, G)$, so that

$$\frac{\partial\text{Intensity}}{\partial s} = \frac{\partial R}{\partial I} * \text{curvat}_\phi(z) * \left[\cos\alpha\cos\lambda_s - \sin\alpha\sin\lambda_s\cos(\phi - \mu)\right]$$

In particular, at a stationary point of intensity along Γ_ϕ, $\partial\text{Intensity}/\partial s$ is zero. If we assume that neither $\text{curvat}_\phi(z)$ nor $\partial R/\partial I$ are zero, we get

$$\cos\alpha\cos\lambda_s = \sin\alpha\sin\lambda_s\cos(\phi - \mu_s)$$

so that $\cot\alpha = \tan\lambda_s\cos(\phi - \mu_s)$

which by lemma 2 is the projection of (λ_s, μ_s) onto the sector containing (α, ϕ), that is, the image plane.

Hence the direction of the sun can be deduced from a stationary point of intensities taken along a section which flanks a bounding contour. Future work will develop the idea that the assumption of a Marr generalised cone allows one to make useful inferences from shading.

ACKNOWLEDGEMENTS

The research reported here was completed while the author was a visiting scientist at the MIT Artificial Intelligence Laboratory. Thanks are due to the many workers at the Laboratory who made the visit so memorable, especially Patrick Winston, David Marr and Berthold Horn. The American referee for this paper made a number of valuable suggestions for improving its presentation.

REFERENCES

[I] Brady, J.M. "Occlusion seems to be hiding" (in preparation) 1979.

[2] Horn, B.K.P. "Understanding Image Intensities", Artificial Intelligence 8, (1978).

[3] Marr, D. "Analysis of occluding contour", Proc. Roy. Soc. Lond. B 197, 1977, pp.44]-

(V| Marr, D. and Nishihara, H.K. "Spatial disposition of axes in a generalised cylinder representation of objects that do not encompass the viewer", MIT AI memo 341 1975.

[5] Woodham, B. "Reflectance map techniques for analysing surface defects in metal castings", (Ph.D) MIT TR-H57, 1979.

TESTING CORRECTNESS OF STRATEGIES IN GAME-PLAYING PROGRAMS

Max A. Bramer
Mathematics Faculty,
The Open University
Milton Keynes,
United Kingdom

This paper considers the distinction between winning strategies in game-playing programs which are either 'optimal' or 'correct[1]'. The relative merits of these two types of strategy are considered and methods are proposed for producing correct algorithms by a process of iterative refinement based on an analysis of 'win-trees'. An example is given of the development of a fully correct strategy for the King and Rook against King chess endgame.

1. Correct and optimal strategies

A problem which has emerged from recent studies of endgames in chess (i.e. subgames with only a small number of pieces remaining) concerns the relative importance of constructing strategies which are either 'optimal' (the stronger side plays perfectly in every position) or 'correct' (the stronger side wins eventually from every theoretically won position, but not invariably in the smallest possible number of moves).

The major advantages of aiming for optimal algorithms are that they can readily be verified automatically given a database of the best move in every position [1], and that they are intrinsically stronger than correct ones. However, the development of such algorithms is open to the theoretical objection that consistently optimal play is probably beyond the ability of even the strongest human players, and the practical objection that it is not feasible to construct either the programs or the databases against which to verify them for any but the most simple endgames.

If it is agreed that the development of correct algorithms is a more appropriate aim, there are two important questions which must be answered and these are addressed in the remainder of this paper:

(i) how can the correctness of an algorithm be tested, either automatically or otherwise?

(ii) how can a 'nearly correct' algorithm be refined to perform progressively better, based on an examination of its non-winning play?

Judging the optimality of any given move is inevitably an error-prone activity, even for experts. Judging the correctness of a strategy might seem to be easier, but it is in fact harder, as a result of the fact that a move can be both optimal and non-correct. Correctness refers essentially to sequences of moves, whereas optimality refers to individual positions, independently of one another. The contaminating effect of even a small number of poor (although possibly win-preserving) individual moves can result in non-correct play from many thousands of positions, the great majority of which will pass unnoticed even in extensive expert testing.

2. Testing the correctness of an algorithm

Modifying the basic method of generating databases of shortest-path winning moves for every position in a given endgame [1] gives an automatic method of testing the correctness of a playing algorithm for a chosen side (say White). The revised method generates an implicit win-tree containing all the positions from which the playing program invariably wins for White, although not necessarily in the smallest possible number of moves. The tree is generated by backtracking from terminal positions which are all wins for White, assuming that Black always plays to defer a loss for as long as possible.

The generation method can be summarized as follows (using WTM and BTM to denote White and Black to move, respectively).

(i) Mark all positions as either a terminal win for White, illegal or 'unevaluated'.

(ii) For all unevaluated BTM positions, compute and store the number of legal moves available. (If zero, Black must be stalemated, so mark as a draw.)

(iii) Backtrack one ply, by generating all legal antecedents (WTM) of the terminal positions. Mark those previously unevaluated

(with WTM) as wins in one ply, provided that the program would have selected the move which was reversed.

(iv) Backtrack from those positions evaluated at the previous level by generating all their legal antecedents (BTM). For those previously unevaluated (with BTM), decrease the counter of legal moves available by one. Mark any position for which the counter is reduced to zero as a win for White at depth two ply.

(v) Continue backing-up, from the positions evaluated at the previous level only, to White wins in 3,4,5,... ply in this way, distinguishing between backtracking to WTM and to BTM positions as in (iii) and (iv) above. This procedure ensures that every winnable position is included in the tree and gives the smallest number of ply needed for White to win against any play by Black,

(vi) Terminate when a level is reached at which no new evaluation takes place. The remaining unevaluated positions are not wins for White with the given playing algorithm. If all theoretically won positions are in the tree at this stage then the algorithm is correct.

3. Examining the win-tree

"he number of theoretically won positions is known [or certain endgames. However an examination of :he win-tree can be used to determine whether a playing algorithm is correct, without this know-Ledge, by the following procedure.

(a) Examine the BTM positions in the tree at each of the ply depths 0,2,4,.... in turn.

(b) At each level generate all the legal WTM antecedents of the BTM positions examined. If all these antecedents are in the tree (at any level) go on to the next level, otherwise note any WTM positions not in the tree and stop. These positions comprise the error set for the playing algorithm.

(c) The playing algorithm is correct if and only if the highest BTM level is eventually reached (with no legal antecedents found that are not in the tree).

If the algorithm is not correct, then the WTM positions in the error set are those at the Lowest level of the tree where White had the Dpportunity to play a winning move but did not Lake it or play any other move which led to a win. Changing White's algorithm to place each such position in the win-tree (by adjusting the nove played to give that BTM successor from which Lhe position was backtracked at stage (b))will generally cause many other positions (both WTM and BTM) also to be included in the tree by virtue of the normal backtracking process of tree generation. Proceeding in this way iteratively will lead to a fully correct algorithm in a finite number of steps. (Since the refinement process has no effect on the positions in the tree at the BTM level from which the previous error set was generated by backtracking, or at any lower levels, the next time the tree examination algorithm is applied, it is only necessary to begin with positions at the next highest BTM level, and so on.)

Changing White's moves at a relatively low level of the tree (i.e. in the 'error set' positions, rather than some other arbitrary set of-positions) is likely to introduce the greatest number of new positions during the subsequent backtracking and thus to lead to a relatively short sequence of iterations before a correct strategy is reached. Moreover, the total number of positions for which White's move is modified (equivalent to introducing "special cases" in the original algorithm) will be substantially less than the number of incorrect positions which occurred with the original version of the playing algorithm.

A semi-automatic alternative to the above approach is to make use of an analysis of the error set (or the union of two or three error sets produced by the 'automatic' refinement process) to suggest modifications to the playing algorithm within its original framework (of patterns, say), on a heuristic basis, with the complete win-tree recomputed at each stage of the iteration.

It is reasonable to suppose that most or all of the positions in an error set will have some feature in common to serve as the basis for a change to the algorithm, for example, the required White move in many positions in the error set will often lead to the same BTM successor. When well chosen, changes to the program's pattern-knowledge are likely to have greater beneficial effect and lead to more rapid convergence than those to individual positions. Naturally, however, if the changes are badly chosen there is no longer any guarantee that the process will terminate.

4. Refinement of a correct algorithm for King and Rook against King (KRK)

Both the automatic and the semi-automatic refinement processes described above were applied to an existing algorithm for KRK, which was thought to be correct, although non-optimal, as a result of extensive testing against human opponents. The algorithm made use of a ranking

of all legal BTM positions for that end-game based principally on a classification of positions into fourteen simple patterns, to find moves for White (the side with the Rook) only. Further details are given in [2]. Surprisingly, the initial version of the win-tree contained only 14,978 WTM and 7,654 BTM positions out of totals of 27,352 and 34,968 legal positions respectively (in a suitable standard orientation). Since it is known that all WTM positions and all but 3,495 (drawn) BTM positions are theoretical wins for White, it follows that in as many as 45% (WTM) and 68% (BTM) of all legal positions the algorithm failed to win when it could.

The first attempt at refining the algorithm was by means of the fully automatic process described previously. The following table summarizes the first six iterations. At all stages of the refinement process the maximum depth of positions in the win-tree remained as 52 ply, compared with the theoretical maximum required for an optimal algorithm of 32 ply.

	WTM moves adjusted		Total added to win-tree		No. of incorrect positions remaining	
Iteration	No.	Depth (ply)	WTM	BTM	WTM	BTM
1	1	9	2,339	3,398	10,035	20,421
2	8	13	1,956	2,466	8,079	17,955
3	89	15	3,000	5,184	5,079	12,771
4	176	17	1,188	2,124	3,891	10,647
5	269	19	2,542	6,138	1,349	4,509
6	190	21	621	1,882	728	2,627
Total	733	-	11,646	21,192	-	-

Although the iterations were not followed through to a final correct algorithm, the table above gives a good indication of the contaminating effect of a relatively small number of wrongly played positions.

The semi-automated approach to modifying the original algorithm began by examining the cause of incorrectness in the 9 positions in the first two error sets identified above. The first change made simply involved a restriction on the highly-valued pattern "King two files apart, with the Rook on the file between them", which led to a dramatic reduction in the number of incorrect positions from 10,035 to 3,291 (WTM) and from 20,421 to 9,513 (BTM). Subsequent minor modifications to a second pattern, again suggested by examination of error sets, gave a final algorithm which plays correctly in all positions, with a maximum depth of 60 ply. The program plays optimally in 18,386 out of the 27,352 legal positions for KRK (67%), and requires only two ply more than the theoretical minimum in a further 4,694 positions. Further details of this experimentation are given in an extended version of this paper [3].

To summarize, the method of win-tree examination can be used not only as a means of testing the correctness of an algorithm without any knowledge of the total number of theoretically won positions, but also to enable a fully correct algorithm to be refined, either automatically or by semi-automatic means, taking account of the overall framework of patterns etc., of an original algorithm. Using these methods would therefore seem to make the development of correct as opposed to optimal algorithms a feasible proposition in future research.

Although the above discussion has been presented solely in terms of simple endgames in chess, it is equally applicable to other problems which can be represented as games of perfect information with two players who move in turn, such that the result is either a win for one player (and a loss for the other) or a draw.

REFERENCES

[1] Bramer, M.A. Computer-generated databases for the endgame in chess. Mathematics Faculty Technical Report, The Open University, Milton Keynes, U.K., 1978.

[2] Bramer, M.A. and Clarke, M.R.B. A model for the representation of pattern-knowledge for the endgame in chess. Forthcoming in Int. J. Man-Machine Studies, September 1979.

[3] Bramer, M.A. Correct and optimal strategies in game-playing programs. Mathematics Faculty Technical Report, The Open University, Milton Keynes, U.K., December, 1978.

Implementing Search Heuristics Using the AL1 Advice-Taking System

Ivan Bratko
Faculty of Electrical Engineering and
J. Stefan Institute, University of Ljubljana
61000 Ljubljana* Yugoslavia

The AL1 System (Advice Language 1) was developed *as* a vehicle for conveying expert knowledge* or advice* to a problem solver. Using chess end-games *as* an experimental problem domain* this paper concentrates on the question: How to implement chess heuristics using the mechanisms of AL1? The spirit of advice programming *as a* sort of "tree-search engineering" is illustrated by developing an AL1 piece-of-advice for solving an example problem from the king-knight vs. king-rook chess ending. Experiments indicate that an advice program for this ending* based on comparatively simple concepts* and the human chess master perform comparably.

t. INTRODUCTION

Two main themes motivated the development of the AL1 system (Advice Language 1 T3,6]): (1) To obtain a linguistic vehicle for conveying expert knowledge* or advice* to a problem solver* and (2) to facilitate formal proving of the correctness of problem solving strategies defined by this knowledge. The second of the above two aspects is considered in [1] while here' using chess end-games as an experimental problem domain* we concentrate on a question of "advice programming": How to implement chess heuristics using the mechanisms of AL1? AL1 as a language for describing strategies is an assertional language! as opposed to procedural languages. Characteristic principles and mechanisms employed in AL1 are:

- the division of problem domain into subdomains (separate "advice tables" handle separate subdomains)l
- production rules* goals* subgoals (goals to be achieved* or maintained* as prescribed by a "piece-of-adv ice")5
- constraints on operators application ("move-constraints" to eliminate meaningless moves from consideration in advance).

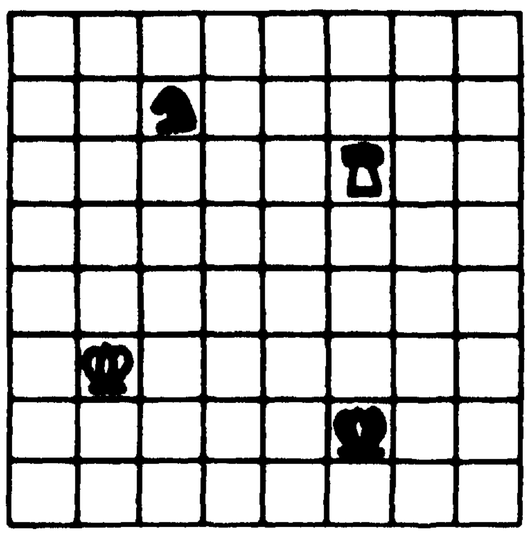
The approach used in our experiments is based on the use of benchmark problems and their solutions for the development of an ALI-like representation of appropriate heuristics. The use of an example problem typically results in the specification of advice* sufficient to solve the problem. The specification of such advice should be mainly based on the concepts already known! new concepts *are* to be introduced only if practically necessary.

2. TREE-SEARCH ENGINEERING IN AL1: AN EXAMPLE

In this section we illustrate the spirit of "advice programming" by an example problem* i.e. a king-knight- king-rook chess position* together with its solution. As shown by Kopec and Niblett C correct play in the KNKR ending is much harder than commonly believed even by che master's standards. The weaker side* i.e. the knight's side* has to fight for getting* or keeping* its pieces together in crder to prevent the opponent's fundamental long-term threat which is* force the king and the knight to separate further and* when the knight is definitely cut-off from its king* attack the knight with both pieces and capture i

Fig. 1 is a good example of what can happen. The natural move is Ke3 *as* it brings the Black king towards the knight and centralizes the king. But Kg3 is surprisingly the only move that *saves* the position. The tree in Fig. 1 illustrates a refutation strategy for Ke3. The depth of this refutation-tree is 20 ply* i.e. the depth at which the knight gets captured. How can we formulate advice in AL1 so that we avoid the 20 ply deep search which is prohibitive.

Two basic mechanisms control the starch in ALi: better-goals and holding-goals* both being predicates on board positions. The result of game-tree search in AL1 is a "forcing-tree" (a concept introduced by Huberman [4]) which represents a detailed strategy for the achievement of better-goals while preserving holding-goals. More precisely? a foreing-tree is a subtree of the game-tree rooted in a current board position. All nodes of the forcing-treei except the root node* satisfy a given holding-goal: all terminal nodes satisfy a given better-goal, and no other node satisfies the better-goal. For each nonterminal "them"-to-move node a forcing-tree contains all legal successor positionsi for each nonterminal "us"-to-move node a forcing-tree contains exactly one successor position. Thus the concept of forcing-tree corresponds to the concept of solution-tree in ANO/OR trees Ce.g. *7]*.



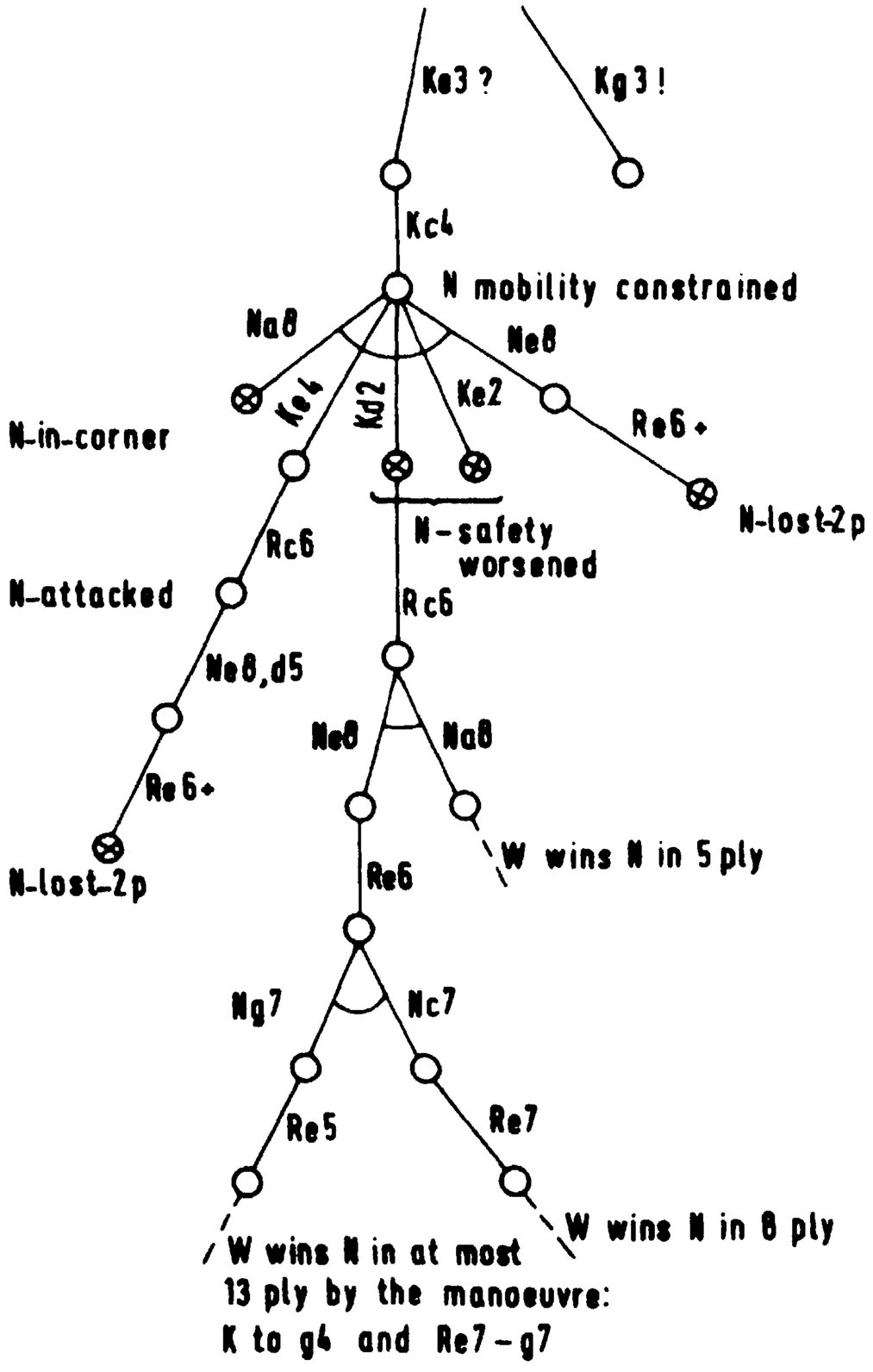


Fig. 1: A refutation-tree for Ke3? . Some obvious branches are omitted.

The following simple concepts suffice to construct an advice which comparatively efficiently solves the problem of Fig. 1'

- our piece attacked ("our" is the weaker side's)?
- "knight-safety" which wilt be for the present purpose defined *s*
  N-safety = distance between their king and knight minus distance between our king and knight (distance is measured in king-moves)!
- N-lost-2ps we to move cannot prevent the loss of the knight in the next two plies?
- N-mobility? the number of safe squares attainable by the knight in one move?
- N-in-corner: the knight in a corner of the board.

A piece-of-advice that solves our problem of Fig. 1 can be stated *as* follows:
Consider the legal moves in the current position in order of decreasing N-safety. Play the first one of those moves which guarantees that for the next 6 plies:

(1) we do not get mated*
and* unless we can capture their rook*

(2) we do not lose the knight*

(3) after the first us-move: N-safety may deteriorate with respect to the current position only if it can be restored on the next move*

(A) the knight is never placed in a corner of the board.

For the sake of the search efficiency we add a better-goal which prescribes the investigation of "dangerous" opponent's moves only: attacking and knight-constraining moves.

The translation of such a piece-of-advice into AL1 is straightforward. In an actual implementationi the advice considered above was slightly generalized in order to broaden its applicability.

An advice program for the KNKR ending [2] was developed along the above illustrated lines. The program contains 16 pieces-of-advice which are different combinations of the concepts, considered in the previous example, or similar ones.

Experimental testing of the program [2] and its behavioral comparison with the human chess master convincingly indicate that the program and the human master perform comparably. In fact, the program's performance was slightly better than the master's in these tests. When applying a piece-of-advice* the program never searched deeper than 6 ply* and typically examined a few hundred of nodes, up to about one thousand. The effective branching factor was thus between 2 and 3.

3. CONCLUDING REMARKS

Of interest are the following observations«

(1) Advice, constructed for solving some particular example problem, was usually applicable to many other problems if only some effort was made to state the concepts used in the advice in a general form. An example is in Fig. 2.

(2) Experience obtained in these experments indicates that, when constructing such knowledge based programsi it is very important to keep the structure of the knowledge neat. This also facilitates formal correctness proofs for strategies, as investigated in [1].

(3) Example of Fig. 1 shows how a*n* automatic facility for learning from examples could be added to the AL1 system in a straightforward manner. In this context* the task of learning from an example can be viewed as one of searching for a compact specification which extracts a proper forcing-tree from the game-tree. This facility has not been programmed! however* part of the KNKR advice program was efficiently constructed by carrying out this process manually C33.

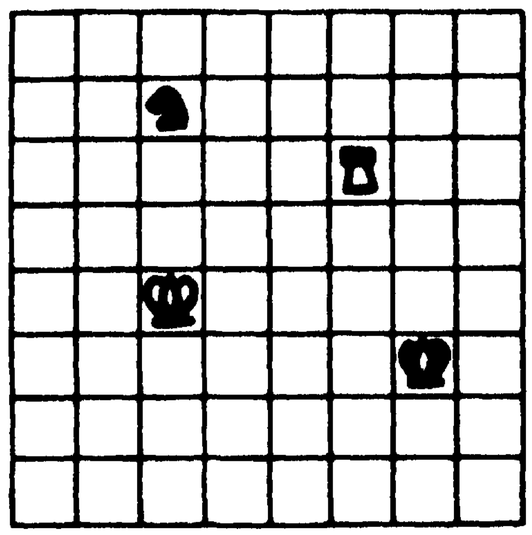


Fig. 2: A position that can occur from the one in Fig. 1, Black to move. The natural 1 ... Kg4 fails to 2 Rc6 Ne8 3 Re6 Nc7 4 Re7 Na8 (or Na6) 5 Rb7 and the knight is stalemated and lost in a few moves. The same advice as for the position in Fig. 1 finds here the only* study-like defence! 1 ... Kh4f .

ACKNOWLEDGEMENT

The author would like to thank professor D.Michie for his continous efforts in Advice Language research.

REFERENCES

[1] Bratko I. (1978) Proving correctness of strategies in the AL1 assertional language* Information Processing Letters* Vol. 7* No. 5* 223-230.

[2] Bratko I. (1979) An advice program for the king- knight vs. king-rook ending* MIP-R-123* University of Edinburgh! Machine Intelligence Research Unit.

[3] Bratko I.* Michie D. (1979) A representation for pattern- knowledge in chess end-games* in Advances in Computer Chess 2 (ed. M.R.B.Clarke)* Edinburgh University Press (in press).

[4] Muberman B.J. (1968) A program to play chess end-games, Technical report no. CS106* Stanford University: Computer Science Department.

[5] Kopec D.. Niblett T. (1979) How difficult is the KNKR ending? in Advances in Computer Chess (ed. M.R.B.Clarke) Edinburgh University Press (in press).

[6] Michie D. (1976) An advice-taking system for computer chess, Computer Bulletin* Ser. 2* 10* 12-14.

[7] Nilsson N.J. (1971) Problem Solving Methods in Artificial Intelligence* McGraw-Hill.

SURFACE-NORMALS FROM CLOSED PATHS

M.J. Brooks
Department of Computer Science,
University of Essex,
Colchester, England.

It is now well established that the shading present in an image of an object is a potential source of 3-D shape information. With this in mind, we seek constraints arising out of the assumption that an image portrays an object with a smooth surface. On obtaining a threshold and a depth-invariance constraint (the latter requiring integration around closed curves) we offer a method of shape from shading for the purpose of demonstrating their potential usefulness.

KEY WORDS

Shape from shading, surface-normal, closed path, constraints, depth-invariance.

TOPICS

Vision and image understanding systems.

1. INTRODUCTION

Given an image of a smooth surface, we consider the problem of assigning to each image point an orientation corresponding to the appropriate object surface-normal, thus obtaining a surface-normal map. This constitutes the core of the shape from shading problem (see [3] and [8]) since 3-D surface data can be recovered from such a map by integration.

Two basic stages can be isolated in the generation of a surface-normal (s - n) map:

(1) The luminance value at each image point defines an orbit in the reflectance map [4] . Thus, for each image point, we produce an infinite, one-dimensional set of surface-normals.

(2) From each of these sets, we select a particular surface-normal such that the resulting surface-normal map represents a smooth surface.

Given a knowledge of the illumination conditions and surface reflectivity, the first stage is relatively straightforward. It is the second stage on which we shall concentrate.

This work is funded by the Science Research Council.

We contend that the assumption that an image depicts a smooth surface is one that has not previously been fully exploited, (For instance, Woodham's program [8J requires substantial clues as to the object shape before being able to run.) Thus we seek constraints imposed by this assumption on dense s-n maps: that is, maps which have an infinitely fine resolution (for we can then use continuous mathematics). These constraints are subsequently translated to discrete form prior to application.

2. CONSTRAINING A DENSE SURFACE-NORMAL MAP

2.1 Smoothness

A dense s-n map, u, represents a smooth surface if, and only if, for any point s in the map and 5 > 0, 3 c > 0 such that, for any 3' in an e-neighbourhood of s, the angle in radians between u(s) and u(s') is less than 6. (See Fig. 1.)

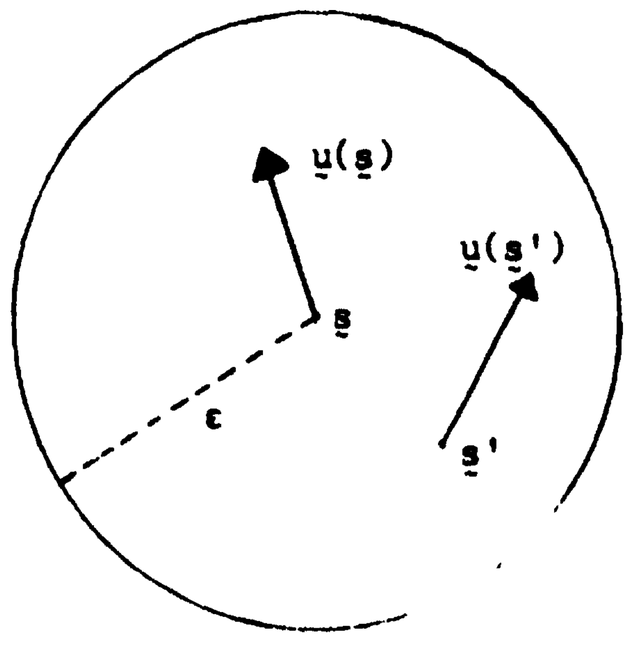


Fig. 1 Smoothness definition

2.2 Depth-invariant Loops

Suppose we were to draw a closed loop, CI, on a continuous surface; we would clearly finish the loop at the same point in space at which we started. Were we now to project the surface orthogonally to form a dense s-n map, then providing all points on the surface were visible with respect to the projection, we would obtain a corresponding closed loop, C2, in the resulting s-n map. If we integrated around *C2* over surface-normals to recover depth, then on completing a tour of C2, we would result in no depth change. Note the similarity between this and the scene analysis concepts of spatial closure expounded in [5] and [2].

This observation leads to a formal constraint: for a dense s-n map, u(s), to represent a continuous surface, the relation

$$\oint_r \frac{\underset{\sim}{u}(\underset{\sim}{s}) \cdot d\underset{\sim}{s}}{\underset{\sim}{u}(\underset{\sim}{s}) \cdot \underset{\sim}{D}} = 0$$

must hold for all closed paths r in the map. Here, s is a point on r with unit derivative ds as shown in Fig. 2; D is a unit vector of depth along the direction of projection. For the derivation, see [I]. The above definition has an analogue in the complex analysis theorem of Morera's; see, for example, p.538 of [6].

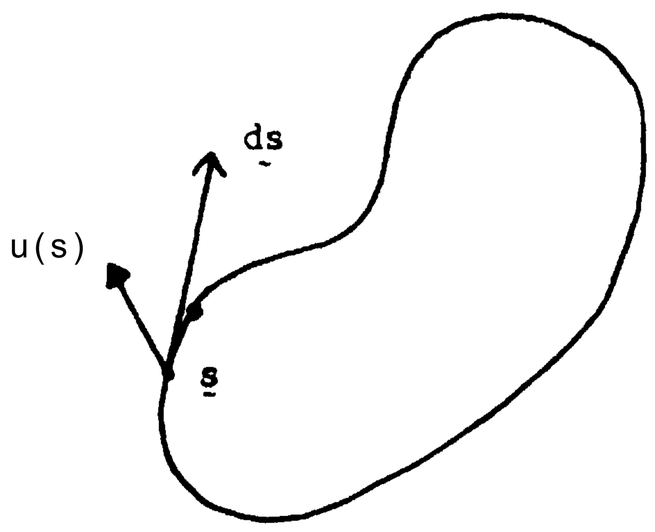


Fig. 2 Integrating around a curve T

3. MAKING THE CONSTRAINTS DISCRETE

Computer vision programs operate on tesselated images and so we require the discrete form of the constraints obtained in section 2.

3.1 Thresholding

In attempting to import the smoothness definition of 2.1 into the discrete world, we are, in a strict sense, doomed to failure since any discrete s-n map can represent a smooth surface providing the surface is allowed to fluctuate sufficiently between the sampling points. Due to a lack of information, we cannot be expected to recover this surface. Indeed, if we are expected to recover the object shape depicted in an image, we are justified in insisting that the image capture the characteristics of the surface. For our purpose, that criterion will be fulfilled if a piece-wise planar solution (one surface-normal per image point) is a satisfactory approximation to the surface.

Suppose we were to generate several images of varying resolutions from a wildly fluctuating surface. If we produced a discrete s-n map for each image and computed, for each map, the maximum angle between any two neighbouring surface-normals, we would find that, in general, this angle decreased as the resolution increased. So by assuming a minimum level of resolution per object scale and per maximum surface curvature, we place an upper limit, or threshold, on the permissible angular difference between neighbouring normals. (The author has recently improved considerably on a blanket threshold by adopting a functional threshold dependent on surface slant.)

3.2 The two by two Integral

We now apply the depth-invariant integral to the discrete case of a 2.2 window of orientations. For a square path of integration (a 2.2 path) and surface-normal quartet as shown in Fig. 3, then for this window to be representative of a continuous surface, the relation

$$\frac{(a_1 + b_1)}{c_1} + \frac{(a_2 - b_2)}{c_2} - \frac{(a_3 + b_3)}{c_3} - \frac{(a_4 - b_4)}{c_4} = \emptyset^*$$

must hold [1].

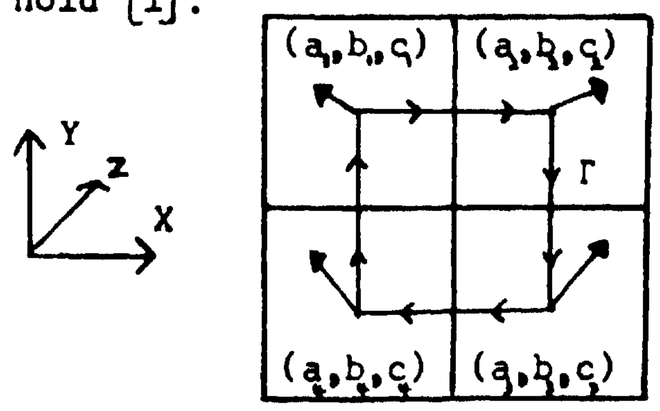


Fig. 3 The two by two integral

---

* The author has recently become aware of [9] in which an equivalent formula is (independently derived.

4. A SHAPE FROM SHADING METHOD

The need to investigate the depth-invariance of all closed paths in a discrete s-n map is an intolerable burden which we finesse by using the following result.

Lemma

If all 2*2 paths are depth-invariant in a discrete s-n map, then so too is any closed path in the map.

Consider a dense s-n map that represents a continuous surface. Then the depth-integral around a closed curve T clearly equals the sum of the depth-integrals around the constituent closed curves r_1 and r_2 as shown in Fig. 4.

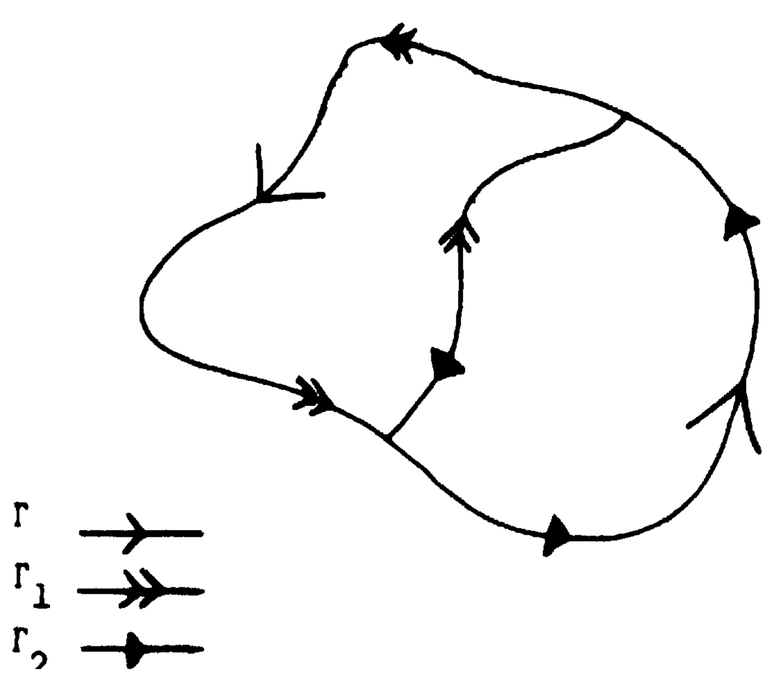


Fig. 4 Constituent curves

This splitting can be repeated as often as required and providing the correct directionality is observed, the sum of the depth-integrals around the constituent curves will equal the depth-integral around r. For a complex analysis analogue, see Goursat's proof, on p.526 of [4].

Now, any closed path in a discrete s-n map will consist of linked horizontal and vertical sections (e.g., Fig. 5). But any such path can be dissected into constituent paths that are all 2*2 (e.g., Fig. 6). Further, if these 2*2 paths all integrate to zero in either direction, then so too, by the above, must the bounding closed path.

We now describe a method of shape from shading in order to demonstrate the potential of these constraints. The method is in two stages.

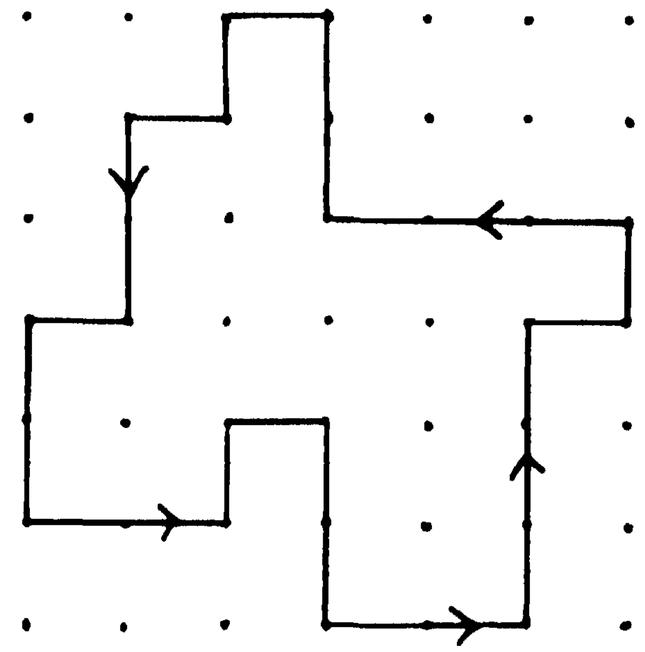


Fig. 5 A discrete s-n map closed curve

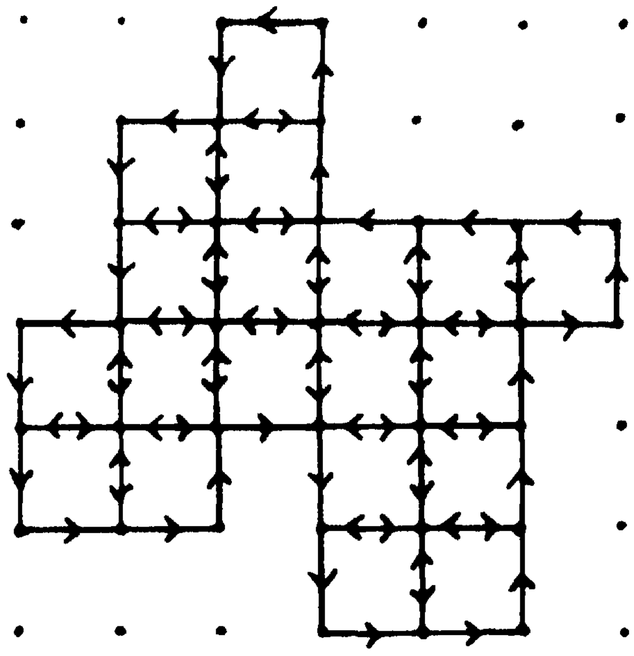


Fig.6 Constituent 2*2 paths

4.1 Obtaining a Set of Interpretations for each 2*2 window

As mentioned earlier, each image point specifies, in general, an infinite, one-dimensional subset of all possible surface-normals. Crucial to our method is the quantising of this interval into a finite set of orientations.

We therefore have a potentially vast number of possible combinations of surface-normal interpretations for each 2*2 window, of which the majority are likely to fail either the threshold or depth-invariance test. There is insufficient space here to detail a speedy means of generating a set of legal interpretations, but note that there is considerable scope for Waltz-like filtration on neighbouring surface-normals (see [7]). Also, note that all 2*2 windows can be processed in parallel. This stage has recently been successfully implemented.

4.2 Grow and Encompass

Having generated a set of legal surface-normal interpretations for each 2*2 window, we now want to obtain a set of interpretations for the whole image. To do this, we start by taking a 2*2 window of interpretations and merging this with an intersecting 2*2 window to form a larger area of legal interpretations. We continue to expand this area until the whole image is covered, thereby obtaining a set of solution s-n maps.

We define the algorithm inductively by considering the merging of the interpretations of an n-pixel area and a partially intersecting 2*2 area.

Suppose, in the n-area, pixels are indexed from 1 to n, and that the 2*2 area intersects the n-area at, say, two image points. Let $(p_{m1}, \ldots, p_{mn})$ be a legal interpretation of the n-area such that p_{mi} is the orientation of the i^{th} pixel, $1 <= i <= n$. Let $(p_{m1}, \ldots, p_{mn})$ for $m = 1, \ldots, k$ be the set of legal interpretations for the n-area, denoted as $(p_1, \ldots, p_n)_k$. Let the legal interpretation set for the 2*2 window be $(q_a, q_b, q_{n+1}, q_{n+2})_\ell$ where $a \neq b$ and $a, b \in [1, n]$.

Now, for all

$$(p_{m1}, \ldots, p_{mn}) \in (p_1, \ldots, p_n)_k$$

and

$$(q_{ra}, q_{rb}, q_{r,n+1}, q_{r,n+2}) \in (q_a, q_b, q_{n+1}, q_{n+2})_\ell$$

we test the relation

$$(q_{ra} = p_{ma} \text{ AND } q_{rb} = p_{mb})$$

which checks whether the surface-normals at the points of intersection are equal. If this relation is false, we pass on to the next combination, thereby rejecting the merger, otherwise we output

$$(p_{m1}, \ldots, p_{mn}, q_{rn+1}, q_{rn+2})$$

as a legal interpretation of the resultant area.

5. CONCLUSION

We have outlined a shape from shading algorithm that converges on legal interpretations without needing clues as to the solution shapes. This was done merely to demonstrate the value of the constraints in section 3. The labourious behaviour of the algorithm is clearly inadequate and the author is currently implementing a cooperative algorithm.

ACKNOWLEDGEMENTS

I am grateful to Pat Hayes, Mike Brady, Tony Cohn, Steven Draper and Paul Robertson for their useful comments and to Berthold Horn for his valuable encouragement,

REFERENCES

[I] Brooks, M.J. "Miscellaneous results'*, unpublished, 1979.

[2] Draper, S. "The penrose triangle and a family of related figures". Perception, 1978, vol. 7, p.283-296.

[3J Horn, B.K.P. "Obtaining shape from shading information", in The Psychology of Computer Vision, P.H. Winston, ed., McGraw-Hill, New York, 1975, p.115-155.

[4] Horn, B.K.P. "Understanding image intensity", Artif. Intell. 8 (April 1977), p.201-231.

[5] Huffman, D.A. "A duality concept for the analysis of polyhedral scenes", Machine Intelligence 8, eds. B. Meltzer, D. Michie, nburgh University Press, p.475-^92.

[6] Kreyszig, E. Advanced Engineerin Mathematics, Wiley,

[7] Waltz, D. "Understanding line drawings of scenes with shadows", in The Psychology of Computer Vision, P.H. Winston, ed., McGraw-Hill, New York, 1975.

[8] Woodham, R. "A cooperative algorithm for determining surface orientation from a single view", in IJCAI 1977, p.635.

[9] Strat, T.M. "A numerical method for shape from-shading from a Single Image", Ph.D. thesis, MIT, 1979.

CONTROLLING QUESTION ASKING IN A MEDICAL EXPERT SYSTEM

Ruven Brooks
Department of Psychiatry and
Behavioral Sciences
University of Texas Medical Branch
Galveston, Texas 77550 U.S.A.

Jon Heiser
Department of Psychiatry and
Behavioral Sciences
University of Texas Medical Branch
Galveston, Texas 77550 U.S.A.

We have discovered what we believe is an essential design problem for medical expert systems, that of controlling the amount and the type of information which the system requests from the user. This problem is inherent in medical expert systems because of the nature of the distribution of clinical states and the nature of the training and background of physicians. The problem also exists for human consultants, and a complete and general solution for computer systems is probably not achievable. However, several techniques show promise for reducing the magnitude of the problem in various clinical domains.

1.0 Introduction

Over the past two years, we have been engaged in the construction of a computer system to advise primary care physicians and specialists in other areas about the use of psychiatric medications. In the course of our first attempts at building the system, we showed it to a few colleagues and discovered, to our surprise, that some of them regarded responding to 30 questions with one-word answers an overly long interaction and the contemplated 100 item sessions clearly unacceptable. Although a portion of the problem was' clumsy use of software on our part, we believe that carefully selecting and controlling the amount and type of information requested from the user is a major aspect of the design of an expert system.

2.0 The Nature of the Problem

Computer based medical expert systems are designed to provide information and advice to physicians or other medical personnel. The Information and advice can be on such topics as diagnosis, test interpretation, medication levels, and patient management. All of these systems request information from the user, ranging from yes-no questions to open-ended prompts for free text.

We believe that limiting the number and content of these requests must be a major, primary design goal in the construction of expert systems. In particular, for optimum user acceptance, two conditions must hold:

1. The number of requests must be the minimum that is consistant with maximum utilization of medical inference.

2. The content of the requests must be perceived by the user as important and relevant to the task of the system.

We contend that these two conditions grow out of two characteristics of medical knowledge and medical expertise (though they probably hold for other fields as well).

2.1 Skewed Distribution of Symptoms

The first of these characteristics is that while the space of symptoms, syndromes, and diseases is very large, the distribution of occurrence in clinical populations is markedly skewed; a *very* few symptoms and diseases account for a large proportion of patients.

This skewed distribution has a profound Impact on the way medical experts convey and express knowledge. One of the more subtle, but significant, impacts is on medical terminology. Medical terms are frequently organized in a hierarchical fashion. Terms near the bottom of the hierarchy refer to single signs or symptoms while those further up the hierarchy *refer* to larger clusters, often with Increased etlologlc or therapeutic implications. This terminological structure often allows complex medical states to be described concisely, with a few well chosen words. This conciseness, in turn, greatly facilitates medical communication.

A second impact of the skewed distribution of symptom occurrence is that information is often conveyed as much by omission as by statement. For example, the lack of any comment regarding delusions or hallucinations in a brief note by an experienced psychiatrist means that, during the course of the evaluation, no delusions or hallucinations were reported or observed; it does not mean that the psychiatrist did not check for them.

A priori, the terse nature of medical communication and the domain sophistication of the user ought to facilitate the building of expert computer systems. The use of a well-defined terminology, together with the fact that the user understands the underlying medical principles. Implies that very simple forms of natural language processing can be used without obviously restricting the range of user responses. To some extent, this expectation has been realized; the MYCIN system [1], for example, asks open-ended questions such as "What is the Gram stain of Organism-1?" without analyzing the input, other than to check the response against a list of legal values.

Problems may arise, however, from several sources: One is, of course, inappropriate or incomplete representation of the medical knowledge within the system. Systems which ask about the current mood of comatose patients are not likely to be well accepted by users.

Other problems in representing medical knowledge are more subtle. An example from our system involves asking about all medications that a patient received during all prior psychiatric episodes; this is a useful piece of information in selecting a current medication. Unfortunately, when questions were asked for each and every prior episode of psychiatric illness, a user was frequently repeatedly asked about medications given years ago, at other hospitals, concerning which little or nothing was known, and the procedure quickly became more irritating than informative.

2.2 Variance iri Physician Knowledge

The second major characteristic of medical knowledge and expertise is that a given physician's level of sophistication may vary greatly depending on the area of medicine. Thus, a second major source of difficulty lies in mismatches between the level of sophistication of the user and that assumed by the system. For example, a system may assume that the user is unable to supply a particular fact or conclusion, and it may, therefore, ask a series of questions to deduce the conclusion. If the user can, in fact, directly supply the needed information, then he or she may become irritated at how "dumb" the system seems. A mismatch in the opposite direction is equally problematic; if the system prompts for conclusions or inferences which a user is not knowledgable enough to supply, the reliability and validity of the system may be considerably reduced.

The mismatch problem is not merely relative to the user's knowledge and experience, but is also relative to the user's perception of what the system is doing. For example, a system which has received inconsistent or doubtful input may, quite appropriately, attempt to deduce some fact or conclusion that a user has already supplied. If the user is unaware of why this is being done, the system's "second guessing" may elicite an unpleasant reaction.

3.0 Solution Strategies

While these problems have been discussed in the context of computer systems, similar problems occur with human consultants. Furthermore, the clinical problems, diagnostic techniques, therapeutic measures and language processes used to communicate these phenomena are so different in different areas of medical practice that it is unlikely that these problems will be solved universally and quickly by any single technique or device. Almost certainly, different combinations of techniques will have to be carefully engineered for each clinical application.

As yet, the repertoire of such techniques is small, and there are no guidelines for identifying when particular techniques should be used. Currently, three techniques show some promise as components of such combinations.

3.1 Guidance from Models of the Domain

One device which can be particularly helpful in eliminating unnecessary questions is a model of the dynamics of a medical state. These models are representations of physiological or biological mechanisms and tie together chains of medical events. Several medical systems have been built which rely on such models [2].

As helpful as dynamic models are, they are currently applicable in only a small number of medical systems. In many medical areas, underlying mechanisms are not yet sufficiently well understood to be useful. An alternative which is more widely applicable is the use of

static models of the relationship between observed signs and underlying disease states. [3.4].

The way in which such static models act to reduce the system's requests for

While static models are, they do not provide an absolute guarantee of user acceptance. If the user's knowledge of the domain is too different from that of the system's, he or she might not recognize the critical nature of some of the information being requested and, therfore, interpret the questions as superfluous.

3.3 User Control over Sophistication Level

While both dynamic and static models act to reduce requests for unnecessary information, they may have little or no impact on difficulties caused by a mismatch in knowledge levels between the user and the system. One approach to this problem is for the system to assume a high level of sophistication on the user's part but to provide the user with ways to indicate that a lower level is appropriate. One such mechanism that we are building into the Psychopharraacology Advisor is to permit users to respond with "LEVEL" to a question which she or he feels is at too high a level. The system responds by breaking the question down into several, simpler questions. For example, answering "LEVEL" to a question about complete auditory hallucinations will cause the Advisor to ask a series of questions about whether the patient has auditory hallucinations, what their form is, and what the content of the hallucinations is.

We believe that this device is a useful addition to conventional "help" system displays of documentation and user options. It involves less user reading and interpretation and, therefore, should be less error prone.

3.4 Supplying Rationales

As noted earlier, in determining user acceptance, the perceived rationale for an Information request is often as Important as the contents of the request itself. Rather than relying on the user to guess the rationale correctly, it may prove more reliable to have the expert system make its rationale available to the user, either automatically or oy request. An example of how this may be done is the explanation capabilities of the rule-based EMYCIN software (Shortliffe, 1976) which we are using as the basis for the Psychopharmacology Advisor. In response to a question from the system, the user may, instead of giving the answer, give commands to obtain Information about why a particular question was asked or how a particular result was achieved.

14.0 Conclusion

We believe that controlling the amount and type of information that a medical expert system requests from a user Is an essential design problem. This problem is inherent in medical expert systems because of the natures of the distribution of clinical states and of the training and background of physicians. The problem is also present in interaction with human consultants, and a complete solution for computer systems is probably not immediately achievable. However, several techniques show promise for reducing the magnitude of the problem. These include the use of dynamic and static domain models, user control over sophistication level, and user access to the rationales behind information requests.

REFERENCES

[1] Shortliffe, E. H. Computer Based Medical Consultations: MYCIN. New York: American Elsevier, 1976.

[2] Gorry, G. A., Silverman, H. & Pauker, S. G. Capturing clinical expertise: A computer program that considers clinical responses to digitalis. American Journal of Medicine, 64(452), 1978.

[33 Pople, H. E. The formation of composite hypotheses in diagnostic problem solving: An exercise in synthetic reasoning. Proceedings Fifth International Joint Conference on Artificial Intelligence, 1977.

[4] Aikens, J. Prototypes: An approach to knowledge representation for hypothesis formation. Proceedings this conference.

The ACRONYM Model-Based Vision System

Rodney A. Brooks, Russell Creiner and Thomas 0. Binford
Stanford Artificial Intelligence Laboratory,
Stanford University,
Stanford, California, 94305, U.S.A.

ACRONYM Is a model-based image understanding system. It demonstrates mechanisms for interpretation of images with generic object classes and generic viewing conditions, in a way that *is* generalizable. It incorporates a powerful geometric modeling capability with a high level modeling language for natural communication with the user In terms of object models A user gives high level descriptions of both generic and specific instances of objects A rule-based Inference system produces a viewpoint dependent symbolic summary of the predicted appearance of the objects. *This* geometric reasoning capability enables the system to incorporate and relate knowledge and information at different levels This summary drives a powerful syntactic matcher to find instances of the objects in preprocessed images.

1. INTRODUCTION

ACRONYM is a vision system based on powerful geometric modeling capabilities. Its geometric modeling system is of Interest in Itself. ACRONYM is also the basis for research In programming robots from a data base of parts models.

ACRONYM is intended to deal with key problems of Interpreting scenes:

I Generalliablity is a central issue. A photointerpreter performs a a broad range of tasks which involve different object classes, which have different contextual Information, and which vary greatly at the Image level because of varied viewpoint, Illumination, sensors, weather, and obscuration and camouflage. Systems for very different tasks should be constructed from a large core of common modules and a small set of task-specific modules For a single system to map this wide range of task elements onto a common set of modules, it is convenient that the modules represent a natural decomposition of the problem into physically meaningful elements, for example, those we use in our own human description of the problem Our basis for generaluability is the use of a tightly structured hierarchy of geometric representations.

2. A variety of information and knowledge is available A photointerpreter solves a puzzle by piecing together selected and multiple chics from current images, background Information, and previous images In doing so, he relies heavily on spatial interpretation from stereo imaging and shadows, and spatial knowledge about structures

This research was sponsored by ARPA contract MDA-903-76-C-0206 and NSF contract DAR-78I59I4

Integrating multiple cues within a single task is a key issue which raises technical questions for representation.

3 It is important that the system be generic with respect to objects and generic with respect to viewing conditions Our approach to generic interpretation is to use object models made from generic parts, and to use generic predictions of appearances of generic parts.

4. Users should be able to specify tasks In a natural and simple way Geometric models are natural for both the user and the vision system Ultimately, users will be able to instruct systems In natural language. The representation hierarchy of ACRONYM is the basis for a Vision Language which could serve as a bridge between natural language and standard programming languages

We describe capabilities for goal-directed model-based scene interpretation based on mapping from object and part models to predicted appearances to picture structures ACRONYM is designed to function in a descriptive sense (bottom up) as well, mapping from picture structures to features of parts to object models In typical situations, both functions are involved.

Model-based vision has major applications In photointerpretation and manufacturing Consider two typical tasks for the system;

I A photointerpreter programs the system to monitor aircraft at an airfield. Aircraft models are entered along with a model of the airfield. The system Is told that aircraft are found at airfields on runways, taxiways, or storage areas.

2 An engineer programs ACRONYM to inspect a part, or to pick the part from a bin of identical parts and place it in an assembly. He uses the part model from a CAD system and specifies a model of the workstation.

The diagram in fig. I shows the main modules, data structures and data flow paths, along with the local environment of ACRONYM. The innner enclosed boxes represent data structures, the other boxes are program modules As of January 1979. implementations of all the modules existed, but not all data paths were in operation.

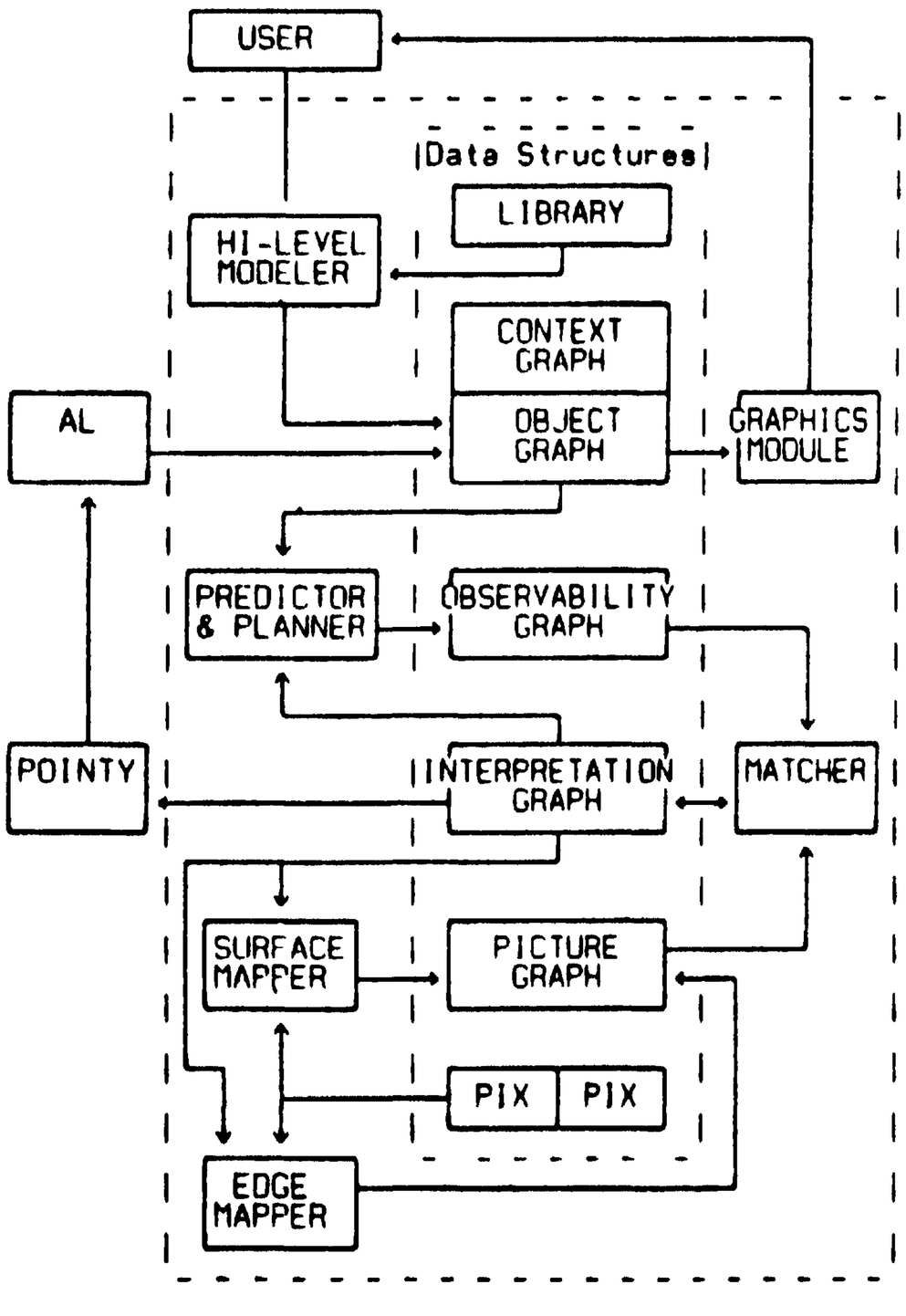


Fig. J. The ACRONYM system.

The user interacts with a high-level modeler to produce both generic and specific models. There will eventually be a library of useful partial models to draw upon. The result of this modeling process is the Object Graph. This graph drives a graphics module which provides the user with visual feedback of shape and spatial relations during the modeling session. The Predictor and Planner module produces a symbolic summary of the way an object modeled in the Object Craph can be expected to appear In an Image. This summary is called the Observability Craph and is used to program the matcher from above The Matcher constructs an Interpretation Craph from the Observability Craph and the Picture Graph, which is a symbolic description computed from the Image by the surface mapper, and the edge mapper. The edge mapper uses techniques of Nevatia and Babu [23] to extract two dimensional shape descriptions from the images This is augmented by depth information from the stereo pair of images (when a pair is used) in the surface mapper, using techniques developed by Arnold [2)

Also included in fig I are modules for AL and POINTY (see Goldman [13]). AL is a high level language for controlling manipulator arms ACRONYM has been interfaced to the AL compiler, so that instead of driving the physical arms, AL drives ACRONYM models of arms. ACRONYM can update its display of an arm faster than once per second on a moderately loaded system, giving real time simulation There is an interface in the other direction to POINTY. POINTY is an interpretive subset of AL, originally designed to be used in a lead and teach mode, to easily obtain spatial information for AL programs. ACRONYM can control the arms via POINTY. So far we have used this interface only for debugging, but it will eventually be used for fully automated assembly tasks, with model-based vision

2. PREVIOUS MODEL-BASED SYSTEMS

ACRONYM provides a unified approach to many aspects of goal-directed vision, and generalizes the approaches of previous vision systems It has a richer three dimensional model representation, a rule-based planning system, and makes more extensive use of shape and relational information than previous systems We briefly survey some existing systems to summarize their use of models and their matching processes.

The MSYS system (Barrow and Tenenbaum *[4])* models objects in terms of their height and orientation, and models three dimensional spatial relationships between them in terms of above, bclow.in front, in back, left and right. Some models (e.g. of a floor) are also constrained to correspond to homogeneous regions within an image. An image is roughly segmented by hand. Simulated range data Is used to determine three dimensional locations and orientations of regions and their spatial relations Horizontal surfaces are identified by height. No shape information is used at all. A matcher matches the data against the model. These simple models are sufficient to identify typical objects in a room scene This demonstrates that spatial relations can be a strong constraint when identifying regions of an image. However the models used rely on a particular viewpoint, and the ability to propagate constraints throughout the whole image. If some other viewpoint is used, or if the scene *is* cluttered with objects not contained in the model, the constraint approach will break down. Tenenbaum and Barrow [25] have also used this system to drive the IGS interpretation guided segmentation system which produces segmentations constrained to be consistent with the high level model.

Carvey [12] developed a system which models objects in terms of their height and orientation, three dimensional spatial relationships with each other and the expected ranges of brightness, hue and saturation of their pixels. Approximate indications of size and shape may also be included. In interactive sessions with sample pictures, the system computes histograms of these features over manually segmented regions Confidence values are given to the spatial relations by the user The system is given the task of finding a given object in an image. It knows the costs associated with testing pixels for each attribute, and so as an initial coarse pass, It tests pixels randomly selected from throughout the Image with tests that will cheaply rule out large areas. These tests are selected by examining the models of known objects for cheap distinguishing features of the desired object. Carvey calls this initial phase acquisition. In the second phase, validation, the models are used to determine more expensive tests, over this reduced area, which will disambiguate the desired object from others which might also have produced the pixels first obtained. Spatial relationships in the models can be used to drive the system when looking for small objects Larger objects, expected to be adjacent In the image (such as a table top when looking for a telephone) are found first, reducing the search area for the small object. The search for the small object can then be carried out at a much finer level. The planning stage and coarse to fine strategy used improve efficiency but this system too suffers from the inability to use general shape information

Ballard, Brown and Feldman [3] describe a general purpose system for programming knowledge-based vision tasks. The knowledge or model Is in terms of nodes describing expected image features, and spatial constraints between them Templates are used to describe shapes. The user must code an executive match procedure for the particular task domain. The system has been used for aerial image interpretation and finding ribs in radiographs of human chest cavities While very general purpose, the system is heavily dependent on using a fixed and known viewpoint, and requires detailed programming for each new domain. Furthermore the user must model objects in the image domain rather than the three dimensional domain

Bolles [9] has a Verification Vision system which relies only on the three dimensional relationships of observables to locate a mechanical part accurately within an automatic assembly work station. The position of the part is known in advance to within a few inches, and its orientation to within about fifteen degrees Thus the models can be very viewpoint dependent. Rather than use shape descriptors, Bolles relies on very local operators such as edge detectors, blob characterizes, and Moravec's [20] Interest operator to characterize the observables. Due to the highly constrained nature of the task these operators have a reasonable chance of matching the correct feature in the image. The system uses a generalized least squares algorithm, and clique finding to match the model of the known spatial relationships of the object to the features identified by the operators. The model can be used to drive a further refinement phase, where the location and uncertainty *region* for further features are predicted. These features can then be searched for *in* the restricted region, with much reduced probability of incorrect identification. Notice that the models used here are simply a collection of points on the surface of the object

Kanade [16] distinguishes the image domain, the domain of observable facts from viewing the scene in either intensity or range data, and the scene domain, where objects are modeled. He uses a 2 1/2 D scene domain. Objects are represented as Image regions. Shape and spatial relations describe the regions. Objects have multiple representations for multiple viewpoints, but these must be explicitly described by the user. The matching process tries to match observed patches against modeled patches.

Rubin's [24] ARGOS system stores multiple representations of buildings, in terms of such things as texture, color, orientation and gross shape features, all gleaned from training examples It also has three dimensional knowledge of the positions of the buildings, which is translated into adjacency information to guide the search for labellings of pixels. The search technique is a very local pixel (or segment) based "Locus" search. This localness means that adjacency is the only meaningful relation which can be used.

These systems make valuable contributions to model-based vision, however they do not generalize in significant ways, with respect to viewpoint, with respect to object classes, and with respect to very large classes of objects. In almost all cases, data and model are matched at the same level, e.g. surface to surface or point to point. ACRONYM adds the capability to reason between different levels of representation, based on a hierarchy of representations.

5. MODELING

The Object Graph contains three dimensional models. It provides volumetric representations of objects, both specific and generic, and three dimensional spatial relations between volume elements. It also provides for multiple levels of representation of objects, from coarse to fine The basic volume primitive used is the generalized cone, first introduced by Binford [5] Spatial relationships of volume elements within an object are defined hierarchically. Both specific and generic volume elements, and relations between them can be modeled. Thus an airplane can be defined to have exactly two wings, while an airport can be described as having from one to four runways.

The Context Graph is not really distinct from the Object Graph. It is only a convenience to talk about objects such as aircraft and their context, airfields. Both use the same representational mechanisms.

3.1 Generalized Cones

Our models are part/whole graphs; the models resemble

stick figures. Parts have subparts, they are subgraphs whose leaves are our primitives, generalized cones. Levels of detail correspond to levels of the part/whole subgraph A natural and useful part/whole segmentation depends on the primitives used for representation of parts Design criteria for representation were presented in Thomas and Binford [26].

A generalized cone is defined by generalized translational Invariance. It is described by three sub-primitives; a spine, a cross section and a sweeping rule. In its most general form, a generalized cone Is the volume swept out by the cross section as it is translated along the spine, while being deformed according to the sweeping rule Elongation is not a necessary property of generalized cones; neither is a circular cross section.

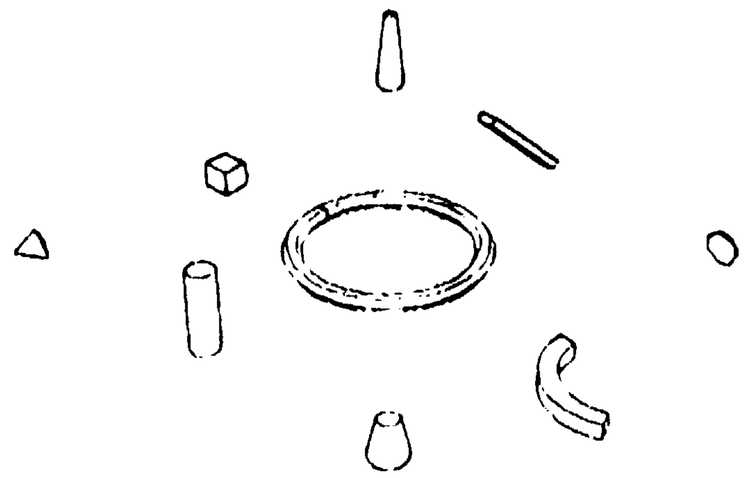


Fig. 2. Some Generalized Cones

Most objects can be segmented In a few ways Into volume elements whose cross sections change smoothly along some space curve (see [21] or Nevatia and Binford [22]). These volume elements can then be modeled by a single generalized cone, with its spine along that space curve Generalized cones accurately represent a variety of fabrication processes Intrinsic properties of a generalized cone can be approximately Inferred from Its two dimensional image under a broad range of viewing conditions (Nevatia [21], Marr and Nishihara [18]). For example, the length to width ratio varies approximately as the sine of the angle between the the cone axis and the ray from the axis to the observer While generalized cones need not be elongated, interpretation is much simpler for elongated objects and current techniques rely on elongation Marr [17] has shown that Interpretation of the occluding contour of an image of an object, under certain restrictive assumptions, is equivalent to interpretation as generalized cones. Thus generalized cones correspond to a natural decomposition of large classes of objects, and they are easy to match to a two dimensional image. Moreover they provide a compact representation for complex volumes. The representation of complex objects requires representation of only a few parts, each of which is about as complex as the representation of a cube

Generalized cones have been used In a number of vision and geometric modeling systems Agin [I] built generalized cone descriptions from Images of simple objects such as a snake and a hammer. Nevatia [21] recognized images of dolls and toy horses, by Indexing into a library of generalized cone models. Miyamoto and Binford [19] built a general modeling and graphics display system based on generalized cones Hollerbach [14] modeled and recognized classes of Greek vases. Marr and Nishihara [18] described a method to determine the orientation of objects by matching them to generalized cone models

Representation of spherical volumes as spheres is Intuitively better than representation as generalized cones, especially for portions of spheres We set out to incorporate two primary geometrical constructions; translation and rotation. Generalized cones are based on generalized translational invariance. Spheres are based on rotational invariance We have a straightforward generalization of rotational invariance, however generalized spheres, as we understand them now, do not seem as powerful an extension as generalized cones', and we require more work to include them in our representation. We have also thought it necessary to represent surfaces, obviously to represent cross sections of cones, but also for nonplanar sculptured surfaces such as wings. Generalized translational invariance is useful here, too. Generalized cones specialize to ribbons, which are defined by an axis and a cross section, both of which are space curves not plane curves, and a sweeping rule (see Brooks, Greiner, and Binford [7]). It *is* not our intention to ignore standard surface spline representations, but rather to explore the advantages and cost of an alternative representation

We use a more general class of generalized cones than previous systems However there are still major restrictions The cross sections must have a boundary which can be decomposed into straight line segments and circular arcs The cross section can be kept at any constant angle while being swept along the spine The sweeping rule must be piecewise linear, and continuous The spine must be continuous and made up of straight line segments Circular arcs can also be used as segments of the spine, so long as the sweeping rule is constant, the cross section has a piecewise linear boundary and is kept normal to the spine. Fig 2 shows some examples This latter class allows for convenient representations of curved roads or taxiways. Analytic work is being done to formalize a natural and adequate extension of this subclass of generalized cones.

The parameters of a cone can describe ranges of possible values, or provide a predicate to test the validity of a proposed value For instance a generic runway can be described in terms of allowable ranges of length and width - possibly dependent on each other.

..2 Model Structure

As in previous modeling systems based on generalized cones (e.g. Miyamoto and Binford [19]) there is a local coordinate system for each volume primitive in a model, rather than a viewer centered, or object centered coordinate system. A complex object which is modeled by more than one

generalized cone has an affixment tree (in general a directed graph - but multiple afflxments can be Inconsistent), where the coordinate system of a cone is that of its parent, modified by the coordinate transform attached to the arc joining them. As in previous systems (see Thomas and Binford [26]) our models also have a hierarchy of level of detail. However we have chosen to separate this hierarchy from the attachment hierarchy.

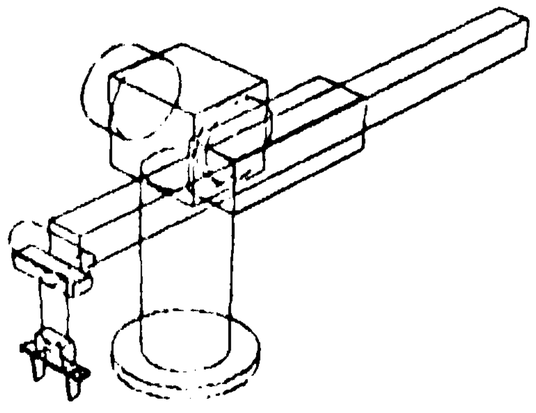


Fig. 3. The Stanford Arm

The following two examples indicate how and why this separation can be useful It is natural to affix the two wings of an airplane to the fuselage, to describe the spatial relation among them However wings appear almost the same size as the fuselage, and the two of them combined (especially when an airplane on the ground is viewed from above) provide an axis of elongation of the same order as that given by the fuselage Thus at the first level of decomposition, we model an airplane as the generalized cones of the fuselage and the two wings, rather than as a single cone. Consider now the Stanford robot arm shown in fig. 3 (this Is a drawing produced by ACRONYM from its model of the arm) A natural *first* level of decomposition of the arm is the upright base, the boom and the hand assembly. The point of Interest here is the relation between the boom and the hand The spatial relation between the boom and the wrist is best described by a series of spatial links through subparts of the hand.

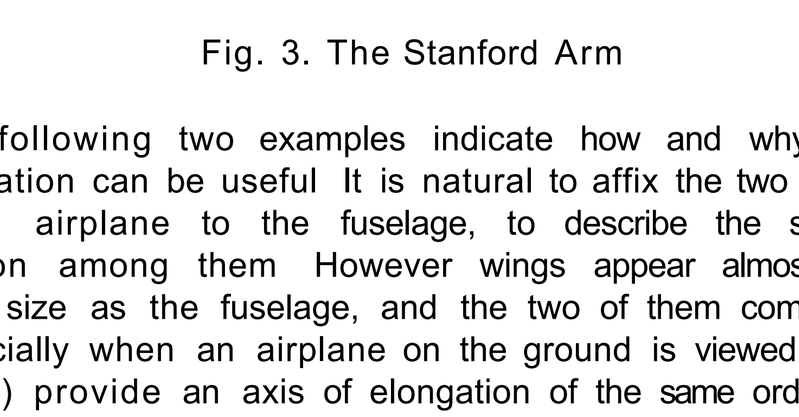


Fig. 4. An LIOII

It is important to represent both generic objects (loosely classes of objects) as well as specific objects For instance we may want to look for airports In general, not only to count the airplanes at each, but also to count the numbers of particular types of airplanes which can be identified Thus a DC-10 say, would be recognized as an instance of the Object Graph node airplane, or as a DC-10. An aircraft with special radar would be identified as a modified form of a prototype If the Object Craph doesn't include a detailed model of a Boeing 767, It might be identified as an airplane, but fail to match precisely any detailed model. A specific model may be a more refined version of a generic model and a specific model can be an example included in a generic model. The modeling system provides language constructs for using a node of the object graph as a prototype for the construction of a new node. This can be done at all levels - from an object down to a sweeping rule.

13 Geometric Editor

The high level modeler provides a high level language which gives the user the ability to manipulate models wiih symbolic names, and provides ways to examine partially built models There is also a graphics module to help the user with detailed modeling of specific objects. It provides the user with visual feedback of what is being modeled and where the local coordinate systems are. The class of cones allowed can generate complex curved surfaces, e.g. the surface generated by a circle swept along a spine at some non-normal angle while varying in size. Rather than approximate these surfaces with planar surfaces, we manipulate Its symbolic description right up until the time to deposit the visible lines in a raster buffer.

We have fast analytic solutions for the appearances of the occluding contours of these surfaces, and have implemented them in a hidden back surface algorithm. It works directly from the object graph representations Early details of that algorithm appeared in our paper [7], Work is proceeding on extending this to a full hidden surface algorithm and the full solutions will appear In a later memo Figs 2, 3. 4 were produced by ACRONYM, requiring approximately 250 milliseconds for the most complex of those figures.

4 PRODUCING THE OBSERVABILITY CRAPH

Given models of objects, ACRONYM attempts to find instances of the objects in images The Observability Graph will tell the matcher how to find instances It is a symbolic summary of the expected appearance of objects in the image It contains generic and specific predictions about shape elements and relations between them, with information about how to find them, what conclusions to draw if identified and what to conclude *it* they are not there. If Information about viewing angles, distance and conditions Is available, it can be used to produce more definite predictions The Predictor and Planner module is a rule based system which uses the Object Graph to produce the Observability Graph

The program must choose features of the object which correspond to image and surface properties which segmentation programs can find We will call such features "observables". The features which we are using first include shape and two dimensional spatial relations of shapes

within the image. When dealing with a stereo pair of pictures we also use three dimensional spatial relations and surface information. We intend to extend the class of features after our initial experiments.

4.1 Observables

Since the exact disposition of objects in the image is not known in advance, the Observability Graph can not contain an exact prediction of what will be seen. Rather, it must consist of predictions which adequately describe a range of possible appearances, generic with respect to both object class and viewpoint. They can best be thought of as supplying constraints on the way the picture can be expected to look. For instance, given that the system will be examining aerial photographs of airplanes on the ground, the Predictor and Planner can tell the matcher that when a candidate fuselage is found, wings should be found adjacent to it, with bilateral symmetry. There will be two possibilities for the relative angle, and once the actual angle has been found, the front and rear of the fuselage will have been distinguished. Then the finer task of locating the rear stabilizers (for positive identification of airplane type) can be carried out, confined to a small part of the image How these are implemented is described in the next paragraph.

The best sort of observables to put into the Observability Graph are those which are invariant. This is a strong requirement; other predictions will be true under a wide range of viewpoints We will call these quasi-invariant observables. Some are quasi-invariant with respect to object class, while others are quasi-invariant with respect to viewing conditions For example all airplanes have a long cylindrical generalized cone as the fuselage (invariant with respect to object class), and from most viewpoints (especially aerial views of airplanes on the ground) the fuselage will appear as an elongated ribbon (invariant with respect to viewing conditions). Observables which are functions can be viewed as quasi-invariants too. For example, the orientations of the wings within an image are functions of the orientation of the fuselage.

The Observability Graph itself consists of nodes, arcs and relations. The nodes are either direct shape descriptors or recursively complete Observability graphs themselves. For instance an airport Observability Graph will contain nodes for runway and taxiway shape descriptors, and a node which is itself a complete Observability Graph for the generic class of airplanes. Each node corresponds to something which should be identifiable in an image. Arcs are binary relations which should hold between instantiations of the nodes they link in the image. For instance an arc might say that any instance of a runway should intersect at least one instance of a taxiway. The Observability Graph also allows n-ary relations - called simply relations These are selected from a library of relations each of which is a function on a list of Observability Graph nodes and their candidate instantiations. Proximity and connected are currently Implemented.

4.2 Control of Rules

The Predictor and Planner is clearly a critical module of the ACRONYM system ACRONYM is the first vision system to incorporate a general reasoning system. It is necessary because we wish to predict the appearance of generic objects, from generic viewpoints This is a difficult task which requires reasoning about a large body of diverse knowledge. We have chosen to use a rule based system to facilitate experimentation, and to provide additivity of new knowledge. We will first describe the control structure we have implemented for our rules, then discuss some of the techniques used in our early sets of rules to produce the Observability Graph

The basic control strategy we have chosen is to have our rules consequent driven (i.e. goal directed, or backward chained). The rules can be interpreted to mean that the consequents should be asserted if the conjunction of the antecedents can be proved This simple strategy is not quite sufficient for the task at hand, however It is possible that once invoked and "fired" a rule may take control for a while, doing some forward reasoning, and possibly setting up new goals which are attempted before control is handed back to the original invoking mechanism Davis, Buchanan and Shortliffe [II] use consequent driven rules in the MYCIN medical diagnosis system. Our first implementation of the rule based Predictor and Planner closely followed the MYCIN model. However, we are working with more complex, structured data, and have gradually moved to a rather different system

Our rules have three compppnents, premises (these are the antecedents), side effects and consequents (possibly multiple consequents for a single rule) Rules are indexed on their consequents. The backward chaining strategy finds all rules which might possibly satisfy the current goal. They are tried in turn until one succeeds. A rule is tried by recursively trying to satisfy its premises, until either one fails (whence the rule is discarded and the next rule tried), or until all premises have been satisfied In this case the side effects are carried out, and then the consequents are asserted - satisfying the original goal. As the name suggests, the side effects of a rule have nothing to do with the basic goal directed control structure.

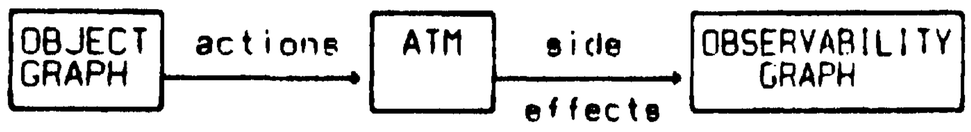


Figure 5. Producing the Observability Graph.

Fig. 5 shows the flow of information from the Object Graph to the Observability Graph. The assertion of the consequents places them into Associative Triple Memory (ATM) The general form of a consequent is a triple consisting of an object (not in the Object Graph sense), an attribute and a value. The attribute is an S-cxpression - currently these are used purely syntactically via tests for equality, but later semantic information may be explicitly

attached; the object maybe a structured fragment of the Object Graph or some entity constructed by the rules. The actual Observability Graph is produced purely by the side effects of rules firing. The goals which direct the control structure are to place given triples in the ATM. For instance the top level goal for producing the Observability Graph for a model called AIRPORT would be to achieve the triple <AIRPORT, (OBSERV GRAPH), T> in ATM.

In producing the observability graph the Predictor and Planner is guided by two classes of quasi-invariants. Object class invariants broadly determine the node structure of the Observability Graph The viewing condition invariants determine the detailed contents of the nodes, and the arcs and relations between nodes. Of course this distinction is sometimes blurred For instance a mixture of the two types of quasi-invariants directs production of the arc which says that runways should intersect a taxiway in the image. The object model says that they should be connected, and a rule which knows that connectivity is invariant under all viewing angles deduces that intersection of the regions corresponding to the nodes will be observable.

5. MATCHING

The Matcher uses the Observability Graph generated by the Predictor and Planner to locate an object in a Picture Graph. The Matcher makes an Interpretation Mapping which maps the Observability Graph into the Picture Graph. It uses a goal-directed matching scheme, searching for features indicated *in* the Observability Graph. It uses relaxation techniques to resolve constraints iteratIvely. To improve efficiency, the Matcher works in a coarse to fine order which is directed by the Observability Graph. For example, In Identifying aircraft, the Observability Graph has several levels of detail, first the aircraft context, an airfield, then the top level of detail of the aircraft, fuselage and wings, then the finer level of detail which distinguishes individual aircraft, I.e. engine number and placement. The Predictor and Planner uses the level of detail of the Object Graph and observability criteria to constrain the Matcher's search space.

Nodes of the Picture Graph correspond to ribbons, surfaces, and curves. Its arcs and relations indicate spatial relations which hold between sets of nodes. We currently *use* intersection, colinearity, and containment arcs. The classes of nodes and arcs are open-ended to accomodate other primitives and relations Ribbons and curves are obtained from an Edge Mapper (see Brooks [6]) which uses a line finder of Nevatia and Babu [23} Surfaces will be obtained from a Stereo Mapper by Arnold [2]

The Matcher attempts to instantiate each node pf the Observability Graph by a set of Picture nodes which satisfy properties of the Observability Node. These local interpretations map a part of the Observability Graph into a limited part of the picture. Context is expressed as Observability arcs and relations among sets of Observability nodes. Local interpretations consistent with Observability arcs form the initial stage of the Interpretation Graph For example, a segment of a picture may appear to be a runway when it *is* considered in isolation. However, this interpretation Is not likely if the proposed runway connects with a freeway system, or has a flow of automobiles upon It. This information must be encoded in the Observability Graph by the Predictor and Planner.

Each test evaluated by the Matcher has a pair of procedures, (provided by the Predictor and Planner) one for success and one for failure which are invoked appropriately. One possible action of a feature-test is to cause this match to fail unconditionally - thereby removing the suggested instantiation (e.g. a particular Observability node to Picture node mapping) from consideration. Usual actions simply modify the bookkeeping associated with the "goodness" of the match. Standard conditions such as Must-Be or Should-Not-Be can be encoded as simple special cases.

The Observability Graph can also specify ways to explain a failed condition. More generally, it can specify consequences of interpretations For example, an interpretation of a ribbon as a fuselage determines the location of the wings. A conjecture may be made which seems plausible at a preliminary stage of the match. If the conjecture is later proven to be false, the conclusions it fostered are re-evaluated, and possibly rejected as well This follows the "Wait and See" philosophy used throughout the matching process.

In the Picture Graph, a set of edges may be structured as ribbons In several incompatible ways An Observability node may map to multiple Picture nodes and a Picture node may map to multiple Observability nodes. The mechanism for enforcing compatibility is uniform at all levels.

The Matcher has storage and retrieval facilities for finding predesignated common subexpressions It is probably too costly to automate the finding of common subexpressions in the Matcher. The process could be partially automated at the level of the Observability Graph or Object Graph, but is probably ineffective if done in a syntactic way. The problem is NP-complete.

The same basic format and evaluating scheme is used by the Observability nodes, arcs and relations as by the full Observability Graph itself. An node may further place certain requirements on the arcs and relations in which it participates. This may be used to specify the number of instances of a given arc in which some node may participate - for example, that a runway should be parallel to at most one other runway in a given airport This quantification is simple to state and to check Far more complex conditions could be enforced with this facility Another feature is the use of a plethora of demons for a variety of cases. There is a growing library of routines, including a transitive closure and a clique finder, which can be used by the Predictor and Planner as appropriate. The range of control structures representable in the Observability Graph, and usable by the Matcher currently far outstrips the ability of the

Predictor and Planner to produce them.

An important aspect of the Matcher *is* its total ignorance. It is a driver function, designed to "evaluate" the Observability Graph In the context of the Picture Graph. All the specific functions, ordering schemes, demons, etc. are contained within the Observability Graph itself. The only routines in the Matcher, proper, are those which access the relevant data, and which know something about the graph structure used. Extensive revisions are planned for the matcher.

6. CONCLUSIONS

The implementation phase has been underway for approximately one year. At the time of writing, ACRONYM had been tested on an example generated by hand from a real image and is being tested on an example generated completely automatically from real images. The High-level Modeler, the Graphics Generator and the Matcher have been tested extensively and are complete for the full-fledged test without further modification. The Predictor and Planner control scheme is complete - it remains to write more rules encoding the the details of transforming the Object Graph into an Observability Graph.

The Matcher has been tested on a hand coded Observability Graph, matching against a hand coded Picture Graph. This test was used during the debugging phase, and was designed to test all modes of operation of the Matcher. Thus the particular Observability Graph used is more complex than can be reasonably be expected to be generated by the Predictor and Planner, at least until we have had considerably more experience. The Matcher was programmed to perform scene labeling for a trihedral blocks world. The nodes of the handcoded observability graph described the possible junctions in a two dimensional line drawing, following the schemes of Huffman [15] and Clowes [10). The arcs constrained the interpretations of the two junctions defining a line to give it the same labeling.

The High-level Modeler has been used to construct a large number of models Figs 3 and 4 show output from the Graphics Generator of two objects modeled in the high level language accepted by the Modeler. Many additional models of both industrial parts and airport related scenes have been built with the system. The Modeler and Graphics controller are completely interactive, enabling the user to "fly" around airports or manipulate the robot arm.

The Predictor and Planner has been run on both the arm model displayed in fig. 3, and a generic model of an airport. Jt produced the complete node structure of the Observability Graphs in both cases, and included quantifiers *in* the airport case, to describe the numbers of allowable numbers of runways and taxiways in a valid airport instance.

In April 1979, the ACRONYM system was tested with a simplified model of an LIOII, and an aerial photograph of an L10II at San Francisco airport. It Identified an aircraft by locating the fuselage, and two wings without help This was a very preliminary coupling of many of the modules of ACRONYM (all were involved except the surface mapper which is not yet implemented). Much more development needs to be done to make the system robust.

Work is proceeding on a smarter geometric editor It will be a knowledge-based editor, which knows about the sorts of operations a user typically wants to perform It will thus be able to act as an intelligent assistant to a user. So far, neither the problems of the intersection of generalized cones, nor a full hidden surface algorithm have been solved because development of the vision system had higher priority than development of the modeling system. These problems will be tackled in conjunction with the smart editor.

The matcher that *is* implemented is very powerful, allowing complex quantifications in its input language The Predictor and Planner does not yet make full use of its capabilities We intend to look for a formal description of the matcher, perhaps modifying it to fit a clean definition of capabilities This will give a better handle on the type of rules we should write for the Predictor and Planner and give an estimate on the ultimate capabilities of the ACRONYM system as a whole

REFERENCES

[!} Agin, Gerald J, "Representation and Description of Curved Objects," Stanford AIM-173, Oct 1972.

[2] Arnold, R. David, "Local Context in Matching Edges for Stereo Vision," *Proc. ARPA Image Understanding Workshop,* Cambridge, May 1978,65-72.

[3] Ballard, D. H., C M Brown and J. A. Feldman, "An Approach to Knowledge-Directed Image Analysis," *Proc. IJCA1-77,* Cambridge, Aug. 1977, 664-670

[4] Barrow, Harry G and Jay M Tenenbaum, "MSYS: A System For Reasoning About Scenes", SRI AI Center, Tech. Note 121.. March 1976

[5] Binford, Thomas O., "Visual Perception by Computer," Invited paper at *IEEE Systems Science and Cybernetics Conference,* Miami, Dec 1971.

[6] Brooks, Rodney A.. "Goal-Directed Edge Linking and Ribbon Finding," *Proc. ARPA Image Understanding Workshop,* Palo Alto, Apr. 1979, 72-78.

[7] Brooks, Rodney A , Russell Grelner and Thomas O Binford , "A Model-Based Vision System," *Proc. ARPA Image Understanding Workshop,* Cambridge, May 1978, 36-44.

[8] Brooks. Rodney A., Russell Creiner and Thomas O. Binford, "Progress Report on a Model-Based Vision System," *Proc. ARPA Image Understanding Workshop,* Pittsburgh, Nov. 1978, 145-151.

[9] Bolles, Robert C, "Verification Vision For Programmable Assembly," *Proc. of IJCAI-77,* Cambridge, Aug. »977, 569-575.

[10] Clowes, Maxwell, "On Seeing Things," *Artificial Intelligence* 2(1971), 79-116.

[II] Davis, Randall, Bruce Buchanan and Edward Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program," Stanford AIM-266, Oct. 1975

[12] Garvey, Thomas D. "Perceptual Strategies For Purposive Vision," SRI AI Center, Tech Note 117, Sep. 1976.

[13] Goldman, Ron, "Rrcent Work with the AL System," *Proc. of IJCAI-77,* Cambridge, Aug 1977, 735-735

[14] Hollerbach. John, "Hierarchical Shape Description of Objects by Selection and Modification of Prototypes," MIT AI-TR-346, Nov 1975

[15] Huffman, David, "Impossible Objects as Nonsense Sentences," In *Machine Intelligence* 6, B. Meltrer and D. Michie eds., Edinburgh University *Press,* Edinburgh, Scotland, 1971.

[16] Kanade, Takeo, "Model Representations and Control Structures in Image Understanding," *Proc. of IJCAI-77,* Cambridge, Aug 1977, 1074-1082

[17] Man, D, "Analysis of Occluding Contour," MIT AIM-372, Oct. 1976

[18] Marr, D. and H. K Nishihara, "Representation and Recognition of the Spatial Organization of Three Dimensional Shapes," MIT AIM-377, Aug. 1976.

[19] Miyamoto, Eiichi and Thomas O. Binford, "Display Generated by a Generalised Cone Representation," *IEEE Conference on Computer Graphics and Image Processing,* May. 1975.

[20] Moravec, Hans P., Towards Automatic Visual Obstacle Avoidance." *Proc of IJCAI-77,* Cambridge. Aug 1977, 584.

[21] Nevatla, Ramakant, "Structured Descriptions of Complex Curved Objects for Recognition and Visual Memory," Stanford AIM-250, Oct 1971

[22] Nevatia, Ramakant and Thomas O. Binford, "Description and Recognition of Curved Objects," *Artificial Intelligence* 8(1977), 77-98

[23] Nevatia, Ramakant and K Ramesh Babu, "Linear Feature Extraction," *Proc. ARPA Image Understanding Workshop,* Pittsburgh, Nov.1978, 73-78.

[24] Rubin, Steven M., "The ARGOS Image Understanding System," *Proc. ARPA Image Understanding Workshop,* Pittsburgh, Nov. 1978, 159-162.

[25] Tenenbaum, Jay M. and Harry G. Barrow, "Experiments in Interpretation-Guided Segmentation," SRI AI Center, Tech. Note 123, Mar. 1976.

[26] Thomas, A. J. and T. O. Binford, "Information Processing Analysis of Visual Perception: A Review," Stanford A1M-227, Jun. 1974.

HIERARCHICAL REASONING IN THE GAME OF GO

David J H Brown
Computer Science Department
Teesside Polytechnic
Middlesbrough, Cleveland, England

The use of a semantic network to describe spatial and functional patterns at various levels of abstraction.. .is described. Particular attention is paid to the use of a network by a deductive problem solver in forming plans and generating moves in playing Go.

1. PATTERN HIERARCHIES

It has often been remarked of Go that it is essentially a visually oriented game! such terms as "good shape", "thickness" and "eye space" abound in the literature. Imagery plays an important role in the conceptualisation of a board position.

RAG - an ongoing effort to program Go by integrating plan formation techniques with a multilevel pattern organisation - represents Go images by means of a semantic network [5] . The nodes of the network denote
(a) organisational units such as points, strings, groups and territorial frameworks and
(b) relations between units in terms of their functional roles (support, influence, etc.) and topology (linkages, cuts, pincers, etc.). Relations are themselves units and as such may be the *parameters* of other relations. For example, "diagonal" is a linkage relation which can exist between two strings and "cut" is a relation which can exist between a string and a linkage relation.

The natural hierarchy and taxonomy of such units is embodied in the network by means of subset, element and part-of relations. In effect, the network provides a continuum of abstraction spaces in which relations are permitted to exist between units at different levels of abstraction.

Fig.2 illustrates part of RAG's representation of the initial situation depicted in Fig.I. The network is implemented using the property-list structure of LISP. Nodes depicted by rectangles are pre-defined atoms; others (such as "BS2" which denotes the string of black stones at Q4, Q3, R3, S3 and S2) are created dynamically. In the present implementation, network updates (as moves are played) are made on an ad hoc basis.

2. AN APPROACH TO REASONING

Because of the standardised representation provided by semantic nets, a uniform reasoning procedure may be employed to perform hierarchical planning. The rule-based inference system employed by RAG [1] is such a procedure. It utilises a set of packages of rules, each package containing rules likely to be applicable to a given class of goals. Packaging of rules is one way of obtaining plausible rules without the need to perform partial matches. This is especially important when rule matching is expensive; RAG's rules are represented as semantic nets, thus the matching problem is one of finding graph injections.

Each rule comprises a CONDITION, an ACTION and a GOAL. In forming plans, RAG matches its current goal with the GOAL of a rule. The CONDITION specifies contextual constraints under which the rule's ACTION should satisfy the instantiated GOAL.

In similar spirit to NOAH [4] , RAG searches at the top level (subgoaling on CONDITIONS) before proceeding to lower levels (subgoaling on ACTIONs). Further, only the temporally most proximate goal of a plan is searched below the current level. The rationale behind this is threefold: (1) It being implausible to consider all continuations, no "proof" could be found for the correctness of a play; (2) Actions which are likely to enable goals are valuable as threats even if counterplay is possible; (3) The inclusion of CONDITIONS in rules enables applicability requirements to be examined abstractly; although Go is

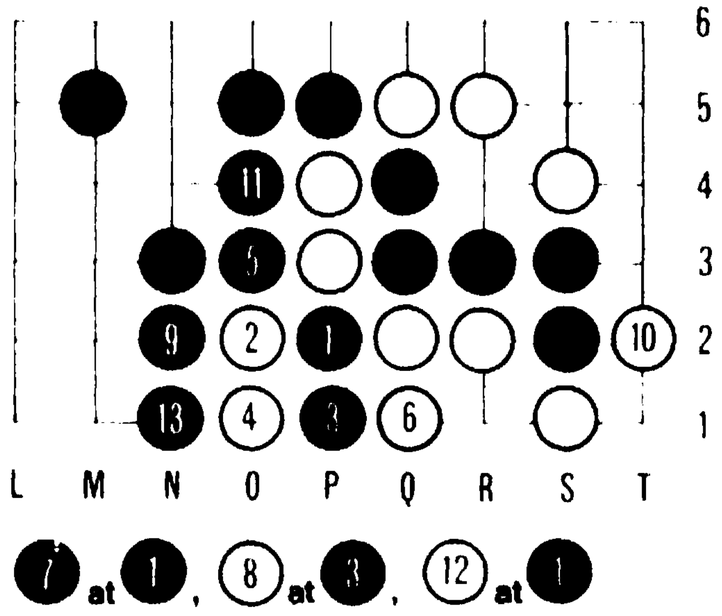


Figure 1: The Two-Stone Edge Squeeze

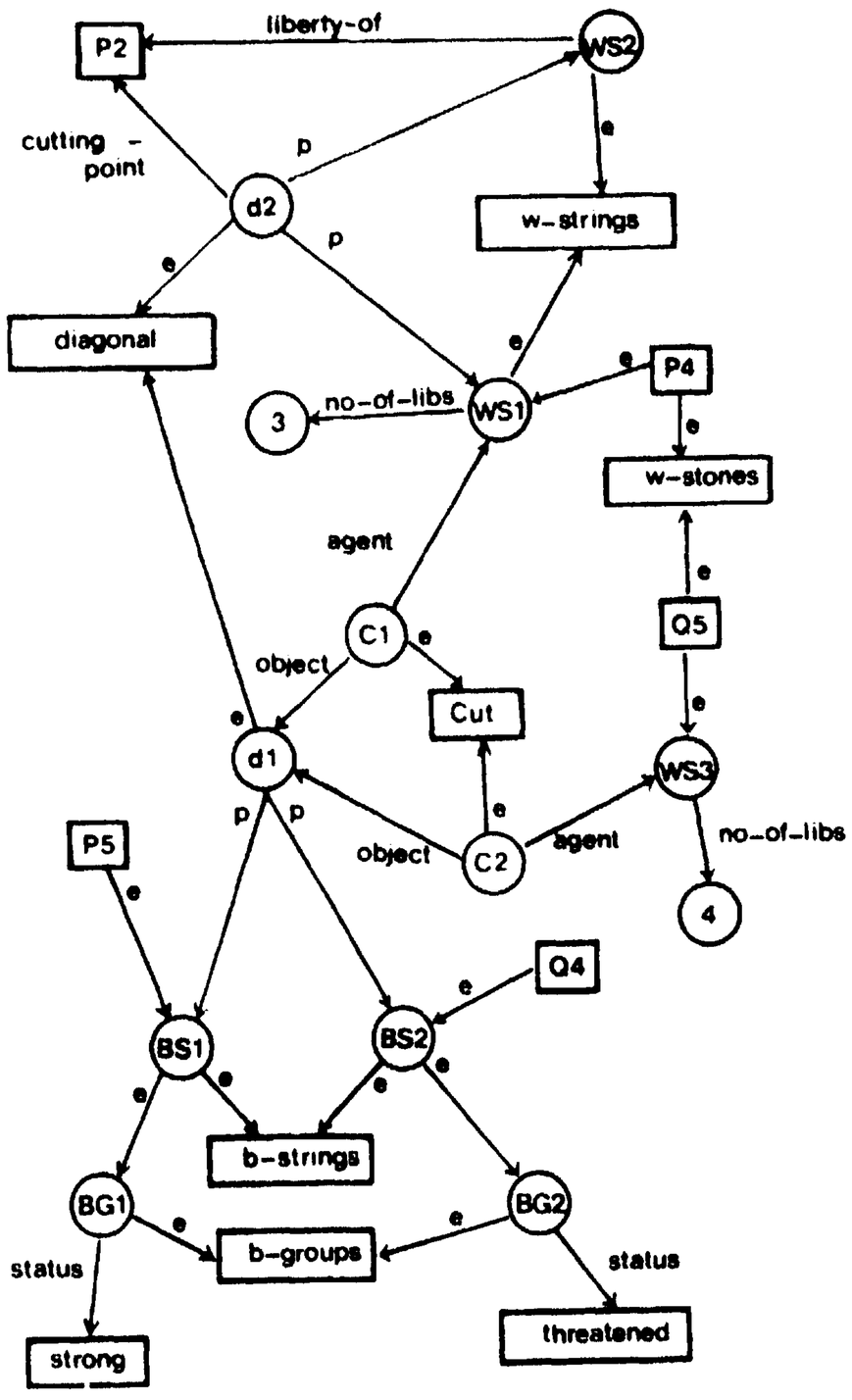


Figure 2: A Fragment of RAG's Representation of the Initial Situation (e=element, p=parameter)

theoretically a game of perfect information it must be treated as one of imperfect knowledge as future events cannot be definitely predicted. Consequently, situations must be judged at an abstract level.

3. LOOKAHEAD

The effectiveness of RAG's rules cannot be guaranteed. It is possible to represent the necessary conditions of a rule's applicability (by its GOAL) but its CONDITION can only partly specify the sufficient conditions, given that a practicable system cannot be enumerative.

At the move generation level, certain sequences can be specified for individual tactics. The representation of Go images by semantic nets enables sequences, such as that in Fig. 1, to be specified in terms of generalised moves. In the example, the first move is specified as:

(CUT (DIAGONAL ?W1 ?W2))

When fitted to the local context, the point P2 is established as the place to play to cut the linkage between WS1 and WS2.

RAG stores generalised move sequences in a database in which each sequence (tree) is indexed by the particular tactic (low-level ACTION) to which it relates. The responsibility for determining the likely relevance of tactics lies with the planning component.

4. SPECIAL TACTICS (TESUJI)

Tesuji are tactical plays known to be effective in certain types of situation. Such situations are characterised by combinations of features and relations in a local context. However, the variety of configurations that can have any one such combination is so large that template matching of stone patterns is infeasible.

Furthermore, each tesuji is inextricably linked with a goal; typical goals being to capture small groups of enemy stones to link up or make eye-shape, isolating weak enemy groups by cutting through linkages and so forth.

Thus the use of stone patterns, as in Zobrist's program [6] , suffers from two deficiencies:
(1) Prohibitively many patterns are required (Zobrist used about 200, which enabled his program to cope with only very elementary tactics); and (2) stone patterns are too primitive to be related to strategic considerations, ie. goals.

Attempting to solve the first of these, Ryder [3] developed a set of recursive definitions for certain forced sequences such as snap-backs and ladders. However, because his program lacked any strategic direction, it wasted many moves defending cuts or making eye-shape when it was unnecessary. The program of Reitman and Wilcox [2], with its multiple perceptual components, comes closest to the solution, but as yet suffers from a lack of strategic direction.

RAG is an attempt to solve both problems in a systematic manner. The planning component utilises rules which suggest various tesuji as a way of achieving certain goals. Each tesuji is passed to a move generator, which attempts to fit generalised sequences indexed by the tesuji into the current position. If a sequence fits, it is expected to achieve the desired goal.

As an example, consider the squeeze tesuji illustrated in Fig.1. This occurs in a race to capture between two eyeless groups. By sacrificing three stones, black can keep ahead in the race and squeeze the life out of white's cutting stones. The generalised move sequence black has comprises a cut (1), descent (3), cut of the newly-created diagonal (5) and throw-in (7) followed by removal of white's liberties.

In the example, the stones at P4 and P3 (WS1 in Fig. 2) are marked as cutting the diagonal connection between the threatened black group in the corner and the strong one on the outside. Q5 and R5 (WS2) are similarly marked, but they have too many liberties for the squeeze to work. Included among rules for saving threatened groups is the "two stone edge squeeze". This rule includes in its CONDITION the requirements? (1) None of the enemy strings connected to the one to be captured (including those connected by transitivity) has more than three liberties; (2) The string to be captured has a cutting point on the second line, with the point below it also vacant; and (3) The cutting point is unprotected (ie. a cutting stone would not be captured immediately).

Whilst there are a few configurations also characterised by these features to which the squeeze tesuji is not appropriate, their testing serves to eliminate countless others. Fitting the aforementioned generalised sequence to the situation in question serves to determine whether and how the original goal can be achieved.

5. SUMMARY

Semantic nets are a powerful re presentational forma I ism and are particularly suited to the description of spatial and hierarchical patterns such as occur in the game of Go. The use of a uniform representation for all levels of descriptive information enables a single deductive procedure to operate effectively in both strategic and tactical decision making.

The combination of graphical pattern matching and generalised sequence fitting enables the testing of low-level tactical goals to be performed, thus providing a verification mechanism for goals formulated at higher levels of abstraction.

ACKNOWLEDGEMENTS

Stu Shapiro first suggested the use of semantic nets for RAG. Don Beal, Tom Mitchell and Jerry Schwatrz provided useful insights.

REFERENCES

[I] Brown, D J H "Reasoning About Games". AISB newsletter 32 (1978) 14-17.

(21 Reitman, Wand Wilcox, B "Pattern Recognition and Pattern-Directed Inference in a program for Playing Go". In Waterman, D and Hayes-Roth, F (Eds) Pat*tern-Directed* Inference Systems. Academic Press,1978, "503-523.

[3] Ryder, K J "Heuristic Analysis of Large Trees as Generated in the Game of Go". AIM-271, Stanford University, 1972.

[4I Sacerdoti, E "A Structure for Plans and Behaviour". AITN-109, SRI International, 1975.

[5] Shapiro, S C "A Net Structure for Semantic Information Storage, Deduction and Retrieval". In Proc. IJCA-71 The British Computer Society, 1971, 512-523.

[6] Zobrist, A R "Feature Extraction and Representation for Pattern Recognition and the Game of Go". PhD Thesis, University of Wisconsin, 1970.

A LOGIC REPRESENTATION OF MANUFACTURING CONTROL

William I. Bullers
Krannert School of Management
Purdue University
West Lafayette, IN 47907

Shimon Y. Nof
School of Industrial Engineering
Purdue University
West Lafayette, IN 47907

Andrew B. Whinston
Krannert School of Management and Computer Science Dept.
Purdue University
West Lafayette, IN 47907

This paper explores some of the typical problems in manufacturing systems planning and control, particularly those pertinent to automatic operation, and describes how artificial intelligence methods can be applied. We demonstrate how predicate logic and theorem-proving techniques using resolution can be used in a manufacturing environment. Assertions of fact and axioms representing the knowledge required are given in an underlying data base. Illustrative examples demonstrate how user problems, such as assignment of Jobs to machines when conflicts occur, can be handled by a decision support system in the framework of resolution in a problem-reduction approach.

1. Introduction

Manufacturing planning and control involves managerial decision-making in three main levels of activities: the strategic level of production and inventory planning; the tactical level of manufacturing operations; and the operational control of processes and material flows. Computerized manufacturing systems (CMS), characterized by multi-purpose, computer-controlled machines connected by automated material handling networks, require a large number of timely decisions to be made at the various levels of operations. See [9] for the information and control aspects of such systems. The purpose of this paper is first to explore some typical manufacturing planning and control problems, particularly those pertinent to the automatic environment, and second, to describe how predicate logic problem-solving methods can be applied to them.

2. Predicate Logic Representation of Knowledge

First-order predicate logic has been used extensively as a knowledge representation language to describe states and problems. Moreover, as shown in *[4]*, predicate logic comprises a programming language for expressing algorithms. Axioms in Horn clause format are interpreted as procedures to be invoked by theorem-proving mechanisms based on Robinson's resolution principle [8]. Deductive query systems such as Chang's [2], and Minker's [5]combine predicate logic with a data base system for knowledge storage and retrieval. These systems use resolution to partially prove theorems, then invoke the data base system to complete the proof. In this paper we demonstrate how a predicate logic representation of knowledge integrated with a data base can be used to model control in a manufacturing environment for the purpose of solving operations management problems.

3. Manufacturing Planning and Control Needs

In the three levels of manufacturing activities previously defined, planning and control are required in order to achieve efficient, productive utilization of resources to meet a schedule of production demands. Strategic and tactical planning and control have been performed quite successfully with the aid of information systems, It is in the operational level of control in which a real-time decision support interface between managers and machines is lacking, particularly in the CMS environment.

Three operational control issues described in[7] include Part Mix, Part Entry and Assignment, and Process Selection. The Part Mix problem is to select particular sets of parts for sub-periods from the larger set of parts scheduled for production in the whole period. The Part Entry and Assignment problem is to sequence individual part entry into the system and assign parts to machines when conflicts occur. The Process Selection problem is to decide which of several alternative processes, all producing the same part but requiring different machines and processing times, should be selected for a given

part.

While off-line analyses using simulation indicate the importance of such control issues, a different approach is necessary when decisions have to be made and implemented in a real-time environment. In the following sections a new approach is developed, and examples given to illustrate how some of the issues described here are handled.

4. Predicate Logic Representation of Manufacturing Control

CMS entities and simple relationships among them are modeled by predicates, which are represented in a network data base system as described in [1]. Complex relationships among the entities are defined by axioms. The axioms extend the data base by supplying procedural knowledge about how the assertional component is used for control tasks.

Many manufacturing control issues require the system status at a given time. Typical inquiries concern what processes are available for each part, what conflicts exist among parts competing for machine servicing, etc. Queries of this type can be handled by problem-solving techniques utilizing a static representation. Generation of the current state of a CMS is done concurrently with actual operation of the manufacturing system by an "operational" mode of the decision support system. Problem-solving on the "operational data base" is performed using a problem-reduction approach [6], in which a problem is given by:

(1) a predicate logic goal statement describing the problem to be solved;
(2) descriptions of problems whose solution is known, primitive problems;
(3) a set of Horn clause axioms, problem-reduction operators, which transform problems into sub-problems.

A top-down proof procedure is applied to carry out resolution, deriving new goal statements from old ones by applying the problem-reduction operators. The primitive problems obtained are then solved by retrieval procedures that evaluate predicates representing assertions in the manufacturing data base.

Control issues in a dynamic domain require a "simulation" mode of the decision support system to forecast future states of the CMS. The predicate TIME(t) is used to denote time, and a state-space approach [6]is used to update the system. The representation of a problem consists of:

(1) the operational data base of assertions denoting the initial state;
(2) a predicate logic statement of a goal state;
(3) a set of Horn clause axioms, state-space operators, to transform 6tates into other states.

A bottom-up proof procedure is applied to derive new assertions from old ones. The operators utilize an ADD and DELETE list similar to that used in STRIPS [3] to handle secondary implications of an operation in addition to the consequent in the Horn Clause axiom. The assertions generated for the "simulation data base" are then used as newly solved primitive problems in a top-down theorm-proof. Comparison of criteria across different simulated states determines operational parameters that should be used in the CMS operation over the forecast horizon.

5. Illustrative Manufacturing Control Problems

Table 1 shows a partial list of predicates needed to represent the machines, parts, operations, etc., and their relationships in a CMS.

Table 1

Computerized Manufacturing System Predicates

Predicates	Interpretation
MCH(mch)	machine "mch"
OPERATION(op)	operation "op"
PART(part,part-type)	part
COMPLETE-PART-OP (part,op,t)	part/op complete at t
MCH-OP(mch,op,op-time)	mch/op process time
PART-ROUTING (part-type,op,next-op)	part routing

The complex relationships are specified by axioms. Problem-reduction operators for the Part Assignment problem determine (1) if the first operation for a part is next to be performed, (2) if a subsequent operation is next, and (3) a possible schedule of a part to a machine for an operation. These three axioms are defined as:

```
(A1) ∀x ∀y ∀z ∀w
  NEXT-PART-OP(part=x,op=y) ←
    PART(part=x,part-type=z),
    OPERATION(op=y),
    PART-ROUTING(part-type=z,op=y),
   -PART-ROUTING(part-type=z,op=w,next-op=y)
   -COMPLETE-PART-OP(part=x,op=y)
(A2) ∀x ∀y ∀z ∃w
  NEXT-PART-OP(part=x,op=y) ←
    PART(part=x,part-type=z),
    OPERATION(op=w),
    OPERATION(op=y),
    PART-ROUTING(part-type=z,op=w,next-op=y)
    PART-ROUTING(part-type=x,op=y),
    COMPLETE-PART-OP(part=x,op=w),
   -COMPLETE-PART-OP(part=x,op=y)
(A3) ∀x ∀y ∀z
  POSSIBLE-SCHEDULE(part=x,mch=y,op=z) ←
    NEXT-PART-OP(part=x,op=z),
    MCH-OP(mch=y,op=z)
```

With these axioms, we can solve a Part Assignment problem such as: "On what machine can part #p2 be scheduled for its next operation?" This problem, expressed as the goal statement:

$\exists_y \exists_z$ POSSIBLE-SCHEDULE(part=#p2,mch=y,op=z)

is reduced to a set of primitive problems by invoking axiom A3, then A1 or A2. A unique "y,z" pair satisfying the primitive problems solves the Part Assignment problem. For multiple "y,z" occurrences, additional axioms specify the particular machine-operation pair to be chosen.

For simulation, a set of state-space operators describe the operation of the manufacturing system. For example, the end of a machine's operation performance is defined by:

(A4) $\forall_x \forall_y \forall_z \exists_{t_1} \exists_{t_2} \exists_{t_3}$

END-MCH-OP(mch=x,part=y,op=z,t=t_3)←
INIT-MCH-OP(mch=x,part=y,op=z,t=t_1),
MCH-OP(mch=x,op=z,op-time=t_2),
ADD(addend1=t_1,addend2=t_2,sum=t_3),
delete list: INIT-MCH-OP(mch=x,part=y,op=z)
CURRENT-PART-OP(part=y,op=z)
add list: MCH-IDLE(mch=x),
COMPLETE-PART-OP(part=y,op=z,t=t_3)

Using such axioms, we solve a problem useful in Process Selection: "Given a shortest processing time scheduling rule, when will part #p2 complete its next operation?" The goal statement:

$\exists_z \exists_w$ NEXT-PART-OP(part=#p2,op=z),
COMPLETE-PART-OP(part=#p2,op=z,t=w)

is solved by first invoking the simulation mode of the decision support system. A bottom-up theorem-proof derives an assertion of the form:

COMPLETE-PART-OP(part=#p2,op=#op_z,t=#t_w)

in the simulation data base. Axiom A1 or A2 is then invoked in a top-down manner to reduce the NEXT-PART-OP sub-problem. Finally, the primitive problems are solved by data base retrieval. Simulating the completion time for a number of alternative scheduling rules enables us to select the rule that provides the best value of a criterion measure of performance for completion of a number of part operations over the simulated time horizon.

6. Conclusions

In this paper we have explored the application of predicate logic and mechanical theorem-proving to the management of a computerized manufacturing system. Predicate logic permits a descriptive non-procedural representation of knowledge related to the operation of the facility. The predicate calculus language allows both static and dynamically changing knowledge to be modeled, while theorem-proving permits "new" knowledge to be deduced from the current knowledge base. From a decision support viewpoint, the resolution procedure allows the knowledge base to be applied to a user's specific manufacturing control problems.

Further work is needed to implement such a system. Questions on how axioms should be stored and how resolution can be effectively carried out must be investigated. From a user's point of view, a more natural, English-like problem statement language is needed. In addition, relationships between the problem-solving techniques described and classical integer programming models of scheduling would be of interest.

ACKNOWLEDGMENTS

This work supported in part by the National Science Foundation (grant MCS 76-24675 for Computerized Decision Support Systems). Mr. W. Bullers gratefully acknowledges the fellowship support made possible by a grant from IBM.

REFERENCES

[1] Bonczek, R., Holsapple, C. and Whinston, A., "The Integration of Network Data Base Management and Problem Resolution", (to appear in Information Systems),

[2] Chang, C. L., "EEDUCE 2: Further Investigations of Deduction in Relational Data Bases", IBM Research Report RJ 2147 (29^10), May, 1978.

[3] Fikes, R.E. and Nilsson, N.J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", Artif. Intell. 2. 1971, p. 189-208.

[4] Kowalski, R., "Predicate Logic as Programming Language", Proc. IFIP 74 , 1974, p. 569-574.

[5] Minker, J., "An Experimental Relational Data Base System Based on Logic", Logic and Data Bases , (Gallaire, H. and Minker, J., eds.), Plenum Press, New York, 1978, p. 107-147.

[6] Nilsson, N.J., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, N.Y. ,

(7] Noi, S.Y. , Barash, M.M. and Solberg, J.J. , "operational Control of Item Flow in Versatile Manufacturing Systems", Proc. 5th Int'l. Conf. on Production Research, 13-16 August, 1979, Amsterdam, The Netherlands.

[8] Robinson, J.A., "A Machine-Oriented Logic Based on the Resolution Principle", Journal of The ACM, 12:1, January 1965, p. 23-41.

[9} Talavage, J.J. and Barash, M.M., "Information and Control in Computerized Manufacturing Systems", Proc. Int'l. Fed, of Automatic Control Conf., Tokyo, Japan, October, 1977.

QUESTION-ANSWERING SYSTEM OF CHARLES UNIVERSITY

Eva Buranova', Svatava Machova, Bohumil Miniberger, and Antonin Riha

Computational Centre, Charles University
Prague, Czechoslovakia

ABSTRACT

This paper describes QUASCU (Question Answering System of Charles University), a system being used to investigate the use of natural language in data-base queries.

The analysis in the QASCU is essentially a syntactlco-semantlc one, its objective being (i) to extract from the query formulated in Czech information which, in the semantic field in question, is sufficient for correct response, and (ii) to translate this information into the query language. Knowledge of the QASCU is stored in a hierarchically organized data base. The semantic field in which the first experiments have been carried out is constituted by the enrolling procedure of students for all the colleges of Charles University.

The programmes of the QASCU are written in the PL/1 language and the system will be implemented on the EC 1040 computer at the Computational Centre, Charles University, Prague, Czechoslovakia.

*Paper not received in time to appear in full in Proceedings

COMPUTER MODELS OF HUMAN PERSONALITY TRAITS

Jaime G. Carbonell
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

This paper presents a goal-based analysis of human personality traits. A process model for automated understanding and interpretation of personality traits is developed. The model Is based on the goal-tree and planning/counterplanning strategies developed to represent subjective beliefs In the POLITICS system. The automated analysis of personality traits plays an integral role in natural-language story-understanding processes and In computer models of human social interactions.

1. Why Analyze Personality TraitaT

Understanding stories requires information and reasoning about the situation, the causal structure of the events, and the characters In the story. Schank [1], Cullingford [2], Rumelhart [3], and Beaugrande *and* Colby [4] have analyzed the narrative structure of stories and developed means of automating the analysis process. Schank and Abelson [5], Wilensky [6], and Schmidt and Sridharan [7] developed means of inferring the goals and plans of the characters in a story from their actions. Both the narrative structures and the goals and plans of the characters *are* crucial in integrating the information contained in stories into a coherent memory representation. Such memory structures are necessary to answer questions about the story in much the same way that people appear to reason about the stories they read.

Character development, however, is *an* important aspect of story understanding that has been largely ignored by Artificial Intelligence researchers. A person reading a story Identifies with one or more characters depending on whether the cha-acters are heros, villains, compassionate, intelligent, unscrupulous, etc., and depending on how the character's personality relates to the reader's self-image and to other people he knows in real life. Furthermore, knowledge of the characters and their personality helps to interpret their actions and induce their goals. Thus, understanding character development is an integral part of processing natural language stories. Here we deal with the most simple form of character development: the attribution of personality traits to actors in simple stories. We analyze personality traits in terms of personal goal trees and predispositions towards applying certain classes of planning and counterplannIna strategies. Goal trees and counterplanning strategies were developed to model Ideological beliefs in the POLITICS system [8]

2. What Information Do Personality Traits Convey?

Consider an example of personality-trait attribution In the following story.

> Bill was very brave, but his brother John was very cowardly. One night the two brothers were walking by the road when a masked bandit surprised them. The younger brother panicked and ran headlong into the forest where he was lost, never to be seen again. The elder brother fought off the bandit, and, in the process, recovered the long lost royal sapphire, stolen years earlier. The king rewarded him handsomely.
>
> QUESTION: Whom did the king reward?

A person reading the above story has little trouble in answering the question: Clearly, the king rewarded Bill. However, it is not particularly easy to see how one goes about formulating the answer. No simple rule will serve. For instance, the last mentioned character in the story before the word "him" is the bandit, but this is obviously not the correct referent. The first step in resolving the referent requires one to understand why the king gave the reward, thus establishing that the elder brother who recovered the sapphire received it. This, however, is only half of the task. How do we know that Bill is the elder brother who deserves the reward?

In order to determine which brother is which we must use the information contained in their respective character traits. One brother is brave; the other is cowardly. Running away in the face of danger is a characteristic behavior associated with cowardly people. Fighting bandits, or otherwise risking one's life for a worthy cause is the type of behavior characteristic of *bravery.* Therefore we determine that Bill, the brave one, must have been the elder brother who fought the

bandit and recovered the sapphire. This determination requires knowledge about some types of actions that are characteristic of bravery and other actions that are characteristic of cowardice. Thus, we *need* to know, or be able to infer, typical behaviors associated with certain character traits. We *need* to answer the general question: If actor X has character trait P, is he likely to do action A in situation S? It seems, therefore, that an investigation of personality traits and their associated typical behavior ought to be a worthwhile pursuit.

3. Goal Trees Representing Personality Traits.

One way of analyzing personality traits is by associating with each trait the goals people described by that trait are likely to have. Once these goals *are* established, certain behaviors can be inferred. Since we have developed mechanisms for understanding goal based events (e.g., PAM [6] and POLITICS [8]), it seems quite fruitful to reduce personality traits to the pursuit of certain types of goals. For the present discussion we borrow three general classes of goals from the Schank and Abelson goal taxonomy [5]: acquisition goals (A-goals), preservation goals (P-goals) and enjoyment goals (E-goals).

Consider the process of understanding a story starting with the following initial segment:

> John is a *very* inquisitive and uncompromising person. He is also rather thrifty in his personal affairs...

There have been no actions thus far in the story, nor any physical or temporal setting that helps the understander establish the situational context. Yet, John's personality traits provide a goal-expectation setting. That is, the understander knows the following information from the above fragment of 6: John's goal of increasing his knowledge about most matters is a goal of *very* high importance. We denote the acquisition of knowledge goal as A-know(John,X,+). (The "+" means John wants knowledge about X. A "-" would signify that John's goal is to actively avoid knowing about X, and a "0" signifies that John ignores new knowledge about X. Thus, if we know that Mary is apathetic, we mean A-know(Mary,X,0).) The fact that John is thrifty tells us that he also has the goal of preserving his money. In fact, the word "thrifty" states a relationship between the P-money goal and the set of A-goals that can be accomplished by spending money. John holds the goal of P-money to be more important than most such A-goals.

Since most personality traits describe deviations from a culturally-defined normative person, we know that John's A-know goal is much more important to him than other people's A-know goals are to them. Similarly, we know that his P-money goal is a little more important to him then is generally the case. We may also infer that John's A-goal of things that cost money, may be a tittle less important to him than other people's corresponding A-goals are to them. The trait "Uncompromising" exemplifies an across-the-board deviation from the norm. John will give higher than normal importance to most of his goals. Using this information, we can build a fragment of John's relative importance (RI) goal tree:

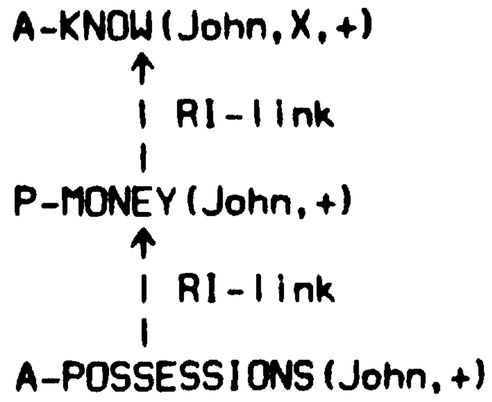


Figure 1: Fragment of John's RI goal tree.

Figure 1 tells us that of the three goals that we know John to have, he considers acquisition of new knowledge as most important, followed by preserving his money, followed by acquiring new material possessions. Since we know that John is a person and a member of western society, we know that he has certain normative goals common to most people in the society. The following list includes some of the prototypical person's goals, arranged in a relative-importance hierarchy in figure 2.

GOAL	EXPLANATION
1) P-health(Self,+)	Self-preservation
2) P-health(Family,+)	Preservation of family
3) A-possessions(Self,+)	Acquire wealth and belongings
4) P-possessions(Self,+)	Preserve one's belongings
5) A-scont(Self,others,+)	High social standing
6) A-know(Self,X,+)	Learn new things
7) E-unpleasant activity(Self,-)	Avoid going through unpleasant experiences
8) E-pleasant activity(Self,+)	Have fun doing enjoyable things
9) P-health(others,+)	Philanthropy
10) P-anything(enemies,-)	Wish doom upon one's enemies

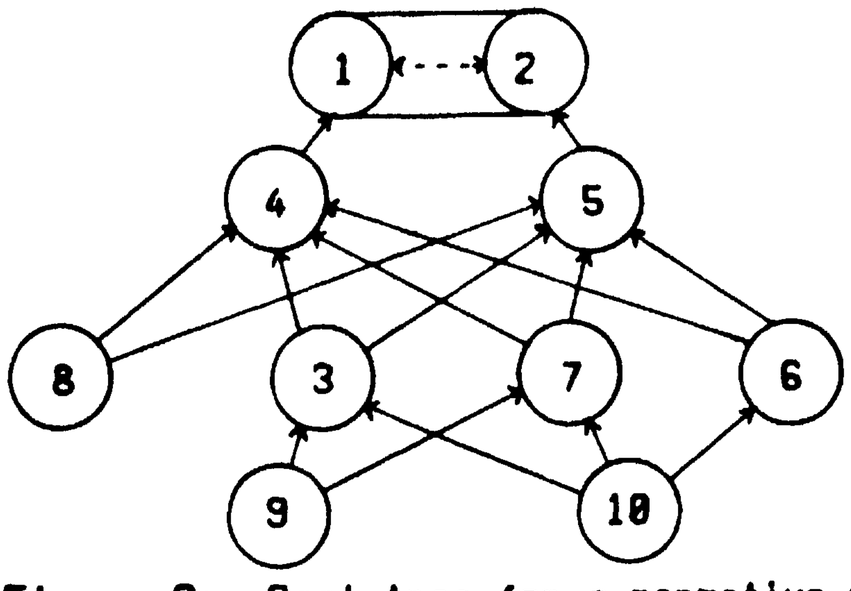


Figure 2: Goal tree for a normative person.

Many personality traits can be represented as deviations from a person's normative goal tree. Table 1 lists a small sampling of such traits and the changes in personal goals that they imply. For instance, "ambition" means that a person holds his A-scont and A-wealth goals as more important, while considering the goals of other people as less important. "Compassion" implies the opposite. Other personality traits refer to the means that a person is willing to use in order to achieve his goals. A more thorough analysis can be found in [8}

Table 1 PERSONALITY TRAIT	GOALS AND THEIR IMPORTANCE (deviations from the socially accepted norm)	
1) Ambitious	A-possessions(self,+)	higher
	A-scont(self,others,+)	higher
	P-anything(others,+)	lower
2) Inquisitive	A-know(self,+)	higher
3) Prudent	P-anything(self,+)	higher
4) Spendthrift	P-money(self,+)	lower
	E-anything(self,+)	higher
5) Vindictive	P-anything(ones who cause goal failure,-)	higher
6) Compassionate	P-health(others,+)	higher
	E-anything(others,+)	higher

A person's goal tree, modified by his personality traits, is used to predict which goals he will strive for in a given situation. Since relative-importance links are transitive, the following rule summarizes the function of personal goal trees:

> If a course of action affects two goals, and no other rules determine which goal to focus on, the effect on the higher-importance goal determines whether the course of action should be pursued.

We developed a set of heuristics for combining personality traits. Consider a person described by traits A and B, where A and B are defined in terms of their deviations from the normative person. A and B consist of a list of attribute-rank pairs. An attribute is the name of a goal. The rank encodes the positive or negative deviation from normative position of that goal in the goal tree. If A and B contain no goals in common, their combination is simply the union of the attribute-rank pairs, otherwise the following heuristics apply:

> CONTRADICTORY TRAITS - If both rankings have a high magnitude, but opposite sign, the two traits cannot be combined. (e.g. A generous miser, and a cowardly brave person *are* instances of contradictory traits.)

> REINFORCEMENT OF EXTREMES - If both rankings have a high magnitude and the same sign, assert the attribute with a ranking slightly larger than the maximum of the two original rankings, (e.g., An unscrupulous, vindictive person is more likely to violate other people's goals than someone who is merely vindictive, or just unscrupulous.)

> DAMPENING MINOR VARIATIONS - If the magnitude cf both rankings is small, but the signs opposite, delete this attribute from the combined trait, as it is of little importance and uncertain consistency.

> PREFERENCE TO EXTREMES - If none of the above rules apply, average the two ratings, but give greater weight to the rating with the higher magnitude.

The heuristic rules were empirically derived by analyzing many personality traits into their component attributes and recombining them in different ways. The analysis of personality traits is part of a larger research effort into subjective understanding of stories and discourse.

1 Schank, R. C. *Conceptual Information Processing.* Amsterdam: North-Holland, 1975.

2. Cullingford, R. *Script Application: Computer Understanding of Newspaper Stories.* Ph.D. Th.f Yale University, 1977.

3. Rumelhart, D. E. Notes on a Schema for Stories In Bobrow, D.G. and Collins, A., *Representation and* Understanding, New York: Academic Press Inc, 1975, pp. 211-236.

4. Beaugrande, R. D. and Colby, B. N. Narrative Models of Action and Interaction. *Cognitive Science* 3, 1 (1979), 43-66.

5. Schank, R. C. and Abelson, R. P. *Scripts, Coals, Plans and Understanding.* Hillside, NJ: Lawrence Erlbaum, 1977.

6. Wilensky, R. *Understanding Coal-Based Stories.* Ph.D. Th„ Yale University, 1978.

7. Schmidt, C, Sridharan, N. and Goodson, *X* The Plan Recognition Problem. *Artificial Intelligence* 11,2(1977). 45-83.

8. Carbonell, J. G. *Subjective Understanding: Computer Models of Belief Systems** Ph.D. Th., Yale University, 1979.

THE COUNTERPLANNING PROCESS: REASONING UNDER ADVERSITY

Jaime G. Carbonell
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

A heuristic model of planning in conflict situations is presented. This strategy-based model accounts for dynamic plan adaptation and replanning in obstructive and constructive counter planning situations. The former situation is characterized by an actor striving to thwart the goals and plans of a second actor. The latter, is the dual situation} it provides general means for an actor to pursue his goal in spite of attempta by others to block his plans. The model has been implemented as part of the POLITICS system, a computer progrem thet understands simple natural language accounts of International political conflicts.

1. Introduction

Planning and problem solving have been active research topics in Artificial Intelligence (e.g., GPS [1] STRIPS *[2]* and NOAH [3]). However, most research efforts only considered one-actor situations, with no adversaries and with a static world model. Plan-understanding systems [4, 5, 6] have also been largely confined to planning scenarios where actors do not scheme to thwart each other's plans while planning to achieve their own goals in light of possible external interference.

We focused our research on situations where the decision maker has to contend with other parties that may be continuously trying to thwart his efforts. In such circumstances, plan formulation and decisions of how and when to implement the plans become much more complex then in simple one-actor planning situations. The *primary* problems that must be addressed In such situations include the non-deterministic nature of planning under uncertain outcomes and changing circumstances, predicting the most likely actions on the part of potential adversaries, formulating alliances with other parties that have common goals, and compromising with one's adversaries when necessary. We discuss part of our theory of counterplanning a*nd* its implementation in POLITICS, a computer program that models ideologically-oriented decision-making tasks in the domain of international diplomacy and political confrontations.

This reserch wee eponiored in pert by the Defence Advanced Research Projects Agency (000), Monitored by the Office of Naval Reeeerch under contract N00014-75-C-1111, and in part by ARPA Order No. 3597, monitored by the Air Force Avionice Leborotory under Contract F33615-7a-C-15II. The views end conclusions contained in this document ere thoee of the author, end should net be interpreted ee representing the official policies, either extpreseed or implied, of the Defense Advonced Reeeerch Project! Agency or the US. Government

2. The POLITICS System

The POLITICS system [7,8]) understands a natural language description of a conflict situation by abstracting its meaning in terms of contextual knowledge of similar conflict situations (encoded as scripts [4, 9]), determining how the situation affects the goals of each political actor, and later applying counterplanning strategies to model the decisions open to the political actors under the circumstances defined by the input conflict situation. POLITICS evaluates possible decisions in terms of their effects upon the goals of the political actor, their likelihood for success (e.g., whether the planned decision cen be implemented, or whether it is likely to be blocked by an opposing political actor.)

In the following example the POLITICS program analyzes a political event first from the point of view of a conservative American policy maker, and later from a liberal viewpoint, to answer some questions illustrating the goal-oriented counterplanning process.

♦(INTERPRET US-CONSERVATIVE)
INPUT TEXT: The United States Congress voted to fund the Trident submerine project

0 Whet did the US Congress do thie for?
A: THE UNITED STATES CONGRESS WANTS THE UNITED 8TATES ARMED FORCES TO BE STRONGER

Q. Why should the US he stronger?
A' THE UNITED STATES ARMED FORCES SHOULD BE STRONG TO STOP COMMUNIS! EXPANSION

0 Whet should tho US do if Russia builds nuclear submarines?
A: THE UNITED STATES SHOULD BUILD MORE SUBMARINES

'(INTERPRET US-LIBERAL)
INPUT TEXT* The United States Congress voted to fund the Trident submarine project

Q< What did the US Congress do (hit for?
A- THE UNITED STATES CONGRESS FEARED FALLING BEHIND IN THE ARMS RACE

Q: What should the US do if RUM* builds Submarine?
A. THE UNITED STATES SHOULD NEGOTIATE WITH RUSSIA TO STOP THE ARMS RACE

In processing the above example, POLITICS inferred that the US wilt build the new weapon system, and that this is an important event whose consequences should be futher examined. At this point the liberal a*nd* conservative interpretations diverged. The liberal analzyed the negative implications towards peace, whereas the conservative focused on the beneficial effects of trident submarines to the US-Soviet strategic balance. Counterplanning is illustrated in the answer to the last question in each interpretation. In the conservative interpretation, POLITICS focused its counterplanning on how to keep the Soviet Union from becoming stronger than the United States. It applied a mutual-exclusion strategy: only one country can be the strongest military power. Therefore, if the US cannot stop the Soviets from building submarines, it should at least build a larger number of submarines and thereby remain stronger. The liberal interpretation focused counterplanning on how to end the arms race, as it considers this to be the most serious consequence of the arms race. Liberal POLITICS believes that both the United States and the Soviet Union want world peace more than military superiority. Liberal POLITICS applied a strategy that states that if a continued bilateral conflict violates more important goals of the conflicting parties, then they should jointly work to terminate that conflict.

The questions answered by the POLITICS system require en inference process that relates the input event to factual information contained in memory, as well as a decision-making process whose focus is to further the goals of the United States as perceived by a conservative (or liberal) American policy maker. Although the POLITICS system is essentially an integrated understanding system, it can be divided conceptually into several modules, including: 1) Natural Language understanding and generation in a semantically-rich domain. 2) A process to focus attention and thereby constrain inferences based in the goal hierarchy representation of political ideologies. 3) A script and situational inference rule applier. 4) A reasoning system based on our heuristic model of counterplanning. This paper focuses only on the counterplanning process, a significant aspect of human reasoning that has not been heretofore directly addressed In Artificial Intelligence.

Planning and Counterplanning require self-knowledge of the goals that one strives for, and at least partial knowledge of the goals of other actors with whom one Interacts. In POLITICS, the goals of all relevant actors form the bulk of the ideological-belief model. For instance, a model of a conservative American asserts that the primary Soviet goal is world domination, and that the primary US goals a*re* communist containment and a strong national defense. The POLITICS model of a US liberal, however, states that the primary US and Soviet goals are the maintainment of world peace, and that humanitarian goals are more important than a strong defense. Because of the varying sets of goals and the different priorities among the goals, POLITICS is able to model different political ideologies using the same reasoning process. This reasoning process consists primarily of the counterplanning heuristics focused on fulfilling the Ideological goals.

3. The Counterplanning Control Process

The counterplanning process is encoded as a set of heuristic strategies applicable to general conflict situations, and a control-flow algorithm that determines when to apply the various classes of counterplanning strategies. We first discuss the control-flow algorithm; later we analyze the structure and content of the heuristic strategies.

There are two general types of conflict situations where an actor may apply the counterplanning process. The first situation is characterized by an actor (X) trying to thwart another actor (Y) from achieving his goal G(Y). X may prevent Y from achieving G(Y) by directly making the goal state impossible to achieve, or by repeatedly blocking Y*s plans to fulfill G(Y). We call this process obstructive counter planning. The second type of counterplanning scenario is essentially the dual of the first type: An actor X is trying to achieve his goal G(X) in spite of attempts by Y to prevent C(X) and to block X's plans for pursuing G(X). This process, called constructive counterplanning. differs from obstructive counterplanning only in terms of the subjective perspective of the counterplanner. The perspective shift causes the counterplanner to apply different strategies at different times, as the application of strategies is goal driven. Figure 1 is a control-flow diagram for the obstructive counterplanning process.

To illustrate the obstructive counterplanning mechanism, consider a simple example. Prison guard X wants to prevent *prisoner* Y from escaping. Hence, GXY) is Y being free outside the prison. We enter figure 1 at the top. Does the guard know the prisoner's escape plan? Let us assume that he does not. His next step is to determine what, if any, plan the prisoner formulated. This plan determination may itself involve some planning: Should the guard ask the prisoner?, Should he threaten him? (Schank and Abelson [4] discuss the social planning units). Let us again assume that the guard fails. At this point we enter the third box in figure 1. The guard can ask himself "what would I do if I was trying to escape?" If he finds a reasonable plan, he should assume that this may be the prisoner's plan a*nd* he should apply the obstructive counterplanning strategies to block the plan. For Instance,

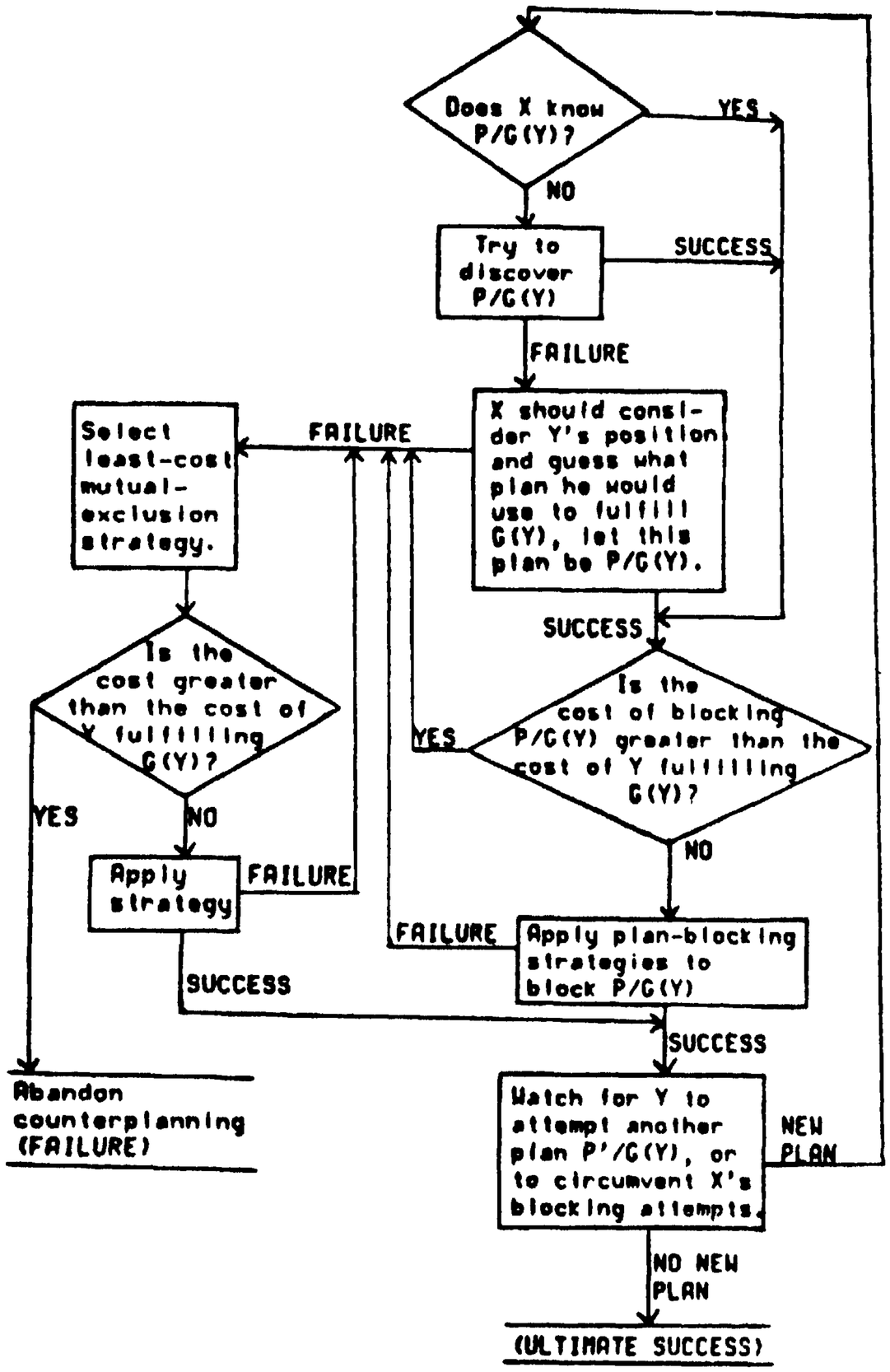


Figure 1: Control structure of the Counterplanning process (obstructive phase). Actor X attempts to block actor Y from achieving his goal G(Y). P/G(Y) is Y's plan for fulfilling G(Y).

the guard may find that stealing the key is a reasonable plan, in which case he may apply the violate-necessary-precondition strategies (discussed later) to conclude that he should keep the keys away from the prisoner.

If no plan presents itself, the only option open to the guard is to take general precautions (i.e., apply mutual exclusion strategies) such as pointing a gun at the prisoner and informing him that either he remains put, or he will be dead. Thus escaping and staying alive become mutually-exclusive states in the prisoner's mind.

There ere exit conditions in the counterplannlng algorithm. For Instance, If the prisoner's plan is to blow uo the prison (and he hes the means to do so), the *guard* may decide that the risk of being blown up is more costly then his goal of thwarting the prisoner. Similarly, if he has to keep a gun trained on the prisoner for the length of the prison sentence, the guard may decide that this Is more costly than letting the prisoner escape.

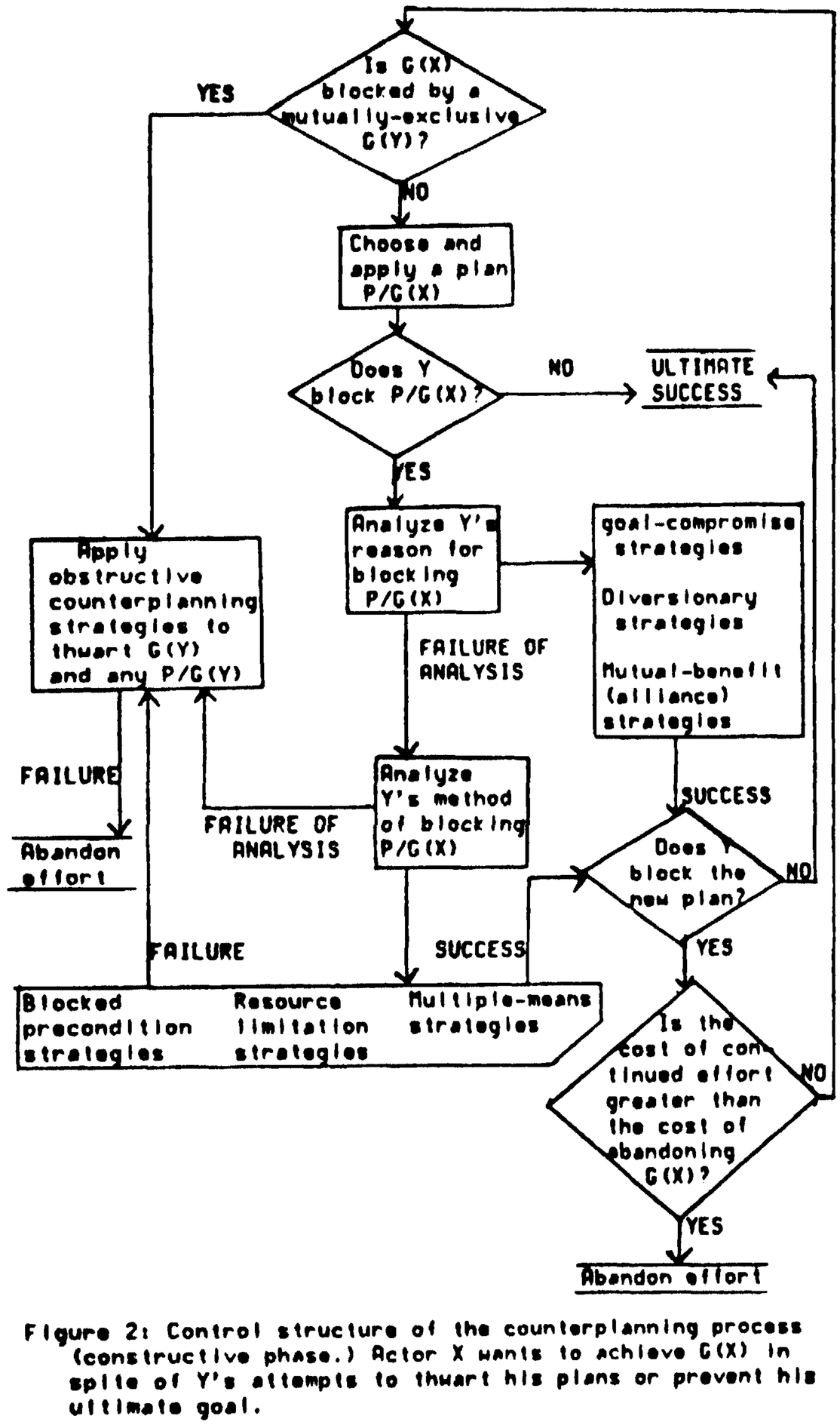


Figure 2: Control structure of the counterplanning process (constructive phase.) Actor X wants to achieve G(X) in spite of Y's attempts to thwart his plans or prevent his ultimate goal.

Figure 2 is the control-flow for constructive counterplenning. Using the constructive counterplannlng algorithm one can trace the options available to the prisoner for formulating an escape plan taking Into account the guard's possible thwarting efforts.

4. The Structure of the Counterplanning Strategies

Our model of counterplanning is essentially a goal-directed rule-based system. Each strategy is a rule that tests the goal conflict state between the two actors, the goal trees of each actor, and the plan that each actor is pursuing. If the test of a rule is true, the action part of the rule suggests a counterplanning method that is likely to succeed. The action part of the rule may have additional conditions or refinements. The structure of a counterplanning strategy is illustrated below.

```
STRATEGY <rule#x>   <title>
      TRIGGER  Conjunction of <test1, test2, ..., testn>
           IF  Conjunction of <testn+1, testn+2, ...>
         THEN  Sequence of <action1, action2, ...>
   REFINEMENT <subrule#x1, subrule#x2, ...>
```

Figure 3: The structure of a counterplanning strategy.

The test part of each rule Is divided into the -TRIGGER" clause and an additional "IF" clause. In order for a rule to apply, both clauses must be true in the counterplanning situation. The reason for our division of the test clause in such a manner is to reduce the time that the process model spends searching for an applicable strategy. All the trigger conditions are "inexpensive" tests: these tests can be applied directly to the situation without requiring further inference or complicated matching. These trigger conditions are compiled into a discrimination network. This organization allows the addition of new strategies without a corresponding linear increase in the search time required to test all the applicable counterplanning strategies.

Once the trigger conditions for a rule have been met, the additional "IF" tests are performed. These tests may be arbitrarily complex, perhaps requiring further inference and the invocation of other counterplanning strategies However, the trigger conditions usually restrict the set of applicable counterplanning strategies to a small number In any given situation (typically three or four). Furthermore, counterplanning strategies reflect common-sense reasoning about how to deal with adverse situations. As such, each strategy is sufficiently general to apply across many reasonable types of human conflict. This means that the total number of counterplanning strategies is relatively small compared to the total number of rules and information contained in the situation-specific scripts. We have found that approximately forty counterplanning strategies suffice to model the counterplanning actions in the situations and goal conflicts we have considered.

The action part of each strategy (preceded by "THEN") is a sequence of counterplanning methods to be applied in the current context by one of the actors. If a strategy is applied, the "refinement" field is checked after the sequence of actions is performed. The refinement contains one or more additional rules that usually provide further detail to the counterplanning situation. These rules are truly subrules to the counterplanning strategies because they are invoked only in the case that all the tests of the strategy are true, and, in addition, the test clause of each subrule is also true. In structure, our strategies are much closer to the rules in expert systems (such as PECOS, [10]) than to more traditional production systems (e.g., the PSG system [11]).

The following sections analyze some of the more significant types of counterplanning strategies. More complete sets of strategies are discussed in [7, 8]. We also discuss the way in which counterplanning strategies are used in text-understanding and question-answoring tasks.

5. Diversionary Counterplanning Strategies

We turn to some of the more specific counterplanning strategies applicable to a mutual-exclusion goal conflict. A frequently-encountered set of strategies operates on the principle of diverting the efforts of an actor in the goal conflict away from direct pursuit of his goal. There are essentially three classes of diversionary strategies, all relying on the fact that if an actor has to divert his efforts to other matters, he is less likely to succeed at his original task. We group the diversionary strategies into the three categories listed below:

DIVERSIONARY STRATEGIES.

1) Threaten higher-level goals of one's opponent.
2) Dissipation of effort: Threaten opponent's other goals.
3) Deception: Misleadingly convince one's opponent that other goals are threatened.

Let us look at one important diversionary strategy and its deceptive option. The process of convincing one's opponent that the actions of a counterplanning strategy have been carried out, without this being the case, is called the deceptive option of that strategy.

```
STRATEGY 1:     THREATEN HIGHER-LEVEL GOAL

      TRIGGER   Mutual-exclusion goal-conflict situation
                between G(Y) and G(X).

           IF   X can find a goal G'(Y) to block, where
                G'(Y) is a goal of higher importance to
                Y than G(Y),

         THEN   X should try to block G'(Y). X can expect
                Y to pursue G'(Y) and abandon G(Y).

   REFINEMENT   X should choose G'(Y) such that:
                1) Y cannot pursue both G'(Y) and G(Y)
                   simultaneously, or
                2) Y can accomplish G'(Y) only if the
                   blocking action is abandoned.
```

Strategy 1 *means* that in a mutual-exclusion goal-conflict situation, one actor (X) *may* threaten a higher-level goal of the second actor (Y), i*n* order to make Y divert his efforts to preserve that higher-level goal. Since *an* actor's attention, time, and material resources *are* limited, Y may not be able to protect his higher-level goal and pursue the goal that conflicts with X's goal simultaneously. Thus, X will be in a better position to win the conflict situation *and* achieve his own goal. The "refinement" part of the rule gives some advice to help X make his choice on which one of Y's higher-importance goals to threaten. X knows which goals Y may consider more important by examining Y's goal tree. Our model assumes that the various actors know about each other's primary motivations. X should threaten a goal that requires Y's full attention, time or material resources to protect. Alternatively, X can threaten an important goal that Y cannot protect. This gives X a bargaining position to tell Y that he will stop his threat only if Y abandons the (presumably less important) goal that conflicts with X's goal.

Let us see the deceptive version of strategy 1. This strategy is based on the same principle of diverting the attention and efforts of the opposing counterplanner away from goal conflict. However, the method used *need* not correlate with reality. As such, one should expect that a deceptive strategy is more likely to fail (e.g., if the opposing actor discovers the truth.).

STRATEGY 2: DECEPTION: FALSE THREAT

Same as strategy 1 but either
1) X only appears to threaten the higher level goal G'(Y), or
2) X threatens to block G'(Y) but has falsely convinced Y that G'(Y) should be one of Y's high-level goals.

There are other diversionary strategies (listed in *[7]h* however they all share a common principle: An actor cannot overtax his attention and resources by simultaneously pursuing multiple courses of action. Awareness of this simple principle applied to other actors (as well as the counterplanner) guides our formulation of the diversionary counterplanning strategies. Each strategy is based on a different method of causing *an* actor to worry about more than one goal at a given time. There are various kinds of limitations on the different items that an actor can focus his attention on at one time. We classify the limitations on the simultaneous pursuit of multiple courses of action into the following categories:

1) Limitations on available time.
2) Mental and physical limitations on the number of actions that can be performed simultaneously.
3) Limitations of material resources.
4) Limitations of ability.
5) Goal of avoiding certain consequences of one's actior
6) Interactions between different courses of actions.

6. Counterplanning Strategies Based on Goal Compromise

Bargaining strategies are characterized by the opponents* willingness to compromise. Willingness to compromise is mediated by many factors such as the two opponents' perceptions of whether compromise is possible, necessary, or desirable. There are two basic classes of bargaining strategies: The first class is based on partial goal fulfillment and the second is based on goal compromise. In order to discuss these strategies, we need to define the notion of partial goal fulfillment. For goals comprising of a conjunction of (possibly recurring) subgoals, partial fulfillment is defined in the obvious manner: The goal is partially fulfilled if some of the subsumed goals *are* fulfilled. For instance consider the following case:

EVENT I: John wanted to marry Susan. They decided to live together instead.

Did John achieve his goal? Strictly speaking the answer is "no". But, if we understand that marriage is a complex goal, we realize that for all intents and purposes John fulfilled most of the goals subsumed by marriage, such as achieving companionship, periodic satisfaction of the sex drive, etc. Since marriage also involves a change in social and legal status not necessarily associated with living together, we say that John partially fulfilled his goal. Therefore, by this measure, partial fulfillment of a goal is the case where the specific goal sought is not fulfilled, but a significant fraction of the underlying reasons for pursuing the goal are fulfilled.

The second way in which partial goal fulfillment is measured applies to goal states that can take a continuum of values. Acquisition of money and achievement of social stature are examples of continuous-valued goals; there ere virtually infinite degrees of social stature and of the amount of money that a person can acquire. We define success differently for preservation goals than for achievement goals (called "P-goals" and "A-goals" respectively in the Schank and Abelson [4] goal taxonomy). Let us call the initial value of the goal state T, the desired value of the goal state "D", and Its resultant value at the time when we must decide whether the goal succeeded "R". For P-goals it is usually the case that I - D, and for A-goals I < 0. The success and failure conditions of continuous-valued goals are given by the following table:

	FAILURE	PARTIAL SUCCESS	SUCCESS
A-GOAL	I = R < D	I < R < D	I < R = D
P-GOAL	R < I = D	R slightly < I = D	R = I = D

Table 1: Partial fulfillment of goal states.

Here are the partial-goal-fulfillment and goal-substitution bargaining, strategies:

STRATEGY 3: GOAL COMPROMISE

TRIGGER G(X) and G(Y) are mutually exclusive and may be fulfilled partially.

IF X cannot achieve G(X) by other counterplanning strategies,

THEN X should try to partially fulfill G(X) by bargaining with Y to compromise mutually on partially fulfilling their respective goals.

Metaphorically, strategy 3 states that if one cannot have the entire pie then one should try to bargain for at least a slice of the pie. Strategy 4 is more cooperative in nature than strategy 3, but involves the same principle of compromise on partially attainable goals:

STRATEGY 4: COOPERATION BY MUTUAL NEED

TRIGGER G(X) and G(Y) are mutually exclusive and may be partially fulfilled.

IF neither X nor Y can independently achieve their respective goals, but can succeed only by pooling their efforts,

THEN X and Y should compromise on partially fulfilling G(X) and G(Y) and plan jointly for their fulfillment.

Strategy 4 states that cooperation may be a necessary course of action in spite of conflicting goals. Event 2 illustrates this point:

EVENT 2: Jesse James and Bill Morgan joined forces to heist the payroll train.

We infer that both actors had the A-goal of acquiring the money in the payroll train. These goals are mutually exclusive but may be partially fulfilled. Suppose we had to answer the question: "Why did Jesse James and Bill Morgan join forces?" The most reasonable answer is: "Probably because neither could heist the train by himself." This answer cannot be inferred from the goal conflict Itself; the existence of a goal conflict suggests competitive rather than cooperative actions. Therefore, the understander has to be aware of strategy 4 - cooperation between actors with conflicting goals is reasonable If neither actor can otherwise fulfill his goal.

The general goal substitution strategy, illustrated in th following event, is presented below.

EVENT 3: Johnny and Billy were arguing over a candy bar. Johnny offered to let Billy ride his new bicycle if he would let Johnny eat the candy. Billy agreed.

STRATEGY 5: GOAL SUBSTITUTION

TRIGGER G(X) and G(Y) are competing goals and X knows about Y's other goals.

IF X can bring about G'(Y), one of Y's goals that is at least as important as G(Y),

THEN X should bargain with Y to substitute G'(Y) for G(Y) as Y's actively pursued goal. (This leaves X free to pursue G(X).)

REFINEMENT Apply this strategy only if G(Y) is not a P-goal.

7. Predictive vt Interpretive Reasoning

For the task of generating hypothetical scenarios, one can only use counterplanning strategies in a predictive manner. However, in story understanding tasks, the counterplanning strategies are used to interpret the actions of the actors. In such situations, it is better to apply the strategies In an interpretive manner. To see what we mean, consider event 4:

EVENT 4: The United States supported Israel in the 1973 Middle-East war. Subsequently the Arabs imposed an oil embargo on the United States and its allies.

QUESTION: Why did the Arabs impose the oil embargo?

Strategy 1 can be used in a predictive framework or in an interpretive manner. In understanding event 4 we need to pose and answer the question "Why did the Arabs impose an oil embargo?" Using knowledge about the goals of the Arabs and the goal of the United States to help Israel, the understander can establish the goal conflict between the Arabs and the United States. The mutual-exclusion goal conflict is between the US goal of aiding Israel and the Arab goal of preventing US aid to Israel.

Having interpreted the situation thus far, an understander can proceed in two different manners. The first manner is to predict all the possible Arab counterplanning actions to make the US stop aiding Israel. If, in interpreting the rest of the event, the understander matches an action with one of the previously predicted counterplanning actions, the understander can conclude that indeed the Arabs were counterplanning against the US and their counterplan was the predicted course of action. Such a process would require the understander to generate vast numbers of hypotheses and subsequently test each hypothesis as a possible explanation of the situation. There is no evidence to suggest that people generate all possible inference paths in a given interpretation in order to discard all but one path that matches reality. It appears more plausible that people only pursue a small number o* relevant inference paths. Therefore, generating all possible plans of action is not a reasonable

psychological model of human thinking, nor does it lend Itself to reasonable constraints on the computational time that the system may require in order to generate and test all alternative actions on the part of the Arabs.

A more reasonable alternative to the generate and test process is the following: Given the existence of the mutual-exclusion goal conflict, we can predict that the two actors may counterplan against each other. No further predictive inferences a*re* generated at this point. The rest of the event should be interpreted in light of the expectation that the two actors may counterplan to resolve their goal conflict.

When the understander learns of the Arab oil embargo, he tries to see if this is a reasonable course of action to take as a counterplan against the US goal of aiding Israel. Counterplanning strategy 1 (refinement 2) matches the type of interaction between the Arab plan and the US goal. The Arabs are threatening a higher-level US preservation goal by cutting off oil supplies, and the US cannot do anything to directly remedy the situation. Now the Arabs can bargain to end the embargo in return for the end of US aid to Israel. Therefore, the understander can establish the Arab counterplanning actions by applying strategy 1 in an interpretive manner. The result of the Arab actions is matched to the action part of the strategy. This match, suggested by our previous expectation that a counterplanning action was likely, allows the understander to infer that the Arabs were invoking counterplanning strategy 1. Therefore, we can say that the reason for the Arab action is the test clause of the strategy. Thus, we have a mechanism for using counterplanning strategies in an abductive manner to interpret actions in conflict situations, as well as a deductive manner to propose counterplans. The general interpretive technique of predicting an overall framework and subsequently instantiating its component parts has been successfully applied in other AI systems such as SAM [9] and HEARSAY-II [12]

The interpretive mode of reasoning is superior to the purely predictive mode because it does not require the generation and subsequent testing of an arbitrarily large set of possible courses of action. It is also a more reasonable model of human thought. Before and during the 1973 Arab-Israeli war, few people foresaw an Arab oil embargo. When the embargo came, however, the reasons for the Arab action became clear.

8* Concluding Remark

We have seen how counterplanning is a necessary process for both decision making in the face of adversity and for understanding natural language descriptions of such conflict situations. The counterplanning strategies discussed in this paper are illustrative rather than exhaustive. More complete sets of strategies are discussed in [7,8]. Counterplanning is a general inference mechanism for understanding human conflict situations. Since most interesting stories and events involve some type of goal conflict, we need to include counterplanning as a necessary tool in the Artificial Intelligence repertoire.

References

1. Newell, A. and Simon, H. A. *Human Problem Solving.* New Jersey: Prentice-Hall, 1972.

2. Fikes, R. E. and Nilsson, N. J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence 2* (1971), 189-208.

3. Sacerdoti, E. D. *A Structure for Plans and Behavior.* Amsterdam: North-Holland, 1977.

4. Schank, R. C. and Abelson, R. P. *Scripts, Goals, Plans and Understanding.* Hillside, NJ: Lawrence Erlbaum, 1977.

5. Wilensky, R. *Understanding Goal-Based Stories.* Ph.D. Th., Yale University, 1978.

6. Schmidt, C, Sridharan, N. and Goodson, J. The Plan Recognition Problem. *Artificial Intelligence* 11, 2(1977), 45-83.

7. Carbonell, J. G. *Subjective Understanding: Computer Models of Belief Systems*.* Ph.D. Th., Yale University, 1979.

8. Carbonell, J. G. *The Counterplanning Process: A Model of Decision-Making in Adverse Situations.* Rep. Computer Science Dept., Carnegie-Mellon University, 1979.

9. Cullingford, R. *Script Application: Computer Understanding of Newspaper Stories.* Ph.D. Th., Yale University, 1977.

10. Barstow, D. R. *Automatic Construction of Algorithms and Data Structures Using a Knowledge Base of Programming Rules.* Ph.D. Th., Stanford University, 1977.

11. Newell, A. Productions Systems: Models of Control Structures In W. G. Chase, *Visual Information processing.* New York: Academic Press, 1973, pp. .

12. Hayes Roth, F., Mostow, D. J., Fox, M S. Understanding Speech in the HEARSAY-II System In L Block, *Speech Communication with* Computers, Berlin: Springer-Verlag, 1978, pp..

FORMAL OBJECTS AND FEATURE ASSOCIATIONS IN "ARGOS-II"

Michel CAYROL, Bernard FADE, Henri FARRENY

Laboratoire "Langages et Systemes Informatiques"
Universite Paul Sabatier, 118 Route de Narbonne
31077 Toulouse Cedex FRANCE

ABSTRACT : ARGOS-II is a problem solving system developed at L.S.I, laboratory. Its purpose is to embody a general model of the decision function of a robot. After a brief global characterization of the system, an overview of the initial specialization process is provided. Then two interesting particularities of the system are emphasized : "formal prospective aptitude" and "automatic control of feature associations". ARG0S-1I generates action plans but also performs the execution monitoring. ARGOS-II is a pattern directed inference system. The "formal prospective aptitude" allows to continue to build plans even when some circumstances are unknown. They permit to avoid some awkwardnesses or failures and are open to learning. Not only does the "feature associations" mechanism deal with the heavy problem of updating the attributes of a group of objects involved in a unique action, but it also allows to easily create powerful demon functions useful for Artificial Intelligence programming.

I. CHARACTERIZATION OF ARGOS-II [1]

a) ARGOS-II is first a problem-solver : it elaborates the action plans by means of an interpreter of production rules. The rules *are* invocated by using problem patterns and a pattern-matching procedure. The search method is, fundamentally, of depth-first type in an implicit and-or graph, with backtracking control facilities. However there is no definite order in the representation of the rules. Their disposition can be modified during the solving process owing to, mainly, the existence of several explicit directives and also efficacious demon families.These directives and demon families are specially designed to deal with rules (a kind of metarules). For example it is possible to inhibit —or to awake— rules conditionnally or not, temporarily or not. All these possible changes of the rule states are pattern directed. The solving process is not considered as terminated when a plan is produced : ARGOS-II realizes the plan monitoring execution (about the process of monitoring plan execution, see [5], ['8], [9]) .Besides, the separation of planning process and execution process is not sharp (as it happens rather frequently) : before the whole plan is built, actions can be performed, if it is necessary (deductions, perceptions, interrogations,...).The ARGOS-II program is written in TLISP-language (see [-4]).

b) ARGOS-II is also a problem expression language for Artificial Intelligence. It contains many of the primitives implemented in classical procedural languages or in more recent systems which use specifications by production rules (see [3],[7],[13]. Among the possibilities to act on rules let us point out, not only the above quoted directives to inhibit,to awake or to fire (immediately), but also the facilities to state spontaneous triggering rules and demon rules.All the standard functions of TLISP may be used.

II. OVERVIEW OF THE INITIAL SPECIALIZATION PROCESS

The system maintains several separated associative networks by means of pattern-matching. Essentially"fact-memory","problem-memory","rule-memory".For embedding ARGOS-II in a real context, the user initially provides four main kinds of knowledge :

a) facts in like-S-expressions form, e.g. :
(BOX3 ASPECT (METAL BLUE) IN (ROOM1 (5 10)))

b) problem analysis rules in form of production rules :

{rule-name SEX_0 SEX_1 SEX_2...SEX_n
problem pattern (α) n fact patterns (β)
trigger part
====> SEX'_1 SEX'_2 ... SEX'_m }
action part

Example :

{R9 (TAKE >X)
(α)
(<X * IN >Y *) not (<Y * STATE RADIOACTIVE *) ⟹
(β)

(γ) (if-patfact (ROBOT * TAKEN <X *) then return)
(δ) (execute (POINT-IN-ROOM (<Y) (>Z)))
(ε) (replace-problem (GO <Z) (APPROACH <X))

(C) (record (plan (GRASP <X)) (update (modify (ROBOT *) (TAKEN <X))))

The words "not", "if-patfact",..."execute",etc... are ARGOS primitives. The symbol "*" represents the operator of undiscriminate absorption (see [14],We find several important knowledge modules in this rule : utility (a), subsidiary conditions for applicability (8), control depending on facts y)f communication with environment (interfaces) or knowledge procedures (5) problem-reduction (in two sub-problems here) (e), primitive sub-problem (terminal act of the considered robot) and world model updating (c),

c) procedures written by means of ARGOS and TLISP primitives,Some of these procedures represent domain knowledge which is useful for internal deductions or computations of the system (e.g. : a method for designing an itinerary, in the case of a moving robot),The others are interfaces between the system and external processes (effectors captors,dialogue with the user).The procedures may be called from the body of the rules by ARGOS -primiti ves as "execute".In this case, the formal call may contain a list of identificators for the results : e.g.: (execute (MEASURE-TEMPERATURE (<OBJ <PLACE) (>X >Y >Z))) becomes after instantiation : (execute (MEASURE-TEMPERATURE (WALLS UP) (>X >Y >Z))). The real list of results returned as value of MEASURE-TEMPERATURE is then matched with the pattern (>X >Y >Z), thus the results are affected to X,Y,Z,

d) problems in like-S-expressions form, e,g. : (FIND (OBJECT (HEAVY SOLID)))

III. FORMAL PROSPECTIVE APTITUDE

In planning phasis, when a production rule is triggered, a part of its arguments receive ground Values. Some other arguments may not be instantiated until a posterior step of plan generation or until execution time, ARGOS-II affects an original symbolic identificator to these latter arguments. Defining and dealing with such "formal objects" have already been done by SUSSMAN (see : "wishful thinking technique" in HACKER [11] and by SACERDOTI (see : "use existing objects critic" in NOAH [9]. In ARGOS-II the formal objects are systematically used, The plan generator can produce sequences like :

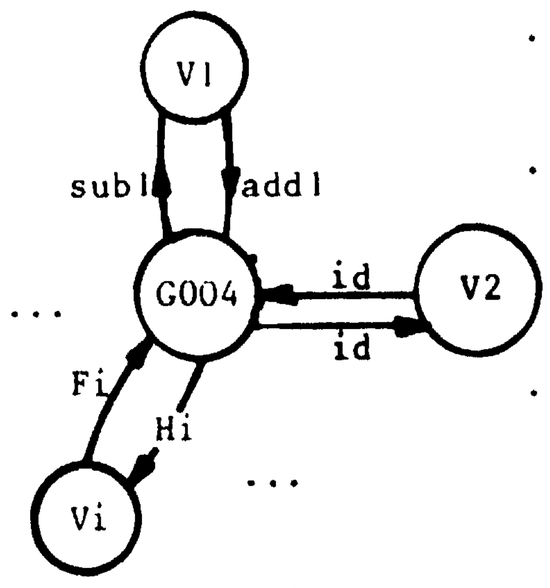
[CHOOSE-FREE (ANVIL)—>(>G005)]
[LOCATE (<G005)————>(>G006)J
[TAKE (HAMMER)————>(>G007)]
[BRING-TO (<G007 <G006)J

where : "CHOOSE-FREE" and "LOCATE" are planning aid procedures : some occurrences of them did not succeed during planning and entailed the insertion of corresponding steps into the plan to execute ; "TAKE" and "BRING-TO" are interfaces with terminal acts of the considered robot ; "(ANVIL)" or "(<G007 <G006)" are entry parameter lists ; after "—>" are described the (symbolic) expected results of the execution ; we note ">X" for a variable "X" which must be instantiated, "<X" the same variable after pattern matching instantiation, "GOOi" the original name of the i^{th} encountered formal object.

After the above fourth step some assertions like "(ANVIL <G005 AT <G006)" and "(HAMMER <G007 AT <G006)" will be recorded in fact-memory. These new data may be, in the following of the ARGOS*s work, matched with patterns like: "(* ANVIL >X *)" or "(* ANVIL ANVIL4 AT >Y *)". Besides, formal problems may be introduced in the data base and then solved. Such a technique often allows to avoid the awkwardnesses or even failures due to a premature bounding. In spite of the lack of some specifications which can only be obtained at execution time, longer (i.e."more prospective") plans can be designed. Besides, some impasses which are independent of the unknown parameters can be detected. Let us emphasize that "formal objects" facilitate intermixing planning and execution monitoring, allowing to include perception-decision points in the plans to execute. At last the formal objects process may introduce some kind of learning : a partially symbolic plan, correctly executed may be interesting to save as a resolution scheme (new rule).

IV. AUTOMATIC CONTROL 0F "FEATURE ASSOCIATIONS"

The problem of the robot which carries a bag which itself contains a box is well-known : we have to conveniently update some related attributes of a group of objects involved in a unique action. In ARGOS-II is implemented a dispositive which allows to link the transformations of a collection of variables with the modification of anyone among them. The chief primitive available for the user for such a work is "associate": e.g.:(associate (VI addl subl) (V2 id id) ... ... (Vi Fi Hi) ...) In the above example is defined an association between the variables VI, V2...Vi...via the functions "add 1","sub1","id", "id",..."Fi","Hi"..,(addl,subl,id : standard-TLISP-primitives).The association is implemented by linking VI,...Vi... to a common variable (G004 in the provided example) created dynamically by the system at every use of "associate":



. initially, the value of G004 equals the value of addl (VI)
. every demand of access to the value of Vi entails an evaluation of Hi(G004)(Hi applicated to value of G004)
. reciprocally,every modification of Vi entails an updating of (only) G004,with the value Fi(Vi).

Let us emphasize that for any access or modifi-

cation of Vi only one computation is done,to relate the Vi-value to the common variable value. Indeed, not only explicit variables (as Vi) but also implicit variables (what we call "features^ may be handled with "associate''. In this case, the formal syntax is of the form : (associate (PI (FN1 FI HI))...(Pi (FNi Fi Hi))..,) where PI ...Pi... are patterns, FNI..,FNi... are feature names and F),H1 ...Fi,Hi... are LISP functions, e.g.;initially, there is in the data base ; (PAINT-BRUSH...COLOUR BLUE...) (PAINT-BOX $_{w}$ COLOUR GREEN...). Let us consider : (associate ((PAINT-BRUSH *)(COLOUR id id)) ((PA1N1VB0X *)(COLOUR id id))).In this example, the first implicitly manipulated feature is the attribute following the prefix "COLOUR" in any datum able to match with the pattern "(PAINT-BRUSH *)". Henceforth the features "COLOUR" of "PAINT-BRUSH" and "PAINT-BOX" will be considered as identical (features "COLOUR", i.e. here, the attributes following the prefixes "COLOUR").Let us note that, by side effects of the functions Fi and Hi, any procedure can be triggered,either when affecting a value to one of the associated variables or features, or at the moment of access to one of these variables (or features).We can write, for example, very practical procedures to control communications between the robot-system and its environment : e.g. :

```
(associate (VI FUNC1 FUNC2))
(define FUNC1 (X) (print PLEASE X ?)(read))
(define FUNC2 (X) (setq HOWMANY (add! HOWMANY))X)
```

Any modification of VI, like "(setq VI BIG-VALUE)" will modify, in fact, the value of the common variable by call to "FUNCI".The value returned by the computation of FUNCJ, i.e.the value of the latter expression "(read)",is affected to the common variable, And for any reference to "VI",the value computed from the common variable by means of FUNC2 is returned.Besides, "HOWMANY" counts the number of entries by "VI". The above example shows that, with "associate" primitive family,it is easy to create a new class of demons ; these demons are able not only to watch over the traffic into and out of data bases, but also to watch over the traffic to and from any variable or feature. The facilities provided by "associate" and other analogous primitives implemented in ARGOS-II extend the "AFFIXMENT" statement proposed in AL (see [6],[12])and the ideas about "representation of actions that have side effects" implemented in AIMDS [10].They are adapted to current mechanisms used in pattern directed inference systems (e.g.:back-tracking)..

V. CONCLUDING REMARKS

ARGOS-1I has been designed to be a general system able to control the behaviour of a robot whose particular description is provided by the user. Industrial applications (computer aided production) are being studied. Among several other worth-considering directions of research for the ARGOS project, let us quote : the learning of plans ; the extension of the system capacities to interact with operator and to explicitate the (deep) reasons of its behaviour ; the fuzzification of the triggering mechanism of the production rules (see [2]).

VI. ACKNOWLEDGEMENTS

The authors wish to thank Professor Beaufils, Director of L.S.I. Laboratory, for his constant encouragements.

REFERENCES

[1] CAYROL M., FADE B., FARRENY H. : Etude en simulation des strategies de contr31e d'un robot Univ.Paul Sabatier Toulouse Tech.Report Juin 1979
[2] CAYROL M., FARRENY H., PRADE H. : Fuzzy production rules directed by fuzzy pattern-matching L.S.I.Univ.Paul Sabatier Toulouse Tech.Rpt.Jan.79
[3] DAVIS R., BUCHANAN B., SHORTLIFFE E. : Production rules as a representation for a knowledge based consultation program, Artificial Intelligence Vol.8 1977
[4] DURIEUX J.L. : TLISP Manual Reference L.S.I.Univ.Paul Sabatier Toulouse Juin 1978
[5] FIKES R.E. : Monitored execution of robot plans produced by STRIPS, Proceedings of the IFIP Congress Ljubljana, Yugoslavia, 1971
[6] FINKEL R., TAYLOR R., BOLLES R., PAUL R., KELDMAN J. : AL, a programming system for automation, Stanford A.I.Lab. AIM-243 Nov.1974
[7] LATOMBE J.C. : Une application de 1'Intelligence Artificielle a la conception assistee par ordinateur - TROPIC, These d'Etat Nov.1977
[8] NILSSON N.J. : A hierarchical robot planning and execution system SRI A.I.Center Tech.note 76 April 1973
[9] SACERDOTI E.D. : A structure for plans and behavior,Elsevier Computer Science Lib.1977
[10] SRIDHARAN N.S., HAWRUS1K G. : Representation of actions that have side-effects, Proc.5^{tn} I.J.C.A.I, pp.265-266 August 1977
[11] SUSSMAN G.J. : A computer model of skill acquisition, Elsevier Computer Sci.Lib.1975
[12] TAYLOR R.H. : A synthesis of manipulator control programs from task-level specifications, Stanford A.I.Lab. AIM-282
[13] WATERMAN D.A., HAYES-ROTH F. : Pattern Directed Inference Systems,Academic Press 1978
[14] WINSTON P.H. : Artificial Intelligence, Addision Wesley Publishing Company 1977

AN APPROACH TO MEDICAL DIAGNOSIS BASED ON CONCEPTUAL STRUCTURES

B. Chandrasekaran, F. Gomez, S. Mittal, and J. Smith, M.D.
Artificial Intelligence Group
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

The power of an experienced medical diagnostician seens to us to arise from a knowledge organization whose very structure helps keep the combinatorial growth of processing under control. Thin structure mirrors the deep conceptual structure of the field. We propose a knowledge representation scheme called conceptual structures, which organizes knowledge in the form of production rules or nore complex procedures In such a way that they are accessed and uaed as needed. This structure is doninantly hierarchical, the successors of a conceptual node standing for subconcepts which refine that concept. Associated with each concept is a set of procedures (experts) which attempt to apply the relevant knowledge to the case at hand; this might include turninc over control to selected subconcepts for more detailed analysis. Thus, abstractly, the conccntual structure organizes the invocation of procedures available to the diarnosticInn. The calling of a special 1st by a physician is akin, we think, to control transfer to a subconcept in the cornItIve structure of a diagnostician during problem solvinp; the difference is in the level of detail of the cogritive map. In this Paper, sore principles governing the design of conceptual structures are presented. We apply the ideas to the design of MDX, a system to perform diagnoses in a syndrone called Cholestasis, and discuss the performance of the system.

1.0 INTRODUCTION

In this paper we shall describe an approach to the design of systems to solve problems in knowledge-rich domains. We shall consider in particular a system intended to perform medical diagnosis related to a liver syndrome named Cholestasis, The underlying metaphor is one of a society of experts among whom the knowledge of the domain and the procedures for acting out that knovleuge are distributed. The organization of knowledge as experts raises many Important questions, such as what knowledge should be organized into an expert and what other knowledge distributed between them; the nature of control transfer and communication between experts; the data-base organization and access; and the relation between redundancy of knowledge and data on one hand and the speed of processing on the other. While the idea of experts is not new, we believe that the eplstemologlcal principles needed as criteria for decomposing a body of knowledge into experts, in order that their interaction is controlled rather than chaotic, have been missing. Our discussion in the next few sections will attempt to highlight some of these principles.

The paper is organized as follows. In Section 2, we discuss how conceptual structures effectively organize a body of knowledge into experts. In Section 3 we discuss, for purposes of illustration, the knowledge representation of a ragment of Cholestasis. This leads, in Section 4, to a discussion of different types oi experts and their structursl representation. In Section 5, a prototype medical diagnosis system based on these principles is described. The description is accompanied by an Illustration of the system dlapnosInf o real medical case. Other contemporary approaches to medical diagnosis using Artificial Intelligence techniques are discussed briefly in Section 6. In the concluding section, we outline some extensions that are contemplated tor the near future.

2.0 CONCEPTUAL STRUCTURES AND ORGANIZATION OK MEDICAL KNOWLEDGE

We adopt the view that a given field of knowledge has an underlying conceptual structure which enables effective use of that body of knowledge. This is the deep structure to which the knowledge representation in the field should strive, and this is what effective teachers communicate and effective learners build. This conceptual structure can he viewed as a way of organizing knowledge in MUCH a way that purposeful, focussed access to the relevant parts of the knowledge structure is achieved. Consider medical diagnosis as an example. It is our view that the power of an experienced medical diagnostician arises from a knowledgs organization whose very structure helps keep the combinatorial growth of processing under control. Knowledge in the form of facts, heuristics, production rules or more complex procedures cannot be effectively uaed unless it is embedded in this underlying structure in such a way as to facilitate access as needed.

What ars the principles by which domains of knowledge are to be organized so that they can be used effectively? Our theals Is that the major criterion has to do with the efficiency of transfers of control or access to different parts of the knowledge structure. There are correct and Incorrect analyses of the conceptual structure of a field. Incorrect analyses lead to a proliferation of control transfers; correct ones should generally lead to focussed and controlled communication between different parts of the structure.

The organization of the medical community provides a concrete caae study in the principles of knowledge structuring. A phenomenological account of medical decision making seems to us to be very revealing in this regard.

One of our guiding notions is that there is a correspondence between the broad conceptual organization In a diagnostician and the organization of specialties in a medical community. The calling of a specialist by a physician is not dissimilar, in our view, to control being handed over to a subconcept in the cognitive structure o;' a diagnostician during the course of his problem solving. Thus thi9 is a concrete realization of the society * of minds paradigm that has been put forward by Mineky and Paper [1]. In order for this paradigm to go beyond being a metaphor, we need to know how a society of minds, whether in. an individual or as a collection of expert solving, a common Problem, should coordinate the activities of and communication among member! to produce intelligent behavior. We feel that the organization of specialties in the medical community reflects the underlying conceptual structure, and, to a certain extent, the requirements of efficiency in control and communication structures as well*. The same principles that lead to effective communication among real experts in a medical community are to be used in organizing and structuring knowledge within a specialty. A simplified rendering of these principles are:

PI) There is some sort of a hierarchical organization with respect to control transfer. That is, the GP who had original responsibility might, after deciding that it is a liver problem, tranfer control to the liver specialist. The liver specialist night in some cases hand over control to a super specialist in a subarea, but it would be highly unusual for him to transfer the patient to a heart specialist. If such a thing is ever done, it'd be usually through return of control to the GP who would then accomplish such a transfer.

P2) The expert who Is contemplating control transfer knows enough about the domain of knowledge of the subaxperts who are likely to be called by him to decide, generally correctly, which subexpert is relevant, but not enough to solve the problem himself.

P3) Each expert knows enough about the domain of others in his level of the hierarchy to decide when he himself has been mistakenly called.

P4) There are some experts who do not belong in the main hierarchy, but who are outside of it, so to speak. They are not generally given control for the purpose of diagnosing the original complaint. They are used as resource experts to accomplish specific sub tasks requiring their expertise. They communicate back to Che expert who called them with the answer. An example of this type of expert Is the radiologist.

P3) The communication between an expert and a subexpert is of several forms: answer specific questions dealing with the subspecialty, confirm or reject possibilities, or, in difficult cases, hand over total control. Further, in some instances a mere number standing for weight or probability Is returned (see P7 below); In most cases, however, the communication is symbolic in nature. In a vocabulary which, in parts, Is particular to the specialty subject matter.

P6) An expert often has enough knowledge of the subject matter of the subexperts to be able to solve easier cases. For example, most GP's know enough about liver diseases to solve "easy" liver cases, enough about heart diseases to solve easy heart cases,etc. Among the criteria that determine which knowledge should be abstracted at this level are: how common is this kind of malfunction, how conceptually complex Is this etc. This sort of organization keeps the combinatorics of control transfer to a minimum, since only in a relatively few cases will there be a large number of accesses to subexperts.

P7) While many significant decisions Involve manipulating symbol structures associated with concepts, there are very definite situations which are best viewed as weighting several pieces of evidence to produce a numerical weight for a hypothesis. This piece of information is eventually Incorporated In symbolic structures which are used for problem solving.

Applying these principles to organization of knowledge within a specialty such as liver disease or Cholestasis, we have found that the conceptual structure can be usefully viewed as a tree, where a node stands for a particular concept and the successors of the node stand for subconcepts that help refine that concept; e.g., hepatitis, whose successors might be acute, fulrinant and chronic types of hepatitis. Associated with each node Is s set of procedures which decide on the applicability of the concept to the case at hand. Part of such decision-making in a node is often the decision to turn the control over to subconcepts and their associated procedures to check on their applicability. In this sense, the conceptual structure organizes the invocation of procedures available to the diagnostician.

Notice that the form of the structure as a tree already exercises a strong constraint over communication. An expert cannot call other experts arbitrarily, but can only communicate via their super and subconcepts. While it is hard at this stage to advance this principle as an absolute requirement, we have found that violations indicate either that the analysis was incorrect or that the concept actually belongs outside the structure in some sense, with specific retricted modes of communication with the experts in the main structure. This is consistent with principle 4 above.

We note in passing that the principles governing the organization of conceptual structures are meant not only to make effective use of knowledge, but also to aid In the acquisition of it. The concepts get refined as a function of further learning, and the node at which the refinement takes place gets appropriate abstractions about the subconcepts. Another point to be noted is that often heuristics are formed which enable the system to go directly to a concept several levels below, sidestepping the methodical, level by level communication. But concentration on this second order optimization device obscures the clarity of the underlying structure. In fact expertise often consists of an accumulation of such heuristics, and, for this

* It might be argued that medical specialties historically grew up around organ systems, i.e., along physiological and anatomical lines, and that control and communication efficiency is not a paramount criterion. In fact all this means is that the criterion of control transfer and communication efficiency is often met by organizing many specialties along these lines- after all one would expect that there would be a certain coherence within each such specialty. Further, not all medical epecialtiee are organ-based; consider rheumatology, radiology, emergency room medicine, clinical pathology, toxicology and infectious diseases as counter examples. Our thesis is that the intuitive notion of internal coherence of a specialty is closely related to the control transfer and communication efficiency, and thus the latter becomes the concrete criterion of natural selection" in the evolution of specialties.

reason, experts tend to be poor elucidatora of the underlying structure. These, however, are issues which art not the main concern of this paper, and will be discussed in detail elsewhere.

In the next section, we illustrate the usefulness of the above ideas by the concrete example of a fragment of the cholestasis system. While we have performed the analysis for cholestasis in general, we present here only a portion thereof, in order to concentrate on the principles.

3,0 CONCEPTUAL STRUCTURE OF CHOLESTASIS

Cholestasis is a condition caused by disruption of the secretion of bile or of its excretion into the duodenum. It happens that while the identification of the cause of cholestasis might be complicated, the determination that It is present or not present can be accomplished with relative ease. This fact makes the cholestasis knowledge base a useful object of study: it Is large enough to be an adequate arena in which to test the ideas, and at the same time, its interaction with knowledge bases concerned with other diseases can be kept under control because of the relative certainty of Its applicability. (We call nodes corresponding to concepts with the latter property "anchor nodes.") It has been found useful to group the causesof cholestasis as occurring within the liver itself(intra-hepatic) or In the biliary tree cutside the liver (extra-hepatic). There are often broad indications of whether a case is extra- or intra-hepatic, which, while not 100% reliable, art generally helpful in deciding v/hich is mora likely* and thus should be investigated first.

Us shall shortly give the details of Procedure Cholestasis, which will be representative of the procedures in most of these conceptual nodes. But first we need to refine the concept of Extra-hepatic Cholestasis in some detail. The movement of bile through the biliary tree can be obstructed by a physical object such aa stone, stricture or cancer of various kinds or by swelling cauaed by inflammation Due to a lumber of causes. Associated with each subconcept are procedures indicating how to establish it by symptoms, signs and lab data; information about deciding which of their subconcepts are applicable, etc. In accordance with the principles set forth earlier, our analysis of Cholestasis recommends the conceptual structure in Fig. 1. Becauae of the procedures attached to each of these nodes, the tree can be viewed aa a structured organization of experts involved in diagnosing Cholestasis. The creation of such a hierarchy is the result of the analysis of the content of a field of knowledge. The difficulties involved in the uncoverins of such a structure can be illustrated by considering the "stone" and "inflammation" (cholangitis) nodes. Textbooks on liver dlseaaes inform us that inflammation may be cauaed by the presence of stones and, conversely, Stones may be formed if there is persistent Inflammation. A direct rendering of this information would result in the mutual calling of these, two experts, violating the hierarchical expert call structure. The difficulty is cleared by noting that these two nodes do not correspond to the general concepts of "biliary Stone" and "bile duct inflammation." Instead,they stand for "stone aa cause of Cholestasis" and "inflammation aa cause of Cholestasis." The "stone" node then has the information about "Inflammation" needed for its work localized there (see III in Procedure Stone-Consistency In sec.4.4) and similarly for the "Inflammation" node. It seems to us that such a decoupling in knowledge organization is essential for operational efficiency.

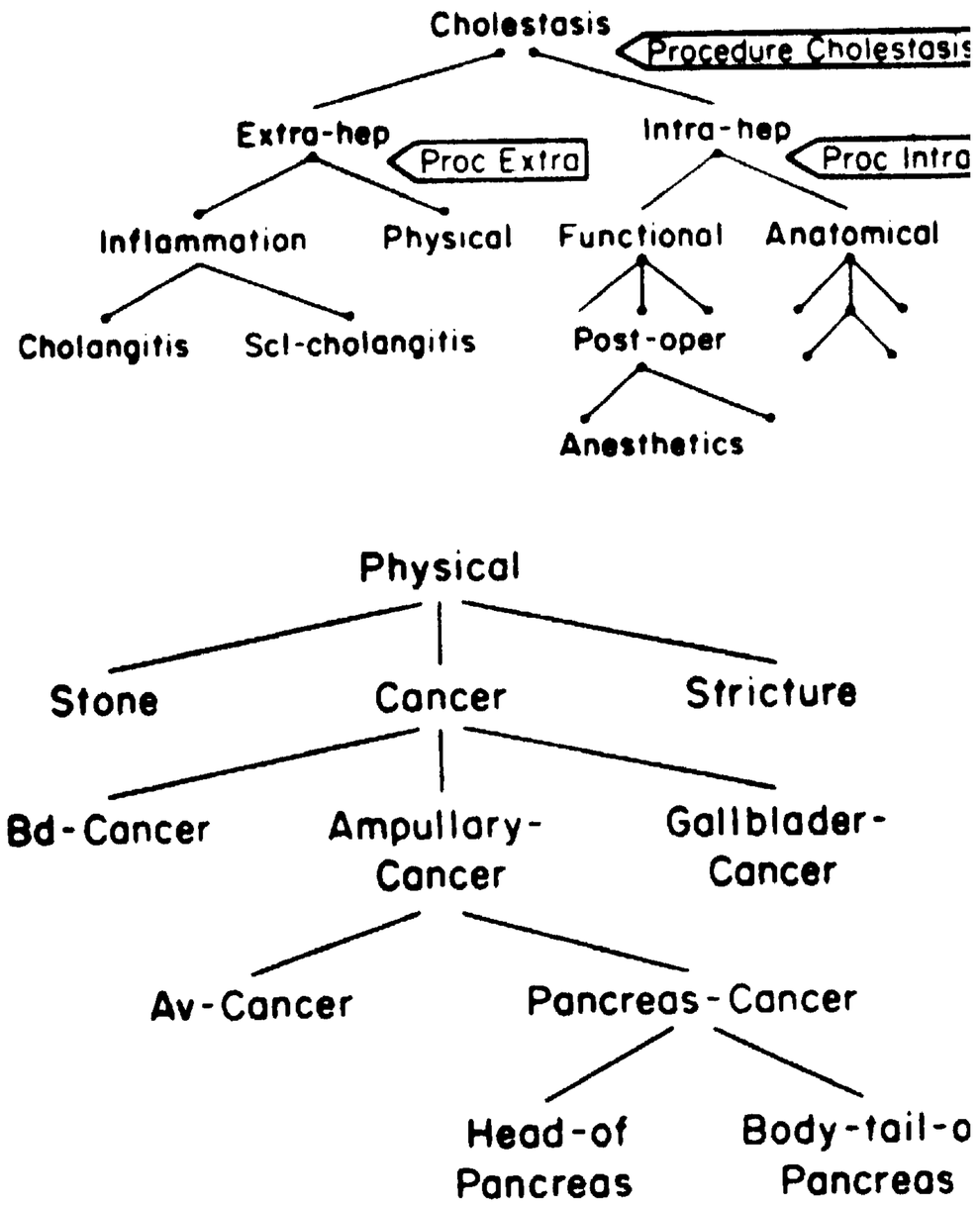


Fig.1 Conceptual Structure of Cholestasis

4.0 PROBLEM SOLVING BY EXPERTS

4.1 Types of Experts

There are two types of experts - characterized by the control they exert over the problem solving process. Experts like Cholestasis, Extra-hep, Intra-hep, etc., play greater role in decision making. They decide, for example, which expert to call next; what sub-problems to solve; decide *it* the problem is solved; mediate between competing advice from subconcepts (in differential diagnosis situations), etc. The other class of experts, such as Stone, Stricture, etc., primarily evaluate the problem-data to determine how well the concepts they represent fit the data and to say what additional data will enable a batter decision.

Concepts such aa Stone are tip nodes In the concept tree (primitive concepts)• Deciding if the presence of •tones characterizes the case is done mainly by weighing different pieces of evidence, both pro and con, into a whole.* This does not entail any search through a space of hypotheses. On the other hand, nodes higher in the

This is not to say that human diagnosticians u numbers in this situation. The Important point la that

* likely" here does not mean that" probabilities are computed; rather, certain aspects of the problem description are looked at for information about which should be given priority In investigation.

tree need to search below in order to fulfill their task. It often happens, however, that a node which was incorrectly analyzed as a tip node is indeed a concept which is sufficiently complex that further refining is necessary.

4.2 Problem Solving Strategy

The basic strategy in problem solving is one of establishing and refining the concepts. Experts associated with the concepts actually carry out these processes; i.e., there is no uniform mechanism operating on different concepts for this purpose. This is very important, as each concept may have different knowledge and thus associated strategies for establishment and refinement.

Some nonprImitIve concepts are anchoring concepts, because they can be established with a high degree of certainty, in pathognomonic situations, they can be established by one datum or one set of data, but generally they are established by accumulating evidence from a small number of highly reliable symptoms/lab data. Once established, the concept Is refined by the process of determining which of the subconcepts, alone or in combination, fit the data which this particular concept was originally asked to explain. These anchoring concepts provide a strong global focus to the problem-solving process. Control Is systenatIcally distributed to other experts so that each may solve a part of the refinement problem. The expert which provides the focus also retains the control over the problem-solving process, thus largely eliminating the "focus of attention" problem.*

How to handle additional data while the diagnosis Is in progress is an issue In any diagnosis system. Our system is designed to begin with some initial data and ask for more, when needed. The new data do not prompt any costly computations or drastic change of focus; Instead, the controlling expert uses them to re-compute only what was affected within its domain. As control is passed back and forth, each expert determines the relevance of the data and acts accordingly but only when it gets the control.

bow the above problem solving strategy is embodied in the system can be best understood by considering sone examples of experts. Ve shall consider Procedure Cholestasis and Procedure Stone "Consistency as examples of non-tip and tip node experts respectively. Cholestasis is also an anchor node and thus provides a good illustration of how control is carefully distributed with a clear focus.

4.3 Procedure Cholestasis

I. CHOLESTASIS has been handed control and asked to first ESTABLISH itself and then REFINE the concept it embodies.

1, Establish cholestasis on historical data, signs, symptoms, ana clinical data, This portion is a combination of heuristic rules and numerical weights. (This is straightforward and can be done quite reliably.)

2. Consider first the disease-hypotheses invoked by patient data in INVOKED-CONC list and made available to CHOLESTASIS by the calling expert. (As a doctor scans through the data, some possibilities come to mind. This is the typical heuristic knowledge of most doctors about cholestatic diseases. However these are merely suggestions to be evaluated.)

*. Generate calls to immediate sub-concepts of CHOLESTASIS, which are the superconcepta of those in the list. For example, if INVOKED-CONC contains SCL-CHOLANGITIS, BD-CANCER, PERICHOLANGITIS and PRIM-BIL-CIRRHOSIS, these calls would be ——

Call EXTRA-REP (consider SCL-CHOL, BD-CANCER)
Call INTRA-HEP (consider PERICHOL, PRIM-BIL-CIR)

(The experts corresponding to these eventually get control and they evaluate data more carefully and return four lists: CONSISTENT-DATA, CONTRADICTING-DATA, RECONSIDERATION- DATA, and RECOMMENDATION, which together consist of weighted recommendation and reasons. Note that CHOLESTASIS knows which subconcepts to call, even though it doesn*t know enough to make a decision about them. This is consistent with P2.)

b. Place members of INVOKED-CONC into three lists: ACTIVE, INACTIVE and UNLIKELY. INACTIVE holds those for which data are insufficient to decide one way or the other.

3. Check if concepts in ACTIVE list can explain all ABNORM-DATA (use CONSISTENT-, CONTRADICTING-DATA lists for this).

I. Yes, also no significant overlap, return ACTIVE list as answer (straightforward cases will be solved at this stage).

II. No, some data in ABNORM-DATA cannot be explained. Go to DEFAULT-PROC.

iii. Yes, but overlap. Go to OVERLAP-HANDLE*, return modified ACTIVE list.

II. DEFAULT-rROC (Control transfers here when INVOKED-CONC falls to explain all facets of cholestasis.)

1. Use, data to decide which of EXTRA or INTRA is more likely. (Imagery information is often very effective in this discrimination. Errors here will delay correct diagnosis, but not prevent it.)

2. Execute EXTRA-IN-CHOLESTASIS or INTRA-IN-CHOLESTASIS .acccording to result from 1. If solved, return ACTIVE list. Else call the other. If that fails go to LOOK-FURTHER.

III. EXTRA- and INTRA-IN-CHOLESTASIS

(EXTRA- and INTRA-IN-CHOLESTASIS contain some knowledge about subconcepts that is of a more usual nature and thus helps deal with a large number of cases. Relatively easy cases will be generally handled by this. Again, putting this knowledge here

symbolic template matching of some kind is needed. In Procedure Stone-Consistency that follows, the numbers really stand for discretized, symbolized degrees of evidence.

* Reference[2] discusses the focus of attention problems that arise in HEARSAY-II, vhich, even though it is not a medical diagnosis project, is of interest due to its experts-based organization. Our additional principles for constraining control and communication seem to make the difference.

* OVERLAP-HANDLE, which deals with the notion of two diseases explaining substantially the same data, is not currently implemented and will not be further discussed here. Details will be given in [3].

reflects P2 and P6) These consist of a collection of rule-based calls of the form:

If history of acute/chronic cholecystitis, Than Call EXTRA (Consider Stone)
If air in x-ray of biliary tree, Then Csll EXTRA (Consider Stricture)
...

IV. LOOK-FURTHER
(Control transfers here when everything CHOLESTASIS knows directly hss been tried, but there is still "significant" data left to be explained.) Order EXTRA and INTRA in terms of likelihood (as in II-I above) and transfer control to one, and then the other if necessary, with request to diagnose unexplained abnormal data.

Principle 1 is implicit in the organisation of the conceptual structure. We saw several explicit examples of P2 in I.2a. and II.2 above. This particular example does not involve P3. Interpretation of imaging data will involve calls to the radiology expert, which is not explicitly Indicated in the above procedure, but which is an example of P4. The different forms of messages, as indicated by P5, are exemplified throughout the Procedure. DEFAULT-PROC employs P6 partially. This procedure has enough knowledge about subconcepts to call them for answers to specific questions, but keep the control for easier cases. The establishment of cholestasis in 1.1 is done in our implementation partly by weights assigned to different symptoms. This is an example of P7.

4,4 Procedure Stone-Consistency*

(We show only thatpart of the expert which determines Che fit; message and explanation details are not shown.)

I. CALCULATE X-RAY EVIDENCE
If Cholangiograms available
then If Stones seen in Biiary-tree
then XEV-<-3; else XEV<- -1
else If plain-film X-ray available
then if stones seen in Biliary-tree
then XEV <--3; else XEV<–O
else XEV«--0
If XEV > 2 return positive consistency evaluation

II. CALCULATE HISTORY EVIDENCE
This decision is made by asking about stone-related diseases: hemolytic- disease, cholecystitis etc.

III. CALCULATE CLINICAL EVIDENCE
This decision is made on the basis of the evidence for Chlsngitls, colicky-pain, vomitting and nausea in a weighted-sum logic.

IV. .CALCULATE SUMMARY EVIDENCE
Evidence from 1, 2 and 3 is combined in s weighted-sum logic to obtain the consistency •valuation.

5.0 PERFORMANCE OF MDX - A PROTOTYPE SYSTEM

5.1 Nature of Infraction

In this section we discuss the performance of the prototype diagnosis system(MDX) on s reel case. Currently the system can only diagnose extra-hepatic Cholestatic syndrome, which consists of about nine diseases represented as terminal nodes. The system is interactive in two senses• The user can enter as much data aa he wants, and later on the system will ask for additional data, if needed. Interaction of the second kind arises when an expert which is not yet implemented is nevertheless invoked by some other expert. In this situation, if a message is sent to an expert which is known to the system but not implemented, the message is routed to the user who can provide answers on behalf of the unimplemented experts (see [3] for details). Part of the attraction of expert systems is that they permit such a modular development in a conceptually natural way. This approach has the additional advantage that the logic of the system's decision-making becomes relatively clear to the physician and he can view the system as s consultant and not aa a competitor. We have space here to discuss only one case in some detail.

5.2 Example Case

This case is discussed in [4].Our discussion will be in the form of computer printout fragments (in Gothic) Interspersed with commentary. User response to queries during diagnosis is preceded by $ sign. In most places, the explanation generated by the system should be sufficient. Patient data are first entered in episodal form. For this case the following deta are entered for past medical history and the At admission episode.

```
                PATIENT RECORD SYSTEM
PATIENTS NUMBER IS PAT01
AGE? 39
SEX? M
ENTER HISTORICAL EVIDENCE
     ? HALOTHANE
     ? (ULC-COLITIS F)
     ? (BIL-SURGERY F)
     ? (CHOLECYSTITIS F)
     ? E
TIME? (AT ADMISSION)
     ? JAUNDICE
     ? PRURITUS
     ? VOMITTING
     ? (WTLOSS (AMOUNT (LB 16.0)))
     ? (PAIN F ABDOMEN)
? (LIVER (SIZE NORMAL))
? (PAIN LIVER F)
? (PAIN T EPIGASTRIUM MILD)
? (SPLEEN (FELT F))
? (URINE (BILIRUBIN))
? (STOOL (BLOOD F))
? (BILIRUBIN (DIRECT 10.799998) 12.200000)
? (PROTEIN 8.1 (ALBUMIN 3.9) (GLOBULIN 4.2))
? (SGOT 32)
? (ALK-PHOS 19.200000)
? (PT 11.399999 (CONTROL 10.5))
? (ESR 16.0)
? (WBC 9200 (PMN(PER 69.0))(LYMPHOCYTE (PER 19.0))
           (MONOCYTE (PER 10.0))
           (BASOPHIL (PER 2.0)))
? (ULTRASONAGRAM (BIL-TREE (IHD DILATE)))
? (PLAIN-FILM (LIVER (NORMAL))(DUODENUM (NORMAL)))
? (ERCP (BIL-TREE (CHEP OBSTRUCTION CONICAL)
                  (CBILE (NORMAL)))
        (GALL-BLADDER (NORMAL))
```

* Each of these four major decisions are made by a separate group of rules, which sre evaluated only when needed. For details aee[3].

```
        (PANCREAS (PANC-DUCT (NORMAL))))
? (GI-XRAY (DUODENUM (COMPRESSION)))
? (LIVER-SCAN (LIVER LARGE) (SPLEEN LARGE))
? (PERC-TRANS-CHOL
   (BIL-TREE (IHD DILATE)(CHEP NARROW)
             (LHEP MULTIFOCAL)(RHEP MULTIFOCAL)))
? (SPLAN-AGRAM
    (LIVER (SURFACE (TUMOR-BLUSH F)
                    (NEOVASCULATURE F))))
END OF DATA ENTRY FOR PATIENT PAT01
```

The entry of negative information, such as (PAIN F ABDOMEN) meaning "no abdominal pain", needs some explanation. Much of this information is obtained by routine physical examination, historical questionnaire and lab tests. However, information such as (ULC-C0L1TIS F) is not routine; it may be available from patient history charts, but more often it is extracted by questions cr tests when a particular hypothesis is being considered. The system is capabJe of asking for more data as needed, though that aspect is not our main concern here. We shall consider the version of our implementation in which any data not given are assumed to be unknown, unless they can be Inferred from known data (with the exception of queries answered by the User for unimplemented experts). In any case, the data that are entered here- both positive and negative- are those directly taken from the case description in the Journal. The important point is that there exists a genuine intellectual problem of putting the pieces together into a coherent whole even when only relevant data are entered. The problem of extraneous data is a bit more subtle. Clearly, totally irrelevant data should not cause any problems, and in fact they don't, No data-directed component has any rules that would fire in this case, and no hypothesis-driven component would look for them. Potentially relevant extraneous data are more problematic, but that applies to our svstem as well as to human diagnosticians. In this case blind alleys are possible, but erroneous diagnoses usually are not. Space limitations prevent a fuller discussion of this issue in this paper.

5.2.1 The Diagnosis

The topmost expert in the system is GP. As the data are entered! some rules are fired, suggesting hypotheses. The hypotheses become the starting point for GP to diagnose the case. In this case, rules suggesting Cholestasis and Post-operative-cholestasis were fired,

```
        BEGIN DIAGNOSIS FOR PATIENT PAT01
TRYING RULES TO INVOKE HYPOTHESIS
TRYING RULE:
 ((( PRESENT? (URINE BILIRUBIN))(NORM? ALK-PHOS AN))
   ((INVOKE CHOLESTASIS)))
SATISFIED
EXECUTING ACTIONS: ((INVOKE CHOLESTASIS))

TRYING RULE:(((HISTORY? ANESTHETICS))
                ((INVOKE (POST-OPER ANESTHETICS))))
SATISFIED
EXECUTING ACTIONS:
  ((INVOKE (POST-OPER ANESTHETICS)))

TRYING RULE:
  (((DEFORMITY? BIL-TREE ? STONE)) ((INVOKE STONE)))
UNSATISFIED:
```

Other rules involving Pruritus, Bilirubin and Jaundice and invoking cholestasis were also triggered.

```
INITIAL DATA INDICATES ---
 CHOLESTASIS
 (POST-OPER ANESTHETICS)
CHOLESTASIS IS INDICATED BY
     PRURITUS PRESENT
     JAUNDICE PRESENT
     URINE BILIRUBIN
     NORM OF ALK-PHOS IS (AN)
     NORM OF BILIRUBIN IS (AN)
     RATIO OF DIRECT VALUE VS
              BILIRUBIN VALUE IS 0.885
(POST-OPER ANESTHETICS) IS INDICATED BY
     HISTORY OF ANESTHETICS
```

CHOLESTASIS being the highest anchor node suggested, control is transferred to it with the message to Establish and Refine itself. The messages and some of the rules* used in establishing cholestasis are shown.

```
          HANDING OVER CONTROL TO CHOLESTASIS EXPERT
EXPERT-CALLED: CHOLESTASIS  CALLED-BY: GP
MESSAGE   (QUERY (ESTABLISH CHOLESTASIS)
                 (REFINE CHOLESTASIS))
```

Procedure Cholestasis described in an earlier section is now operational.

```
TRYING TO ESTABLISH CHOLESTASIS
ENTERING TABLE LOGIC
TRYING RULE:  (((DUCT? IHD BIOPSY BILE-STASIS)
                (PRESENT? JAUNDICE)) ((T 3)))
UNSATISFIED:
EXITING TABLE LOGIC
PATHOGNOMONIC EVIDENCE IS 0
```

Similarly rules such as that involving large liver size result in accumulating physical evidence, and rules triggered by bilirubin, SGOT and other lab data are used to construct a more structured clinical evidence.

```
PHYSICAL EVIDENCE 4
CLINICAL EVIDENCE (8 4 0 2)
```

A symbol-matching table such as in Procedure Stone-Consistency is finally used to decide cholestasis on a scale of -3 to 3. Notice the difference between the use of rules for suggesting cholestasis in GP and for establishing cholestasis in this node. In the latter, evidence is methodically accumulated and weighted to reach a decision.

CHOLESTASIS FIT: (ESTABLISHED CHOLESTASIS 3)

GP had passed two related suggestions to Cholestasis, namely Post-Oper and Anesthetics as invoked suggestions (I.e. suggestions made in a data-directed fashion)and these experts are now called(see 1,2 in Procedure Cholestasis). The message actually gets route** to the

* In the rest of the discussion only a few of the rules from each conceptual group will be shown. Each decision within an expert is made by a rule-group or a Procedure. The representational details will be described in[3]. Rule conditions are actually logical queries to the data-bass. The details of the data-base organization and queries can be found In[5].

**The routing was done automatically by the communication system and CHOLESTASIS did not know that POST-OPER and its ancestor INTRA-HEP arc not implemented. The answer docs indicate, however, that the decision was made by the USER. Because POST-OPER is the immediate super-concept of ANESTHETICS, only one message was sent, though individual answers were returned.

USER who makes the decision that Post Oper ls_ unlikely .

```
EXPERT-CALLED:  INTRA-HEP  CALLED-BY:  CHOLESTASIS
MESSAGE   (OUERY  (REFINE  POST-OPER)
                  (CONSIDER  ANESTHETICS))
EXPERT-CALLEU:  POST-OPER  CALLED-BY:  INTRA-HEP
MESSAGE   (OUERY  (REFINE  POST-OPER)
                  (CONSIDER  ANESTHETICS))
FOR EACH DISEASE LISTED INDICATE WHETHER IT IS
ESTABLISHED
UNLIKELY OR UNKNOWN . ANSWER Y-R-U
POST-OPER?$N
ANESTHETICS?$N

EXPERT-CALLED:  INTRA-HEP  CALLED-BY:  USER
MESSAGE (ANSWER  (UNLIKELY  ANESTHETICS  BY-USER)
                 (UNLIKELY  POST-OPER  BY-USER))
```

Control now passes to II.1, i.e., CHOLESTASIS now tries to decide which of EXTRA or INTRA is more likely.

```
TRYING TO DECIDE BETWEEN EXTRA AND INTRA-HEPATIC
CONDITION:  (DEFORMITY?  IHD  (CT-SCAN  ULTRASONAGRAM)
                         DILATE)
VALUE:  T
CONUITION:  (MORPH?  BIL-TREE  PERC-TRANS-CHOL  (NORMAL))
VALUE:  F
CONUITION:  (MORPH?  BIL-TREE  ERCP  (NORMAL))
VALUE:  F
VALUE RETURNED:  (EXTRA-HEP  T  F  F)
SELECTED:  EXTRA-HEP
```

Following II.2, control transfers to EXTRA-IN-CHOLESTASIS. Here heuristic rules are tried to suggest possibilities from among the subconcepts of extra-hepatic cholestasis.

```
TRYING RULES TO INVOKE HYPOTHESIS
TRYING RULE:
  (((PAIN?  ABDOMEN  COLICKY))  ((SUGGEST  STONE)))
UNSATISFIED:

TRYING RULE:  (((MORPH?  GALL-BLADDER  ?  GB-SHADOW))
               ((SUGGEST  G8-CANCER)))
UNKNOWN

TRYING RULE:  (((DEFORMITY?  BIL-TREE  ?  (NARROW)))
               ((SUGGEST  SCL-CHOLANGITIS  BD-CANCER)))
SATISFIED
EXECUTING ACTIONS:
  ((SUGGEST  SCL-CHOLANGITIS  ED-CANCER))
SUGGESTIONS IN EXTRA-CHOL  (3D-CANCER  SCL-CHOLANGITIS)
```

CHOLESTASIS first calls BD~CANCER(via EXTRA-HEP).

```
TRYINu TO ESTABLISH BD-CANCER
EVIDENCE-SITE OF BIL-TIMCR  (T  (CHEP))
TRYING TO INFER BILIARY OBSTRUCT ION
EVIDENCE-SITE OF BIL-OBSTRUCTION  (T  (CHEP))

TRYING RULE:

  (((AND-TRI  (CAR  TUMOR)  (CAR  OBST)
              (AROUND  (CADR  TUMOR)  (CADR  OBST)))
   f(T 3)))
SATISFIEu:

TRYING RULE:
  (((DEFORMITY?  BIL-TREE  BIOPSY  TUMOR))  ((T -3)))
UNKNOWN:

EVIOENCE-OF TUMOR IN BILE-UUCT 3
ESTABLISHED BO-CANCER 3
```

```
MESSAGE  (ANSWER(ESTABLISHED  BD-CANCER  3))
```

reaches CHOLESTASIS via EXTRA-HEP.
BD-CANCER made the decision after asking questions about where the obstruction and tumor in the biliary-tree may be(both questions are complicated queries posed to the Radiology expert , not shown here, which examines the relevant radiological information to make the Inference).

SCL-CHOLANGITIS is called next, but returns an unlikely evaluation after detailed processing of relevant information.

```
TRYING TO ESTABLISH SCL-CHOLANGITIS
TRYING RULE:
  (((AND-TRI  (DUCT?  BIL-TREE  BIOPSY  FIBROSIS)
              (DUCT?  BIL-TREE  BIOPSY  (CARCINOMA
                                         ATY-EPITH-CELL))
         (HISTORY?  ULC-COLITIS)))
   ((T 3)))
UNSATISFIED:
PATHOGNOMONIC  EVIDENCE  IS  0
```

X-ray evidence is evaluated to be -2 by use of a logic similar to that shown in the decision between EXTRA and INTRA earlier.

```
XRAY EVIDENCE  IS  -2
SUMMARY EVIDENCE  IS  -2
UNLIKELY  SCL-CHOLANGITIS  -2

EXPERT-CALLED:  EXTRA-HEP  CALLED-BY:  SCL-CHOLANGITIS
MESSAGE   (ANSWER  (UNLIKELY  SCL-CHOLANGITIS  -2))
```

At this point, there is one refinement of EXTRA-HEP in ACTIVE list and CHOLESTASIS has to decide if all Important data have been covered by the active hypotheses. This decision is currently made by the user.

```
ACTIVE:  BD-CANCER  EXTRA-HEP
UNLIKELY:  SCL-CHOLANGITIS  POST-OPER  ANESTHETICS
UNKNOWN:
```

The problem of when to stop is still an open problem in diagnostic knowledge-based systems, though some approaches based on thresholding confidence measures have been trled(6,7]. We are working on a symbolic matching criterion for this purpose and preliminary design has been completed for explaining the major findings in extra-hepatic cholestasis (see [3] for details). In this case, as the CPC discussion shows(4), all the data have been explained. However, in order to demonstrate the rest of the system, let us say NO to the question:

```
ARE ALL DATA EXPLAINED BY ACTIVE DIAGNOSIS?$N
```

US now are in IV in Procedure Cholestasis. LOOK-FURTHER hands over control to EXTRA-HEP, asking it to see if it can refine the diagnosis any further (but limiting its search by asking It to consider only the typical diseases). EXTRA-HEP tries its own heuristic rules, but no suggestions are made. It then falls back on an exhaustive search and calls * each typical expert under it, one by one. As it turns out, all return an unlikely evaluation.

```
EXPERT-CALLED:  EXTRA-HEP  CALLED-BY:  CHOLESTASIS
MESSAGE   (OUERY  (REFINE  EXTRA-HEP)
                  (CONSIDER  TYPICAL))
TRYING RULES TO INVOKE HYPOTHESIS
```

* Details of calls and processing by these experts have been skipped.

```
TRYING RULE:
   (((EQUAL? JAUNDICE INTENSITY SEVERE)
     (EOUAL? JAUNDICE TREND UNRELENTING))
    ((SUGGEST AV-CANCER)))
UNKNOWN
SUGGESTIUNS IN EXTRA-HEP NIL
```

EXTRA-HEP returns control back to CHOLESTASIS with the result of its processing namely failure to establish any more concepts, but able to pretty much rule out all the other possibilities. The point of continuing the diagnosis, even though the correct diagnosis had already been made, was to demonstrate the role of suggestion rules at all levels In the conceptual hierarchy as a means of achieving efficiency. If bd-cancer had not bean suggested at the level of CHOLESTASIS, It would have been established when control was handed over to EXTRA-HEP. The same effect occurs at all levels(including within tip-node experts, which usually have pathognomonic rules) in the hierarchy.

```
EXPERT-CALLED: CHOLESTASIS CALLED-BY: EXTRA-HEP
MESSAGE (ANSWER (REFINED EXTRA-HEP))

ACTIVE: Bu-CANCER EXTRA-HEP
UNLIKELY: AV-CANCER AMP-CANCER GB-CANCER STRICTURE
          STONE SCL-CHOLANGITIS POST-OPER ANESTHETICS
UNKN0WN:
     ARE ALL DATA EXPLAINED BY ACTIVE DIAGNOSIS?$YES
```

The diagnosis is summarized on a scale of -3(Reject) to 3(Definite).

```
EXPERT-CALLED: GP CALLED-BY: CHOLESTASIS
MESSAGE (ANSWER (ESTABLISHED CHOLESTASIS 3))

DISEASES CONSIDERED AND FOUND UNLIKELY

AMP-CANCER -2                SCL-CHOLANGITIS -2
GB-CANCER -2                 POST-OPER BY-USER
STRICTURE -3                 ANESTHETICS BY-uSER
STONE -2                     AV-CANCER -3

          CONSIDERED AND UNABLE TO DECIDE
THE FINAL DIAGNOSIS IS —
  CHOLESTASIS 3
  EXTRA-HEPATIC INDICATED WITH OBSTRUCTION AT (CHEP)
  BD-CANCER 3
```

There are three important points to note about this example. First, the diagnosis is correct (though systems based on other approaches may have done as well In this case). Second, the diagnostic knowledge was used at the correct place, once the proper context had been established. For example, it would be premature and often disastrous to try to establish tip-level concepts like bile-duct cancer, unless cholestasis had been established and extra-hepatic indicated as more likely than intra-hepatic. Finally, the problem of coping with a large volume of data at a single level was avoided by using the patient data at the appropriate level of specificity and importance. In other words, the system tries to explain different types of data at those levels in the conceptual hierarchy where they are moat relevant,

6.0 COMPARISON WITH RELATED SYSTEMS

6.1 INTERNIST

INTERNIST is perhaps the most comprehensive AI medical program: it deals with 80% of Internal medicine. The diseases are organized in an orgnn-baaed hierarchy. Disease nodes are linked by causal and associative links. In addition, each datum (type) has "manifestation of" and "invoked by" links with disease nodes (with some weights attached to these links), INTERNIST performs diagnoses by calculating a numerical score for diseases baaed on how many manifestations are present, fraction of prototypical findings present, support from causal diseases and a measure of how much unexplained data is explained by a hypothesis. The disease hierarchy and a partitioning of current hypotheses into complementary and supplementary diseases helps t6 create some focus and cut down the complexity of proceasing. The basic problem-solving strategy is one of pursuing the highest scoring hypothesis in a global way. A new datum introduced at any time leads to a recomputation of all eatlmates. This could also lead to sudden shifts of focus. Much of the Intelligence of the program is embodied in the complex scoring function. Thus explanation of diagnosis is conceptually opaque.

6.2 Present Illness Program (PIP)

This program is designed to suggest Possible diseases in the area of renal diseases. Knowledge about clinical and physiological states (not all being diseases) Is organized into frames. Each frame contains information about relationships with other states; triggers to Invoke statea and differential diagnosis; and a table of probabilistic information to estimate the likelihood of the state, baaed on patient data. There are also rules which can determine the likelihood of states or rule them out in some cases.

The basic strategy is similar to that of INTERNIST, i.e., the invoked hypotheses are ordered by their likelihoods and the top one at any given time is pursued. A hypothesis is accepted or rejected if It croases a threshold. The likelihood estimate includes the support from complementary states, match with the prototypical findings, and the degree to which patient data are explained by the hypothesis.

Again, as in INTERNIST, a new datum leads to potentially expensive recomputatlon of all estimates, and possible focus shifts. Further the use of Bayesian estimation In a global way would add to the computational and focus problems. Discussion of some problems associated with Bayesian estimates can be found In [11] and [12]. Aa a last comment, the frames are interpreted in a uniform way. It would be difficult to use specialized knowledge to bear on the problem at the appropriate place, unless It can be caat in the general mold.

6.3 MYCIN

MYCIN advises clinicians in the area of infectious diseases. The knowledge is stored in the form of deductive rules, with some associated certainty factor. The program also has available to it a context hierarchy, which allows it to select rules based on the context and ask for appropriate data.

The rules are used in a backward-chaining deductive fashion, to first find the cause(s) of infection and then suggest therapy for it. Much of the real power of the system seems to us to come from the ability of the context hierarchy to use the rules in a controlled way. More generally, while rules are a convenient way to express certain kinds of knowledge, many other kinds of knowledge are not easily encoded in rule form (see [13] for a case study In this regard).

6.4 CASNET

CASNET performs diagnoais and recommends therapy in

glaucoma diseases. The knowledge Is represented in a causal network of dysfunctional states with wall defined starting (etiologic) and ending states. Evidence about the likelihood of these states can be obtained from another network relating states to tests. ▲ test may be a patient datum or combinations of those. Diseases are defined as paths in the causal net.

The system calculates the status of each state, baaed on the test values. This computation is repeated as new data are entered. It also computes the weight of each mode, which ia Independent of its status, but depends on those nodes which occur in the same path. Diseases are hypothesized end verified on the basis of which paths are most likely or potentially able to account for all confirmed states. There is a computational burden in calculating and recalculating the status and weight of each node in the network. Given the exponential nature of Such computation, it seems unlikely that such an approach would be eaaily extended to larger domains. Further, the use of patho-physlologlcal states is, perhaps, not possible for most of medicine, as such machanisms are known only spottily. A relatively minor problem is created by the fact that evidence from different tests can be combined by creating a "higher" teat-entity. In many medical situations, where patient data combine in different ways to suport or disprove disease hypotheses, many new test-entitles would need to be created for such combinations.

7.0 CONCLUDING REMARKS

While the preceding pages can be viewed as a partial attempt at uncovering the structure of human cognition In the diagnosis taak, the real teat of the ideas proposed is in how well the system baaed on them works. In addition to the case presented in an earlier section, several other extra-hepatic cases (both from medical journals as well aa from a practicing hepatologlst) have been tried, and in all caaea the diagnostic behavior was natural and closely corresponded to the physicians' reasoning. The system is constantly being modified, but not in the sense of "patching" but to increaae the quality of the knowledge. For instance, it has been found useful to place certain rules higher in the hierarchy. In soma cases, the table logic of some tin nodes has been found inaccurate, and, after consultation with medical texts and experts, modifications have been made. These seem to us to be natural ways of improving the power. Of courae, the modifications to local procedures should converge eventually. We cannot say that the system has yet passed this test.

We envisage the next stage in our work to be enlarging the system to handle the whole of the Cholestasis syndrome. We hope to study the effects of upward scaling in this enlargement. For one thing, a more powerful data baae organization for temporal phenomena will be needed. However, we shall not opt for a separate temporal expert, aa this would violate our constraint on control transfers. Instead temporal knowledge will be distributed as needed.

Our current approach consists of giving the system all the data that are available in the journal description of the caae or from the collaborating hepatologlst. In either caae, the case Is in a sense "complete." In reality, however, aa the diagnostic process unfolds, tests are ordered to confirm or reject hypotheses. Thus the ability to order tests would be e useful extension to our system. The conceptual information needed is partially implicit in the knowledge structure (queries to the data baae can be converted to orders for tests). Similarly, since the inter-expert messages contain purposes and responses, the basis for an explanation capability is already present. However, more work is needed to realize these facilities in an elegant manner, i.e., without too much detail and with appropriate abstractions. Explanation and test-ordering facilities would enhance the use of the system as a consultant.

In summary, we have attempted, in this paper, to lay the foundations for a knowledge representation scheme, which derives its power because of its ability to capture the structure of a body of knowledge at an appropriate level of depth. There are many interesting issues related to conceptual structures we have not had occasion to discuss in this paper. These and our experiences with problem-solving systems for larger domains of knowledge will be the subject of future reports.

ACKNOWLEDGMENT: We are indebted to Douglas Levin,M.D., for his interest and consultations.

REFERENCES

[1] Minsky,M. "Plain Talk about Neurodevelopmental Epistemology." In Proc. IJCAI-77. MIT, Cambridge, Mass., August, 1977, pp.1083-1092

f2] Hayes-Roth,F.,Lesser,V.k. "Focus of Attention in the Hearsay- II Speech Understanding System." In Proc. IJCAI-77. MIT, Cambridge, Mass., August, 1977, pp.27-35

[3] Mittal,S. Ph.D. Dissertation(forthcoming), Dept. of Com p. and Info. Sc, The Ohio State University, Dec., 1979

[4] "Case 45-1977, Massschussetts General Hospitsl." New England Journal of Medicine. Nov.10, 1977, pp.1054-1059

[5] Mittal.S., Chandrasekaran ,B. "Conceptual Rapresentation of Patient Data Bases." OSU-TR-79, Dept. of Comp. and Info. Sc, The Ohio State university, April,1979

[6] Pople,ii.E.,hyerB,J.D.,iiiller,R.A. "DIALOG: A Model of Diagnostic Locic for Internal Medicine." In Proc. IJCAI-75. Tbilsi, USSR, Sept.,1975, pp.846-655

[7] Shortliffe.E.H. Computer Cased Medical Consultations: MYCIN Elsevier -North-Holland Inc., 1976

[8] Pople,H.E. "The formation of Composite Hypothesis in Diagnostic Problem Solving: An Excercise in Synthetic Reasoning." In Proc, IJCAI-77. LIT, Cambridge, Mass., August, 1977, pp.1030-1037

[9] Pauker,S.,Gorry,G.A.,i:a3Sirer , J., Schwartz, W. "Towards a Simulation of Clinical Cognition." The American Journal of Medicine 60(1976)981-996

[10] Weiss,S.n. ,Kulikowski,C.A. ,Amarel,S. ,Safir,A. "A Model-based Method tor Computer-Aided Medical Decision-Making." Artificial Intelligence. 11$2(1978) 145-172

[U] Szolovits,P.,Pauker,S.G. "Categorical and Probabilistic Reasoning in Medical Diagnosis." Artificial Intelligence. 11:2(1978) 115-144

(12] keinstein,A.K. "Clinical Blostatistics XXXIX. The haze of bayes, the Aerial Palaces of Decision Analysis and the Computerized Ouija Board." Clinical Pharmacology and Therapeutics. 21:4(1977) 482-490

[13] Reggie,J.A. "A Production Rule System for Neurological Localization." In Proc. Second Annual Symposium on Computer Applications in Medical Care. November, 1978, pp.254-260

RESOLUTION PLANS IN THEOREM PROVING

C. L. Chang
IBM Research Laboratory
5600 Cottle Road
San Jose, California 95193

Briefly, an example of a resolution plan can be described as follows: Let LI v A and L2 v B be two clauses, where LI and L2 are literals, and A and B are clauses. If LI and L2 can be made complementary by some substitution, we shall call A v B (L1,L2) a resolution plan. In general, a resolution plan is represented by

C (LI,MI)...(Lr,Mr) ,

where LI,MI,...,Lr,Mr are literals, and C is a clause. If C is empty, (L1,M1)...(Lr,Mr) is called a total plan. The total plan corresponds to a proof if there is a unifier which simultaneously make (Li,Mi) complementary, i=I,...,r. As shown elsewhere, the total plan approach can eliminate redundancies. In this paper, we shall show that all the strategies developed for resolution can be used to generate total plans.

1. INTRODUCTION

In [Chang and Slagle 1977], the concept of total plans for proving theorems in first order logic was proposed. A total plan is denoted by a sequence of pairs of literals, (LI,MI)... (Lr.Mr). If there is a substitution 0 that can make each of (L1,M1)9,...,(Lr,Mr)0 a complementary pair of literals, the plan is said to be acceptable, and 0 is called a solution. In [Chang and Slagle 1977], a method based upon rewriting rules for generating total plans has been proposed. In this paper, we shall show that we can use resolution strategies [Chang and Lee 1973, Kowalski and Kuhner 1971, Kowalski 1975, Robinson 1965a, 1965b, Wos et al. 1965] to generate total plans. This is done by generating resolution plans. Formal definitions of resolution plans will be given in the sequel. They are defined in terms of notation of literals, instead of literals themselves. Because of this, unification is not actually carried out during generations of resolution plans. This is in constrast to resolution where unification has to be performed. Therefore, It is faster to generate resolution plans than resolvents.

In the following, we shall first give connection graphs, then we shall define various resolution plans such as binary resolution plans, hyper resolution plans, linear resolution plans, etc.

2. CONNECTION GRAPHS

The concept of a connection graph has been used in [Kowalski 1975, Sickel 1976]. Essentially, a connection graph is a data structure for a set of clauses indicating possible refutations. We shall now give a definition of a connection graph.

A pair of literals LI and L2 are called potentially complementary if LI and L2 can be made complementary by applying some substitution, after renaming variables so that LI and L2 share no variables.

A connection graph for a set S of clauses is constructed as follows:

(1) Every clause in S appears in the graph exactly once;

(2) For every clause, Ll v...v Ln, in S, where Ll,...,Ln are literals, it is represented in the graph as

Ll Lr

Fig. 1

(3) For every pair of potentially complementary literals, draw an edge connecting the literals. (Note that edges between literals in the same clause are allowed.)

Example 1. Fig. 2 is a connection graph for the set, {P(x) v Q(y), iP(a), iQ(b)J.

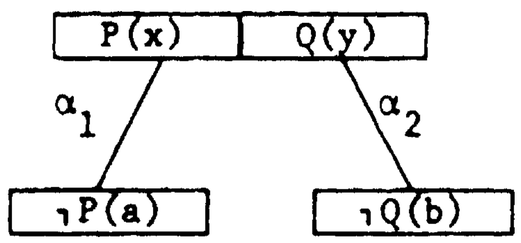


Fig. 2

Example 2. Fig. 3 is a connection graph for the set, (¬P(X) V P(f(x)), P(a), iP(f(f(a)))}. Note that there is an edge connecting literals iP(x) and P(f(x)) in the same clause.

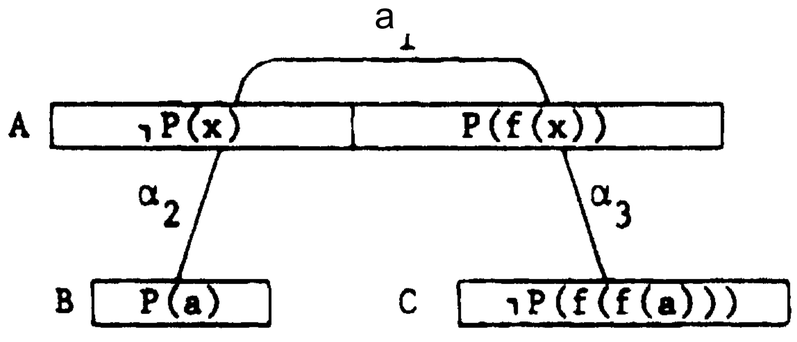


Fig. 3

In the sequel, we shall label a clause by a distinct name and then refer to a literal In the clause by its position in the clause. That is, if C is a clause, then Cn is the n-th literal (counting from the left) of clause C, where n is an integer. By this convention, Fig. 3 will be represented by Fig. A.

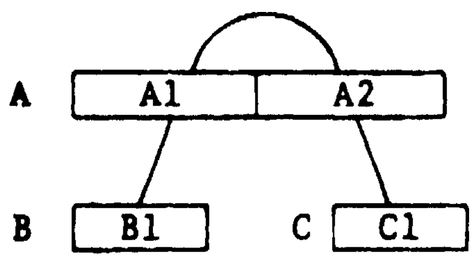


Fig. 4

3. BINARY RESOLUTION PLANS

In terms of a connection graph for a set S of clauses, we now give the following definitions.

A clause plan is represented by a clause followed by a sequence of pairs of literals. A clause plan can be represented as

Ll v ... v Lp (Ml,Nl)...(Mq,Nq) ,

where Li, Mj and Nj, i=l,...,p, j=l,...,q, are literals. We call Ll v ... v Lp a clause segment, and (Ml,Nl)...(Mq,Nq) a plan segment of the clause plan. A clause plan is called a total plan if and only if its clause segment is empty.

Definition. Let two clause plans A and B respectively be denoted by

A=C P
B=D Q,

where C and D are the respective clause segments of A and B, and P and Q are the respective plan segments of A and B. If there are literals Ll,...,Lm in C, and Ml,...,Mn in D such that there are edges in a connection graph connecting every literal of Ll,...,Lm to every literal of Ml,...,Mn, then the clause plan

(C-C') v (D-D')(L1,M1)(L1,M2),...(Lm,Mn)P Q ,

where C'={Ll,...,Lm} and D'={Ml,...,Mn}, is called a binary resolution plan of A and B. Note that (Ll,Ml)(Ll,M2)...(Lm,Mn) is a cartesian product of C' and D', and may be represented by (C',D').

Example 3. Let A=P(x) V Q(x) and B=¬P(a) v R(x). Using the positions of literals in the clause to name the literals, A and B are represented as

A = A1 v A2
B = B1 v B2,

where A1, A2, B1 and B2 are P(x), Q(x), ¬P(a) and R(x), respectively. Note that using the notation in the above definition, C={A1,A2} and D={B1,B2}, and both P and Q are empty. Since A1 and B1 are complementarily unifiable, we choose C'={A1} and D'={B1}. Therefore, a resolution plan is obtained as follows:

(C-C')U(D-D')(C',D')P Q
= ({A1,A2}-{A1})U({B1,B2}-{B1})({A1},{B1})
= {A2}U{B2}(A1,B1)
= {A2,B2}(A1,B1)
= A2 v B2(A1,B1).

We can use resolution strategies to generate total plans for refutations. That is, given a set S of clauses, we first build a connection

graph for S. Then, to check whether or not a pair of literals are potentially complementary, we need only to look at the connection graph to see if there is an edge connecting the literals. Once we have a connection graph, resolution plans can be generated faster than resolution because unification does not have to be performed. A total plan is a clause plan whose clause segment is empty. We can use resolution strategies to generate total plans. Once a total plan is generated, we then perform unification at the last step to check whether or not the plan is acceptable.

Example 4. Consider the set S of the following clauses,

```
¬T(x,y,u,v) v P(x,y,u,v)
¬P(x,y,u,v) v E(x,y,v,u,v,y)
 T(a,b,c,d)
¬E(a,b,d,c,d,b).
```

The connection graph for S is shown in Fig. 5.

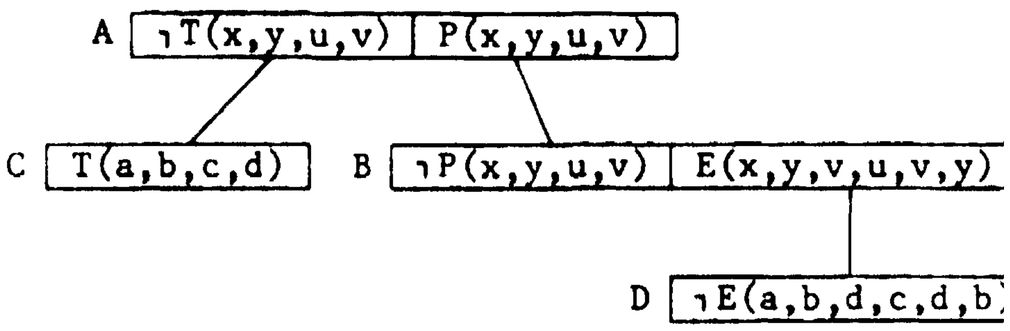


Fig. 5

Using literal names, Fig. 5 can be represented by Fig. 6.

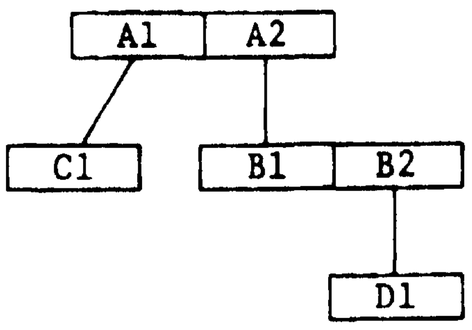
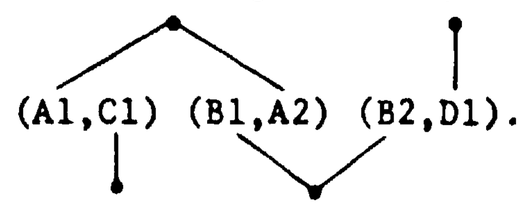


Fig. 6

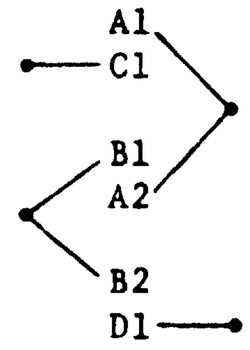
Now, using Fig. 6, we can generate the following resolution plans:

(1) A1 v A2	a clause in S
(2) B1 v B2	a clause in S
(3) C1	a clause in S
(4) D1	a clause in S
(5) B1 (B2,D1)	a resolution plan of (2) and (4)
(6) A1 (B1,A2) (B2,D1)	a resolution plan of (1) and (5)
(7) (A1,C1) (B1,A2) (B2,D1)	a resolution plan of (3) and (6).

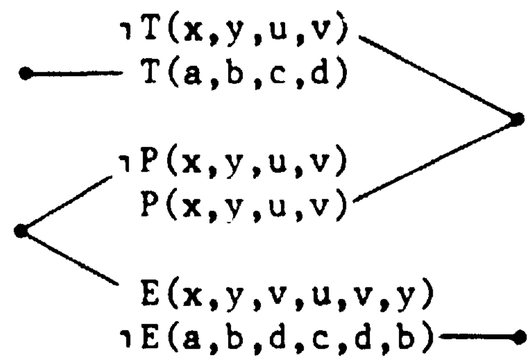
Now, since we want to perform unification, we first need to rename variables. As discussed in [Chang and Slagle 1977], for the total plan (A1,C1)(B1,A2)(B2,D1), all literals belong to a clause are linked to a common dot. Therefore, we obtain the following linked plan,



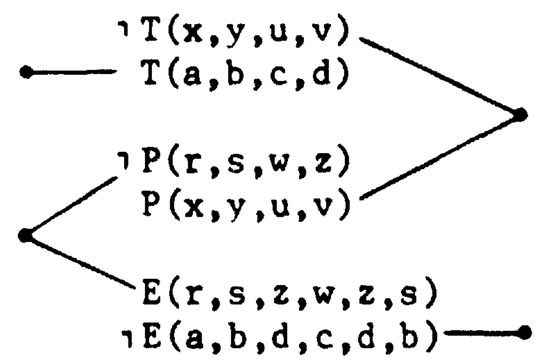
The above linked plan can also be represented as



Substituting the real literals for A1, A2, B1, B2, C1 and D1 in the linked plan, we obtain



Renaming the variables so that no clauses share variables in common, we obtain



Unifying the two expressions,

(T(x,y,u,v), P(r,s,w,z), E(r,s,z,w,z,s)) and
(T(a,b,c,d), P(x,y,u,v), E(a,b,d,c,d,b)),

we obtain a substitution

θ = {a/x, b/y, c/u, d/v, a/r, b/s, c/w, d/z} .

This substitution is a solution of the set M of the following clauses,

$$\begin{cases} \neg T(x,y,u,v) \vee P(x,y,u,v) \\ \neg P(r,s,w,z) \vee E(r,s,z,w,z,s) \\ T(a,b,c,d) \\ \neg E(a,b,d,c,d,b) \end{cases},$$

because Mθ is

$$\begin{cases} \neg T(a,b,c,d) \vee P(a,b,c,d) \\ \neg P(a,b,c,d) \vee E(a,b,d,c,d,b) \\ T(a,b,c,d) \\ \neg E(a,b,d,c,d,b) \end{cases},$$

and is truth-functionally unsatisfiable.

4. LINEAR AND SET-OF-SUPPORT RESOLUTION PLANS

Definition. Given a connection graph for a set S of clauses, C_n is called a linear resolution plan if and only if it is obtained as shown in Fig. 7, where CQ is a clause in S, and for $1=0,1,\ldots,n-1$, (a) C_{i+1} is a resolution plan of C_i and B_i, and (b) each B_i is either in S, or is a C_J for some j, $j<i$.

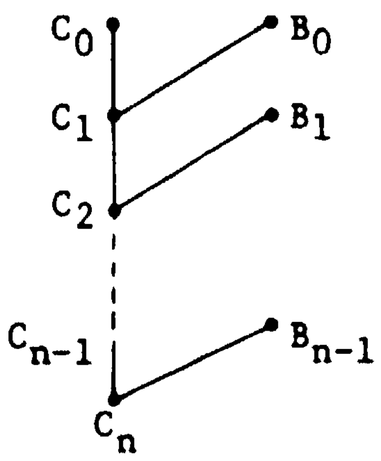


Fig. 7

Example 5. The deduction obtained in Example 4 is a linear deduction as shown in Fig. 8. That is, the total plan, (A1.C1)(B1,A2)(B2.D1), is a linear resolution plan.

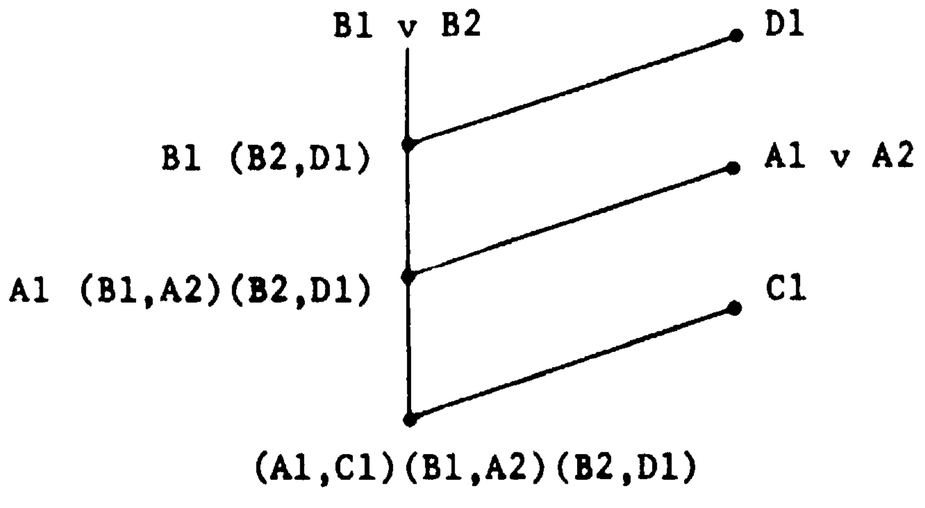


Fig. 8

Definition. A subset T of a set S of clauses is called a set of support of S if (S-T) is satisfiable. A set-of-support resolution plan is a binary resolution plan of two clauses that are not both from (S-T).

Example 6. Consider the set S of clauses given in Example 4. Now, if we let T be the set $\{\neg E(a,b,d,c,d,b)\}$, then the deduction given in Fig. 8 is also a set-of-support deduction. (Note that in general, a set-of-support deduction needs not be a linear deduction.)

5. HYPER RESOLUTION PLANS

In order to define a hyper resolution plan, we shall name atomic formulas in every literal of a clause, instead of naming the literal itself as given in the previous sections. That is, if C is a clause, then the n-th literal of clause C is named as Cn if the literal is positive, and named as ¬Cn if it is negative. For example, if

$$A = P(x) \vee Q(x) \text{ and}$$
$$B = \neg P(a) \vee R(x),$$

then A and B can be respectively represented as

$$A = A1 \vee A2 \text{ and}$$
$$B = \neg B1 \vee B2.$$

We say that a clause plan is positive if its clause segment Is positive. Otherwise, it is negative.

Definition. Let C be a negative clause plan, and let $B_0, B_1, \ldots, B_{n-1}$ be positive clause plans. Then, C_n is a hyper resolution plan, as shown in Fig. 9, if C_n is a positive clause plan obtained by a deduction shown in Fig. 10, where C_i is a binary resolution plan of C_{i-1} and B_{i-1}, i = 1 .

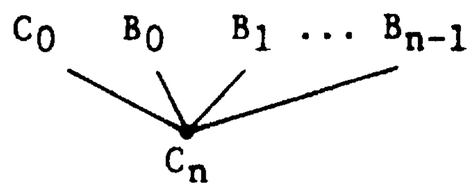


Fig. 9

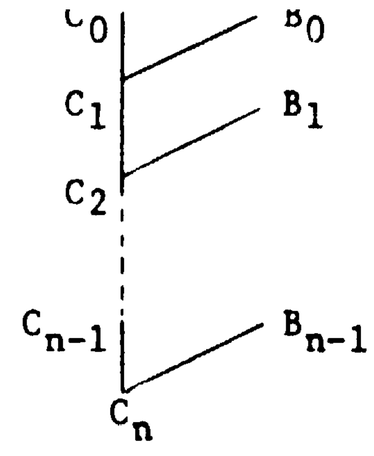


Fig. 10

Example 7. Consider the set S of clauses given in Example 4. Using the clause names in

Fig. 5, we can represent the clauses in S as follows:

(1) ¬A1 v A2
(2) ¬B1 v B2
(3) C1
(4) ¬D1.

Then, the deduction shown in Fig. 11 is a hyper deduction, while the one shown in Fig. 12 is not.

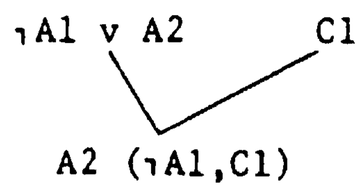


Fig. 11

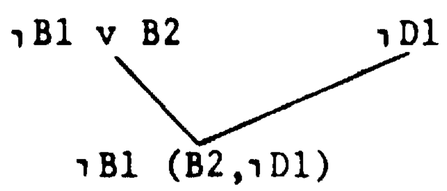


Fig. 12

6. RENAMING VARIABLES AND HANDLING DUPLICATE CLAUSE PLANS

One of the important steps in theorem proving is renaming variables. In ordinary resolution, before a resolution is performed, variables in clauses need to be renamed so that the clauses share no variables in common. Herbrand's Theorem says that a set S of clauses is unsatlsfiable if and only if there is a finite unsatisfiable set S' of ground instances of clauses of S. Renaming variables is closely related to Herbrand's Theorem because more than one ground instances in S' may come from the same clause C in S, and the ground Instances are essentially obtained by first renaming variables in copies of C.

Given a set S of clauses, we first generate a deduction of a total plan from S, and then check if the total plan is acceptable by renaming variables and by performing unification. In the deduction of the total plan, the same clause plan may appear more than once. In this case, we shall use primes to distinguish the different copies of the clause plan. This is illustrated by the following example.

Example 8. Consider the set S of clauses given in Example 2. That is, S consists of the following clauses

¬P(x) v P(f(x))
P(a)
¬P(f(f(a))).

The connection graph of S is shown in Fig. 3. Using literal names, Fig. 3 can be represented by Fig. 13.

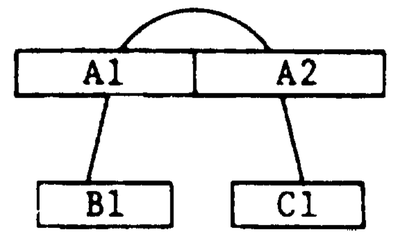


Fig. 13

Using Fig. 13, we can generate a deduction of a total plan shown in Fig. 14.

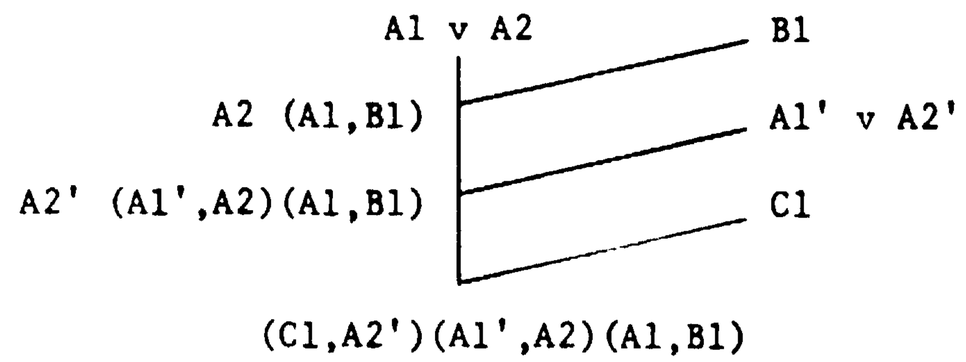
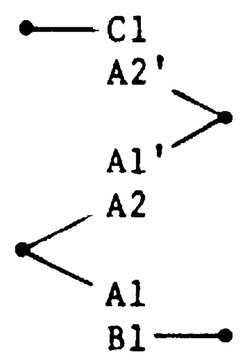
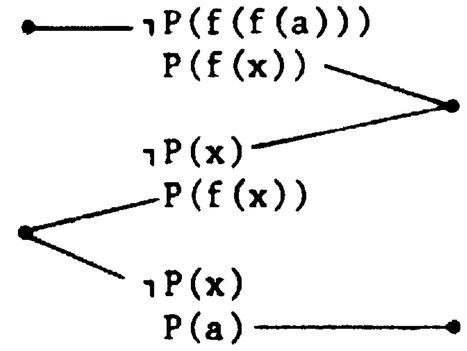


Fig. 14

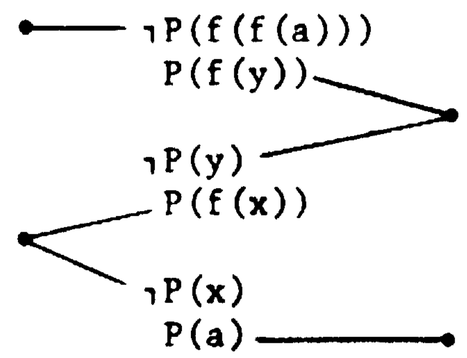
Note that two copies of (Al v A2) are used in the deduction, and we use (Al* v $A2^1$) to represent the second copy. By linking literals belonging to the same copy of a clause, we obtain



Substituting the real literals for A1, A2, A1', A2', B1 and C1, we obtain



Renaming the variables so that no clauses share variables in common, we obtain



Unifying the two expression,

$$(P(f(f(a))), P(y), P(x)) \text{ and}$$
$$(P(f(y)), P(f(x)), P(a)),$$

we obtain a substitution

$$\theta = \{f(a)/y, a/x\} .$$

This substitution is a solution of the set M of the following clauses,

$$\begin{cases} \neg P(x) \vee P(f(x)) \\ \neg P(y) \vee P(f(y)) \\ P(a) \\ \neg P(f(f(a))), \end{cases}$$

because $M\theta$ is

$$\begin{cases} \neg P(a) \vee P(f(a)) \\ \neg P(f(a)) \vee P(f(f(a))) \\ P(a) \\ \neg P(f(f(a))), \end{cases}$$

and $M\theta$ is truth-functionally unsatisfiable.

7. CONCLUSIONS

We have defined various resolution plans. These plans are obtained from the connection graph of a set S of clauses. To generate resolution plans, no unification needs to be performed. Therefore, it is faster to generate a resolution plan than to generate a resolvent. Unification is performed only at the end of a proof. One of the advantages of using the resolution plan method is that all the strategies proposed for resolution can be directly applied to generate resolution plans.

ACKNOWLEDGMENTS

The author would like to thank Dr. J. R. Slagle of Naval Research Laboratory, Washington, D.C. for his conversations.

REFERENCES

Chang, C.L., and Lee, R.C.T. (1973): Symbolic logic and mechanical theorem proving, Academic Press, New York.

Chang, C.L., and Slagle, J.R. (1977): Using rewriting rules for connection graphs to prove theorems, IBM Research Report RJ2117, San Jose, Ca. (To appear in Artificial Intelligence, 1979.)

Kowalski, R., and Kuhner, D. (1971): Linear resolution with selection function, Artificial Intelligence 2, 227-260.

Kowalski, R. (1975): A proof procedure using connection graphs, JACM 22, 4, 572-595.

Robinson, J.A. (1965a): A machine-oriented logic based on the resolution principle, JACM 12, 1, 23-41.

Robinson, J.A. (1965b): Automatic deduction with hyper-resolution, Internat. J. Comput. Math. 1, 227-234.

Sickel, S. (1976): Clause interconnectivity graphs in theorem proving, IEEE Transactions on Computers, C-25, 823-834.

Wos, L., Robinson, G.A., and Carson, D.F. (1965): Efficiency and completeness of the set of support strategy in theorem proving, JACM 12, 4, 536-541.

SOLUTION TO MANY CHESS PAWN ENDGAMES'

Kenneth W. Church
NE43-308
MIT
Cambridge, MA. 02139

Abstract: The solution to a number of Artificial Intelligence problems depends upon Knowledge representation. The theory of *Co-ordinite Squires* is a frameworK for representing the relevant ideas in a large class of King and pawn endings. Some endgame problems which require twenty to thirty plies of analysis using a more traditional representation, a branching tree of moves, can be efficiently solved using this theory, which is analogous to Waltz* scene analysis algorithm.

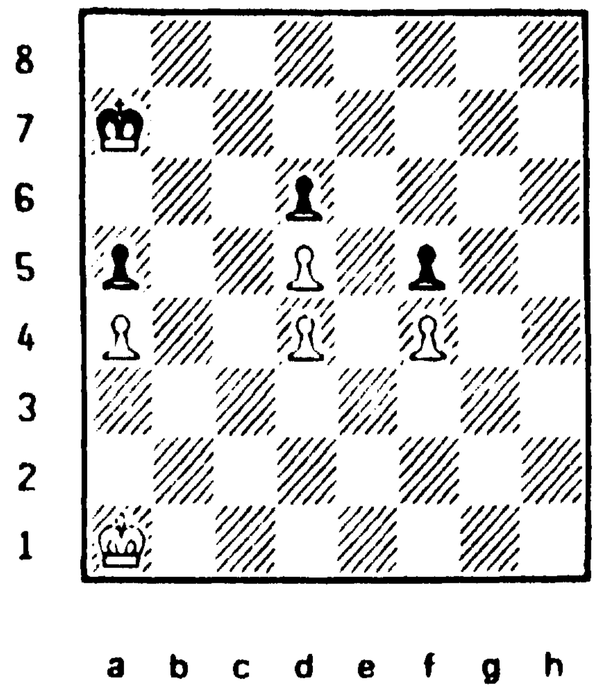
The Co-ordinate Squares theory, which has received considerable attention in the chess literature over the last half century, is Implemented using a constraint propagation technique. The constraint is that the defending King must precisely co-ordinate with the attacKing King to avoid losing. For any particular square the attacKing King occupies, there are only a few possible (co-ordinating) squares the defending King could occupy without transposing into a Known lost position. As the attacking king moves from one square to another, the defending king must co-ordinate, move from a co-ordinate of the one square to a co-ordinate of the other. We will give a reduction algorithm which propagates this constraint in order to determine the co-ordinate squares.

Introduction

We will begin by defining a simplified version of King and pawn endgames, which we will call COORD. Although this game can be solved by traditional methods, it is a good example for demonstrating co-ordinate squares. Then we will explore some aspects of King and pawn endgames which are not included in COORD.

COORD is played on a chess board with Kings and pawns which move in the usual way. In the initial position, the pawns are all locked against each other so there are no pawn moves. White wins if he captures a black pawn. Black draws if white never wins. The task is to find the next move and prove that it is optimal. Consider the following COORD problem which will be used throughout this paper.'*

Figure 1



white to move wins; black to move draws

This famous problem has a rich history in the chess literature ([Fine, problem *70], [AverbaKh, problem *671], and *many* others). Computer chess researchers have also studied this problem. Newborn, the author of Peasant*'*, estimates that this problem would require 25,000 hours of cpu time [Newborn, page 129} Chess 4.5 has more recently found the solution with a 26 ply search [Frey2}. Using co-ordinate squares, we can find the solution in about thirty seconds on a Lisp Machine [Weinreb).****

* This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

** All chess moves and squares within this paper are given in *Algebraic Notation,* the most common notation in the world. Each square is referenced with a letter, *a* through *h* and a number, *1* through *8* as shown below figure 1. In this notation, king moves and captures are denoted by a capital *K* followed by the destination square. Pawn moves *are* denoted by the destination square. A square followed by an *x* followed by another square denotes a pawn capture.

*** Peasant, one of the few chess program designed to play only king and pawn endgames, uses a depth first alpha-beta branching tree strategy.

***t More careful analysis of the program performance can be found in [Church]. The co-ordinate squares approach is solution is at least as good as Peasant's on 16 problems from [Fine). The time cost is shown to be bounded by a constant for king and pawn endgames where pawn moves are irrelevant. Algorithmic analysis of the pawn moving heuristics have not been performed. In practice, the time costs are well within normal chess tournament constraints.

Most chess programs, including Peasant and Chess 4.5, use a depth first search (DFS) which can be hard to contain. DFS might, for example, explore a queen endgame that could follow from a long series of blunders.

Another problem with a DFS is that there are often a large number of paths between any two positions. The DFS algorithm has recently used a transposition table to catch positions that have been previously explored. For solving COORD problems, a transposition table is more expensive than necessary.

A DFS transforms the search space into a tree. A better representation is probably a graph of positions connected by full moves (a half move for each side). In COORD there are only 4096 positions since the pawns are fixed and there *are* only 64 ways to place each King. Furthermore, the maximum out-degree is 64 since each king can make at most 8 legal moves. This very regular graph is probably more efficient than a more general transposition table.

Given any position, it is theoretically possible to determine who will win. This determination can be made independently of the context, the moves that have been played previously. A position has no memory. A DFS maintains this context on the stack. Perhaps this is a mistake.

To conclude this section, we have motivated that the traditional method, a DFS, may not be optimal for solving COORD problems. In the next section, we will consider an alternative approach which does not suffer from the drawbacks stated above.

An Alternative to DFS

What does determine the value of a position, whether or not white will win? A winning position either is *obviously* winning (inductive basis) or can be forced into another winning position in'exactly one move (inductive step). The decisions are local; they could be performed by a parallel machine consisting of 4096 gates comparing each of the 4096 positions with its neighbors. The locality is important because it allows us to resolve the positions in any order without changing the results.

To determine whether a position is obviously winning requires chess domain heuristics which will be discussed later. We will assume for now that an obviously winning position is one where the white king has *captured* a pawn. Although these domain heuristics strongly influence the results, this paper is more concerned with the inductive reasoning. Demonstrating that this induction is correct requires showing that any unresolved position which leads to a known winning position in exactly one move is itself winning (easy) and that any unresolved position, which doesn't lead to a resolvable winning position in exactly one move, isn't winning (more difficult). Due to space considerations, the demonstration will not be given here.

Even though we may be able to compute the value of every position, we still don't know which move is best. However, we do know that the best move will preserve the value. This provides a very strong constraint for selecting the best move. We may still need a DFS to choose the best move, but the DFS is over a much smaller search space. The implemented program calculates these constraints, without performing the final DFS to choose a move.

We have motivated an alternative to the traditional DFS. Before describing the algorithm more specifically, we would like to show how this relates to the chess notion of *co-ordinite squires.*

Co-ordinate Squares

We have defined drawing positions to be those positions from which white cannot force a win. In other words, no matter what white does from a drawing position, black has a adequate reply. The black king is *co-ordinating* with the white one. White can more this way or that, left or right, but he cannot out maneuver the black king. The chess technique of co-ordinate squares as advocated by Averbakh and Maizelis [Averbakh] locates many of the co-ordinating squares, using some unspecified procedure, in order to simplify the search process. In this work, we have refined the procedure to an algorithmic process.

The Reduction Algorithm

If black is co-ordinating at one time and a move later he is not, then he must have made a mistake. Similarly, if white is winning and a move later, black is back defending successfully, then white must have missed an opportunity. Co-ordination is invariant. The end result cannot change if both sides play perfectly. Consequently, to decide if white is going to win, all we have to do is decide if he is winning in the initial position.

To decide the value of the initial position, we have to decide the value of the neighboring positions. Chances are that this procedure will request the value of every position. So far this is very similar to a DFS. The difference is that this procedure does not specify the order in which the positions *are* considered. How can we efficiently classify the 4096 positions into winning and losing? We will use a reduction algorithm. Initially, classify the 4096 positions into one of two categories: either obviously winning for white or unknown. Each snapshot depends upon its neighbors. That is, if an obviously winning position can be forced from an unresolved position, then it too is obviously winning for the attacker (white). The algorithm keeps checking positions with their neighbors until no more unresolved positions can be classified winning. At that point, the unresolved positions are classified co-ordinating because the attacker cannot force a win from them.

The Triangulation Criterion

We use the chess term triangulation loosely for the comparison of a position with its neighbors. This step accounts for arbitrarily complex king maneuvers, some of which chess players call triangulation, opposition, distant opposition, co-ordination, tempo-ing, or zugzwang*. The constraint is that the defender must have a reply to any move by the attacker. Assume that the attacker is on a_{attack} and the defender is on a_{defend} IF the attacker can move to b_{attack}, then the defender

must be able to reply by moving to a square b_{defend} which defends b_{attack}. If the defender doesn't have such a reply, then his defense on a_{defend} has been refuted.

Since all defenses of *an* attacking square (placement of the attacking king) must meet this constraint, we can optimize the comparison procedure with the following reformulation. If the attacker can move from one square to another then all defenses of the former must be a move from a defense of the latter. This formulation of triangulation motivates the notion of a defense set (dset), the set of squares which defend against an attacking square. A implementation is given below using the primitives `fetch-dset` (to retrieve the dset of an attacking square) and remove (to remove a defense from a dset). The predicate `can-move?` decides if a player (either white or `black`) can move from square a to square b when the other king is on square `oking`.

```
can-move? (player, a, b, oking) =
  adjacent (a, b) & legal (player, b, oking)

triangulate:proc (attacker, a_attack, b_attack)
 A := fetch-dset (attacker, a_attack)
 B := fetch-dset (attacker, b_attack)
 defender := other-side (attacker)
 for all a_defend in A do
  if ~∃b_defend∈B
    [can-move? (attacker, a_attack, b_attack, a_defend) →
     can-move? (defender, a_defend, b_defend, b_attack)]
    then remove (a_defend, A)
```

♦ The following are common chess terms.

`triangulation` - Taking a longer than necessary route toward a target so that the position will recur with other side on the move.

`opposition` - The state where the two kings are on the same rank or file and there is exactly one square in between.

`diagonal opposition` - The state where the two kings are on the same diagonal and there is exactly one square in between.

`distant opposition` - The state where the difference in ranks of the two kings and the difference in files are both even.

`co-ordination` - (relatively new term) The state where there exists a defensive reply to wherever the attacking king moves.

`tempo-ing` - A catch-all term to cover arbitrarily complex attempts to repeat the position with the other side on the move. Although triangulation is the most common method, there are other possibilities such as pawn maneuvering.

`zugzwang` - A state where the side to move has only bad moves and would be well off if only he didn't have to move.

Figure 2

The black kings mark the possible defenses (before triangulation); everything else obviously loses since the white king is closer to a black pawn. (This computation is described later in more detail.) Since the white king on c4 (left) is threatening to move to bS (right), all defenses of c4 ({a7, b7, c7, a6, b6, a3}) must either prevent the threat or respond to it. a6 and b6 prevent the white king from moving to bS. None of the other defenses work because black cannot respond to the threat by moving from one of them to a defense of bS. At this point A - {a6, b6}.

These reductions tend to propagate. Now that a number of defenses of c4 have been eliminated, it is possible that one of c4's neighbors might require further reduction. This process is defined to terminate when no more reductions are possible. The propagation effect explains why a6 is eventually removed from the defense set of c4. If black were on a6, white could beat black to the f5 pawn. 1. Kd3, Kb6; 2. Ke3, Kc7; 3. K13, Kd7; 4. Kg3, Ke7; 5. Kh4, Kf6; 6. Kh5, Kf7; 7. Kg5, etc. The algorithm determines this by triangulating the dset of g5 with that of h5 and then the dset of h5 with that of f6 and so on. The propagation continues until the map has settled.

A Solution to Figure 1

Since the order of triangulating will not change the final results as long as we keep triangulating until the propagations have settled, we will adopt the following non-optimal implementation for expository purposes only:

Initialization: assign the first guess dsets (using chess domain heuristics):
Do until no dset changes
triangulate each dset with all its neighbors

The first guess dsets are shown below. For the sake of brevity, we will show only the more interesting sets.

b5 {}
c4 {a6, b6, a7, b7, c7} (ignoring defenses like a3)
b3 {a6, b6, a7, b7, c7, d7, a8, b8, c8, d8}
c3 {a6, b6, a7, b7, c7, d7, a8, b8, c8, d8}
d3 {a6, b6, a7, b7, c7, d7, a8, b8, c8, d8}
e3 {b6, b7, c7, d7, e7, b8, c8, d8, e8}
f3 {f6, c7, d7, e7, f7, c8, d8, e8, f8}
g3 {f6, g6, d7, e7, f7, g7, d8, e8, f8, g8}
h4 {f6, g6, h6, e7, f7, g7, h7, e8, f8, g8, h8}
g5 {}
h5 {f6, e7, f7, g7, h7}

h6 {f6, e7, f7, g7, h7}
b2 {a6, b6, a7, b7, c7, d7, e7, a8, b8, c8, d8, e8}
c2 {a6, b6, a7, b7, c7, d7, e7, a8, b8, c8, d8, e8}
d2 {a6, b6, a7, b7, c7, d7, e7, a8, b8, c8, d8, e8}
a1 {a6, b6, a7, b7, c7, d7, e7, f7, a8, b8, c8, d8, e8, f8}
b1 {a6, b6, a7, b7, c7, d7, e7, f7, a8, b8, c8, d8, e8, f8}
c1 {a6, b6, a7, b7, c7, d7, e7, f7, a8, b8, c8, d8, e8, f8}
d1 {a6, b6, a7, b7, c7, d7, e7, f7, a8, b8, c8, d8, e8, f8}

Triangulate the above sets with their neighbors.

Iteration 1
c4 {a6, b6} (constrained by the dset of b5)
h4 {f6, g6, h6} (constrained by the dset of g5)
h5 {f6} (constrained by the dset of g5)
h6 {f6} (constrained by the dset of g5)
all others unchanged

On the next iteration, by triangulating all the dsets again, the constraints propagate another square away from the targets.

Iteration 2
b3 {a6, b6, a7, b7, c7}
c3 {a6, b6, a7, b7, c7}
d3 {a6, b6, a7, b7, c7}
h4 {g6, h6}
h5 {}
h6 {}
g3 {g6, h6, e7, f7, g7, h7}

This process continues until no more defenses can be removed. These iterations have been deleted for brevity.

Summary
b5 {}
c4 {b6}
d3 {c7}
c3 {b7}
b3 {a7, c7}
d2 {c8}
c2 {b8}
b2 {a8, c8}
d1 {c7}
c1 {b7}
b1 {a7, c7}
a3 {b8, b7, d7, d8}
a2 {b8, b7, d7, d8}
a1 {b8, b7, d7, d8}

White wins if he has the move because because black's King on a7 does not defend a1. Me must play moves which prevent black from co-ordinating. One winning line is: 1. Kb1, Kb7; 2. Kc1, Kc7; 3. Kd1, Kd7; 4. Kc2, Kc7; 5. Kd3 and black has to move. If he plays Kb8, Kc8, Kd8, or Kd7 white wins the a5 pawn with Kc4. On any other black move, white wins the f5 pawn with Ke3. The algorithm does not tell us how to win; it only decides which moves preserve the win. In this case, white has only one move (1. Kb1) which preserves the win. Black can reply to 1. Ka2 or 1. Kb2 by moving to a co-ordinate, {b7, b8) and (a8), respectively. From these co-ordinating positions, no matter where white moves, black will be able to move to another co-ordinating position.

In the same position with black to move, black can co-ordinate by moving to either b7 or b8, since both defend a1. Neither Fine nor Averbakh suggest Kb8 because that defense is somewhat "anti-theoretical" probably because it gives away a twisted distant opposition*. Kb8 may look unusual because it unnecessarily moves away from the action.

The Order of Triangulating Dsete

Even though the order will not change the final results, it can influence the running time. For best efficiency, the most constrained sets should be compared with their neighbors first because they are most likely to shrink their neighbors. We can assure that the most constrained sets will be considered first if we keep a sorted queue" of sets to be considered. Initially all the sets are on the queue. In each iteration, pull a set off the queue and triangulate it with its neighbors. If triangulation should remove any elements from a neighboring set, then check for the propagations by re-queuing those neighboring sets.

Mow do we decide if one set is more constrained than another? One answer is to employ some domain restricted knowledge. For example, we might use a heuristic which suggests that attacking squares closer to pawns tend to have more constrained defense sets. However, in this work, we have chosen a much simpler and domain independent ordering heuristic. The smaller dsets are more constrained than the larger ones. This completes the description of triangulation.

The Triangulation Criterion Viewed as a Waltz Filter

It is useful to compare this approach to previous work in other areas of Al In this section, we will show that this reduction technique is very similar to Waltz' filter. Waltz [Winston, chapter 1] developed a filtering process to search a line drawing scene. The problem is to find all possible ways to label the lines (- for concave, ♦ for convex, and either <-- or --> for a boundary) in a scene so that lines meet in physically realizable vertices. Waltz's algorithm first assigns each vertex a set of possible labels, i.e. physically realizable. The next and final step is the filtering process which assures that connected vertices are labeled consistently, that two vertices connected by a line are labeled the same way at both ends. The procedure is to iterate through the vertices checking each possible label for consistency with each connected neighbor. If any possible label is found to be inconsistent with a neighbor, that label is removed from the possible labels. It is possible that the effects of removing a label might propagate to other vertices. After *every* removal, the possible propagations are immediately considered.

t Opposition can be twisted by certain pawn formations. Since the opposition is usually an asset, it is understandable why both Fine and Averbakh might overlook the fact that the opposition is not necessary for black to draw.

tt See [Meckworth).

The triangulation procedure is analogous to Waltz's filter. The vertices correspond to the possible locations of the attacking king. A defense set can be thought of as a set of physically realizable ways to label a vertex. Two attacking squares *are* connected iff the attacker can move from one attacking square to the other. Just as two vertices connected by a line must be labeled consistently, the defense sets of two attacking squares must be consistent. That is, the defender must be able to move from any square in one defense set to a square in the other defense set.

This analogy shows us that we can apply the triangulation constraint in time proportional to the number of vertices (~64) times the number of possible labels (~64). Using the queue mechanism discussed previously, it is possible to improve the average cost. In Waltz's filter, since the propagation possibility is checked immediately after a set is reduced, it is unlikely that they will be checked in the best order. (The order wasn't as important to Waltz because the line drawing world has only 4 possible labels whereas the chess endgame world has 64.) Another advantage of the queue is that it can avoid considering some vertices multiple times. The Waltz filter will check a set for propagations that it will have to check later anyways. The queue mechanism avoids this by preventing • set from appearing twice on the queue. It is very difficult, though, to see exactly how good the queue is.

Finding Possible Defenses

In this section we will discuss the chess domain heuristics involved in deciding the "obvious" cases. These are only heuristics; although the more complete heuristics discussed in [Church] have no known limitations for COORD examples, they do not solve all king and pawn endgames. That work illustrates the limitations in more detail.

One criterion which will work for COORD, but will be difficult to extend to practical chess endgames, is the following: *Target positions (the white king occupies the same squire as a black fawn did) are obviously winning. All others might defend.* In this subsection, we will define a more useful criterion.

For now we will assume that the defender will lose unless his king is as close to every target, black pawn, as the attacking king. This definition uses the notion of distance. White's distance between two points is the number of white king moves it would take to travel the distance. Black's distance measures black king moves. The two are not the same because a king cannot move on top of a friendly pawn or into an enemy pawn's zone of control. There is an algorithm for computing the distance from one point to all others.

Closeness

A player's king (on square a) is closer to the target t than the other player's king (on d) if his distance to the target is less than the other player's.

```
closer (player, a, d, t) =
    distance (player, a, t) <
    distance (other-side (player), d, t)
```

Defense Set

A square d possibly defends an attacking square a if it is legal for the the defender and close enough to the targets.

```
dset (attacker, a) =
    {d | legal (other-side (attacker), d, a) &
        ∀t ∈ targets ~ closer (attacker, a, d, t)}
```

Getting back to Figure 1, let us find the defense set of c4 where white is the attacker. c4 is two moves from target a5, three squares from d6 (c5 is illegal for white), *and* five from f5. According to this first criterion, black can defend c4 from a7, b7, c7, a6, b6, or a3. (See Figure 2 shown above.) In other words, if white's king is on c4, then black's pawns are safe only if the black king is on one of the above defenses a7, b7, c7, a6, b6, or a3. We have seen that triangulation wilt remove many of these defenses.

Extended COORD

As we have said, COORD is simpler than chess king and pawn endgames. In this section, we will explore methods to extend co-ordinate squares. One problem with COORD is that black cannot win. One solution is to do the calculation twice, once where white is the attacker (as in COORD) and once where black is the attacker. The game is drawn if both sides are co-ordinating with each other. It should be impossible for neither side to be co-ordinating with the other. If only one side is co-ordinating with the other, then he is winning. There are probably more attractive solutions.

There *are* quite a few other differences between COORD *and* chess king and pawn endgames which must be considered. We have two basic strategies to extend co-ordinate squares. The first, which is untested, increases the search space. Co-ordinate squares is *very* good in manipulating the kings in one pawn configuration. However, there are ways to jump into another pawn configuration, by either capturing or moving a pawn. Once a player jumps into another pawn configuration (hyper-space), he can *never* return because there is no way to move a pawn backwards or to bring back a captured pawn from the dead. The graph of hyper-spaces is therefore acyclic. We could solve each hyper-space one at a time, starting with the leaves until we have solved the root space (the initial position.) The problem with this idea is that the number of hyper-spaces can be very large, one for each possible pawn configuration. Five pawns on the second rank would generate 5^7 spaces. There must be heuristics to reduce the number of hyper-spaces.

The other approach augments the existing search with the necessary chess heuristics. We [Church] have solved a large class of king and pawn endgames with this approach although there are problematic endgames for which the heuristics do not apply. The optimal solution is probably a compromise between the two approaches. It would be worthwhile to experiment with the hyper-space approach.

Conclusion

In conclusion, although this work has shown that co-ordinate squares can solve several endgames otherwise requiring 20-30* plies of analysis, the question remains, what are the limits to the approach? At first, the algorithm appears to be exponential with the number of potential pawn moves. That is, one way to solve problems with pawn moves is to map the co-ordinate squares for any pawn structure that could result. However, if we think about what the pawn moves mean, then we can do considerably better. For example, a tempo, having one more pawn move than the opponent, is an option to pass. We can show how to incrementally modify the co-ordinate squares map to account for the tempo [Church]. Similarly, the effects of a passed pawn can be considered without building separate co-ordinate squares maps for each square the passed pawn could occupy before it queens. It should be possible to state exactly how pawn moves and king moves are coupled.

At first it was believed that the strategy would break down when more pieces *are* introduced. Even that is not so clear; pieces, especially slow ones such as knights, in many cases must avoid zugzwang by co-ordinating with each other. In fact, Averbakh has written another book about knights [Averbakh, *Knight Endgames].*

The key observation that the strategy depends upon is that all these positions have a very limited number of terminal nodes, distinct positions that could result after an arbitrary number of moves. Since there *are* so few terminal nodes, It is possible to apply a static evaluation function to each one *and* then observe how the nodes are connected (triangulation).

Using the queue improvement to Waltz's algorithm, co-ordinate squares starts with the most constrained nodes and works backwards toward the least constrained. It happens that the constraints tend to be centered *near* the targets, and that the initial placements of the kings tend to be relatively far from the targets, in the more difficult endgame problems such as Fine *70 (Figure 1). Working backward from the target positions to the initial positions, co-ordinate squares is somewhat similar to backward reasoning, just as a depth first tree search from the initial position toward the target positions is forward reasoning. The usefulness of this analogy is in the observation that it is often possible to halve the exponential growth of a search by building backwards from the terminal nodes and forward from the initial position simultaneously. If this could be applied to co-ordinate squares as described above, then the defense sets would not contain a number of unobtainable defenses. [Church] describes some heuristics for quickly removing such useless defenses.

We have designed a system that thinks about the chess domain more as a regular graph of possible positions than as a tree. Consequently, when the number of positions is small, as it tends to be in king and pawn endgames, the procedure works surprisingly well. In general, as the number of interesting positions grows, the space tends to look more and more like a tree. In a general chess position, co-ordinate squares is not the optimal representation. Within our restricted endgame domain, which has long been a weak point of chess machines, the co-ordinate squares procedure shows great promise. As e generalization of Waltz's filter, this work may have tome implications in other domains besides chess endgames.

References

Averbakh, Y. and Maizelis, I., *Pawn Endings,* Chess Digest, Inc., 1974 (English Translation).

Averbakh, Y. *and* Chekhover, V., *Knight Endings,* B. T. Bettford Ltd., 1977 (English Translation).

Church, Kenneth W., *Co-ordinate Squares:* a *Solution to* /Many *Chess Pawn Endgames,* Bachelor's Thesis at MIT, 1978.

Fine, Reuben, *Bisic Chess Endings,* David McKay Company, Inc., 1941.

Frey, Peter W. (editor), *Chess Skill in Man and Machine,* Springer-Verlag, 1977.

Frey, Peter W., personal communications, 1978.

Mackworth, Alan K., *Consistency in Networks of Relations,* AI Journal, February 1977.

Newborn, Monroe, *PEASANT: An endgame program for kings and pawns,* chapter 5 of *Chess Skill in Man and Machine.*

Perdue, Crispin and Berliner, Hans J., *EG — A Case Study in Problem Solving with King and Pawn Endings,* IJCAI, 1977.

Tan, S. T., *Representation of Knowledge for Very Simple Pawn Endings in Chess,* University of Edinburgh Memo M1P-R-98, 1972.

Weinreb, D. *and* Moon, D, *Lisp Machine Manual,* MIT, 1978.

Winston, Patrick Henry (editor), *The Psychology of Computer Vision,* McGraw-Hill Book Company, 1975.

Acknowledgments

Tim Anderson, Kurt Vanlehn, and my especially my father, Russell Church, were among a large number of very supportive readers. Hans Berliner has provided much encouragement. And, of course, I mustn't overlook Richard Greenblatt, my thesis adviser and friend.

DIALOGUE MANAGEMENT FOR RULEBASED TUTORIALS

William J Clancey
Computer Science Department
Stanford University; Stanford, CA 94305

Mixed-initiative discussion requires that a tutorial program have the means to manage its share of the dialogue. Dialogue management includes operations ranging from record keeping and context focusing to heuristics for directing the dialogue economically according to the needs of the student. This paper illustrates how knowledge for generating remarks in a tutorial dialogue about rule-based domain knowledge can be represented conceptually as a network consisting of transitions between dialogue situations in which the links are various kinds of management heuristics. The chief finding is that a network of procedures is a useful representation for organiiing heuristics for carrying on a structured dialogue.

1 Introduction

This paper discusses a kind of mixed-Initiative tutorial dialogue that concerns a single, complex task to be solved by a student under a program's guidance Sequences of student/tutor remarks can be grouped into "discourse situations" or recurrent patterns In the discussion, making the mixed-initiative nature of this tutorial more complex than in previous work Typical discourse situations are: examining the student's understanding after he asks a question that shows unexpected expertise, relating an inference to one just discussed, giving advice to the student after he makes an hypothesis about a subproblem, and so on

The general problem of sharing initiative and making provisions to carry out one's discourse goals is here termed 'dialogue management" We illustrate principles for taking initiative that are desirable for tutoring rule-based diagnostic problems An important consideration is the possibility of 'alternative dialogues" We focus on the problem of making appropriate, prolonged presentations that go beyond interruption and repetitive question/answering, but use them as components in a larger scheme

Most research with Intelligent Computer Aided Instruction (1CA1) programs has emphasized the use of an underlying expert artificial Intelligence program for generating subject material to present to a student and for modelling his understanding. Indeed, researchers call this work "generative" CAI to underscore the constructive operation of creating questions or tasks for a student and creating a differential model that relates the student's behavior to the program's model of expertise [18] While one distinguishing feature of these programs is frequently cited to be their capability to engage the student in a mixed-initiative dialogue [I6J [17], we find that the "natural language understanding" In these programs Involves parsing student Input [5], not conversational interaction.

In the "reactive environment" of the early SOPHIE lab [2], the tutor never takes the Initiative at all; and in the games of WEST [6] and WUMPUS [13], the tutor's remarks are all Interruptions or reactions to the immediately preceding move taken by the student SCHOLAR [10] and WHY [22], the geography and meteorology tutors, follow a Socratic dialogue format of repetitive questioning using topic selection rules and strategies for testing a student's understanding; the session itself Involves no overarching problem or shared task to be discussed Thus, ICAI research has emphasized the use of domain expertise for modelling the student and tutoring principles for correcting his misconceptions, but it has dealt only tangentially with the problems of carrying on a prolonged, *purposeful* tutorial dialogue that as a whole considers a single task.

The discourse problems dealt with here include maintaining and sharing context as solution of the task proceeds, and providing means for the student to express initiative as he unfolds the complexity of the problem and encounters limitations in his understanding. Researchers who specialize In Natural Language studies have reported various theoretical aspects of task-oriented dialogues, such as recognizing and generating Intentions and plans [12] [7] and focusing on objects and subtasks [II] [15]. We distinguish our work from other Natural Language research in two ways. First, we view the discourse problem as requiring (a) Interpretation of intent and (b) reasoning about this understanding in order to act. We have minimized the Interpretation problem by restricting the student to a command-oriented input, allowing us to concentrate on the tutorial knowledge for generating remarks and guiding the dialogue Second, we are formalizing tutorial knowledge In terms of specific performance rules for various situations, rather than studying human dialogues and enumerating general dialogue Interaction principles.

In this paper, after setting the scene by a short overview of rule-based tutoring, as exemplified by a particular system, GUIDON, we present a transition diagram that we find useful for illustrating dialogue management issues Examples of the system in operation are given The second half of the paper deals with the problem of topic relationships, for example, how the dynamics of the Interaction complicate separation of discourse knowledge Into modular procedures. A final section discusses potential applicability of GUIDON to other representations and problems

2 Rule-based Tutoring: GUIDON

GUIDON [9] *is* a tutorial system that can be built on top of any MYCIN-like expert system [20], The knowledge base of a MYCIN-like system is in the form of situation-action rules, such as, "If the Infection that requires therapy Is meningitis and the patient has symptoms of mumps, then this Is weakly suggestive evidence that the type of the infection is viral' In this paper, all examples will be drawn from the MYCIN infectious disease consultation program, though the

* This research was sponsored in part by grants from ARPA (Contract Title MDA 9O3-77-C-0322) and NSF (MCS 77-02712)

GUIDON system Is capable of tutoring any knowledge base of this form.

The purpose of a MYCIN consultation is to determine the organisms that might be causing an infectious disease and to prescribe antibiotic therapy for them. The MYCIN interpreter invokes Its rules in a goal-directed way, chaining backwards to evaluate the predicate of a rule The program requests information (case data") from the user as it becomes appropriate in order to apply a rule The result of a consultation Is a record of rules that were applied, acquired case data, and conclusions inferred by the application of rules This record is configured into an explicit AND/OR tree for use In tutorials AND nodes are rules and OR nodes are the goals hanging below them.

In a GUIDON tutorial a student plays the role of a consultant In a diagnostic game The dialogue deals exclusively with a particular consultation that has previously been stored in the MYCIN system. The student is presented this same case That Is, he is given some information about a patient suspected to have an Infectious disease, and Is expected to request case data, as he sees necessary, to draw conclusions about the cause of the infection. The topics of this dialogue are precisely the goals that are determined by applying MYCIN rules, e.g., the type of the infection As the dialogue progresses. GUIDON is able to collect Information about the student's knowledge of the case and Its domain. The purpose of a GUIDON tutorial is to make the student aware of any gaps or inconsistencies in his knowledge with respect to the underlying rule base, and to correct these deficiencies Note that the course of the dialogue need not pursue the exact sequence of rule invocation and question-asking that MYCIN used in this case, and in general it does not.

The GUIDON system has already been described in some detail In *[9]* in which a framework was established for the development of the program. Briefly, the framework Incorporates: I) discourse knowledge in the form of sequences of domain-independent tutoring rules (hereafter, "t-rules") organiied Into stylized discourse procedures (patterns for tutorial Initiative); 2) domain knowledge In the form of the MYCIN-like rule base and records from the consultation to be discussed with the student; and 3) a communication record for recording dialogue goals and the dynamic state of the tutorial In this paper we will show how this framework has been Instantiated. The dialogue transition diagram presented in Section 3 represents the calling structure of the discourse procedures (comprising 200 t-rules) currently implemented in the program. Section 4 gives several examples of the topic relationship problems In dialogue management.

GUIDON is an example of an ICAI tutorial system. The central feature of such a system is the presence of an underlying "expert program" that can solve the problems that are posed to the student. A "differential student model" (6) is formed by comparing the student's problem-solving actions to those of the expert program. From this comparison of behavior the tutorial program infers whether the student knows about and uses the problem-solving knowledge incorporated in the expert program's actions. Thus, the model of the student's knowledge is an "overlay" [8] on the expert knowledge base. In a rule-based system like GUIDON, this consists of a marked subset of expert rules that the student is thought to know.

Tutoring consists of posing problems and/or making remarks that are Intended to *bring* the student's knowledge into alignment with the expert system; thus apparently missing rules are presented and misconceptions are corrected One of the major research problems Is the design of tutorial remarks: when to say something and what to say [4] GUIDON'S discourse procedures are designed to deal with this problem

Dialogue management Involves coordinating tutorial goals with the constraints Imposed by; I) time available for the session, 2) student initiative and conversational (social) postulates [14], 3) the communication channel, and 4) human memory and learning capability In early development of GUIDON, the limitation of time and verbosity have been the chief forcing functions: It is not possible to explicitly discuss every Inference that the expert program attempts By using heuristics developed for economical presentation and the student initiative options presented below, the crux of dialogue management in GUIDON, session time for relatively simple cases has been reduced from about five hours to less than one (for students who know how to use the program).

The main Issues of dialogue management discussed below are.

— On what basis does a tutor select alternate presentation techniques?

— How does a tutorial program maintain and share dialogue context?

— How can we provide for and cope with student Initiative?

— How can we ensure dialogue connectedness and comprehensIbIIIty?

3 The Dialogue Transition Diagram

In this section we present the concept of alternative dialogues, different ways something can be said and possible subdtalogues that can occur. The concept is illustrated by a state transition diagram that represents the invocation structure of the discourse procedures In GUIDON. The links represent control expressed by tutorial rules within the procedures. These links signify choice points that lead to alternative dialogues, dictated by domain logic, economy, or tutorial considerations Thus, these represent the management decisions In which the tutor takes the initiative to control dialogue situations

Fig. I illustrates a portion of a dialogue with GUIDON. After a case was chosen (the current system requires that the user select a case from the on-line library), the student was given Initial Information, such as the fact that there Is a pending culture in the laboratory and no organisms have been reported The discussion has reached the point of considering clinical Information (non-laboratory data) to determine the organisms that might be causing the Infection

The student asked whether the patient has a rash, and the tutor gave the reported case data (lines 9-11) (Student input is In the form of keywords or simple sentences) At this point several remarks were made by the tutor: I) lines 13-24, the datum requested by the student was related to the current topic by reciting the path that connects them in this case rash -> Herpes Zoster virus -> infection type -> organisms; 2) lines 26-28, the tutor focussed on the topic "Herpes Zoster virus." and stated a final conclusion; 3) lines 30-31, the tutor explicitly returned the student's attention back to the original topic of determining the organisms causing the infection.

5 You should now proceed to ask questions that will enable you to make an hypothesis about the organisms tnat might be causing the infection.

** RASHES

10

Pt538 does not have a rash.

Your question is indirectly relevant:

15 Whether Pt538 has a rash can be used to determine whether Pt538 has symptoms of Herpes Zoster virus...
which can be used to determine the type of
20 the Infection...

and this will enable us to determine the organisms that might be causing the infection.

25

The fact that Pt538 does not have a rash is evidence that Pt538 does not have symptoms of Herpes Zoster virus (RULE3691

30 Back to our discussion of the organisms that might be causing the infection...

Fig I Relating a factor to the current topic
("Pt 538" replaces the name of the patient.)

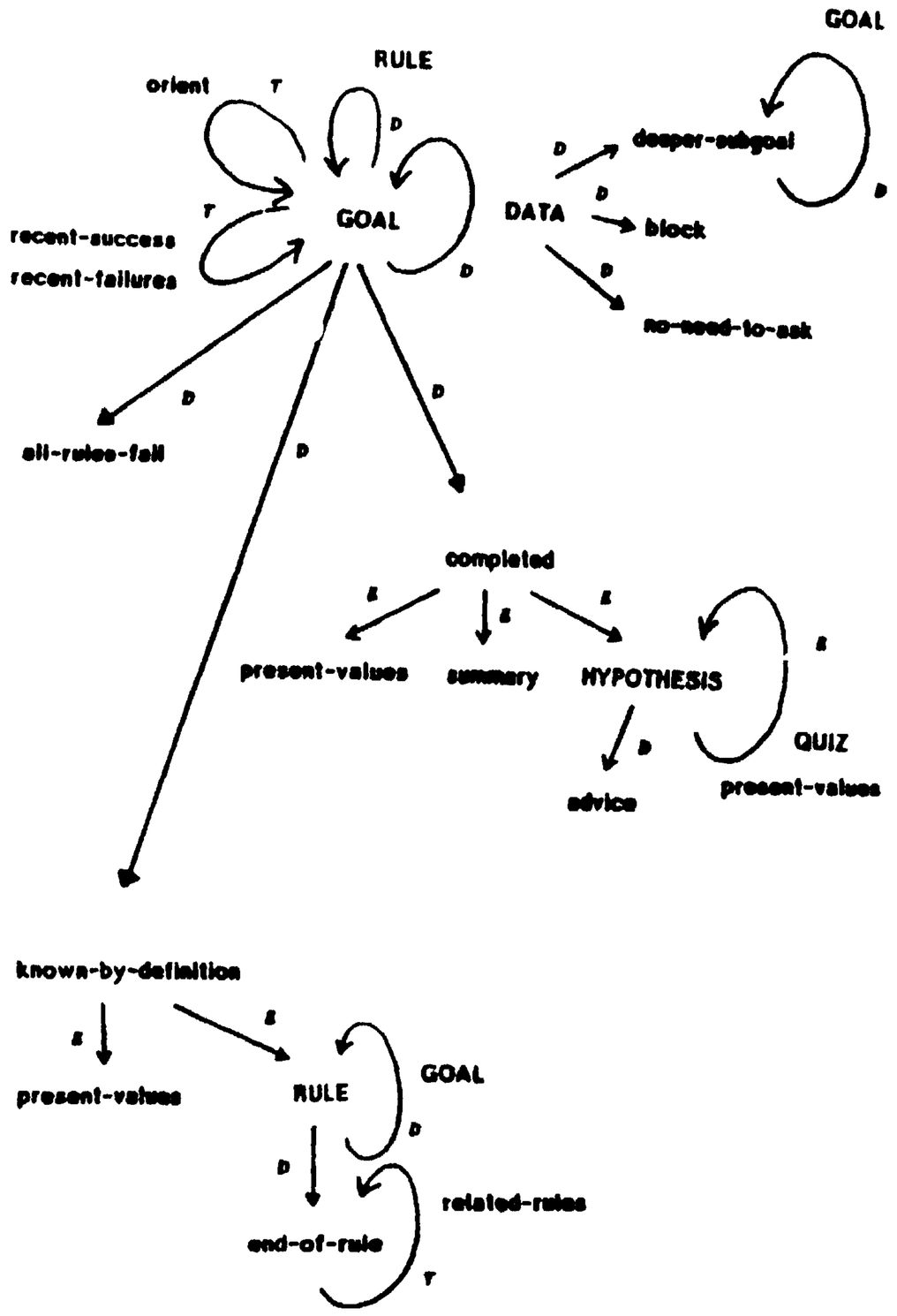


Fig 2 Dialogue Transition Diagram

It now may be helpful to look at the dialogue transition diagram (Fig 2). Each node In this diagram stands for a discourse procedure, or sequence of t-rules. For example, when lines 5-9 were printed In Fig. I, the procedure GOAL was being followed, the basic procedure for discussing any MYCIN goal which appears In a domain rule. To answer the question about rashes, the DATA procedure was applied. Whether or not the patient has a rash is a "deeper subgoal" In that it does not appear in any rule that can determine the current goal, but appears deeper in the AND/OR tree of goals and rules. The DATA procedure printed line 11 and set up the second-deepest subgoal as the new current topic (whether the patient has symptoms of Herpes Zoster virus). An arrow that loops back to a node signifies that the procedure that labels it is called one or more times, as one might expect. For example, we see that one or more "related rules are presented at the end of discussion of a given rule. The Italicized labels stand for the basis of the transitions—economy, domain logic, and tutoring goals—these distinctions are described below.

Returning to our example, the tutor observed that the new topic (Herpes Zoster—see lines 15-17 in Fig. I) was "completed" because all of the case data that the expert needed to make a final conclusion had already been given to the student Here a choice had to be made. Should the tutor present the final conclusion (as It did in lines 26-28)? Or should a summary of evidence be offered? Or should the tutor ask the student to make an hypothesis (as to whether or not the patient has symptoms of Herpes Zoster)? Fig 3 and Fig. 4 show the procedure COMPLETEDCOAL. We see that T-RULE5.02 was applicable: a single domain rule was used by the expert program to make a conclusion and the student model indicated that the student knew this rule, so it was simply stated The fact that the dialogue could have taken a different form at this point, at the tutor's Initiative, illustrates the possibility of alternative dialogues. There are many more alternative dialogues than those Illustrated by the different paths in Fig. 2. For example, the procedure for discussing a rule (shown here as RULE) incorporates 18 different methods, e.g., clause-by-clause discussion, supplying case data and then asking for an hypothesis, and discussing a failed subgoal and then mentioning the conclusion that can't be made.

COMPLETEDGOAL.PROC005

Purpose: Discuss final conclusion for a goal

Step I: <Decide whether to finish with a summary>

Step2: <Discuss the final hypothesis for the goal>

Step3: <Wrap up discussion or record completion>

Fig. 3 A Discourse Procedure

Apply the first rule that is appropriate:

T-RULE502 <Directly state single, Known rule>

If; I) There are rules having a bearing on this goal that have succeeded and have not been discussed, and
2) The number of rules having a bearing on this goal that have succeeded *is* I, and
3) There Is strong evidence that the student has applied this rule

Then: Simply state the rule and its conclusion

T-RULE5.03 <Request hypothesis when rules may be unknown>

If: You have examined the rules having a bearing on this goal that have succeeded and have *not* been discussed, and have found a rule under consideration for which there is not strong evidence that the student has applied the rule under consideration

Then: Substep i.

If: I) An Introductory remark is to be made before requesting an hypothesis from the student, and
2) The student has not requested help for forming an hypothesis

Then: Say: hypothesis-ready

Substep Ii. Discuss the student's hypothesis for the goal currently being discussed [ProcO14]

Fig. 4 T-rules for Deciding How to Discuss • Final Hypothesis (Fig. S, Step 2)

The COMPLETEDCOAL procedure illustrates two Important constraints for choosing among alternative dialogues on the basis of the principle of economy. Specifically, "surface complexity" features of the knowledge being discussed (e.g., the number of rules involved, whether a rule is definitional, that all relevant rules fail, and how many subgoals remain to be completed before an Inference can be drawn) and the student model combine In these tutorial rules to select a presentation method that affects how extensive the discussion will be.

When a conservative programmed tutor Is not sensitive to the complexity of a situation or the student's knowledge, tedious, overly detailed discussion results; the program belabors topics that can be dealt with swiftly by a single remark or simply skipped over. For example, a striking Improvement to GUIDON tutorials was based on the recognition that many MYCIN topics are "definitional," so the usual process of discussing all of the evidence explicitly and asking the student to make an hypothesis can be bypassed when the model Indicates that the student knows the relevant rule.

Besides economical considerations, the dialogue transition diagram illustrates two other kinds of transitions: logical and tutorial The links leading from the DATA procedure are distinctions based on domain logic, as are the three straight links leading from GOAL. These links are based on domain facts and relations (e.g., that a topic Is the name of a block of case data) and decisions made by the expert program (e.g., why a question need not be asked). In following these transitions, the tutor is reasoning about the subject material.

Tutorial transitions are based on the tutor's goals for teaching particular material to the student and/or refining its model of his knowledge. The examples in Fig. 2 are: 1) quizzing the student about rules that "recently" succeeded or failed because of case data Just given to the student ("recent-success" and "-failure" in Fig. 2); 2) providing initial orientation to get the student started on a topic that is new to him ("orient"); and 3) quizzing about rules related to the one Just discussed ("related-rules). We call these instances of *opportunistic tutoring* because they signify initiative taken by the tutor at "appropriate" times with the purpose of fitting as much information into the session as is practical These quizzes constitute subdlalogues that can be quite similar in motivation and format to SCHOLAR [10], WHY (22), and A BLOCKS [3] tutorials.

The appropriateness of the interruption is determined by Its consideration within the sequence of a discourse procedure, a conventional dialogue pattern, and by heuristics that determine whether a remark should be made The basic idea is that the program has certain topics that it wants to bring up (eg., a couple of rules relevant to the current case that the student may not know), so it checks to see if the kind of remark that is appropriate at a given time provides the opportunity to raise a particular topic Thus, after the tutor returned attention to the previous topic In Fig. I (lines 26-28), it checked to see if there were any rules that it wanted to mention that had Just then been applied by the expert program (there were none).

4 Managing Topic Relationships

A number of complications have been glossed over in the previous discussion, primarily problems of relating topics to one another during the dialogue. The subsections below deal with I) maintaining and sharing dialogue status information; 2) providing for and coping with student initiative; and 3) making situation transitions coherent and arguments clear.

4.1 Dialogue Status Information

GUIDON's dialogue status Information is in two parts: I) records of Inferences associated with each topic and rule, and whether they have been discussed in the current session; and 2) a simple record of the context of the dialogue.

Recall that every topic In the dialogue is a goal for which a conclusion is drawn from MYCIN rules and/or case data has been reported. GUIDON has to keep track of how MYCIN gained information and correct the student if he asks questions that are not related to the current topic or whose answers can be Inferred from what has been given.

Associated with each goal pursued by MYCIN, and hence each potential topic in the tutorial dialogue, is a dynamic inference and discussion record of data given to MYCIN and the student, conclusions made by MYCIN and conclusions believed to have been made by the student, and whether a rule or topic has been discussed in the current session. This information is combined to fill in the AND/OR tree to maintain models of the expert's and student's current knowledge; for example a rule is marked as "fired" when

MYCIN has marked all of its subgoals as known. Some of the information is used in fairly complex tutorial strategies, such as providing assistance according to an hypothesis revision strategy.

Various options make it possible for the student to access this record to keep track of the evidence that has already been discussed and what remains to be considered. For example, the PENDING option causes GUIDON to state subgoals that remain to be discussed, case data that can still be requested, and subgoals for which the student has enough Information to form a final hypothesis.

Another student-controlled means of sharing status Information directs the program to indicate when a conclusion can be made, e.g., "MYCIN just made a conclusion about the type of the infection." In terms of teaching strategy, this kind of remark is Intended to encourage the student to examine his own understanding at this point, perhaps leading him to request more information However, note that some students may not care what MYCIN *is* doing, preferring to proceed on the basis of their own knowledge.

The focus record consists of variables that are set when the student uses various options and when the tutor makes hints about topics for the student to pursue. By using this record when providing subsequent assistance to the student, the tutor's choice of topics is connected to what has gone on before. This method for keeping the dialogue "focussed" is admittedly simple, but because of the other constraints on the dialogue (i.e., stacked topics and the option-oriented format), it seems to be sufficient.

4.2 Student Initiative

An essential part of tutorial dialogue management is making provision for the student to express himself. When one first sets out to construct a mixed-initiative program, it may not be clear that provision must be made for every potential kind of initiative that the student will be allowed to make. This includes being able to refer back to an early topic and provide more details (given that explicit presentation of evidence may be non-exhaustive), allowing the other participant to change the topic, and so on We might summarize this by saying that we must allow the student to specify what he knows, what he wants to know more about, and what he wants to ignore.

GUIDON provides for student initiative by making available a variety of student options and by indexing tutorial remarks so that the student can easily refer to them later (use them as arguments to options). Options are grouped by the kind of initiative, some examples are shown in Fig. 5.

Option type	Examples
Get Case Data	BLOCK ALLDATA
Information Retrieval	PENDING DETAILS
Dialogue Context	RULE TOPIC
Say What You Know	IKNOW HYPOTHESIS
Request Assistance	HELP HINT TELLME
Change the Topic	DISCUSS STOP
Special	? PROFILE JUSTIFY

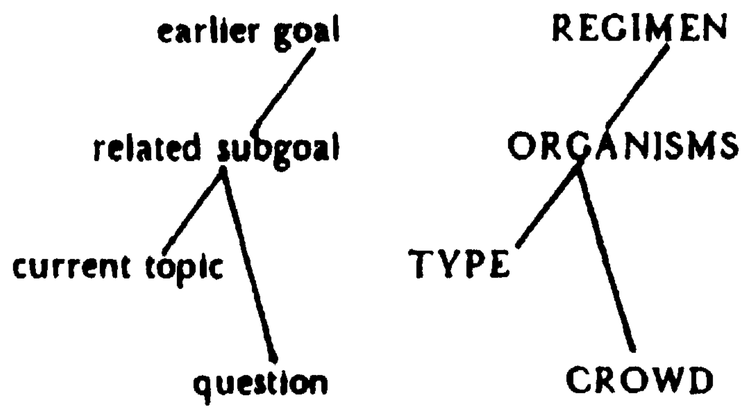
Fig. 5 Options Available During GUIDON Dialogues

Fig 6 illustrates the typical use of options Each option generally has a discourse procedure associated with it For example, the procedure for IKNOW invokes HYPOTHESIS if the expert program hasn't made a final decision yet; otherwise COMPLETEDGOAL is invoked. T-rule5.02 was applied in Fig. 6 when the student used the IKNOW option.

```
** FACTORS

The following factors will be useful to determine
the type of the Infection:

     3a.  whether the meningitis is partially treated
     3b.  whether a lumbar puncture has been
          performed on Pt538
     3c   the duration of the neurological signs
     3d   the CSF protein
(*)  3e.  whether Pt538 has a rash consistent with
          Herpes Zoster

  Factors marked with V have already been stated.
Based on what we have already mentioned, you should
be able to conclude about: 3b.

** USE 3C

The duration of the neurological signs is 7.2 hours.

Using other Information available to you now, the
duration of the neurological sigm enables you to
conclude that the type of the infection is bacterial
(.2) viral (.2) fungal (-.4) tb (-.4) [RULE524J.

** IKNOW 3B

Good MYCIN has also made a decision.
The site of CULTURE-1 is evidence that a lumbar
puncture has been performed on Pt538 [RULE1121

Back to our discussion of the type of the Infection...
```

Fig 6 Sample Use of Options

GUIDON allows a student to explicitly change the topic by the DISCUSS option. However, student requests for data can also (implicitly) change the topic if the datum requested is not relevant to the current topic (cannot be used directly in any Inference). Fig. I illustrated a dialogue that occurred when the requested Information was relevant to a deeper sub goal If requested Information is relevant to a previous, shallower subgoal, the tutor states this relation so that it is clear to the student what topic Is currently being pursued (Fig. 7).



** DOES THE PATIENT LIVE IN A CROWDED ENVIRONMENT?

Pt538 does not live In a crowded environment.

Whether the patient does live in a crowded environment is not relevant to determining the type of the infection.
It Is a consideration we can use later when we return to our discussion of the organisms that might be causing the infection.

Fig 7 Crossing Topic Paths in a Tree of Subgoals

4.3 Situation Transitions

Clarity of presentation is critical In productive teaching, as in any dialogue. In GUIDON tutorials this involves providing coherent transitions between dialogue situations and comprehensible summaries of inferences

Providing clear transitions requires that discourse procedures take Into account the context in which they are invoked, so that the tutor's remarks are not disconnected or redundant The chief design principle we have followed Is that a tutorial rule that Invokes a new discourse situation provides transition to the new situation and return to the former context. For example, T-RULE 5 03 prepares the student for the request for an hypothesis Note that the predicate "an Introductory remark Is to be made..." is a flag set by the calling discourse procedure, e.g., it is false if the student has explicitly requested the opportunity to make an hypothesis.

Fig 8 illustrates the kind of disconnected dialogue that arises from Inadequate communication between discourse procedures Following the procedure that Is Invoked after an hypothesis ("advice" in Fig 2), the tutor asks the student if he wants to continue discussion of this topic or if he would like to wrap It up at this point. The student wishes to continue, so the dialogue shifts to discussion of this topic (the procedure GOAL), and the first thing the tutor remarks about Is that the topic is essentially done (attempting to make an economical transition based on a comparison of the expert program's current hypothesis to the final conclusion it will reach after all evidence has been considered).

A smoother dialogue would result if the procedure for giving advice after an hypothesis looked to see if the topic was substantially completed and so the student would only be asked once If he wanted to go on. The interaction shown here is typical, and many of the changes made to the system after each trial tutorial session are designed to improve the dialogue In similar ways. While we are constantly allowing for more subtle variations in the dialogue, the framework of rules organized Into procedures is working well

** HYPOTHESIS 3A

Is the meningitis partially treated?
** YES

You're right

The time since therapy with thf cephalothin was started Is evidence that the meningitis Is partially treated (.95) [RULEI45]

It remains to collect further evidence for determining whether the meningitis Is partially treated.
Would you like to see a summary of all of the evidence that can eventually be collected (as opposed to working it out yourself)?
** NO

Would you like to see a summary of the evidence we have discussed before going on to ask more questions?
** NO

We have substantially completed our discussion of whether the meningitis Is partially treated
Would you like to make an hypothesis now (as opposed to gathering more Information)?
** NO

Fig. 8 A Disconnected Dialogue

Finally, when tutorial remarks are limited to lines of print on a teletype page (a constraint this research has adopted), there Is no opportunity to pause or to emphasize words (by changing voice volume or inflection) Therefore, it is worthwhile to incorporate visual keys that help to organize presentations, and provide clear transition between segments of the dialogue. For example, when summarizing evidence, GUIDON regroups goal/subgoal relations, separating "key factors" from contextual information The usual, MYCIN-style listing of these same rules could require several pages.

5 Limitations and Scope

As stated previously, GUIDON is not intended to be a complex natural language converser, such as GUS [I] In order to concentrate on formalizing specific discourse procedures, we have avoided the problem of recognizing new dialogue situations In free text by providing the student with an option-oriented vocabulary. Moreover, generation of remarks is on the level of choosing the course of the dialogue, rather than building up output from grammatical components. We devote just a small amount of computation towards determining the student's Intentions, chiefly by relating student requests for data to the current topic. However, we do Intend to build a more complex model of the student's strategies, and this might Involve parsing complex Input (as in Sleeman's program for understanding student explanations [21]) Furthermore, the AND/OR tree of topics and rules strongly constrains the tutor's choice of dialogue situations In less rigidly pedantic dialogues, generation of conversation Is sensitive to considerations such as the social context of the participants and emotional connotatioa

(Some of Burton and Brown's *kibitzing* strategies are like this [6])

Second, besides the limitations of GUIDON's mixed-Initiative conversation capability, It can only discuss rule-based knowledge It Is clear that not all domain knowledge can be expressed as topics that are determined by applying situation-action rules; but it Is plausible that this will be an adequate framework for discussing a wide variety of diagnostic problems For example, suppose that the representation of knowledge consisted of relations that were inference triggers (as In INT FRNIST [19]), we still might find It convenient to think of the dialogue as being structured into goals (something to find out), case data, possible outcomes (alternative hypotheses), and inference relations It seems probable that many of the Issues of dialogue management we have discussed will be important, though we may need to augment the situations we have formalized (Fig. 2) For example, for an hypothesis-oriented discussion we would refine the GOAL procedure to focus on one outcome value at a time, and Incorporate transition t-rules for detecting and discussing shifts to another outcome (hypothesis revision)

Third, some aspects of our work have been motivated and aided by the particular rule bases available to us The design of GUIDON's dialogue implicitly makes use of the fact that MYCIN'S AND/OR tree is never more than five topic levels deep and the current topic Is usually only two levels below the top goal of determining therapy for the patient. A significantly deeper tree might prohibit a stacked topic" design. Moreover, the success and usefulness of GUIDON's tutorial ability depends to some extent on the universality of the rule set, i.e., how the subject matter has been structured by the rule authors. The domain rules were designed to be comprehensible to potential users of the consultation system who are not experts, but it is Inevitable that some of the concepts represented in the rules are artifacts of the particular representation and the requirements of making everything explicit.

6 Closing Remarks

While most ICAI research has focussed on representation of domain expertise and construction of a student model, we have shpwn that there Is a group of issues that center about the problem of carrying on a coherent, task-oriented mixed-initiative dialogue with a student We have named this collection of issues the dialogue management problem and discussed *it in terms* of tutorials based on MYCIN-like rules. We presented a representation of dialogue knowledge in the form of a transition diagram in which the nodes are discourse situations and the links represent selection of alternative dialogues based on domain logic, economy of presentation, and tutorial objectives. Other issues were described in terms of managing topic relationships and sharing initiative.

We stated that the main forcing function behind improvement of the program was the impossibility of explicitly discussing each inference made by the expert program. Verbosity continues to be a problem for GUIDON. It is interesting to note that the most effective "pruning" of topics in the current version of the program is a result of the student's initiative. Perhaps it is not necessary or desirable for the program to attempt to manage the dialogue too severely; deciding what should be discussed is naturally a shared task. We hope that a proposed "case lesson plan" [9] will give the tutor additional leverage for non-exhaustive discussion by providing reasonable, time-sensitive goals for the session.

References

[I] Bobrow, Daniel G., Kaplan, Ronald M., Kay, Martin, et al (1977). GUS, A Frame-Driven Dialog System. Artificial Intelligence, Vol 8 No 2, 155-173

[2] Brown, J.S., Rubenstein, R., Burton R., (1976). Reactive Learning Environment for Computer-Aided Electronics Instruction. BBN 3314

[3] Brown, J.S., Burton R, Miller M, deKleer, J., Purcell, S, Hausmann, C, Bobrow, R. (1975). Steps toward a Theoretic Foundation for Complex, Knowledge-Based CAI. BBN 3135

[4] Brown, J.S (1977). Uses of AI and Advanced Computer Technology In Education In Computers and Communications: Implications for Education. Academic Press, Inc New York.

[5] Burton, R. (1976) Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems. BBN 3453.

[6] Burton, R. (1979). An investigation of Computer Coaching for Informal Learning Activities The Int J or Man-Machine Studies II.

[7] Carbonell, Jaime R (1978). Intentionality and Human Conversations Theoretical Issues in Natural Language Processing-2, July 1978, 141-148.

[8] Carr, Brian and Goldstein, Ira (1977). Overlays: A Theory of Modelling for CA1. MIT AI Lab Memo 406.

[9]Clancey, William J. (1979). Tutoring Rules for Guiding a Case Method Dialogue The Int J of Man-Machine Studies 11,25-49.

[10] Collins, Alan. (1976). Processes in Acquiring Knowledge. In Schooling *and* Acquisition of Knowledge, (Anderson, Spiro and Montague, eds), Erlbaum Assoc., Hillsdale, NT.

(II)Deutsch, Barbara G. (1974). The Structure of Task-Oriented Dialogs. IEEE Symposium for Speech Recognition, 250-253.

[12] Faught, William S. (1977). Motivation and Intensionality in a Computer Simulation Model Ph. D. Dissertation, Stanford Unlversity,AIM-305.

[13] Goldstein, Ira. (1977). The Computer as Coach: An Athletic Paradigm for Intellectual Education. MIT AI Lab Memo 389.

[14] Gordon D and Lakoff, G (1971). Conversational Postulates. Papers from Seventh Regional Meeting, Chicago Linguistic Society, Chicago. *University* of Chicago Linguistics Department.

[15] Grosi, Barbara J (1977). The Representation and Use of Focus In a System for Using Dialogues. Proc. of 5th IJCAI, 67-76.

[16] Hart, Peter E. (1975) Progress on a Computer Based Consultant. Proc of 4th IJCAI, 831-841.

[17] Hart, R O. and Koffman, E B. (1975). A Student-Oriented Natural Language Environment for Learning LISP. Proc. of 4th IJCAI, 391-396.

[18] Koffman, Elliot B, and Blount Sumner E. (1973). Artificial Intelligence and Automatic Programming in CAI. Proc. of 3rd IJCAI, 86-94.

[19] Pople, Harry E. (1975). DIALOG A Model of Diagnostic Logic for Internal Medicine. Proc. of 4th IJCAI, 848-855

[20] Shortliffe, E H. (1974) Computer-Based Medical Consultations: MYCIN, American Elsevier, New York, 1976.

[21] Sleeman, Derek. (1977). A System which Allows Students to Explore Algorithms, Proc. of .5th, IJCAI, 780-786.

[22] Stevens, Albert L., and Collins, Allan. (1977). The Goal Structure of a Socratic Tutor. BBN 3518.

MECHANIZING A PARTICULARLY EXPRESSIVE MANY SORTED LOGIC

Anthony Cohn
Department of Computer Science
University of Essex, U.K.

ABSTRACT

A particularly expressive many sorted logic is presented along with an outline of an implementation. The logic allows functions and predicates to be sorted polymorph!cally and the sortspace to be partially ordered. It is shown that truth values may sometimes be inferred solely on the basis of sort information.

1. INTRODUCTION

A problem arising in many AI systems is that of matching two structures. The simplest form of matching is a purely 'syntactic' comparison of nested patterns organised hierarchically, see [6]. Tips of the patterns are either constants (which match only if equal) or variables (which match any pattern in which they do not occur). We consider matching processes which utilise sort information; these are still relatively simple and syntactic (and thus cheap) but offer considerable selectivity, since the mismatch of sorts can prevent unifications which lead to useless inferences (contrast, e.g., KRL [I] where 'matching' covers arbitrarily complex inferencing). Moreover, it is sometimes possible to infer the truth value of an expression merely on the basis of the sorts of some of its subexpressions. E.g., if c is female then FATHER(c) is definitely false,

2. ASSIGNING SORTS TO EXPRESSIONS

Mechanized sorted logics are not new (e.g., [2] , [8], but typically their expressive power has been limited by the restrictions on how expressions may be sorted; in particular, variables must usually be quantified over a single sort thus precluding polymorphism (though [8] allows a very limited polymorphism).

Defining polymorphic functions is a common and useful style of programming. We want to use polymorphism in axiomatisations. Hayes (V) describes a logic allowing polymorphically sorted relations and functions, (In [7] Strachey distinguishes 'parametric polymorphism' in which, e.g., maplist has sort (a +3)+ list(8), from 'adhoc polymorphism[1] or overloading; our implementation only handles overloading). Rather than variables having sorts, each relation and function h has an associated sorting function h' which specifies which sorts are legal arguments and for each legal combination of argument sorts, the sort of the value of the function. [5] makes heavy use of this logic.

Thus the sort restrictions are specified once only for every non-logical symbol and the sorts a variable may have in any wff are determined by context. This is more convenient than specifying the sorts locally in every wff.

As a simple example consider the following naturally polymorphic relation
spouse: (MANxWOMAN "+ U) + (WOMANxMAN -► u)
(the choice of u as the range is explained in sec. 4).
In a monomorphically sorted logic, we must define spouse: HUMANxHUMAN-> U
which fails to capture all the constraints of our earlier definition; alternatively, if we allow a partial ordering on sorts (see sec, 3) then it it possible to define SPOUSE in terms of WIFE and HUSBAND with tighter sort restrictions, at the expense of forcing a two way split in the search space, depending on whether we assume WIFE(x,y) or HUSBAND(x,y). Our implementation allows the or-branching implicit in a polymorphic function to be maintained internally in a single derivation rather than forcing a split into several derivations, increasing the search-space branching rate.

However, contrary to Hayes' claim in [4], mechanizing such a sort structure is not trivial. E.g., if g:(SIxS2 -»SI)+(S2xSI -->S2) then (P(g(x,y),g(y,x)),P(u,u)} should fail to unify,

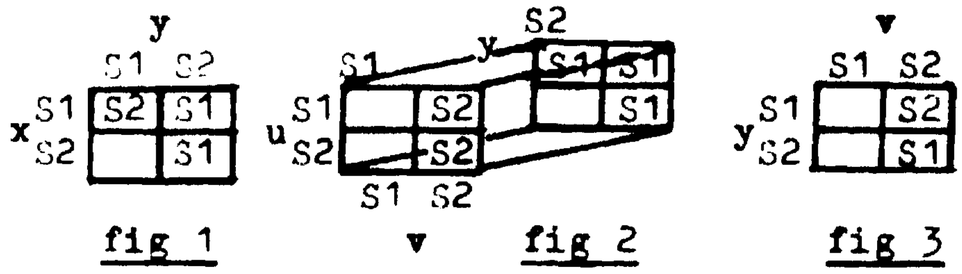
as should $\{Q(y,g(x,y),g(w,x)),Q(u,v,g(v,u))\}_t$ because the resulting terms (g(x,x) in the former and g(g(x,y),x) in the latter) both violate g's sort constraints. The difficulty occurs when merging two variables, because the checks cannot be kept local to the part of the proof tree being operated upon, as the scope of a variable is that of the whole proof tree. E.g., merging x/u would mean the erstwhile legal proof tree containing the terms g(x,y), g(y,z) and g(z,u) would become illegal; locally each terms looks correct; inconsistency is only detected when the three terms, which may be widely separated in the proof tree, are considered together.

We need to be able to check the sort restrictions globally. For every well formed expression (wfe) b, we can define a partial function B:F-+S where F=V->S, V is the set of variables and S the set of (for the moment) disjoint sorts, such that B(f) is defined and gives the correct sort for the wfe for the combination of sorts on variables specified by f. We can regard f c F as specifying an environment in which every variable is bound to some sort. B then specifies which environments are permissible for b, and what the sort of b would be in that environment.

In our implementation, we choose to represent B as an n-dimensional square array called a sort array(SA), with as many rows and columns as there are sorts and n=|v|. Thus every position represents an fc F. We define domB={f:B(f)€ S). If sorts are being used to good effect then dornB| will be small and thus the array sparse. If" domB=Φ then no combination of sorts of variables is legal: b is informed.

We can compute the SA for a proof tree incrementally as the tree is built; the operations on SAs that we need to perform mirror the operations on the tree: substituting for variables and attaching new expressions to the proof. The primitive building blocks are the SAs representing the sorting functions of nonlogical symbols.

The following example illustrates two of the basic operations that we need to perform on SAs. Suppose the SA for term g(x,y) is as in Fig. 1, then binding x to g(u,v) gives the SA for g(g(u,v),y) in Fig. 2. Binding a variable to a a term with k new variables increases the dimen-. sionality of the SA by k-1. Merging two variables decreases the dimensionality by one, by taking a diagonal slice through the array. So substituting u/y picks out the rows on the y=u diagional, giving the SA for g(g(y,v),y) (see Fig. 3). Binding v to a constant c of sort SI gives a vector with no defined entries; thus g(g(y,c),y)) is not a wfe.



3. NON-DISJOINT SORTS

A sort specifies a subset of objects in the universe. If we allow non-empty intersections then we naturally induce a partial ordering (i.e., set inclusion) on S. Matching the sorts of two terms when unifying is then set intersection.

Inclusion induces a partial ordering on F and F→S in the usual way, e.g.:
$\forall f1,f2 \in F\ f1 \sqsupseteq f2 \equiv \forall v \in V\ f1(v) \sqsupseteq f2(v)$
We require B in F→S to be monotonic:
$f1 \sqsupseteq f2 \Rightarrow B(f1) \sqsupseteq B(f2)$
and sensible:
$\forall d \subseteq \underline{domB},\ d \neq \phi:\ \exists z = \bigsqcup d \Rightarrow \exists y = \bigsqcup\{B(f): f \in d\}$
With these (natural) restrictions, the implementation sketched above can be extended to the case of non-disjoint sorts. Considerable computational and space savings follow from the observation that one can often calculate some entries in a SA, given others. Thus only part of each SA - already sparse - needs to be explicitly stored in memory. Details of such normal SAs are in [3].

4. SORTING WFFS.

The value of a wff is true or false. Defining a single sort $\mathfrak{U}$ = {true,false} leads to problems: we can only sort MOTHER and FATHER loosely as (HUMANxHUMAN→$\mathfrak{U}$) to make sense of
$\forall x,y$ PARENT(x,y) ≡ MOTHER(x,y) ∨ FATHER(x,y)
otherwise x∈MAN∪WOMAN! We shall define the functionality of SAs for wffs as F→BOOL where BOOL = {$\mathfrak{T}$,$\mathfrak{F}$,$\mathfrak{U}$} and $\mathfrak{F}$ = {false} and $\mathfrak{T}$ = {true} We can induce a lower semi-lattice on BOOL as we did on S. We now sort MOTHER and FATHER as (MANxHUMAN→$\mathfrak{F}$)+(WOMANxHUMAN→$\mathfrak{U}$) and (MANxHUMAN→$\mathfrak{U}$)+(WOMANxHUMAN→$\mathfrak{F}$) respectively.

An advantage of this technique is that characteristic predicates need no special treatment by the inference machine; e.g., we can sort MAN: (MAN→$\mathfrak{T}$)+(⊥\MAN→$\mathfrak{F}$).
To demonstrate a second advantage let r={B(f): f∈ F} where b is some wff; if r = {$\mathfrak{F}$} then b is a contradiction (e.g., MOTHER(John,x)) and if r = {$\mathfrak{T}$} then b is a tautology (e.g., ¬MOTHER(John,x)). Similarly if we are trying tc prove ∀ x,y MOTHER(x,y) and we resolve against the above assertion then we need only resolve away the subgoal PARENT(x,y) since FATHER(x,y)

can be inferred false by the sort structure. This technique generalises the concept of insignificant literal in [2].

5. THE EFFECT OF SORTS ON THE LOGIC

We can only sketch this interaction: full details and proofs, including completeness and soundness results will be found in [3].

Every SA represents a set of axioms which would otherwise have to be explicitly asserted; e.g., if b: (S1xS2→S3)+(S4xS5→S6)
then the missing axioms are (assuming disjoint sorts): $\forall x,y$ S3(b(x,y)) ≡ S1(x) ∧ S2(y) and $\forall x,y$ S6(b(x,y)) ≡ S4(x) and S5(y)
We must take account of these implicit axioms if we are to prove (say) S2(c), for a proof of S3(b(c,y)) would imply S2(c). That a given SA B is correct may be shown by proving that the equivalent axioms follow from the sort lattice axioms, the primitive SAs for all the nonlogical symbols and from b itself.

The axioms which define the structure of the sort lattice are eliminated in two ways.
(1) Use of lattice theoretic operations (e.g. ⊓, ⊔) on SAs (see definitions below).
(2) By extending resolution in the following way. We must allow Si(c) to match with any positive literal Sj(t) (where Sj ⊔ Sc ⊏ Si, t unifies with c, Sc, Si, Sj ⊏ S and Sc(c)). This includes the conventional case where Sj=Si. It must also match Sj(t) where Sj ⊔ Sc ⊏ Si provided we also prove ¬(Sj ⊓ Si)(c)σ, where σ is the substitution unifying t and c. Similar but dual techniques apply when trying to prove ¬Si(c).

We now present a formal definition of how to compute the new SA D after substituting e for x in a proof tree b. We need only consider the modified SA B for which {B(f):f ∈ domB}={⊥}, since for all other f ∈ domB we can already infer b's truthvalue. So clearly if f ∈ domD then D(f)=⊥. There are two cases to consider
(1) e is a variable then
$\forall f \in domB\ \exists z=f(x) \sqcup f(e) \Rightarrow$
$(\lambda v.\, v=e\rightarrow z,\ v\neq x\rightarrow f(v)) \in domD$
(2) e is not a variable
$\forall f \in domB,\ g \in domE\ f(x) \sqsubset E(g) \Rightarrow$
$(\lambda v.\, v\neq x\rightarrow (f+g)(v)) \in domD$
If necessary b and e must be standardised apart before computing D. Subsequently the erstwhile shared variables must be remerged, diagonalizing in each case.

(Both the above operations have to be modified slightly to propogate normality.) Definition (2) only allows a term to match a variable of a lesser sort, though we can remove this restriction by replacing the left hand side with '$\exists z=f(x) \sqcup E(g)$', providing that we also prove that e is of the more restricted sort; the resulting logic is more expressive (noticed first by Plotkin). E.g., suppose we wish to prove $\exists x P(x)$ where P is only defined on abelian groups. If we know that R(g) and that $\forall x R(x) \Rightarrow P(x)$ (where R is defined on groups and g is some group) then we can complete the proof providing we also prove that g is abelian (clearly abelian-group ⊐ group). Polymorphism complicates the precise definition of this extension so we do not give it here.

6. CONCLUSION

The difficulty experienced in mechanizing a theory or language is usually proportional to its generality or expressiveness. We have proposed a particularly general many sorted logic: allowing polymorphic functions and a partially ordered sortspace; furthermore, the sort of an expression may sometimes be determined solely from sort information.
A pilot implementation which works for the basic theory is currently being extended. At present SAs have to be ground instances, i.e., sort metavariables, allowing parametric polymorphism are not permitted. It is hoped to include this facility.

7. ACKNOWLEDGEMENTS

Phil Collier and Jon Cunningham made useful comments on a draft; I am particularly indebted to Pat Hayes for his help and guidance.

REFERENCES

[1] Bobrow, D. and Winograd, T. "An overview of KRL", Cognitive Science, vol. 1, No. 1,(1977).

[2] Champeaux, D. "Dating semantic networks", Proc. AISB/GI, Hamburg, 1978.

[3] Cohn, A. "Mechanizing and implementing many sorted logic", Ph.D. thesis. In preparation.

[4] Hayes, P. "A logic of actions", Machine Intelligence 6.

[5] Hayes, P. "Naive Physics 1: Ontology of liquids", Essex University, 1978.

[6] Robinson, J. "A machine oriented logic based on the resolution principle", JACM 22, pp.572-596.

[7] Strachey, C. "Fundamental concepts in programming languages", Notes for the Int. summer school in computer programming, Copenhagen, 1967.

[8] Weyhrauch, R. "Lectures notes for summer school on AI in Pisa", 1978.

MANIPULATION EXTRAPOLATION
A SYSTEM FOR CONTROLLING TRAINABLE ROBOTS

Ronald W. Colman
Computer Science Department
California State University
Fullerton, California 92634

Program synthesis from exemplary data is employed as a context for the design of trainable robots. A method for inferring programs from sample execution traces is illustrated. The method provides a possible model of a way humans are trained-by-exaraple. The convergence characteristics and the conditions sufficient to ensure them are briefly discussed.

1. INTRODUCTION

When training another human, we frequently demonstrate, by example, how the first few of a sequence of similar problems are solved. The trainee induces a program from such a demonstration. For example, a program to insert shims in an assembly might be as follows:

1) Test fit and exit If within tolerance.
2) Select largest shim not overcompensating.
3) Insert selected shim. Go to step (1).

An industrial trainee would likely learn this by repeatedly

a) observing the Initial state of an assembly.
b) observing a sequence of actions by a trainer which modify this state. And perhaps,
c) hearing comments such as, "Always use the largest shim possible.

This paper proposes a model for such an interface between a human trainer and a computer. The most general context, program synthesis, is employed. The input for the i-th problem is assumed to be as follows:

a) an initial state, s_i, of data in the memory of a computer.
b) an execution trace, t_i, of a hypothetical program to be induced. This is the sequence of instructions executed by the program in transforming s_i to a final state, from which control commands are deleted. And perhaps,
c) identification by the trainer of those variables of significance at decision points. This reduces combinatorial explosion and is not required by the example to follow.

A training sequence, as follows, is assumed:

$T_N = [(s_1,t_1),(s_2,t_2), \ldots, (s_N,t_N)]$

2. MANIPULATION EXTRAPOLATION

The process to be presented will employ two subsystems, both well-deiined in the literature elsewhere. Grammatical inference, the problem of inducing a grammar from a sample of sentences, is solved by a variety of published methods for regular languages [4]. One by the author has convergence characteristics especially suited to this application [I]. The calculation of discriminant functions for numerical vectors has also been extensively studied [2], An algorithm for the linear functions sufficient for the following is found in [5]. More general polynomial functions may use linear programming as demonstrated by the author [I].

2.1 The Process

In the following, G is a formal grammar induced to generate the set of execution traces. Alternative productions in this grammar (productions with identical left parts) are selected by discriminant functions defined over the set of partially transformed states occurring at the associated point in the calculation.

The process is specified in detail by specifying the inference procedures to be used for G and the discriminant functions. This is done in (1) where convergence is established.

A micro-processor controlled manipulator is envisioned, communicating interactively with a large computer where its program would be induced as follows:

0) INITIALIZE: Set grammar, G, to generate the empty language, T_o = nil and i = 0.
1) INCREASE SAMPLE: Set i = i+1 and $T_i = T_{i-1} \cup [(s_i, t_i)]$
2) TEST G: If t_i is in the language generated by G, go to step 4.
3) INFER GRAMMAR: Reform G to generate at least t_j j=1,2,...,i and go to 5.
4) TEST STATE DISCRIMINATORS: If discriminant functions involved in processing s_i select the correct path to generate t_i, go to 1.
5) COMPUTE DISCRIMINANT FUNCTIONS: Reform discriminant functions for each set of productions altered in step 3 or found to be improperly selected in processing s_i. Go to 1.

3. ILLUSTRATIVE EXAMPLE

The computation of N-factorial will provide a simple illustration. Let the data state consist of two elements, (x1,x2), with x1 initialized at i-1 for the i-th problem (the computation of (i-1)-factorial) and let x2 be initialized at 1. Associated with each such starting state will be the shortest sequence of instructions possible to transform x2 into the factorial of the initial value in x1. The training sequence for the first three problems is as follows:

T_3 = [((0,1), nil), ((1,1), nil), ((2,1), load-x1-store-s2)]

Each t_i will be treated as a 'sentence' over the 'alphabet' of commands shown in figure 1.

CHARACTER	INTERPRETATION
1. load x1	(load x1 into accumulator)
2. load x2	(load x2 into accumulator)
3. store x1	(deposit accumulator in x1)
4. store x2	(deposit accumulator in x2)
5. decr x1	(decrement contents of x1)
6. decr x2	(decrement contents of x2)
7. mul x1	(multiply accumulator by x1)
8. mul x2	(multiply accumulator by x2)

Figure 1: EIGHT 'CHARACTERS' IN ALPHABET OF G

A grammar generating this language, {nil, load-x1-8tore-x2}. is shown in figure 2. The language is more concisely represented using the numbers of the commands in figure 1, {nil,14}, instead of the commands above.

```
<start> ::= < 1 >
  <1>   ::= < end >
  <1>   ::= < 2 >
  <2>   ::= load-x1-store-x2 <end>
```

Figure 2: PRODUCTIONS OF GRAMMAR INDUCED FROM T_3

The fourth and fifth problems yield the training sequence:

$T_5 = T_3 \cup [((3,1),145184),((4,1),1451845184)]$

Each of these new sentences is found to be absent from the language generated by the grammar induced from the preceding training sequence and a new grammar is induced. The resulting grammar induced from T is shown in figure 3. This grammar will be seen equivalent to the flowchart in figure 4. It generates a solution for each of the infinite set of problems, N-factorial for N=0,1,2,.... Branches in the flowchart correspond to alternative productions in the grammar. These must now be associated with discriminant functions to transform this grammar into a program.

```
<start>::= <1>
    <1>::= <end>
    <1>::= <2>
    <2>::= 1 4 <3>
    <3>::= <end>
    <3>::= < 4 >
    <4>::= 5 1 8 4 <3>
```

Figure 3: PRODUCTIONS OF G INFERRED FROM T_5

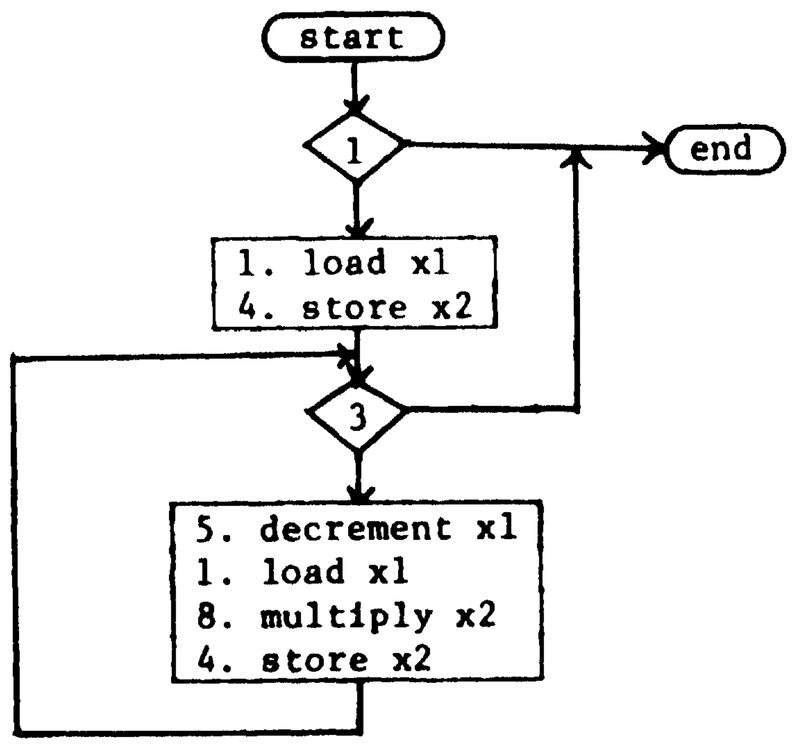


Figure 4: FLOWCHART EQUIVALENT TO GRAMMAR

Figure 5 shows the data states associated with each of the branch points in the flowchart. Each of these sets of states is dichotomously partitioned into mutually exclusive subsets associated with a particular alternative production in the grammar. A linear discriminant function, F(xl,x2) - A*xl+B*x2+C, is adequate tor both branches. For branch 1, (A,B,C)=(3,-2,-2) will cause F(xl,x2) to be negative for states associated with rule, <!>::«= <end>, and positive for states associated with its alternative, <1>::- <2>. For branch point 3, (A,,B,C)=(2,0,-5) will similarly select the correct 'sentence' for each problem. An algorithm exists, guaranteed to find such linear separators if they exist [5],

ALTERNATIVE PRODUCTIONS	ASSOCIATED STATES
Branch Point 1	
<1>::= <end>	(0,1),(1,1)
<1>::= <2>	(2,1),(3,1),...,(i,1),...
Branch Point 3	
<3>::= <end>	(2,2),(2,6),...,(2,i!),...
<3>::= <4>	(3,3),(3,12),(4,4),(4,20),...

Figure 5: DATA STATES ASSOCIATED WITH ALTERNATIVE PRODUCTIONS (GRAMMAR) AND DECISION POINTS (FLOWCHART)

3.1 Interaction with Trainer

Trainees often ask questions. Some grammatical inference methods use interrogation to determine sentences to be excluded from the language generated by the grammar. This is not appropriate to the application represented here and the grammar inference process used in [I] avoids a need for this by assuming that the training sequence contains all sentences shorter than the longest sentence present. The trainer would, however, likely have insights about which variables in the state vector are irrelevant at decision points. Interrogation of the trainer for this purpose, while not yet implemented, would greatly reduce the complexity of the discriminant function calculations.

4. CONVERGENCE

While flowcharts are equivalent to regular grammars, execution traces, representing a subset of all possible paths through the flowchart, are often a proper subset, selected by predicates at decision points, and are not regular languages. The regular grammar induced from such execution traces will, in general, generate a language larger than the solutions to all problems under consideration. A convergence criteria "approach from below," captures the notion appropriate here [I]. A general discussion of such convergence is in [3].

Convergence of the grammar inference process is necessary but not sufficient. Separability of states at decision points in the induced grammar must be assured. The assumption that the training sequence could have been generated by a program with separable decision points is not sufficient since the induced grammar may be structurally dissimilar to the hypothetical generator.

A constructive grammar inference process devised by the author has been shown to preserve polynomial separability under the assumption of the existence of such a hypothetical program (1). Polynomial separability is reasonably general but excludes some frequently used tests such as "branch if variable x is even."

REFERENCES

[I] Colman, R. W. "Inductive Extrapolation." Ph.D. Dissertation, Info, and Computer Science Dept., University of California at Irvine, January, 1977.

[2] Duda, R. 0. and P. Hart. Pattern Classification and Scene Analysis. J. Wiley and Sons, New York, 1973.

[3] Feldman, J. A. "Some Decidability Results on Grammatical Inference and Complexity." Information and Control 20 (1972) pp. 244-262.

[4] Fu, King-Sun and T. L. Booth. "Grammatical Inference: Introduction and Survey," IEEE Transactions on Systems, Man and Cybernetics. January, 1975, pp. 95-111.

[5] Nilsson, N. J. Learning Machines: Foundations of Trainable Pattern Classifying Systems. McGraw-Hill, 1965.

HIERARCHICAL PLANNING IN A DISTRIBUTED ENVIRONMENT

Daniel D. Corkill

Computer and Information Science
University of Massachusetts
Amherst, Massachusetts, 01003

Planning is a crucial aspect of many applications which are naturally suited to the use of distributed processing hardware. Use of a centralized planner is generally incongruous with effective distributed problem solving systems, motivating generalization of centralized planning techniques to accomodate multiple and distributed centers of planning control.

Such a generalization of Sacerdoti's NOAH (Nets of Action Hierarchies) planning system is described. This generalization involves distribution of NOAH's criticism and world model mechanisms.

1. INTRODUCTION

Planning is a crucial aspect of many applications which are naturally suited to the use of distributed processing hardware. These applications often occur in situations where sensory devices, processing capability, and devices to be controlled have wide spatial distributions and are even mobile.

Use of a centralized planner in these applications is incongruous with the development of distributed problem solving methodologies, Due to the high cost of interprocessor communication, transmitting environmental information to a centralized planner and distributing completed plans to appropriate processors is potentially expensive. The inappropriateness of centralized planners in most distributed applications motivates generalization of centralized planning techniques to accomodate multiple and distributed centers of planning control.

2. A DISTRIBUTED NOAH SYSTEM

2. 1 NOAH as a Framework for Distributed Planning""

Sacerdoti's NOAH [1] is a suitable candidate for generalization to a multiple center distributed planner for several reasons:

a. Plan expansion is already localized to the expansion of individual actions. Distribution of plan expansion is simply a matter of locating plan expansion (SOUP) procedures at each processor. Separation of plan expansion from consideration of action interdependencies (criticism) allows expansion to be performed locally, prior to the necessarily non-local analysis of interactions.

This research was supported by National Science Foundation Crant MCS78-04212.

b. The integration of nonlinear and hierarchical planning techniques reduces the combinatorial growth of planning. This reduction in the size of the planning space potentially lowers the amount of interprocessor communication required in the distributed system.

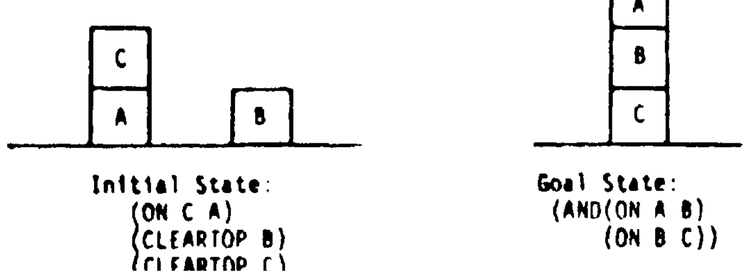
c. The generated plans retain their- nonlinear representation, allowing for parallel distributed execution without additional processing to detect potential parallelism.

Intelligent allocation of planning task responsibility among individual processors is an important aspect of distributed planning. In the distributed NOAH system, certain high-level conjunctive subgoals are allocated to individual processors. Division between conjuncts is appropriate because conjunctive goals are initially assumed independent by NOAH. To the degree that the conjuncts are indeed independent, there, is no need for interaction between processors to linearize the subplans, and the subplans can be executed in parallel. However, this approach does not guarantee a close correlation between the spatial distribution of the planning process and the spatial distribution of environmental information and of effectors required during plan execution. In the following sections, a suitable subplan to processor allocation mechanism (a Decompose Plan critic) is assumed.

To complete distribution of NOAH's planning phase, the world model and general criticism mechanisms must be distributed. The distributed critics are extensions of NOAH's original general purpose critics. The centralized versions of these critics are retained in the distributed NOAH system and are used to operate on the local plans, with interprocessor criticism performed by the distributed criticism techniques.

2.2 Distribution of Resolve Conflicts

To show how a distributed Resolve Conflicts critic can be implemented, we look at a simple problem involving three blocks and two planning processors. Although illustrative of the interactions between distributed planning elements, blocks world problems have a high degree of interaction between actions, and the resulting plans tend to be linear in nature. The initial state of this problem (which is assumed to be known by both processors) and the goal state are:



A single processor, P1, is given the goal state, which is entered on Pi's local procedural net (Fig. la) and expanded into a conjunction at the second hierarchical level (Fig. lb).

Pi's Decompose Plan critic allocates one of the conjuncts, (ACHIEVE(ON B C), to a second processor, P2 (Fig. Pa). The other conjunct, (ACHIEVE(ON A B)), remains P1's responsibility (Fig. 1c). The subplan which P1 will generate to achieve (ON A B) is modelled by P2 as the MODEL node (PLAN:1a). MODEL nodes are similar to PHANTOM nodes, except that their add and delete lists represent expressions that are to be made true by actions of another processor. In this case, (PLAN:la) contains (ON A B) on its add list. The subplan which P2 will generate to achieve (ON B C) is modelled by PI as (PLAN:2a).

P1 expands (ACHIEVE (ON A B)) to the third level, as shown in Fig. Id. * Since (PUT A ON B) deletes the expression (CLEARTOP B), PI must inform P2 of this change by sending the message (DENY(CLEARTOP B)).

?2 performs an analogous expansion (Fig. 2b), sending (DENY(CLE ARTOP C) to PL **

P1 receives (DENY(CLEARTOP C)) from P2 and enters it into the delete list of (PLAN:2a). P? does likewise with (DENY(CLEARTOP B)), entering the expression on the delete list of (PLANr:la).

P2's Resolve Conflicts critic notices a conflict between the denied expression (CLEARTOP B), in (PIAN:1a), and the PHANTOM node (CLEAR B). In effect, the expression (CLEARTOP B) is a resource that will be used by both processors during plan execution. ?2 will require temporary use of (CLEARTOP B), while PI will require permanent use (deletion). Therefore, the distributed planning system must allow P2's temporary use before granting P1's deletion. Resolve Conflicts establishes this ordering by synchronizing the distributed conjuncts, to insure that (PUT B ON C) will be executed before (PUT A ON B). This synchronization is performed by inserting a signalling action, (SIGNAL:2a), into the plan (Fig. 2c) and sending (WAIT:2a(DENY(CLEART0P B))) to PI. When PI receives this message it inserts a wait action, (WAIT:2a), preceding (PUT A ON B), as shown in Fig. le. *

During execution, (SIGNAL:2a) causes P2 to transmit a proceed message to P1. After transmitting the message, P2 continues execution. The effect of (WAIT:2a) is to suspend Pi's plan execution until a proceed message is received from PP. If the message has already been received when (WAIT:2a) is executed, plan execution resumes immediately.

In addition to (SIGNAL:2a), P2's Resolve Conflicts critic inserts a second MODEL node, (PLAN:lb), into the net (Fig. 2c). This node models those actions planned by P1 which will be executed following the (WAIT:2a)/(SIGNAL:2a) synchronization. The denied expression (CLEARTOP B) is copied from (PLAN:1a) into (PLAN:1b), indicating that it will occur after the synchronization. Temporally unrelated (incomparable) model nodes model actions to be executed by other processors in relation to actions in parallel portions of the local plan. Therefore, incomparable model nodes can overlap (even totally) in the actions they model.

P1 inserts a similar MODEL node, (PLAN:?b) (Fig. le), to model those actions planned by P2 which will be executed prior to the synchronization. There is an ambiguity on Pi's part as to when the denial of (CLEARTOP C) occurs in relation to the (WAIT:2a)/(SIGNAL:2a) synchronization. (PLAN:2b) represents those actions which must execute before the synchronization. (PLAN:2a) also represents those actions, as well as actions that are unsynchronized. Without additional information, P1 must leave (CLEARTOP C) on the delete list of (PLAN:2a), because that MODEL node is the least temporally specified. The expression is tagged to indicate the ambiguity.

• For expository ease, we assume that the distributed processors do not proceed to more detailed planning levels until all messages relating to the current level have been received. Although such coordination is not required, it does simplify the planning process. Planning activity without such level synchronization is described in [2].

** Such DENY messages can be eliminated by having each processor infer the changed world state from the high level subgoal specification of the other processor. In this way, only (transparent) effects which cannot be inferred need be communicated.

* In these problems, certain node names are identified by processor and an index (a, b, c, etc.). The same processor/index identification is coincidental—except with WAIT and SIGNAL actions which are paired using this identi fication.

P1 finds no additional criticism to perform at the third level, and expands the plan as shown in Fig. If. The MODEL nodes are copied into level four in the same manner as PHANTOM nodes are copied. P1's Resolve Conflict critic notices a (CLEARTOP C) conflict between (PLAN:2a) and (PUT C ON OBJECT:1a). Resolve Conflicts inserts (SIGNAL:1a) into the plan and sends (WAIT:1a(DENY(CLEARTOP C))) to P2 and places (CLEARTOP C) on the delete list of (PLAN:2c) (Fig. 1g).

From P2's viewpoint, its planning has been completed at level three. However, when it receives the message (WAIT:1a(DENY(CLEART0P C))) from PI, P? must insert (WAIT:1a) before the violating action (PUT B ON C). This is shown in Fig. 2d.

At this point, both planners have completed all expansions and criticism. The plan has been completely "linearized" (synchronized) as the block movement actions:
(PUT C ON OBJECT:1a)(PUT B ON C)(PUT A ON B).

A look at the completed plans (Figs. 1g and 2d) shows that (PLAN:2a), (PLAN:2b), and (PLAN:?c) model P2's action (PUT B ON C). (PLAN:1c) models P1's action (PUT C ON OBJECT:1a), and (PLAN:1b) models (PUT A ON B). (PLAN:1a) models both (PUT C ON OBJECT:la) and (PUT A ON B).

2.3 Distribution of NOAH's Global World Model

In the above problem, a complete and correct inital world model was assumed at both

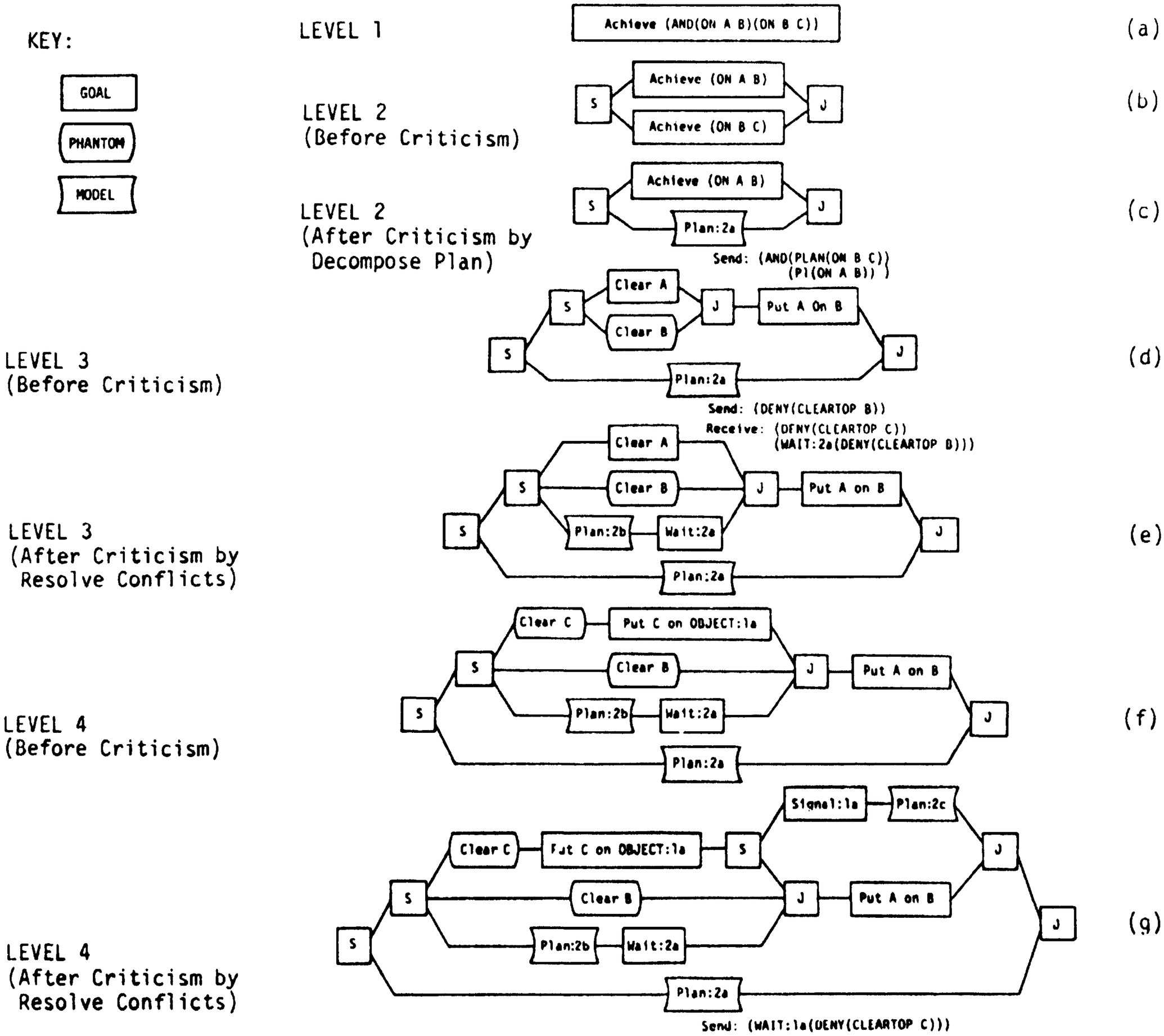


Figure 1. Three Blocks Problem: Processor P1

One approach to obtaining additional world model expressions is for each processor to announce (broadcast) relevant portions of the partial world model generated from its local sensory data to processors which are known to require this information during planning. Each processor then integrates the information it receives from other processors into its own partial world model. This integration may be as simple as taking the union of the world model expressions or as involved as a complete problem in distributed interpretation (such as described in [3]). This approach is appropriate when there is a good chance the announced expressions will be used by the receiving processor(s).

A second approach to obtaining additional initial world model expressions is to transmit requests for them. If a planner is unable to determine the value of an expression locally, it broadcasts to other processors a request for

2.4 Distribution of NOAH's "Distributed" World Model

Expressions whose values are changed during local planning are removed from the local initial world model, with the add or delete list of the action indicating the new values. Denied expressions, received from other processors, are handled in a similar fashion: the expression is deleted from the local initial world model and the new value indicated on the delete list of the appropriate MODEL node.

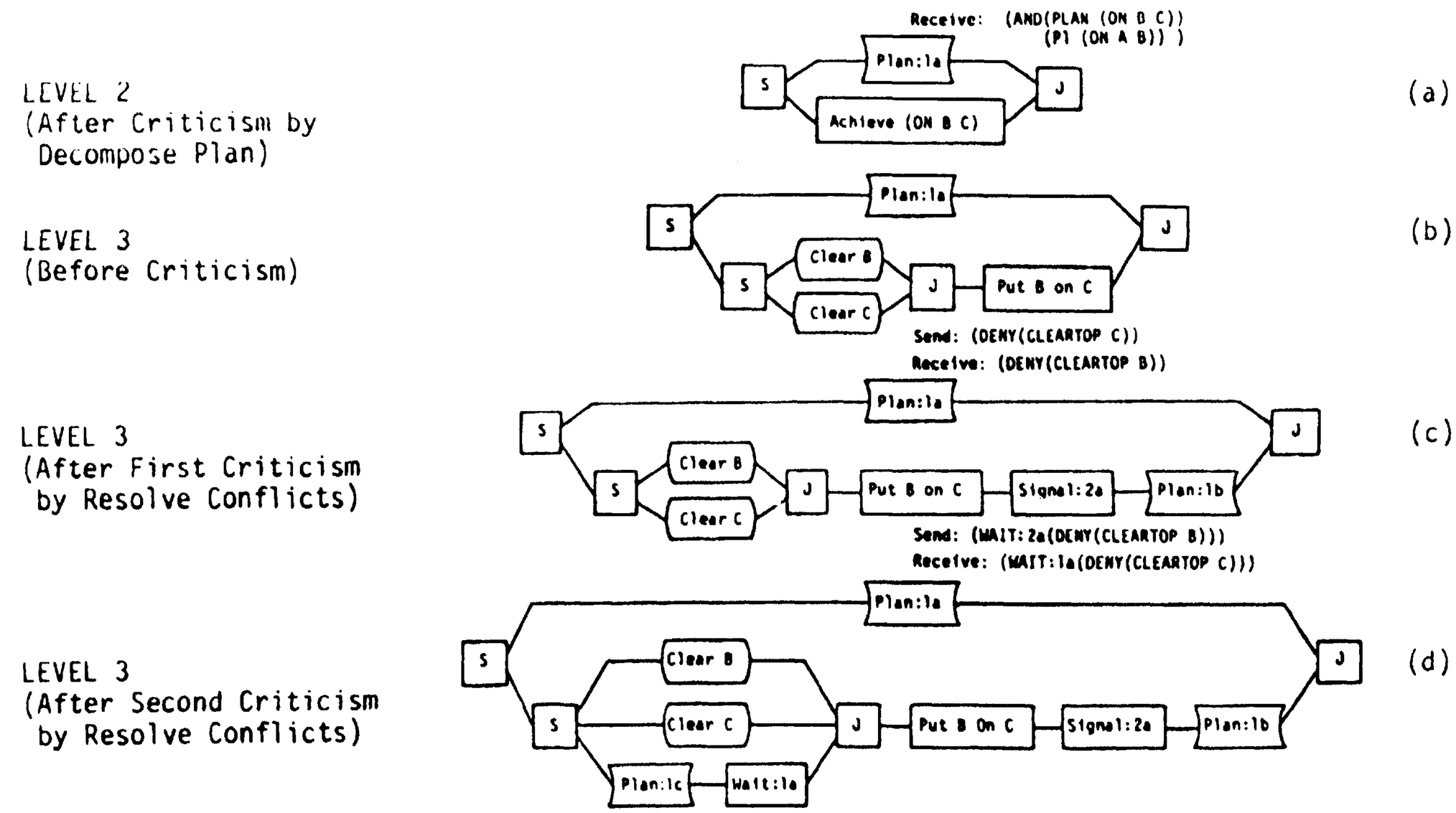


Figure 2. Three Blocks Problem: Processor P2

Spatial awareness of the planning areas of the various processors is also important in reducing the communication of expression changes during planning. If a processor knows that the areas of direct and indirect effects of its actions and those of another processor do not interact, there is no need to communicate the effects of its actions.

Synchronization by Resolve Conflicts leads to expression ambiguity in MODEL nodes and can require additional processor interaction in its resolution. In the problem of Section *2.2,* P1's Resolve Conflicts critic inserted (PLAN:2b) into the net as a result of the (WAIT:*2a*)/(SIGNAL:2a) synchronization. Planning then proceeded with the denied expression (CLEARTOP C) assumed to be in only (PLAN:2a). This was an incorrect assumption: the denying of (CLEARTOP C) actually precedes the (WAIT:2a)/(SIGNAL:2a) synchronization in P2*s plan and therefore also belongs in (PLAN:2b). Fortunately, the incorrect assumption did not effect the need for a second synchronization because (PUT C ON OBJECT:la) is temporally incomparable with both (PLAN:2a) and (PLAN:2b).

However, if P1 had an interacting action following the (WAIT :2a)/(SIGNAL:2a) synchronization, it would have been necessary to determine if (PLAN:2b) should, in fact, contain (CLEARTOP C) on its delete list. Without such a determination, P1 would generate an incorrect synchronization which would lead to an unnecessary double cross.

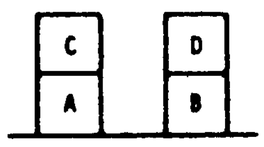
To handle these ambiguities, processors issue <u>conditional</u> requests for synchronization. In the problem, P1 would send the message:

(IF(AFTER(DENY(CLEARTOP C))
(SIGNAL:2a))
(WAIT:1a(DENY(CLEART0P C)))).

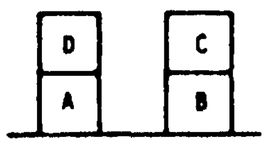
If (CLEARTOP C) is time-ordered after (SIGNAL:2a), then (WAIT:la) is inserted by P2. Otherwise, P2 returns the message (BEFORE(CLEARTOP C)(SIGNAL:2a)). Notice that the choice between conditional and unconditional synchronization requests is made locally by P1.

2.5 Distribution of Resolve Double Crosses

A second problem illustrates implementation of a distributed Resolve Double Crosses critic. The initial state and goal state for this problem are:



Initial State:
(ON C A)
(ON D B)
(CLEARTOP C)
(CLEARTOP D)



Final State:
(AND(ABOVE D A)
(ABOVE C B))

Processor P1 is given the goal state and expands the plan to level 2 (Figs. 3a) where (ACHIEVE(ABOVE C B)) is given to P2 (Fig. 4a).

Both processors expand the plan to level 3 (Figs. 3b and 4b). P1's Resolve Conflicts critic requests synchronization to protect the PHANTOM node (CLEARTOP D) (Fig. 3c). P2 does likewise to protect (CLEARTOP C) (Fig. 4c).

Once two temporally related (comparable) WAIT and SIGNAL actions occur in a plan, it is necessary to insure that they are not part of a double cross. Attempting to synchronize such a double cross leads to <u>deadlock</u> during plan execution. In this problem, P1 will not arrive at (SIGNAL:1a) because it is waiting at (WAIT:2a) (Fig. 3d). P2 will not arrive at (SIGNAL:2a) because it is waiting at (WAIT:1a) (Fig. 4d). To detect this deadlock situation during planning, * PI must determine if (WAIT:1a) is time-ordered before (SIGNAL:2a) in P2's plan. PI already "knows" that (SIGNAL:2a) is the last action in (PLAN:2c), that it cannot execute (WAIT:la) because of (WAIT:2a), and that no other processors can issue a (SIGNAL:la) proceed message. If (WAIT:la) is also in (PLAN:2c) there will be an execution deadlock.

P2 needs the analogous information about (WAIT:2a).

This ordering information is obtained by announcing the location of any wait actions that are time-ordered before a signal action. In this problem, P1 sends the message (BEF0RE(WAIT:2a)(SIGNAL:la)) and P2 sends the message (BEFORE(WAIT:1a)(SIGNAL:2a)). Once these messages are received, both processors detect the double cross (deadlock) and take steps to avoid it.

PI can determine locally (because WAIT:2a synchronizes the use of (CLEARTOP C)) that the denying of (CLEARTOP C) formed its contribution to the double cross. As in the centralized NOAH system, an examination of variable bindings shows that (ON C A) should not be true when (PUT D ABOVE A) is executed. PI inserts a new GOAL node, (ACHIEVE(NOT(ON C A))), into the plan in an attempt to eliminate the double cross. The (WAIT:2a) synchronization requested by P2 to protect the denying of (CLEARTOP C) is now inappropriate and is removed from the plan. The MODEL node (PLAN:2c) is merged into (PLAN:2a), and PI sends the message (UNDENY(CLEARTOP C) to P2. These actions are shown in Fig. 3e

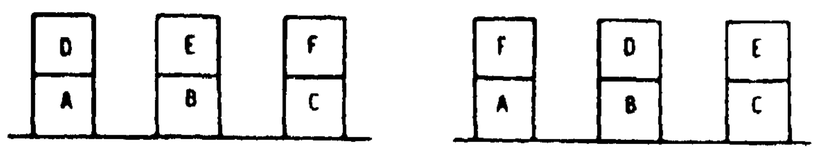
P2 proceeds similarly, as shown in Fig. 4e.

When the UNDENY messages are received, the superfluous signal actions are removed (and the appropriate MODEL nodes are merged) by both processors (Figs. 3f and '4f), and the plans are expanded to completion (Figs. 3g and 4g).

If more than two time-ordered synchronizations are required between processors, double cross (deadlock) detection may involve following chains of synchonization orderings. This may

* In the general case, deadlock detection is undecidable [5]. However, the loop- and condition-free nature of these plans allows for straightforward deadlock detection.

involve a combination of synchronization orderings and local linearization orderings. Consider the blocks world equivalent of a circular shifting of the contents of three registers:



Initial State:
(ON D A)
(ON E B)
(ON F C)
(CLEARTOP D)
(CLEARTOP E)
(CLEARTOP F)

Final State:
(AND(ABOVE F A)
(ABOVE D B)
(ABOVE E C))

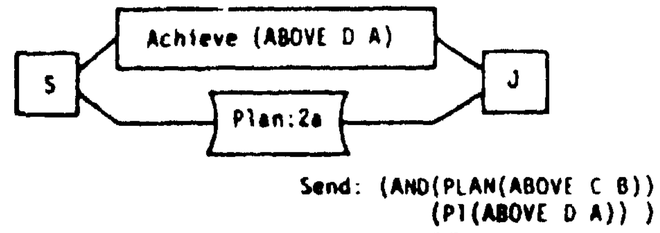
If three processors are each assigned one of the conjunctive goals, the following synchronization pattern develops:

P1: (WAIT:3a) ... (SIGNAL:1a)
P2: (WAIT:1a) ... (SIGNAL:2a)
P3: (WAIT:2a) ... (SIGNAL:3a).

Double cross detection by P1 requires the knowledge that (WAIT:1a) is time-ordered before (SIGNAL:3a). P3 requires the knowledge that (WAIT:2a) is time-ordered before (SIGNAL:1a). This problem can be generalized to a circular shifting of the contents of N registers.

No attempt was made to modify the double cross resolution mechanisms of the centralized NOAH system. Development of more sophisticated distributed double cross resolution techniques remains an important issue.

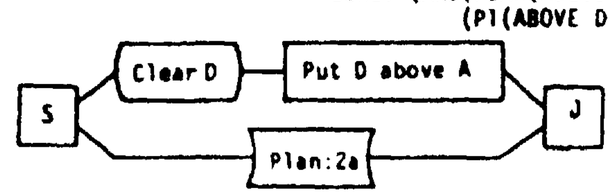
LEVEL 2
(After Criticism by Decompose Plan)



(a)

Send: (AND(PLAN(ABOVE C B))
(P1(ABOVE D A)))

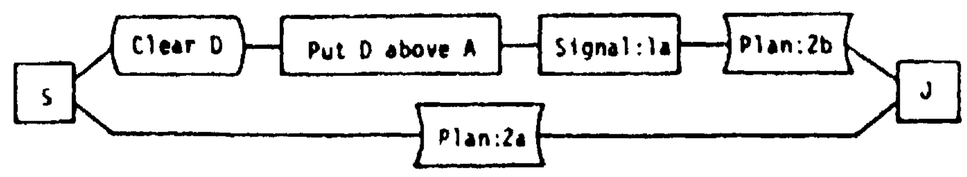
LEVEL 3
(Before Criticism)



(b)

Send: (DENY(CLEARTOP C))
Receive: (DENY(CLEARTOP D))

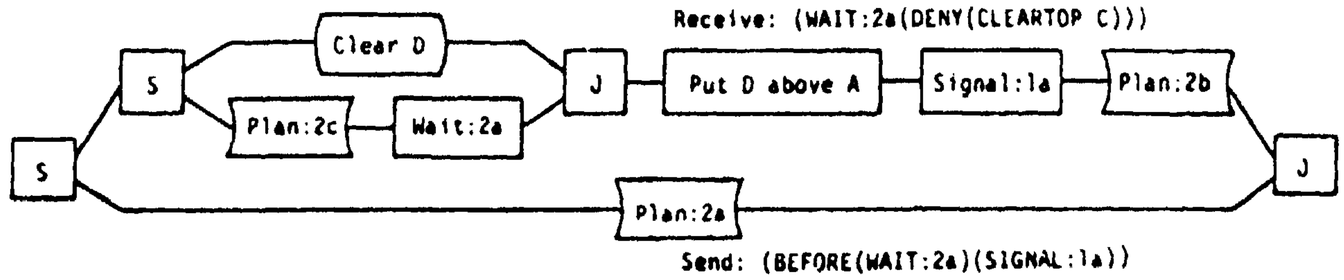
LEVEL 3
(After Criticism by Resolve Conflicts)



(c)

Send: (WAIT:1a(DENY(CLEARTOP D)))
Receive: (WAIT:2a(DENY(CLEARTOP C)))

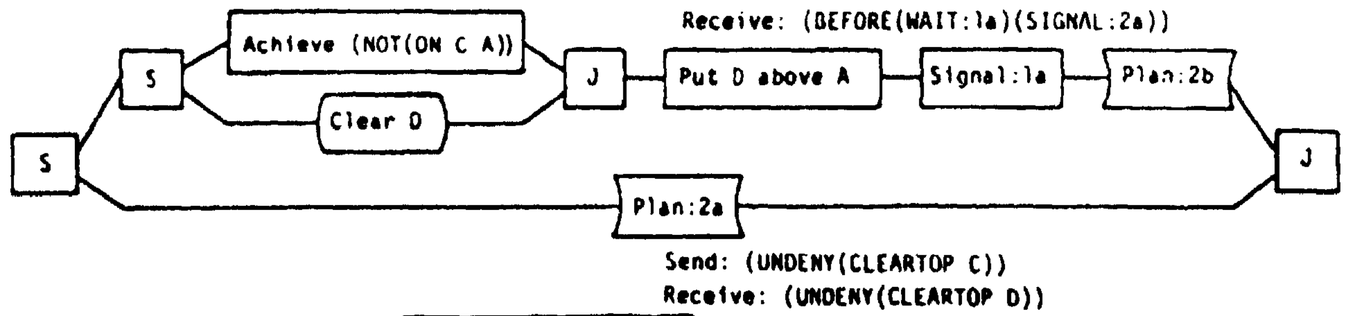
LEVEL 3
(After Second Criticism by Resolve Conflicts)



(d)

Send: (BEFORE(WAIT:2a)(SIGNAL:1a))
Receive: (BEFORE(WAIT:1a)(SIGNAL:2a))

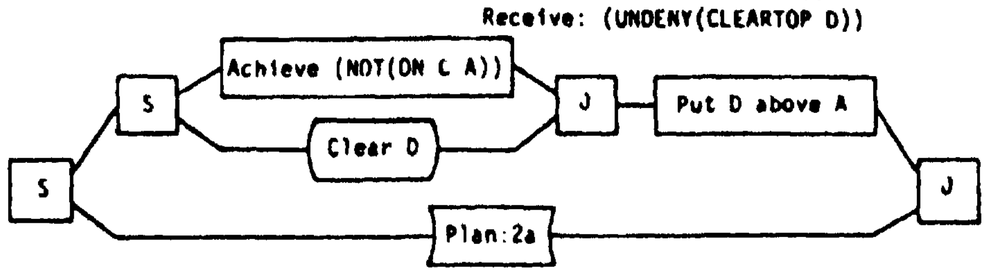
LEVEL 3
(After First Criticism by Resolve Double Crosses)



(e)

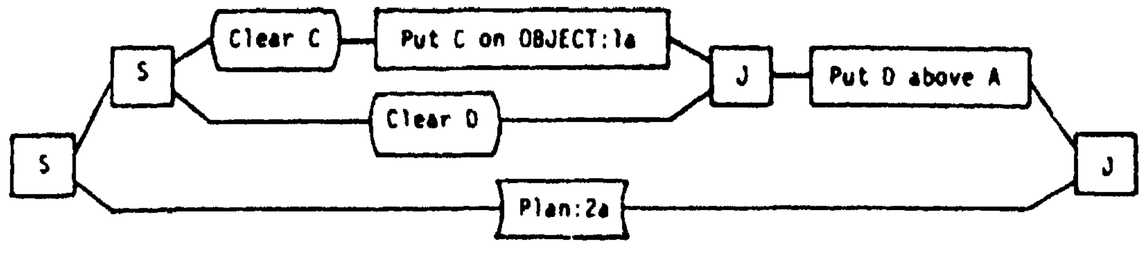
Send: (UNDENY(CLEARTOP C))
Receive: (UNDENY(CLEARTOP D))

LEVEL 3
(After Second Criticism by Resolve Double Crosses)



(f)

LEVEL 4



(g)

Figure 3. Swapping Blocks Problem: Processor P1

2.6 Distribution of Eliminate Redundant Preconditions

In order to detect redundant precondition achievement, it is necessary for processors to know of assertions made by other processors in a manner analogous to the use of denial information in conflict detection. Therefore, both denial and assertional messages are exchanged. PHANTOM precondition assertions, however, are not exchanged. Unless converted to GOAL nodes, interprocessor redundancies between PHANTOM nodes are not eliminated.

As in conflict resolution, ambiguity in the location of assertional expressions may exist and synchronization of actions may be required. A longer version of this paper [6] further describes the distributed Eliminate Redundant Preconditions critic.

2.7 Distribution of Use Existing Objects

The distributed Use Existing Objects critic analyzes all uninstantiated formal objects contained in the local portion of the plan, based on incoming assertional/denial information. If a formal object can be instantiated to achieve the same effects sent by another processor--and if the action involving the formal object can be time-ordered no later than the non-local action that achieves these effects--Use Existing Objects performs the instantiation and transmits a PHANTOMIZE request to the other processor.

As with Eliminate Redundant Preconditions, additional synchronization may be required to insure that subsequent actions by the other processor do not proceed until the instantiated action has been executed.

Figure 4. Swapping Blocks Problem: Processor P2

2.8 Additional Issues in Distributing NOAH

Integrating a distributed NOAH system into a complete distributed problem solver involves a number of additional issues: plan to processor allocation; planning versus execution-time criticism and synchronization; distributed detection of plan completion followed by plan execution versus integrated planning and plan execution; and distributed monitoring and plan modification. These issues are addressed in [?].

3. CONCLUSION

NOAH is suitable for generalization to a distributed planning system. The generalization requires introducing mechanisms for criticism and world model distribution.

ACKNOWLEDGMENTS

This paper is built on Earl Sacerdoti's milestone in planning: NOAH. The use of NOAH as a basis for distributed planning evolved during discussions with Victor Lessor and Nils Nilsson. Victor Lesser's comments on earlier drafts and during numerous discussions have been both useful and appreciated. John Lowrance, Scott Reed, and Jack Wileden also made helpful suggestions for this paper.

REFERENCES

[1] Sacerdoti, E.D. A Structure for Plans and Behavior. New York: American Elsevier, 1977

[2] Corkill, D.D. "Hierarchical Distributed Planning." Technical report, COINS, Univ. of Mass., Amherst (in preparation).

[3] Lesser, V.R., and L.D. Erman. "An Experiment in Distributed Interpretation.'' Technical Report CMU-CS-79-120, Computer Science Dept., Carnegie-Mellon Univ., May 1979.

[4] Smith, R.C. "A Framework for Problem Solving in a Distributed Environment." Ph.D. thesis, technical report STAN-CS-78-700, Computer Science Dept., Stanford Univ., Dec. 1978.

[5] Holt, R.C. "Some Deadlock Properties of Computer Systems." Computing Surveys, 4:3 (1972) 179-196.

[6] Corkill, D.D. "Hierarchical Planning in a Distributed Environment." Technical report, COINS, Univ. of Mass., Amherst, Feb. 1979-

PROPOSITIONAL ATTITUDESi FREGEAN REPRESENTATION AND SIMULATIVE REASONING

Lewis G. Creary
Artificial Intelligence Laboratory, Computer Science Department
Stanford University; Stanford, California 04306

ABSTRACTi This paper describes briefly a general notation that has been constructed for representing information about propositions! attitudes, and a method that is being developed for using this notation to reason about such Information. The notation Is Fregean in spirit, and Is based on McCarthy's first-order theory of individual concepts and propositions, extended to permit adequate representation of Iterated propositlonal attitudes, and with a semantically simpler representation of quantified propositions and "definite-description'' concepts of Individuals. The notation provides a systematic and Intuitively motivated way of dealing with apparent failures of extensionality and quantification Into Intentional constructions. The reasoning method Is knowledge-based, and uses a program's own ordinary inferential apparatus to "simulate" the reasoning of other intelligent organisms. The method thus avoids any need for redundant re-representation of Information about propositions at higher levels in the hierarchy of concepts, and provides a natural approach to making realistic inferences about the propositions! attitudes of others.

Topics and Key Words: Representation of knowledge, formal representations, propositional attitudes, Intentional contexts, reasoning about knowledge and belief.

Human beings are able (unconsciously) to represent and use effectively a wide range of information concerning people's beliefs, knowledge, preferences, and goals -- both their own, and those of others. Just how they manage to do this, however, is not at all well understood. Investigators In artificial intelligence, for example, have just begun to confront seriously some of the more complex and subtle problems Involved in explicitly representing and reasoning about such information. These problems Include those of I) dealing with apparent failures of extensionality, II) quantifying Into intentional constructions, iii) representing Iterated propositional attitudes, and Iv) making realistic inferences about the reasoning of Intelligent organisms. This *paper* discusses an approach to these problems that Is being developed by the author.* We shall Indicate the main desiderata that motivate our approach, describe briefly a general notation we have constructed for representing information concerning propositlonal attitudes, sketch a method that we are developing for the use of this notation in reasoning about such Information, compare our approach with some alternatives, and Indicate the present status and future direction of our research along these lines.

1 Desiderata

Largely because we desire a representation and reasoning system of potentially great generality, we adopt the following, somewhat idealized, criteria to guide the system's development:

a. The *primary* representation employed should be declarative, should have at least the logical power of a first order predicate calculus, and (with suitable choice of vocabulary) should be capable of representing almost any Information concerning propositlonal attitudes that a person could understand or express.

b. Reasoning sbout propositional attitudes should be based on separately represented knowledge concerning the organisms holding the attitudes; empirical assumptions should not be built Into either the top-level reasoning mechanism or the basic representation language.

2 Representing Propositlonal Attitudes

Our representational scheme has Its roots In the semantic theory of Gottlob Frege [8], and stems more directly from John McCarthy's recent use of Fregean ideas In the construction of a first-order theory of Individual concepts and propositions [McCarthy 1977, 1979]. Our language Is a modification and extension of McCarthy's. In order to use most efficiently our limited space here, In this section we shall first describe the philosophical framework within which our own representation language Is developed, then explain the main features of our language, and finally Indicate Its major points of difference from McCarthy's language and the reasons for them.

2.1 The Philosophical Framework

The basic Idea of our representational scheme Is that propositional attitudes are relations between people (or robots) and propositions, with both terms of the relation being taken as members of the domain of discourse. We regard propositions (and also concepts) as abstract entities, In much the same way as we do numbers, sets, etc. Unlike the latter, however, propositions (and concepts) are for us abstractions of things psychological; they are language-Independent (except for the language of thought) and person-Independent components of situations involving belief, knowledge, desire, and the like. Because It is possible for a person to discover that one of his beliefs Is logically equivalent to a proposition that he has never believed, we are led to reject logical equivalence as a criterion of Identity for propositions. Though this criterion is often adopted for logical purposes, our acceptance of such an unrealistic idealization would violate the second desideratum mentioned above.

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 2494, Contract MDA903-76-C-0206.

In general, we view the objective concepts that figure In our semantic theory as abstractions of possible subjective concepts, which are In turn possible configurations of Information and associated information-processing procedures In the memories of Individual organisms. Thus, organism O's having formed an objective concept C Is the same as O's having computed for itself a configuration of Information and associated procedures that Is an Instance of the abstraction C. Needless to say, details of the structures and contents of concepts, both objective and subjective, are very much In need of Investigation. This viewpoint is somewhat more psychologistic than anything that either Frege or even McCarthy has seemed willing to embrace. Nevertheless, we believe that this framework Is entirely appropriate as a basis for work in artificial intelligence and information-processing psychology, and expect it to be a fruitful source of heuristic insights for the development of technical details and the resolution of difficulties.

Since we take propositions to be composed of concepts in a structured way (and also to be themselves concepts of truth values), our representation scheme reflects this philosophical perspective: It Is only through the mediation of concepts that intentional attitudes bring those holding them Into relation with the (frequently non-conceptual) situations and objects with which those attitudes are concerned.

2.2 The Representation Language

With Frege, we distinguish the *sense* (intensional meaning, connotation) of an expression from its *denotation* (extension). Thus, the expressions "7+6" and "4x3" differ In sense but have the same denotation, since they both denote the number 12. In general, we take the sense of an expression to be a concept of appropriate type,* and In particular we follow Frege In taking the sense of a sentence to be the proposition (for him, Gedanke) that the sentence is used to express. For example, consider the formula:

1. wants(mike, Meet(Mike, Wife Jim»

   [i.e., mike wants to meet Jim's wife *as such.]*

Here, the subformula of the form "Meet<—>'' Is a complex name of the proposition that mike meets jim's wife, compounded of simple names of the relation-concept Meet, the function-concept Wife, and individual concepts of mike and Jim. The proposition named Is the sense of the formula "meet(mika, wife Jim)", while the denotation of this latter formula Is its truth value. In general, capitalized names denote concepts of the things denoted by the corresponding uncapltalized names. (In order to avoid confusion In the present discussion, we shall follow this rule even In the English sentences commenting on our formalism.)

It is well known that linguistic constructions describing proposltional attitudes often create contexts within which substitution of equals for equals, a standard rule of Inference for Identity, appears to break down. Thus, the sentences

2. mike wants to meet Jim's wife.
3. Jim's wife Is sally's mother,

might both be true, yet

4. mike wants to meet sally's mother.

might under the same interpretation be false because of mike's unawareness that Jim's wife is a mother. However, on a Fregean analysis, such apparent failures of extenslonallty turn out to be Illusory; the rule of substitution in question simply does not apply In such cases, and the Inferences it seemed to sanction turn out to be fallacies of equivocation. Thus, In our Fregean formalism we have (in addition to 1) the following formulas:

5. wife Jim = mother sally

   [I.e., Jim's wife is sally's mother.]

6. wants(mlke, Meet<Mika, Hothar Sally»

   [i.e., mike wants to meet sally's mother *as such.)*

Note that 6 does not follow from 1 and 6, even though the contexts Involved are all extensional. The reason is that "Wife Jim" denotes, not the person denoted by "wife Jim", but only a concept of her, so there is no way here to substitute equals for equals. This is as It should be, given our Initial interpretation of the sentences 2-4. However, It Is worth noting that there Is a second Interpretation of 4 under which It *can* reasonably be Inferred from 2, 3, and appropriate background Information, though the logical structure of the Inference may not Immediately be clear. This inference Is explicated In our present formalism as follows: From 1, 6, and the presumably true

7. conceptof(Wifa Jim, wife Jim)

   [I.e., Wife Jim is a concept of Jim's wife.]

we may easily deduce the conclusion

8. 3P.wants(mike, Meet<Mika, P» A
   concept of (P, mother sally)

   [i.e., mike wants to meet a specific person
   who Just happens to be sally's mother.]

In three steps by conjunction, substitution of equals for equals, and existential generalization. This conclusion Is our formalization of the second interpretation of 4; among other things, It shows the way in which our notation permits quantification into intentional contexts, and Illustrates the value of this kind of "quantifying In" as a representational technique.*

So far, our departures from [13] are relatively minor; the most significant differences arise only in connection with the representation of Iterated propositions! attitudes, a matter to which we now turn. As indications of the way In which our notation handles this phenomenon, we offer three quite different Interpretations of the ambiguous sentence:

0. pat believes that mike wants to meet Jim's wife.

* Our employment of the term "concept" in this context conforms with a fairly standard modern usage, but it should be noted that Frege's use of "Begriff", the German equivalent, Is quite different [Frege 1891, 1892a]. An excellent overview of Frege's semantics Is provided by the editor's introduction to [8].

* One can imagine circumstances in which the formula 8 would be clearly true, but In which the truth of the indicated English Interpretation of it might nevertheless be questionable. The issue involved Is whether the truth of 4 (on the second Interpretation) requires that mike be represented as having a concept *of a certain kind* of the person he wants to meet (for discussion of such issues, see [9]). However, the refinements of 8 that might thus be suggested would not take us outside the basic representational framework that we have indicated, and could best be developed in the context of an application of the framework to a specific class of reasoning problems.

(In the formulas to follow, "Mike$" denotes a concept of Mike, while Mike is in turn a concept of the person mike. Also, Mike$ is the *sense* of the concept-name "Mike", just as Mike is the *sense* of the personal name "mike". Analogous comments apply in the case of Meet$, Wife$, etc. The symbol "Exist" is an operator that binds concept-variables to form names of propositions.)

9.1 believes(pat, Wants(Mike,
Meet$(Mike$, Wife$ Jim$)))
[here, pat is represented as attributing to mike a desire to meet jim's wife *as such*.]

9.2 believes(pat, Exist P$.Wants(Mike,
Meet$(Mike$, P$)) And
Conceptof(P$, Wife Jim))
[here, pat is represented as believing that mike wants to meet a specific person (e.g., his daughter's piano teacher) who just happens to be jim's wife.]

9.3 ∃P$ P.believes(pat, Wants(Mike,
Meet$(Mike$, P$))) ^ conceptof(P$, P)
^ conceptof(P, wife jim)
[here, it is represented that the specific person whom pat believes that mike wants to meet just happens to be jim's wife.]

Still other readings of 9 are expressible in our notation, but we omit them here for the sake of brevity. The reader should note the relationship between the formulas 9.1 and 1, and between 9.2 and the result of substituting "wife jim" for "mother sally" in 8.

2.3 What Is New to Al Mert, and Why

The three readings just formalized provide a fairly stringent test of a notation's expressive power; the representational scheme of [13], for example, is Incapable of capturing all three. Since the contributions of our notation to representation theory He mainly In certain Improvements to McCarthy's scheme that are responsible for the expressive power just Illustrated in 9.1 - 9.3, we shall briefly outline the two main improvements, and Indicate why they are needed.*

2.3.1 A Hierarchy of Concepts

The addition of a notational system for a potentially Infinite, multlbranched hierarchy of concepts Is our major Improvement; It Is needed to enable McCarthy's language to adequately represent Information concerning iterated proposltlonal attitudes. The language of [13] provides for propositions and for concepts of Individuals, but not for concepts of propositions or for concepts of concepts of Individuals. However, It turns out that concepts of the letter two sorts are in fact required for the adequate treatment of constructions containing proposltlonal attitudes nested to a depth of 2, as Illustrated above In 0.1 - 0.3. Moreover, the obvious Inductive generalization holds for proposltlonal attitudes nested to depths of 3, 4, etc. Though the number of levels of concepts required Is thus In principle Infinite, In the vast majority of applications only a few of the lowest levels of the hierarchy will actually be used. The nature of our notational hierarchy is illustrated by the following initial sequence of formulas:

10a. teacher wife jim (a sentence)
10b. Teacher Wife Jim (a name of a proposition)
10c. Teacher$ Wife$ Jim$ (a name of a concept)
10d. Teacher$3 Wife$3 Jim$3 (")
10e. Teacher$4 Wife$4 Jim$4 (")

Our language, like McCarthy's, is embedded In a multi-sorted first-order logic with functions and Identity. However, because of the hierarchy of concepts, we make a more extensive use of sorts than McCarthy does. Thus, we have different sorts not only for Individuals, truth values, concepts of individuals, and propositions *(AS* McCarthy does), but also for concepts of concepts of Individuals, concepts of propositions, and so on up the hierarchy. This aspect of our language has much In common with Alonzo Church's logic of sense and denotation [Church 1951, 1973, 1974], though In other respects there are major differences from Church's development. Church's logic Is presently one of the most fully Investigated formal treatments of concepts and propositions in the literature, and his papers are a valuable source of Insights and technical Ideas for those concerned with the application or development of Fregean Intensional logic.

One aspect of our language development that has not yet received detailed attention Is the matter of avoiding antinomies, especially semantic ones. The main reason for this is our desire to maintain flexibility In the formulation of axioms, rules of Inference, and other features, as our language Is developed through a series of applications to specific reasoning problems in a computational setting. Unlike formal developments in pure logic, a language designed for use In artificial Intelligence must be convenient to apply and lend Itself to efficient computer implementation. Yet, It Is not possible to anticipate *a priori* what the optimal language organization will be for a given class of reasoning problems. When the lessons of developmental application have been learned, studies can then be undertaken to determine what technical means of avoiding antinomies Is most appropriate.

2.3.2 Semantically Simpler Concept Names

In order to provide for names denoting quantified propositions and "definite-description" concepts of Individuals, our language must contain some special-purpose name-forming machinery. McCarthy ([1979], pp. 13-16) has Introduced such machinery, but In order to avoid any extension of the underlying logic, he has resorted to a notational system whose semantics is rather obscure and cumbersome, requiring that both possible worlds and "propositions with inner variables in them" be Included in the domain of discourse. We believe that the needed conceptual names can be supplied more simply and naturally through a small expansion of the underlying logic that Is completely extensional, preserves its essentially "first-order" character, and Is semantically and syntactically transparent. What we have added for this purpose are three variable-binding, term-forming operators "All", "Exist", and "The", together with associated axioms. The operators "AH" and "Exist" form names of quantified propositions and thoir higher-level relatives in the conceptual hierarchy, as Illustrated above for "Exist" In the formula 0.2. The operator "The" forms names of "definite-description" concepts of individuals and their higher-level relatives. Thus, "The P.Child of (P, Mike>"

* A paper giving a much more complete and detailed account of our Improvements to McCarthy's representational scheme is currontly in preparation. It will Include precise statements of the syntax and semantics of the notations Introduced, and formulations of some aasoclated axioms. It will also Include discussion of some Improvements not mentioned here.

denotes the concept expressed by the phrase [H]the child of mike".

3 Reasoning about Propositlonal Attitudes

Our approach to reasoning about propositions! attitudes has its source In a familiar common-sense heuristic that people often use when thinking about the mental states of others: they ask themselves "What would I believe (or Infer, or feel) in that person's situation if I were psychologically very much like that person?" The Idea behind this heuristic plays a central role in our current efforts to construct an effective and natural computational method for reasoning about knowledge, belief, desire, and the like. In this section we shall indicate the lines along which our approach is being developed, discuss its relation to what has been called the "data base approach," and explain the sense In which it Is simulative.

3.1 Characterization of Our Approach

In accordance with the second desideratum noted earlier, our approach to reasoning is knowledge-based, with some of the required knowledge being represented as formulas, some as production rules, and some as LISP programs. The approach has these salient features:

a. A program's reasoning about the propositlonal attitudes of other organisms is facilitated by using the program's own "mental machinery" to "simulate" the thinking of these organisms. This avoids any need for redundant re-representation of information about propositions at the second and higher levels In the hierarchy of concepts, and, for example, permits logical relations among beliefs to be determined directly by the program's ordinary Inferential apparatus.

b. We require an access/maintenance system for an associatively indexed data base of logical formulas that has a CONNIVER-like context mechanism [16], and the ability to associate with each formula an indicator of its eplstemle status, together with the rules and other formulas on which this status Is based. The context mechanism is to be used both as an efficient way to keep track, for specific inferential purposes, of what Is (or Is not) believed by a given person, and as an aid In presumptively Including common knowledge and opinion In the corpus of each person's beliefs.

c. As a supplement to deductive rules of the usual sort, we make use of "inference by evaluation," a procedure whereby a sub-expression of the form "f(a1, a2, ... an)" is replaced by the result of evaluating It as a LISP expression.* Thus, "believes<mike, Uncle<JIm, Sally>>" may be an immediate consequence of "belleves(mike, sense "uncle(jim, sally)")". This use of content-dependent inference rules can lead to substantial efficiencies in the reasoning process.

To illustrate the Intended style of reasoning, let us consider how a system S would use it to determine whether or not mike believes that Jim is sally's uncle. Interest Is thus focussed on the formulas

11. believestmike, Uncle<Jim, Sally)), and

12. believes<mike. Not Uncle<Jim, Sally)).

Having determined (we suppose) that neither 11 nor 12 nor their negations *are* either explicitly stored in or trivially inferrable from Its knowledge corpus, S then "pretends" that it is mike, and tries to figure out what Its state of belief is concerning the proposition that Jim is sally's uncle.* It does this as follows: S computes from 11 the sentential formula

11 b uncle(jlm, sally),

and then constructs two contexts: C_{bm}, containing (sentential) formulas expressing the propositions that it initially thinks are believed by mike and relevant to 11.b, and C_{bm}, containing formulas expressing the propositions that It Initially thinks are not believed by mike and relevant to 11 b. In constructing these contexts, S is guided not only by specific knowledge about mike's beliefs, but also by general principles concerning what people commonly do and do not know and believe, and by an explicitly represented corpus of such common beliefs. Wherever needed, sentential formulas are computed directly from the names of the propositions they express, as In the case of 11.b.

At this point S sets up two tasks, T1 and T2. In T1, 11.b Is added to mike's belief-context C_{bm} and the resulting premise-set is taken as the starting point for an attempt to Infer either: i) a contradiction (in which case it might be concluded that mike believes that Jim is not sally's uncle), or ii) members of mike's non-belief context C_{bm} (In which case it might be concluded merely that mike does not believe that Jim is sally's uncle). In the companion task T2, the negation of 11.b is added to C_{bm} and a similar attempt Is made to infer either: I) a contradiction (in which case It might be concluded that mike believes that Jim Is sally's uncle), or ii) members of mike's non-belief context C_{bm} (in which case It might be concluded merely that mike does not believe that Jim is not sally's uncle). Measures of the resources expended by the inference process are computed for use In conjunction with knowledge about the level of mike's reasoning abilities; the indicated conclusions about mike's belief-state will actually be drawn if mike can reasonably be expected to see the logical relationships involved. At certain points in the reasoning, S may add a formula to C_{-bm} after determining that there is no reason (for mike) to believe the proposition expressed by the formula.

The tasks T1 and T2 are to be pursued in parallel until either: I) a complete characterization of mike's state of belief with respect to the proposition in question has been inferred, or ii) the resources allotted to the tasks are exhausted. In many cases, a good initial heuristic for S will be to try to attribute some minor variant of *S's* own state of belief concerning the proposition In question (and the reasons for It) to mike. The combinatorics inherent In these tasks are to be controlled by using a knowledge-based theorem prover of the general sort envisioned In [19], and by keeping the contexts involved fairly small, allowing them to grow larger only when the theorem prover indicates a need for It, and then only by Judicious choice of relevant propositions. The overall simulative process must be callable recursively, in order to permit reasoning about iterated propositions! attitudes.

* Essentially this mode of inference has been implemented in the "semantic attachment" feature of the FOL proof checker; see [20].

* We assume here that an organism believes that P If and only If It either has P explicitly stored as a belief that is readily accessible, or can easily Infer P from explicitly stored beliefs by a relatively routine inference procedure.

3.2 Relation to the "Data Bate Approach"

Our use of contexts, illustrated above, to represent subsets of beliefs and non-beliefs for Inferential purposes Is related to what Robert Moore has called the "data base approach" to representation and reasoning about knowledge, which he characterizes as follows:

> [An] ... Idea which Initially seems very appealing Is to use the multiple data-base capabilities of advanced Al languages to set up a separate data base for each person whose knowledge we have some information about. We then can record what we know about his knowledge in that data base, and simulate his reasoning by running our standard inference routines In that data base. This Idea seems to have wide currency In Al circles, and I advocated it myself in an earlier paper [16]. Unfortunately, it doesn't work very well. ([17], p. 224.)

Moore goes on to point out some difficulties with the "data base approach" that he evidently regards as casting doubt on its viability. We shall discuss this matter here, in order both to clarify the degree of our own Involvement with the data base approach, and to rescue the Idea behind its second part from any presumption of guilt by association with the quite different Idea behind its first part.

Within the data base approach as characterized by Moore, sets of formulas (data bases) are assigned two distinct functions: they are to serve both as a general means of representing facts about knowledge, and also as premise sets for simulative Inferential processes. However, it appears that the difficulties raised by Moore for this approach cause serious trouble only in relation to the first function. The two main difficulties concern the representation, using multiple data bases, of logical compounds (disjunctions and negations) of knowledge attributions [17], p. 224; [18], sec. *2.2).* These problems do not arise within our own approach in connection with the first of the two functions mentioned above (that of a general representation medium), since we use the Fregean formalism described In Section 2, *not* multiple data bases, to perform that function. We do make use of what are in effect small temporary data bases to perform the second of the two functions mentioned (that of premise sets), in the way Illustrated In Section 3.1. For this use of "data bases," however, the problems raised by Moore appear to be quite manageable.

One such problem concerns the use of separate data bases to represent what a given person does and does not believe. Moore (ibid.) Implies, without further explanation, that the problem of utilizing the information in a person's belief and non-belief data bases In a single simulative reasoning process Is not easily solved, The illustration in Section 3.1 is directly relevant to this problem, and shows that a solution lies In running the simulative theorem prover In the belief data base (context), while using the non-belief data base in an appropriate way as a source of goal statements for the theorem prover.

A second problem concerns the combinatorlally large number of possibilities for mike's knowledge state compatible with a description of it containing multiple disjunctions of knowledge attributions. The resulting uncertainty can require that many different knowledge contexts bo constructed and Investigated in order to draw useful conclusions about what mike has inferred from his knowledge. However, this difficulty is not peculiar to our approach to reasoning, but rather Is inherent In the given type of reasoning problem Itself. Our approach helps to mitigate such problems by using specific knowledge of the Inference task to minimize both the number and size of the contexts constructed.

3.3 Indirect Simulation

While the type of reasoning about belief Illustrated In Section 3.1 may *properly* be called "simulative," It is not *directly* simulative. That Is, conclusions about mike's beliefs a*re* not drawn simply by running a supposedly accurate simulator of mike and directly reading off the conclusion. Rather, an admittedly approximate simulator of mike is run, and then conclusions about his beliefs are *inferred* from from observed characteristics of the simulation, together with knowledge of the simulator and of mike. In this way the raw results of the simulation can be corrected on the basis of known differences between the simulator and mike. This sort of *indirect simulation* (as we shall call it) has the advantage that a single simulator can be used to learn about the beliefs of many different organisms. It has the obvious limitation that great differences between the processes of the simulator and those of the organism simulated can make it difficult to draw accurate conclusions.

4 Comparison with Some Alternative Approaches

As Indicated earlier, our Progean approach to the representation of propositional attitudes is closely related to that of McCarthy [1977, 1079]. With the Improvements to McCarthy's scheme discussed above in Section 2, our representational method goes a long way toward satisfying the first desideratum of Section 1, though much more experience with it is needed to gain a clear Idea of its limitations. Unlike straightforward modal representations of proposItlonal attitudes, a Fregean scheme permits the use of a fully extensional logic, and allows quantification over concepts and propositions, as In 9.3 and In the formalization of "All of mike's beliefs are well-founded."

When It comes to reasoning about proposItlonal attitudes, our approach Is quite different from the one currently taken by McCarthy, which relies heavily on principles relating statements about propositions! attitudes to the truth values of the proposItlonal objects of the attitudes In various possible worlds. While this "possible worlds" approach provides a powerful tool for systematizing reasoning about knowledge [McCarthy 1976; Moore 1977, 1979], In Its simplest form It is committed to some quite strong idealizations that would preclude its applicability in many practical situations, and that violate the second desideratum adopted above in Section 1. For example, the following principle plays a fundamental role in this approach:

> PW1: If a proposition P Is true In *every* possible world compatible with what a given person knows, then that person knows P.

This entails (most implausibly) that a person knows all the logical consequences of his knowledge. Moore [1977, 1979] acknowledges this difficulty In principle, and proposes to solve It by regarding the conclusions sanctioned by PW1 as only presumptive in nature, to be overridden by better supported Information to the contrary. However, his technical developments thus far take no account of this caveat, and Instead assume PW1 In Its simple, unqualified form.

A similar assumption can serve to simplify our simulative approach to reasoning. In Its crudest form, the assumption

is that an Intelligent program's own reasoning mechanism, used to Investigate the beliefs of another organism, is sufficiently accurate as a simulation that it requires no correction. In the terminology of Section 3.3, this amounts to assuming that such a program can use a *very* convenient form of *direct* simulation to Investigate the beliefs of others. This simplified simulative approach would be on a par with existing possible-worlds reasoning methods with respect to our second desideratum. However, we suspect that a simulative system could be more readily and naturally extended to take realistic account of the reasoning abilities of other organisms than could a possible-worlds system incorporating the principle PW1. The reason Is that, since PW1 makes no direct reference to logical Implication or inference, the most efficient revision might be to replace PW1 outright rather than to reinterpret it as a merely presumptive rule. Of course, the correctness of our suspicion can be decided only when extensions of the sort It envisions have been seriously attempted.

6 Further Research

Development of our approach Is presently proceeding on three fronts:

1. Epistemologlcal investigation to determine what knowledge and inferential procedures are appropriate for the solution of particular reasoning problems within the framework we have established.

2. Investigation of alternative designs for an Initial implementation of our Ideas in which all reasoning is deductive.

3. Studies leading to the design of a computational framework that Integrates deductive and non-deductive reasoning, and deals in a principled way with degrees of belief, *and* uncertainty of evidence and inference.

An Intermediate goal of our research Is the construction of a problem solver of the "advice taker" type [McCarthy 1960; McCarthy & Hayes 1969], to serve as a tool for the detailed study of knowledge, beliefs, goals, and normative precepts, as they function in planning and decision-making.

Acknowledgments

At various stages in the work reported here, I have benefitted from Interactions with Robert Filman, Reed LetsInger, John McCarthy, Robert Moore, Richard Weyhrauch, and an anonymous referee, but of course, they are not responsible for the result.

References

[1] Church, A. "A Formulation of the Logic of Sense and Denotation." *Structure Method and Meaning: Essays in Honor of Henry M. Sheffer.* Edited by P. Henle et al. New York: Liberal Arts Press, 1951.

[2] Church, A. "Outline of a Revised Formulation of the Logic of Sense and Denotation (Part I)." *Nous.* 7 (1973) 24-33.

[3] Church, A. "Outline of a Revised Formulation of the Logic of Sense and Denotation (Part II)." *Nous.* 8 (1974) 136-166.

[4] Frege, G. "Function und Begriff." Address given to the *Jenaische Gesellschaft fur Medicin und Naturwissenschaft.* January 9, 1891. Translated by P. T. Geach under the title "Function and Concept." [7].

[6] Frege, G. "Uber Begriff und Gegenstand." *Vierteljahrsschrift fur wis sense haftliehe Philosophie.* 16 (1892) 192-206. Translated by P. T. Geach under the title "On Concept and Object." [7].

[6] Frege, G. "Uber Sinn und Bedeutung." *Zcitschrift fur Philosophie und philosophische Kritik.* 100 (1892) 26-60. Translated by H. Feigl under the title "On Sense and Nominatum." *Readings in Philosophical Analysis.* Edited by H. Feigi and W. Sellars. New York: Appleton-Century-Crofts, 1949. Also translated by M. Black under the title "On Sense and Reference." [7].

[7] Frege, G. *Translations from the Philosophical Writings of Gottlob Frege.* Edited by P. Geach and M. Black. Oxford: Basil Blackwell, 1962.

[8] Frege, G. *The Basic Laws of Arithmetic. Exposition of the System.* Translated and edited by M. Furth. Berkeley and Los Angeles: Univ. of California Press, 1967.

[9] Kaplan, D. "Quantifying In." *Words and Objections: Essays on the Work of W. V. Qitine.* Edited by D. Davidson and J. HIntikka. Dordrecht, Holland: D. Reidel Publishing Co., 1969. Reprinted in *Reference and Modality.* Edited by L. LInsky. London: Oxford University Press, 1971.

[10] McCarthy, J. "Programs with Common Sense." *Proceedings of the Symposium on Mechanisation of Thought Processes.* Teddington, England: National Physical Laboratory, 1969. Reprinted In *Semantic Information Processing.* Edited by M. Minsky. Cambridge, Mass.: The MIT Press, 1968.

[11] McCarthy, J. "An AxiomatIzation of Knowledge and the Example of the Wise Man Puzzle." Unpublished memorandum. 1976.

[12] McCarthy, J. "Epistemologlcal Problems of Artificial Intelligence." *Proceedings of the Fifth International Joint Conference on Artificial Intelligence.* Cambridge, Mass.: 1977. 1038-1044.

[13] McCarthy, J. "First Order Theories of Individual Concepts and Propositions." Artificial Intelligence Laboratory Memo AIM-326. Stanford University, March 1979. Also forthcoming In *Machine Intelligence* 9. Edited by D. Michle. 1979.

[14] McCarthy, J., and Hayes, P. "Some Philosophical Problems from the Standpoint of Artificial Intelligence." Artificial Intelligence Laboratory Memo AIM-73. Stanford University, November 1968. Also In *Machine Intelligence* 4. Edited by D. Michle. New York: American Elsevier, 1969.

[16] McDermott, D., and Sussman, G. *The Conniver Reference Manual,* Ai Lab Memo 269. Massachusetts Institute of Technology, May 1972.

[16] Moore, R. C. "D-SCRIPT: A Computational Theory of Descriptions." *Advance Papers of the Third International Joint Conference on Artificial Intelligence.* Stanford, Calif., 1973. 223-229. Also In *IEEE Transactions on Computers.* C-26:4 (1976) 366-373.

[17] Moore, R. C. "Reasoning About Knowledge and Action.*' *Proceedings of the Fifth International Joint Conference on Artificial Intelligence.* Cambridge, Mass., 1977. *223-227.*

[18] Moore, R. C. *Reasoning About Knowledge and Action.* Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, (forthcoming).

[19] Nilsson, N. *A Production System for Automatic Deduction.* Computer Computer Science Department Report No. STAN-CS-77-618. Stanford University, July 1977.

[20] Weyhrauch, R. *A Users Manual for FOL.* Artificial Intelligence Laboratory Memo AIM-236.1. Stanford University, July 1977.

OUANTIFICATION IN A THREE-VALUED LOGIC FOR NATURAL LANGUAGE OUESTION-ANSWERING SYSTEMS

Veronica Dahl
Professor
Departamento de Matematica
Facultad de Ciencias Exactas
1428 Buenos Aires, Argentina

This paper examines quantification in relation to a typed three-valued logical system which was developed to serve as the internal query language for virtual data bases accepting natural language (NL) consultation. Such a svstem is capable, among, other things, of reflectinp certain NL presuppositions, handling relations among sets and coping with certain NL ambiguities, within a simple though natural and fairly vast NL subset.

In our approach, quantification is dealt with through a single mechanism by which all NL quantifiers introduce the formula "those(x,p)", denoting the set of all those x's in x's associated domain which satisfy statement p. The meaning of each particular NL quantifier— including any presuppositions it may induce— is rendered through particular constraints upon the set (e.g., cardinality constraints).

1. INTRODUCTION

The idea of using Logic as a conceptual framework in question-answering systems is not a new one. The fact that it can formally deal with the notion of logical consequence makes it particularly attractive to represent meaning.

Nevertheless, efforts in this direction had been more or less abandoned in Artificial Intelligence research, mainly because standard predicate calculus (PC) has been shown to be unsuitable for an accurate representation of NL. Its main inadequacies seem to relate to a general inability to incorporate relevant semantic features into the logical representation of a sentence.

Recent linguistic research ((1] ,[2]) has arrived at interesting results concerning the extension of standard PC in order to provide a better formal model of language.

It is our thesis that research in this direction, together with the fact that programming in logic ([3] ,[4]) has become possible since the development of the PROLOG programming language [5], is likely to yield some very good results concerning NL processing.

On the other hand, the evolution in data base technology has been drifting more and more towards the use of logic, both for data description and for queries I 6] . Consequently, it is tempting to explore the possibilities of using logic throughout a NL data base system, in the hope of reducing interfaces to a strict minimum.

These considerations have led us to develop a rigorous logical svstem with a clearly defined semantics, which has been successfully used as the internal query language into which a computer translates NL input in order to consult different data bases (| 7] ,| 8] ,(9] ,[10]) .

In our approach, a data base is actually an interpretation according to which a well-formed formula in our logical system is evaluated. Its evaluation directly produces the answer, since it yields either a truth value, which corresponds to a yes-no question, or the extensional representation of a set, which cor responds to a wh-question.

This paper focusses on our treatment of quantification within this logical system. The most important theoretically related work is [1] ,[21 and [11) . Related computer systems are mentioned throughout the paper.

Section 2 discusses the need of typing variables in connection with meaningfulness and ambiguity solving. Section 3 motivates our approach to quantification and the need of three logical values in connection with existential and number presuppositions, and then describes

the process of translating NL quantifiers into logical ones. Section 4 discusses the results obtained in concrete applications.

For space limitations, we are forced to omit a formal and complete definition of our logical system. Those of its features that are relevant to our subject, however, are intuitively described in Section 3.

Spanish is used throughout the paper as the concrete point of reference with which to exemplify our theoretical considerations.

2. MEANINGFULNESS, AMBIGUITY AND SEMANTIC TYPES"

A NL processing system must have a means for checking semantic as well as syntactic accord, in order to reject anomalous sentences such as "Which dogs speak Latin?".

A widely spread solution to this problem consists in first generating a "deep structure" *of* the sentence, taking only syntax into account, and then performing all the necessary semantic operations and checkups on it.

As has already been observed [12], this often implies a tradeoff between syntactic and semantic complexity, when it is overall simplicity and efficiency that are important.

Moreover, linguists themselves are not unanimous as to whether the semantic component should be separate or intermingled with the syntactic one | 13).

Where logic is concerned, there is a simple and elegant way of dealing with meaningfulness: by using types. Types, by the way, are also a useful means for associating the universe of PC to the relations in a particular data base. If we associate a type to each data base domain, and model the relations in a data base through predicates whose arguments are restricted to values in specific domains, the analyser can check that the arguments in each predicate generated by an input sentence are of the expected type.

Incidentally, types are also useful to improve efficiency: a) by narrowing the search space, since only those values in a variable s associated domain need to be considered, and b) by avoiding sterile data base consultations, since absurd queries can be rejected by the analyser on the grounds of domain incompatibility.

Types have been proposed by other authors (e.g. [14] ,[15] ,[16] ,[17]) in connection with efficiency and meaningfulness, but they have still another interest: they allow to automatically solve certain NL ambiguities. Take for instance the query:

tCual es el salario del empleado que vive en Lomas?—Which is the salary of the employee who llves in Lomas?

A priori, a machine has no way to decide whether the antecedent of the relative clause is "the salary of the employee" or "the employee". But in a type-checking system in which the first argument of the relation "live" is associated to the HUMAN domain, and in which employees— and not salaries— are known to belong to this same domain, the first alternative is not even possi b1e.

Ambiguities concerning different aceptions of a word can generally also be. solved through domain checking. This is particularly true in our implementation, in which variables are associated to a type contextually, during the analysis of an input sentence (of. Section 3.1).

3. PRESUPPOSITIONS, QUANTIFIERS AND A THREE-VALUED LOGIC

Typed calculus in itself is not enough to make all sentences meaningful. We actually need more than two logical values, since in NL there are two ways in which a statement may fail to be true: either because its negation holds, or because something presupposed by the statement fails to be satisfied. In the latter case, the statement is felt to be pointless rather than false.

There is another reason why it must not be considered false. Take for instance the statement:

El sombrerero loco odia a Alicia.
The mad hatter hates Alice.

Tn a context in which no hatter is mad, it is obviously not true. However, we can not either consider it false, since then the statement

El sombrerero loco no odia a Alicia.
The mad hatter does not hate Alice.

would have to be considered true.

The inexistence of a referent for the definite noun phrase makes the whole sentence meaningless. The existence of more than one referent

would also make it meaningless, since we could not tell which of the referents the sentence is talking about. This is because the Spanish singular definite article induces a presupposition of existence and unicity upon the noun phrase's referent.

Our treatment of quantification has been devised to account for those presuppositions induced by NL quantifiers. We prefer to call them "determiners", since they include all articles, cardinal numbers and words such as "some", "many", etc.

If a sentence contains a determiner, a quantification of the form "those(x,p)'' is introduced, where x is a typed variable and p is a logical formula in our system. Its evaluation yields the set of all x's in x's associated domain which satisfy p.

According to the determiner's meaning, number and existence presuppositions on such a set are represented within the output formula. For instance, "Three blind mice run" is represented as

card(those(x,and(and(mice(x),blind(x)),run(x))),3)

(the cardinality of the set of those blind mice who run is 3)

Definite articles introduce the formula "if(fi , fi)", whose value is "undefined" whenever f_i fails to be satisfied, and has the same value as f_2 if f_1 is true. Here is an example, using the easier-to-picture tree representation:

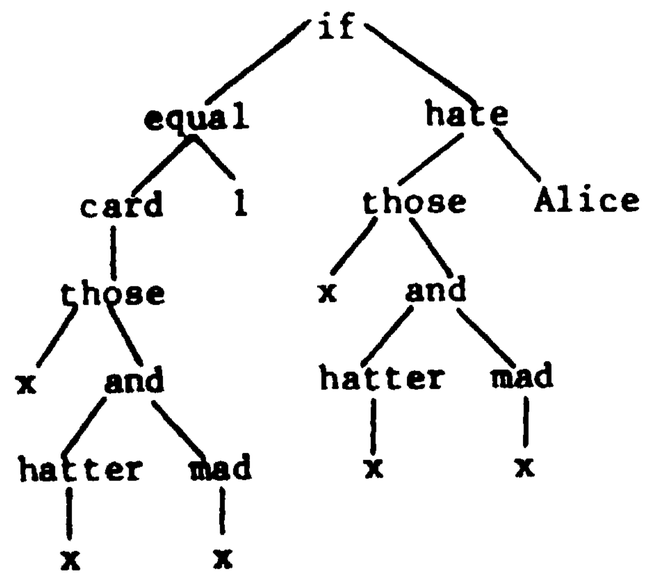


Fig. 1 Representation of the sentence "The mad hatter hates Alice"

The formula represented in Fig. 1 will evaluate to "undefined" if the set of mad hatters does not contain exactly one element.

3.1 Spanish determiners and their translations

We can now examine the general process by which a determiner introduces a "those" formula. Let us consider a sentence consisting of a noun phrase followed by a verb phrase, in which the noun phrase contains a noun introduced by a determiner. We can first represent it through a three-branched quantification of the form:

$$q(x,f_1,f_2)$$

where q is a quantifier into which the determiner translates, x is a typed variable, f_i is the noun phrase's translation, and f_2 is the verb phrase's translation. Intuitively, f_i specifies the domain of quantification, and q states what portion of the domain f_2 holds for.

Our previous example, for instance, can first be represented:

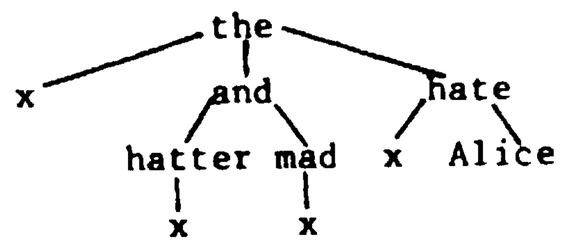


Fig. 2 A first representation of "The mad hatter hates Alice"

In our implementation, a variable's associated domain depends both on f_1 and f_2 in the following manner: each predicate known to the data base system has a domain associated to each of its arguments. For instance, "mad" and "hatter" could require its arguments to belong to the HUMAN domain, while "hate" could require its first argument to belong to the ANIMAL domain.

When a three-branched quantification is generated, the variable it creates is typed by the intersection of all those domains it has been associated with by the predicates appearing in either f_1 or f_2. In our example, x's type would be HUMAN (the intersection between the HUMAN and the ANIMAL domains).

Instead of generating a different quantifier for each determiner, it is useful to represent all quantifications through a single one of

the form

para(x,p,c)
for

whose intuitive meaning is: "c holds for the set of all x's in x's domain which satisfy p".

Each quantification is thus assigned an equivalent "for" expression, in which the determiner's meaning is represented. Here are the representations of the Spanish determiners "un" (a), "unos" (some), "el" (singular the), "los" (plural the), "ningún" (no) and "i" (any cardinal number). The rest are considered in Section 3.1.2.

un(x,f_1,f_2)=unos(x,f_1,f_2)=for(x,and(f_1,f_2),greater-than(card(x),0))

el(x,f_1,f_2)=for(x,f_1,if(equal(card(x),1),f_2))

los(x,f_1,f_2)=for(x,f_1,if(greater-than(card(x),1),f_2))

ningún(x,f_1,f_2)=for(x,and(f_1,f_2),equal(card(x),0))

i(x,f_1,f_2)= for(x,and(f_1,f_2),equal(card(x),i))

Note that we have chosen to identify "un" (a) and "unos" (some). This is because "un" is frequently used to denote "at least one". In order to avoid ambiguity, "1" should be used to mean "exactly one". This convention is particularly useful when negation is involved. For instance, "No tengo un centavo" (I have not a cent) would have a wrong representation with the "exactly one" interpretation: it would state that the cardinality of the set of cents I possess is not 1, which means it can either be 0,2,3,etc.

Finally, any formula of the form "para(x,p,c)" can be replaced by just the formula c, in which all occurrences of x have been replaced by the formula:

aquellos(x,p)
those

representing the subset of x's domain whose elements satisfy p.

The reader can now verify that the representations shown in Figs. 1 and 2 are equivalent.

3.1.1 Quantifier hierarchy

Quantifier hierarchy obeys three perhaps too simplistic rules, but which have proved to be useful within our NL subset *.

- Rule 1: A determiner in a verb's subject introduces a quantification which dominates all quantifications introduced by the verb's complement(s). For instance, "Toda rosa tiene (algunas) espinas" — Every rose has (some) thorns— is represented:

toda(x,rosa(x),algunas(y,espinas(y),tiene(x,y)))
every rose some thorns has

Note that the representation

some(y,thorn(y),every(x,rose(x),has(x,y)))

would be incorrect, since it rather means: "There exists a certain set of thorns which every rose has".

- Rule 2: Whenever a noun has a complement, the quantification introduced by the complement's determiner dominates the one introduced by the noun's determiner. For instance, "Sabato autografia el libro de cada visitante"— Sabato autographs the book of each visitor— is represented:

cada(x_2,visitante(x_2),el(x_1,libro-de(x_2,x_1),autografía(Sábato,x_1)))
each visitor the book-of
autographs

- Rule 3: When a referential word (that is, a verb, a noun or an adjective) has more than one complement, quantification takes place from right to left: the rightmost complement generates a quantification which dominates the quan-

* As a formal characterization of our NL subset is beyond the scope of this paper, we shall informally state that it has a fixed vocabulary—determiners, prepositions, the conjunction "y" (and), relative and interrogative pronouns and the word "no"; and a variable vocabulary—nouns, proper names, adjectives and verbs in the third person of a simple tense . Sentences are either declarative or interrogative, in the active voice, and restricted relative clauses are allowed and can be nested with no other limit than the computer's capacity.

tification(s) introduced by the leftmost complement(s). For instance, "Raúl regala un espejo a cada niño"-- Raúl gives a mirror to each child-- is represented:

cada(x_2,niño(x_2),un(x_1,espejo(x_1),regala(Raúl,
each child a mirror gives
x_1,x_2)))

3.1.2 Determiners with a negative implication

As a general rule, the negation introduced by "no" in a sentence is translated by placing the operator "no" (not) right after the quantification introduced by the subject. For instance, "Los chicos no conocen al padre de Martín" (The children do not know Martín's father) is represented:

los(x_1,chicos(x_1),no(el(x_2,padre-de(Martín,x_2),
the children not the father-of
conoce(x_1,x_2))))
know

But negation is not always explicit. The Spanish determiner "ningún" (no) can be regarded as an implicit negation, since it expresses that no portion of the domain of quantification satisfies the statement involved.

In a non-inverted subject position (e.g., "Ningún elefante vuela"-- No elephant flies--), it generates a special quantifier called "ningún", whose representation takes this fact into account (see Section 3.1).

There are two other cases, however, in which "ningún" coexists with an explicit negation. These cases require a different quantifier, since otherwise the negation would be represented twice. These cases are:

a) In a subject position, plus subject-verb inversion: the "ningún" determiner is assimilated to the "every" quantifier. For instance, "No vino ningún hombre"-- No man came-- is represented:

todo(x,hombre(x),no(vino(x)))
every man not came

b) In a position other than the subject: the "ningún" determiner is assimilated to the indefinite article's quantifier. For instance, "Carlos no tiene ningún hijo"-- Carlos has not any child-- is represented:

no(un(x,hijo(x),tiene(Carlos,x)))
not a child has

Another special case is the negation preceding the "todo" (every) determiner (e.g. "No todo pájaro canta"-- Not every bird sings--). The analyser considers "no todo" as a single determiner generating its own associated quantifier:

no-todo(x,f_1,f_2)=for(x,and(f_1,not(f_2)),greaterthan(card(x),0))

4 SOME PRACTICAL CONCLUSIONS

The ideas previously discussed have been confronted with several successful PROLOG minicomputer implementations of NL consultable data base systems, in which NL analysis is performed through a metamorphose grammar [18] , a powerful PROLOC tool for defining context-sensitive rewriting rules.

The first one concerned just one particular application: the representation of the hardware and software catalogues for the SOLAR 16 series of computers, in order to generate, from a user's question in French, the characteristics of a configuration suiting his particular needs ([7] ,[8]).

Our first conclusions were reinforced and improved by further research concerning a more general data base system implemented both for Spanish and French consultation ([9] ,[10]).The Spanish version of this system has been adapted to Portuguese by H. Coelho and L. Pereira (personal communication, 1978). Their work confirms our expectations regarding the generality of our ideas— and of our implementation—, since remarkably few modifications of the program proved necessary, and they were strictly related to lexical differences, whereas rules concerning structure remained unchanged.

Similar ideas to the ones discussed above concerning quantification have inspired other NL data base systems, namely LSNLIS [19] and PHLI0A1 ([17] ,[20]).But in spite of the points in common, our general approach is markedly different. We have tried to incorporate all relevant semantic as well as syntactic NL features into a single formalism, in order to do without intermediate sublanguages and have a single process perform the analysis of an input sentence. LSNLIS, on the contrary, first generates deep structures and then maps them into a semantic representation. PHLIQA1 has several successive levels of semantic analysis, each requiring a special formal language. Some of them are meant to deal with ambiguity, which in our approach, as we have seen, is dealt with through the contextual typing of variables during the quantification process.

Another important point in our approach, which for reasons of space we must be content with just stating, is a high degree of formalization through logic: because of the use of PROLOG, important theoretical aspects such as a rigorous characterization of our NL subset and of the syntax and semantics of our internal query language need not be dissociated, as is generally the case, from those practical aspects concerning the implementation.

From the performance point of view, no frustrating delays in answering have been encountered so far in the systems we have implemented, and the conciseness obtained is rather great: the complete listing of our Spanish analyser, including the vocabulary associated to a small sample data base, holds in just four pages.

Among our limitations, let us mention the fact that we have not attempted to use NL for data base creation or modification, and that interrogation is limited to the context of one isolated sentence which can not refer to other previous questions or answers.

ACKNOWLEDGEMENTS

The research partially reported here was conducted at the Aix-Marseille II University, under the supervision of A. Colmerauer.

REFERENCES

[1] Keenan, E. L. "On semantically based grammars". Linguistic Inquiry, 1972.

[2] Hausser, R. "Ouantification in an extended Montague grammar". Dissertation, University of Texas at Austin, 1974.

[3] Kowalski, R. A. "Logic for problem solving". DCL Memo 75, Department of Artificial Intelligence, Edinburgh, March, 1974.

[4] Van Emden, M. H. "Programming with resolution logic". Report CS-75-30, Dept. of Computer Sciences, Univ. of Waterloo, November, 1975.

[5] Roussel, Ph. "Prolog: manuel de reference et d'utilisation". GIA, Univ. d'Aix-Marseille II, September, 1975.

[6] Gallaire H. and Minker J (eds.) Logic and data bases. Plenum Publishing Corporation, N.Y., 1978.

[7] Dahl, V. and Sambuc, R. "Un systeme de banques de donnees en logique du premier ordre, en vue de sa consultation en langue naturelle". DEA report, GIA, Universite d'Aix-Marseille II, October, 1976.

[8] Colmerauer A., Dahl V. and Sambuc, R. "Consultation en francais d'un catalogue industriel: configuration de la serie d'ordinateurs SOLAR 16". Final report of contract BNIST N'291767, GIA, Univ. d'Aix-Marseille II. To appear.

[9] Dahl, V. "Un systeme deductif d'interrogation de banques de donnees en espagnol". These de Doctorat de Specialite en Intelligence Artificielle, GIA, Univ. d'Aix-Marseille II, November, 1977. .

[10] Dahl, V. "Some experiences on natural language question-answering systems". Preprints of the international workshop "Logic and data bases", CERT, Toulouse, November 1977, XXVIII, pp. 1-9.

[11] Colmerauer, A. "An interesting natural language subset". Preprints of the international workshop "Logic and data bases", CERT, Toulouse, November, 1977, XXIX, pp. 1-32.

[I2]Petrick, S. R. "On natural language based computer systems". IBM Journal of Research and Development, July, 1976.

[13]Lakoff, G. and Ross, J.R. "^Es necesaria la estructura profunda?" In Semantica y sintaxis en la linguistica transformatoria". Alianza Editorial de Madrid, 1974, pp. 226-231.

[14] Mc. Skimin, J. and Minker, J. "A predicate calculus based semantic network for question-answering systems". Preprints of the workshop "Logic and data bases", CERT, Toulouse, November, 1977, VI,pp. 1-37.

[15] Pirotte, A. and Wodon, P. "A comprehensive formal query language for a relational data base: FQL". R.A.I.R.O. Computer Sciences, 2:2,1977.

[16] Reiter, R. "Deductive question-answering on relational data bases". In: [6] .

[17] Scha, R. "Semantic types in PHLI0A1". In Proc. 6 International Conf. on Computational Linguistics, Ottawa, 1976.

[8] Colmerauer, A. "Grammaires de metamorphose". To appear in: Natural language communication with computers. Lecture Notes in Computer Science, Springer Verlag.

[19] Woods, W.A. , Kaplan, R.M. and Nash-Webber, B. "The Lunar Sciences Natural Language Information System: Final report", BBN Report 2378, Bolt Beranek and Newman, Inc, Cambridge, Mass, 1972.

[20] Landsbergen, S.P.J, and Scha, R.J.H. "Formal languages for semantic representation", M.S. 10.259, Philips Research Laboratories, Eindhoven, Holland.

BROWSING IN URGE DATA BASES

Douglas D. Dankel II
Assistant Professor
Computer and Information Sciences Department
university of Florida
Gainesville, FL 32611

The formation of data bases containing information on mechanical systems for troubleshooting purposes has become increasingly popular and important. Examination of these data bases by humans can be very costly. A system called BROWSER was developed to heuristically search a data base containing information on U.S. Navy aircraft with little or no human intervention. BROWSER searches the data base guided by models and heuristics looking for interesting patterns or configurations. The user is then notified of the existence of these patterns.

1. INTRODUCTION

Consider the tasks of trying to find similarities between the causes of the crashes of several aircraft of a common type, the causes or early warning signs for various diseases, the characteristics of stocks with high growth potential or advanced warning signs of severe weather conditions. All of these tasks require a large amount of data and valuable time spent sifting through the data. Browsing or troubleshooting computer systems of the future will eventually perform these tasks. BROWSER is designed to be the prototype of such systems.

BROWSER is an automated system for trouble-shooting with data bases. It is based on Douglas Lenat's automated discovery in mathematics [1] [2] and is designed with a modified production system architecture [3] [4]. BROWSER explores a data base using models describing the data and a large collection of data-dependent and data-independent heuristics.

An example of a BROWSER data base is a data bass containing information on maintenance actions performed on U.S. Navy aircraft [5]. Th3 data base is explored through the definition of data subsets and their examination using simple statistical techniques. Within a data subset BROWSER looks for regularities such a recurring sequences of data or unusually high occurrence rates of a particular datum. BROWSER attempts to find sequences of maintenance actions which precede a particular maintenance action. Such an action might be the failure of a component in a system. If such a pattern can be found, it could be used to give warning of the possible failure of that component. When one of the preceding events occurs the part which is due to fail could be repaired or replaced eliminating possible adverse conditions which might later occur due to the failure of the component (e.g., the aircraft crashing).

When comparing two data subsets BROWSER looks for statistically significant differences in the occurrence rates of a particular datum. BROWSER can compare aircraft with very good maintenance histories to aircraft with very poor maintenance histories to find differences. Discovery of these differences might lead to modifications on the aircraft with poor histories which will improve their system performance.

All discoveries or facts about the data are reported to the user. The user could be given the option of creating alerters [6] [7], which would monitor for the occurrence of these facts in new data entering the data base. (Alerters are not currently implmented.)

2. IMPORTANT BROWSER FEATURES

BROWSER contains three significant features. First is the representation of facts and knowledge known about the data base. This knowledge is contained in models and data-dependent heuristics. Second is the controlled execution of tasks within the system. For this, BROWSER uses an agenda, an ordered list of all tasks which are to be executed, data-

This work was supported by the Office of Naval Research under Contract N00014-75-C-0612 while the author was at the University of Illinois at Urbana-Champaign.

independent heuristics, and a controller, which executes the agenda tasks. The final significant feature is the representation of new knowledge. New knowledge is represented and stored in concept trees. Each concept represents a subset of the data and contains several facets which describe the subset.

2.1 Data Base Knowledge

Knowledge of the data base is stored in models and data-specific heuristics. Five general types of models exist in BROWSER. The first four model types provide detailed knowledge about the data in the data base while the fifth provides general knowledge on data structures useful for examining the data.

The lowest level models are the Data Base Models. One Data Base Model exists for each type of data file in the data base. Each model contains a description of the data fields which compose the data file, including the field's retrieval name (used in the data base query language), an equivalent English name, the starting position of the field in a data record, the length of the field, and the type of value contained in the field (numeric or character). The Data Base Models are used mainly by the data base management system for retrieving data from the data base.

The Data Specific Models provide information on how the various data fields are related to each other. These models show the semantics of the data fields. Again, one Data Specific Model exists for each type of data file in the data base.

The Specific Models are of two varieties. The first shows how the data records, described semantically in the Data Specific Models, combine to form larger units called Conceptual Units. For the aircraft data base an example Conceptual Unit is a maintenance action performed on an aircraft. Each performed maintenance consists of several actions, such as the removal of an aircraft part, the repair of a part or the installation of a part, where each action is described by a single data record in the data base. The second type of Data Specific Models describes the objects from which the data was gathered. For the aircraft data base these models are of the F-4 and A-7 aircraft.

Typical Models are the most general data specific models. They describe generalizations of the Specific Models. Generalized knowledge which applies across Specific Models is stored in the Typical Models. BROWSER'S Typical Models include a model of a typical aircraft and the model of an aircraft's history. The typical aircraft model represents common knowledge known about all aircraft. The aircraft's history model shows how the Conceptual Units of performed maintenance combine to form the overall maintenance history of an aircraft.

The final model type is that of the General Models. They provide information on the typical types of data structures which exist within the data. Included are models of a network, a tree, and a list.

Data-dependent heuristics are contained within the models. These heuristics provide knowledge about the data values in the data fields and other knowledge not represented by the structure of the model. Interconections between models are provided so this heuristic knowledge can be shared.

2,2 Controlled Execution of Tasks

The second significant feature of BROWSER is the use of an agenda to control the execution of tasks within the system. BROWSER'S basic operation is to define new subsets of data and explore them. At any one time BROWSER might have 50 or more subsets of data being considered for examination, many additional subsets currently being examined, and some subsets which have already been fully examined. The system uses an agenda to keep track of all of the tasks associated with the subsets which must be performed. These tasks are of two types: those which fill in the facets of a concept and those which explore and examine the data subsets associated with the concept.

The tasks on the agenda are ordered. This ordering is determined by several factors. Certain tasks must be executed before others: e.g., the definition of the data contained in a subset must exist before that subset can be retrieved from the data base, and the subset must be retrieved before it can be examined for regularities. Additionally, some tasks appear to be more interesting to perform than others because of the results of previous tasks. Each task has a list of reasons why it should be executed, each reason has a value and the sum of all of the values, the task priority, gives the position of the task within the agenda.

The basic procedure of BROWSER is as follows. All tasks are sorted according to their priorities. The highest priority task is removed to be executed. All heuristics, both data-dependent and data-independent, which are relevant to the execution of this task are gathered. These heuristics are then executed one by one. When all of the relevant heuristics have been executed the cycle is repeated. This

cycle is continually repeated until all tasks have been executed.

2.3 Representation of New Knowledge

The final significant feature of BROWSER is the representation and storage of new knowledge. New knowledge consists of facts gathered by BROWSER from the data base and is stored as faceted concepts on concept trees. Initially the data base is described by a small set of concepts: ALL-PLANES (the entire data base), CRASH (aircraft which have crashed), NON-CRASH (aircraft which have not crashed), HI-MAIN (aircraft requiring excessive maintenance) and LO-MAIN (aircraft with good maintenance histories). BROWSER creates more subsets and their associated concepts by examining existing subsets of data and offering ideas on how the existing subsets might be restricted to form smaller subsets or, possible, generalized to form larger subsets. Each new subset is described by a concept in a concept tree with its facets holding the important and relevant information desired about the subset. Note that the subsets of data described by the concepts can intersect in their coverage of the data base, and that new subsets are always created from existing subsets.

The concepts hold the relevant information needed about the data subsets. They represent the subsets and show their relationship to the other subsets. Each concept consists of a number of facets or characteristic knowledge about the subset. Typical facets include: NAME - tne name used to refer to the subset; DEFN - the definition of the data contained within the subset; INTEREST - a numeric measure of how interesting the subset of data is; GENERALIZATIONS - pointers to more general (larger, containing) subsets; SPECIALIZATIONS - pointers to more restricted (smaller, contained) subsets.

5. SUMMARY OF RESULTS

The initial implementation of BROWSER was designed to test the feasibility of automatic browsing. It was implemented in MACLISP and ran on a KI DEC-10 computer, occupying a total of 400 pages of memory (this includes the data base management system). The data base consisted of one year of maintenance authorization records for six aircraft. For this initial implementation two of the data related models were implemented: the Data Base and Data Specific Models for the particular data files used.

The initial testing occupied 24.5 CPU hours. The data base management system used 54% of this time retrieving the data. Approximately one half of the remaining time was used examining the data returned. The examination suggested the creation of *224* data subsets. I found 45 of the 115 subsets actually created to be of interest. Of the subsets suggested, 163 were suggested because data values occurred at statistically significant rates. T judged one half (82) of these subsets to be worthwhile to create. Several interesting differences were found between the data subsets, differences too complex to summarize here [8].

While the initial implementation of BROWSER was limited in its scope, it has provided some important ideas and insights for developing more complex systems. It is hoped that in the near future such systems will be able to provide valuable information to users, information which currently is not available or is very expensive to obtain.

References

[1] Lenat, D.B., "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search", SAIL-AIM-2S6, Stanford University, July 1976.

[2] Lenat, D. B., "Automated Theory Formation in Mathematics", in Proc. IJCAI-77, MIT, Cambridge, Mass., Aug. 1977, pp. 333-842.

[3] Davis, R. and J. King, "An Overview of Production Systems", STAN-CS-75-521, Computer Science Department, Stanford University, Oct. 1975.

[4] Lenat, D. B. and J. McDermott, "Less than General Production System Architectures", in Proc. IJCAI-77, MIT, Cambridge, Mass., Aug. 1977, pp. 928-932.

[5] Tennant, H., "The PLANES Database", Working Paper 14, Advanced Automation Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, June 1978.

[6] Buneman, 0. P. and H. L. Morgan, "Alerting in Database Systems: Concepts and Techniques", The Wharton School, University of Pennsylvania, Dec. 1975.

[7] Cohen, S. F., Alerters in Network Databases", The Wharton School, University of Pennsylvania, Feb. 1977.

[3] Dankel, D. D. II, "Browsing in Large Data Bases", Ph. D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, to be completed Fall 1979.

SUB-PROBLEM FINDER AND INSTANCE CHECKER
TWO COOPERATING PREPROCESSORS FOR THEOREM PROVERS

Dennis de Champeaux
Bedrijfsinformatica
University of Amsterdam
Jodenbreestraat 23 K 3123
1011 NH Amsterdam

Two pre-processors for theorem provers are described, which when applicable, will lead to search space simplification.
Both were implemented and integrated with an existing resolution type connection graph theorem prover.
Examples are provided which confirm our claim of search space simplification.
Key words and phrases: Theorem proving, pre-processing, search space simplification.

1. Pre-processors for theorem provers in general.

The main paradigm in automatic theorem proving is or should be that search is to be avoided, postponed, or else to be minimized. This should be done by any means one can lay one's hand on while maintaining completeness, generality and not succumbing to the ad-hocness as advertised by the proceduralists.

Search space simplification has been a major goal in the resolution school. Exploiting 'larger' more numerous operators (derivation rules) is the main activity in the natural deduction school. There are however many more options to be developed with which the role of search can be limited to those circumstances where there is no sensible alternative.

Already slumbering for some years is the application of multi-level search in theorem proving (Sacerdoti, 1973 [7]). it is incomprehensible that the support from a model technique (Gelernter, 1959 [3], has not been pursued. Search guidance with heuristic functions and automatic improvement of them is still inconclusive, although it has become clear that syntactic features can contribute only modestly to such functions. Chosing an appropriate representation (within one language say predicate calculus) is a wide open problem. Determining the right representation language is not yet a problem since it is not known whether multiple representation languages are necessary.

The former two issues belong to the preprocessing repertoire. We mention a few more. Selection of relevant axioms and/or definitions and/or already proven theorem/lemmas to prove a conjecture; finding a counter example of a conjecture to make sure that a proof is impossible; finding a similar already proven theorem to see whether its proof can be generalized and/or modified to handle a conjecture; instantiating a second order-theorem (e.g. induction scheme); reducing a conjecture to independent sub-problems (reduction to weakly dependent sub-problems can be done in the multi-level search framework); recognizing that a conjecture is an alphabetic variant and/or an instantiation of an axiom or an already proven theorem; etc.

This paper reports the results of implementing the two last mentioned. The next section describes the independent sub-problem recognizer. Section 3 deals with the instance checker. Section 4 discusses how they are cooperating and how they should be related from a process point of view. Two examples are given in section 5.

2. The independent sub-problem recognizer.

The sub-problem recognizer we developed is based on components of a predicate calculus-conjunctive normal form translator. Our translator was inspired by the procedure as described in [Manna, 6]. A small improvement to this translator led to sub-problem decomposition. First we present the original translation. Then we expand one of the steps (as was also partially done in [Loveland, 5] page 34). A subset of these translator rules make up the sub-problem recognizer.

The translator in Manna, 6] omitting here irrelevancies, consists of the following steps:

1- Eliminate 'if ... then' and 'if and only if'.
Replace A⊃B by V(∿A,B), and
A≡B by Λ(V(∿A,B), V(A,∿B)).

2- Move 'not' inwards.
Replace ∿(x)A by (∃x)∿A,
∿(∃x)A by (x)∿A,
∿V(A1, ..., An) by Λ(∿A1, ..., ∿An),
∿Λ(A1, ..., An) by V(∿A1, ..., ∿An), and
∿∿A by A.

3- Push quantifiers to the right.
Let (Qx) be (x) or (∃x), and let XX be Λ or V;
replace (Qx) XX(A1, ..., Ai, ..., An) by
XX{Ai, (Qx)XX(A1, ..., Ai-1,Ai+1, ..., An)}
if x not free in Ai.

4- Eliminate existential quantifiers (introduction of Skolem functions). Pickout the leftmost well formed part (∃y)(B(y)) and replace it by
B(f(x_{i1}, ..., x_{in})) where
a) x_i, ..., x_{in} are the free variables of (∃y)(B(y)) which are universally quantified to the left of (∃y)(B(y));
b) f is a 'fresh' n-ary function constant.

5- Eliminate universal quantifiers.

6- Distribute 'and' over 'or'.
Replace V(A1, ..., Ai, Λ(B1, ..., Bk),
Ai+1, ..., An) by
Λ{V(A1, ..., B1, ..., An),
..., V(A1, ..., Bk, ..., An)}.

If step (i) can be reapplied it has precedence over step (i+1).

Remark: A sequence of say universal quantifiers should get special attention in step 3. E.g. (x)(y){VP(x,y), Q(y)} can be replaced by (y)V{Q(y), (x)P(x,y)}.
Thus every universal (existential) quantifier in a sequence of universal (existential) quantifiers should be moved to the right of the sequence in its turn to check whether it can be pushed further to the right.

This procedure can be improved by expanding step 3 with:

3.1 Straighten out 'and's and 'or's.
Let XX be Λ or V.
Replace XX(A1, ..., Ai, XX(B1, ..., Bk),
Ai+1, ..., An) by
XX(A1, ..., Ai, B1, ..., Bk,
Ai+1, ..., An).

3.2 Distribute 'and' over 'or' or vice versa.
Replace
(x)V(A1, ..., Ai,Λ(B1, ..., Bk),
Ai+1, ..., An) by
(x)Λ(V(A1, ..., B1, ..., An),
..., V(A1, ..., Bk, ..., An)) and
(∃x)Λ(A1, ..., Ai,V(B1, ..., Bk),
Ai+1, ..., An) by
(∃x)V(Λ(A1, ..., B1, ..., An),
..., Λ(A1, ..., Bk, ..., An)).

3.3 Distribute quantifiers over connectives.
Replace
(x)Λ(A1, ..., An) by Λ((x)A1, ..., (x)An) and
(∃x)V(A1, ..., An) by V((∃x)A1, ..., (∃x)An).

Example: Using the extended step 3 the formula:
(x)[P(x)Λ(y){Q(x,y)Λ(∃z)R(y,z)}]
can be rewritten into:
(x)P(x)Λ(y)(x)Q(x,y)Λ(y)(∃z)R(y,z).

Observe that this replacement simplifies the Skolem function that has to be generated for the elimination of the (∃z)-quantifier.

Remark: It is worthwhile to check for redundancies (using the instance checker) after application of step 3.2.
E.g. V((x)A(x), A(a), ...) can be simplified to V(A(a), ...) while
Λ((x)A(x), A(a), ...) can be replaced by Λ((x)A(x), ...).

Remark: For easy readability we used in 3.3 the same variable 'x' in all the terms of the connective in the replacement. This does not hurt the translator. The instance checker however can get confused and demands step 3.3 to be modified such that each term in the replacement has a new fresh variable.

The INdependent SUB-problem REcognizeR - INSURER - consists of step 1 upto the modified step 3 and with step 6. It is obvious that in case INSURER applied on a problem P produces a form with leading connective 'and' its terms are independent sub-problems and if all sub-problems can be proven the P has been solved. The reverse statement that INSURER will find a maximal decomposition if P can be decomposed we leave as an intriguing task for Aiello/Weyrauch's program FOL [1].

We end this section with the 'real-life' example given in box 1.

(1) (x)(y)(z) x(yz)=(xy)z
(2) (x) xe=x
(3) (x) ex=x
(4) (x) xI(x)=e
(5) (x) I(x)x=e
(6) (H)(SUBGR(H)↔{(∃x)H(x)Λ
(x)(y)[H(x)ΛH(y)→H(xy)]Λ
(x)(H(x)→H(I(x)))})
(7) (H1)(H2){SETEQ(H1,H2)↔(x)(H1(x)↔H2(x))}
(8) (g)(xx)(H)(COSET(g,xx,H)↔
{SUBGR(H)Λ
(x)(xx(x)↔(∃y)(H(y)Λx=yg))})
(9) (g)(xx)(H)(COSET(g,xxH)→{H(g)↔SETEQ(xx,H)}).

box 1 Axioms (1-5), definitions (6-8) and a theorem (9) from group theory.

(1-5) are non-minimal axioms defining a group; (6-8) are definitions of respectively sub-groups, equality of subsets and of right-cosets; (9) is a theorem expressing a property of cosets.
Observe that subsets are represented by 1-argument first order predicates, and that SETEQ and COSET are 2nd order predicates. Direct translation of (1-8) and the negation of (9) into conjunctive normal form yields 39 clauses with together 109 literals.
INSURER however recognizes that (9) can be decomposed into:

(10) $(g)(H)\{H(g)\vee(xx)[\sim COSET(g,xx,H)\vee \sim SETEQ(xx,H)]\}$ and

(11) $(g)(H)\{\sim H(g)\vee(xx)[\sim COSET(g,xx,H)\vee \sim SETEQ(xx),H)]\}$.

Working on (10) (not done by INSURER, but by another program component) the definition of COSET, SETEQ and SUBGR are respectively substituted. The result is negated and together with (1-5) translated into conjunctive normal form yielding 14 clauses with 23 literals. After each substitution of a definition INSURER is called to check whether further decomposition is possible. When working on (11) this strategy is succesful after substituting away SETEQ. Two new sub-problems are found both ending up with 14 clauses and 21 literals.

Although our connection graph theorem prover COGITO is not yet able to handle these three sub-problems, the chance to find a solution has increased with an 'infinite' amount when compared to the non-decomposed situation.

INSURER also can handle the sorted predicate calculus that was introduced in [Champeaux,2]. The same coset example formulated in sorted predicate calculus - without decomposition - yields 28 clauses with 61 literals. INSURER finds here also three sub-problems each having 12 clauses with respectively 16, 14 and 14 literals. A significant reduction again, possibly bringing this problem within reach of the with paramodulation extended COGITO.

3. The instance checker

The instance checker (INSTANCE) we designed and implemented is in fact a special case theorem prover. It is an instrument with which a conjecture can be recognized as being an alphabetic-variant or as a special case of an already accepted theorem. Conjecture and theorem are expressed as closed, slightly restricted - see below - predicate calculus formulas.

Let T and K be respectively a conjecture and an accepted piece of 'knowledge'. The input of the recursive INSTANCE consists of three elements:

— two forms T and K for which must hold that they do not share variables and that disregarding permutations of sequences of quantifiers and/or arguments of 'and' and 'or' it is the case that T - INSURER(T) and = - INSURER(K); and
— a list of variables, VR, to be explained in the sequel, which at the top level call is empty.

Although at the top level one is mostly interested in a yes-no answer, for reasons that should become clear in the description of INSTANCE, the output is more substantial in the positive case. The output of INSTANCE is:

— NO, signifying that T is not an instance of K, or else
— a list of triples, where each triple is of the form {a, Ta, Ka} with a a substitution, Ta a "without loss of generality substitution instance" of T being a logical instance of the special case Ka of K. It should now be clear that if NO ≠ INSTANCE(T,K,0) then KKT.

An example where more than one triple will be returned by INSTANCE is:
T = $(\exists x)Ax,x)$,
K = $(y)(A(p,y)\wedge A(q,y))$, with the output:

$$\left(\left\{\begin{matrix}x\leftarrow p\\ y\leftarrow p\end{matrix}, A(p,p), A(p,p)\wedge A(q,p)\right\}\right.$$
$$\left.\left\{\begin{matrix}x\leftarrow q\\ y\leftarrow q\end{matrix}, A(q,q), A(p,q)\wedge A(q,q)\right\}\right).$$

We give a simplified description of INSTANCE omitting subtilities that have to do with the equality predicate. $\alpha\varepsilon y$ stands for: the identifier α occurs in the expression y; $S^{\alpha}_{\beta}y$ is the result of substituting α for β in the expression y; thus $S^{\alpha}_{\beta}y = y\sigma$ when $\sigma=\{\beta\leftarrow\alpha\}$.

We assume that the employed unification algorithm is willing to accept also non-literal predicate calculus formulas and so we get for example $\{z\leftarrow a\}$ when calling
UNIFY($(x)A(x,a)$, $(x)A(x,z)$, $\{z\}$).

```
INSTANCE(T,K,VR):=
if T and K are unifiable, taking into account VR,
under substitution σ
  then return with ({σ, Tσ, Kσ})
  else
if K = (α)(Form(α))
  then U:=INSTANCE(T, Form(α), VRU{α})
    if U=NO then return with NO
            else (thus U=⋃_i{σ_i, x_i, y_i})
            return ⋃_i{σ_i, x_i, ỹ_i},
            where ỹ_i=(α)y_i when αεy_i or else y_i
  else
if T = (∃α)(Form(α))
      then U:=INSTANCE(Form(α), K, VRU{α})
```

```
        if U=NO then return with NO
                else return ⋃_i{σ_i, x̃_i, y_i}
                where x̃_i=(∃α)x_i when α∈x_i or else x_i
  else
if T = (α)(Form(α))
      then let c be a fresh constant
           U:=INSTANCE(S^c_α(Form(α), K, VR)
           if U=NO then return with NO
              else return ⋃_I{σ_i, x̃_i, y_i}
              where x_i=(α)S^α_c x_i when c∉y_i
              or else x_i
  else
if K = (∃α)(Form(α))
      then let c be a fresh constant
           U:=INSTANCE(T, S^c_α Form(α), VR)
           if U=NO then return with NO
              else return ⋃_i{σ_i, x_i, ỹ_i
              where ỹ_i=(∃α)S^α_c y_i
                 when c∉x_i or else y_i
  else
if K = V(K_1, ..., K_n)
   then return ⋃_i{σ_i, x_i, y_i}where x_i=Tσ_i, y_i=Kσ_i
   and for all j x_i is an instance of K_jσ_i or if
   no such triple exists NO [see the appendix
   for a more detailed description of this case]
  else
if T = Λ(T_1, ..., T_n)
   then return ⋃_i{σ_i, x_i, y_i} where x_i=Tσ_i,
   y_i=Kσ_i and for all j T_jσ_i is an instance
   of y_i or if no such triple exists NO [we omit
   specification of this case since it runs
   parallel to the former case]
  else
if T = V(T_1, ..., T_n)
   then U: = ∅
     for all T_j do
     U2:=INSTANCE(T_j, K, VR)
     if U2≠NO thus U2=⋃_j{σ_i, x_i, y_i} then
            U:=UU⋃_i{σ_i, Tσ_i, y_i}
     if U=∅ then return NO else return U
  else
if K = Λ(K_1, ..., K_n)
   then U: = ∅
     for all K_j do
     U2: = INSTANCE(T, K_j, VR)
     if U2≠NO then U: = UU ⋃_i{σ_i, x_i, Kσ_i}
     else return NO.
```

As mentioned in the former section INSTANCE is called in INSURER after application of step 3.2. The effectiveness of doing so was proven by the conjecture of the second example in section 5: (x)0y){A(Equal(x,nil) ... Without INSTANCE INSURER will decompose this example into eight (8!) sub-problems of which six are redundant. When INSTANCE is incorporated the two non-redundant sub-problems only remain. INSURER and INSTANCE can also be coupled in another way as we will describe in the next section

4. Interplay between INSURER, INSTANCE and COGITO and what is to be desired.

INSURER, INSTANCE, COGITO and the predicate calculus - conjunctive normal form translator - were embedded in a 'fixed' regime. Input for the prover consists of axioms, supporting theorems, definitions and the conjecture. For the next description we want tp remind that activation of the connection graph theorem prover COGITO should be postponed at all costs.
Roughly a supervisor triggers the following activities :

step 1 : If the conjecture is an instance of an axiom, a theorem or an already proven theorem (see step 2) then return with success.

step 2: If the conjecture decomposes into the sub-problems CI, ..., Cn
then for each Ci go (recursively) to step 1 if the value returned for treating
Ci is succesful
then add Ci to the collection of already proven theorems
else quit with failure
return with success,

step 3: If the conjecture contains a predicate defined in one of the definitions then substitute for each occurrence in the conjecture the instantiated body of the definition and go to step 1.

step A: Translate the axioms, supporting theorems and the negation of the conjecture into conjunctive normal form, call COGITO and return the value returned by it. (COGITO gets a resource parameter ensuring termination) .

The results reported in the next section have been obtained with this, slightly different, regime. Although this particular fixed connection between INSURER, INSTANCE and COGITO makes sense and is certainly effective we do not like it. We prefer considering INSURER, INSTANCE and COGITO as being members of a potentially larger family of deductive, cooperating, independent specialists. This would require the supervisor to be implemented as a multi-processes scheduler. The overall structure would be more transparant, making more easily addition of a new specialist. Not having available language as QLISP, INTERL1SP and MAGMA-LISP prevented us of doing so.

5. Examples

Our first example looks terribly simple but a straightforward treatment after translation, by COGITO had not yet found a contradiction after

Recursive definitions are not allowed

generating 35 clauses. It consists out of only:
definition: (s)(t){SETEQ(s,t)↔(x)(xεs↔xεt)}
and conjecture: (u)(v){SETEQ(u,v)↔SETEQ(v,u)}.
INSURER immediately recognizes that the conjecture decomposes into two sub-problems of which INSTANCE shows that one is an alphabetic variant of the other.
Remains to be proven:
(u)(v){∿SETEQ(u,v)VSETEQ(v,u)}.
After substituting away SETEQ with the definition INSURER finds again two sub-problems and - surprise - INSTANCE also finds that one is a variant of the other.
Remains this time:

$$(u)(v)\{V(\exists y1)(y1\varepsilon v \wedge y1\notin u)$$
$$(\exists y2)(y2\notin v \wedge y2\varepsilon u)$$
$$(y3)(y3\varepsilon v \vee y3\notin u)\cdot$$

Negating this conjecture and translating into C.N.F. yields four clauses. The 2nd clause generated by COGITO was already the empty one. Generating the non-solution with 35 clauses was 20 times more costly than finding this solution.

The next example was taken from [Green, 4] and was already worked on as reported in [Champeaux, 2].
definition:
(x)(y){R(x,y)↔(Same(x,y)∧Sd(y)).
axioms:
(1) (x)(y){Sd(y)→Sd(merge(x,y))},
(2) (x)(y)(u){(Sd(y)∧Same(x,y))→
→Same(cons(u,x), merge(u,y))},
(3) (x){Equal(x,nil)→R(x,nil)},
(4) (x){∿Equal(x,nil)→
→Equal(x,cons(car(x)cdr(x)))},
(5) (x)(u)(v){Equal(x,u)∧Same(u,v)→Same(x,v)}.
conjecture:
(x)(∃y){(Equal(x,nil)→R(x,y))∧
(((∿Equal(x,nil))∧
R(cdr(x),sort(cdr(x))))→
→R(x,y))}.

The conjecture - already mentioned in section 3 - decomposes into two sub-problems of which INSTANCE finds that one is an instance of axiom(3). The remaining sub-problem was solved as well as without definition substitution (by adding the definition to the axioms) as with substitution. In both cases a contradiction was found more easily than in the non-decomposed case as can be seen from the g-penetrance values in table 1.

program and strategy	input + generated clauses	g-penetrance
QA3 [Green, 4]	286	0.091
COGITO (plain)	38 (25)	0.579 (0.680)
+ sub-problem recognition	28 (17)	0.785 (0.882)
+ definition substitution	20 (12)	0.800 (0.917)

Table 1 Showing the effectiveness of INSURER and INSTANCE. The numbers between brackets refer to values obtained when the sorted predicate calculus is used [Champeaux, 2]. (The g-penetrance is defined as # clauses in proof/ # (input+generated clauses).)

6. Summary

Two algorithmic (always halting) components from the natural deduction realm are described. Both were implemented and integrated as pre-processors with a resolution type, connection graph theorem prover. Examples are given that illustrate augmented power of this complex with respect to the sole theorem prover.

Factoring out other algorithmic components and/ or adding plan/strategy generators to the pre-processor family (possibly leading to multi-level search [Sacerdoti, 7], we consider a more promising future task than investing more intelligence deep inside resolution and/or natural deduction theorem provers.

Excellent typing was done by Pom van der Horst

Appendix

We give here a more detailed description of the sub-case
INSTANCE(T, V(K1, ..., Kn), VR).

```
INSTANCE(T, V(K1, ..., Kn), VR):=
    INSORK(({(K1, ..., Kn)(∅,T,∅}))
```

So we have now to describe the procedure INSORK. Its input consists of a non-empty list (Task1, ..., Taski, ...), where an element Taski is of the form:
{(K̃m, ...K̃n)(σ, T̃, (K̃1, ..., K̃m-1))}, with K̃j=Kjσ, T̃=Tσ. [T̃ is already an instance with respect to V(K̃1, ..., K̃m-1).]

```
INSORK(⋃_1 Taski):=
    Newtask:=∅
    for each Taski do
        U:=INSTANCE(T̃, K̃m, VR)
        if U≠NO  thus U=U{σ_j, x_j, y_j}
            then for each element of U do
                NN:={(K̃m+1σ_j, ..., K̃nσ_j)
                    (σ+σ_j, x_j, (K̃1σ_j, ..., K̃m-1σ_j,y_j))}
                Newtask:=NewtaskUNN
    if Newtask=∅ then return NO
    if m=n thus all elements Ki have been treated
        then U2:=∅
            for each element of Newtask do
            (which have the form
              {∅,(σ,T̃,(K̃1, ..., K̃n))})
               U2:=U2U{σ,T̃,V(K̃1, ..., K̃n)}
            return U2
        else return INSORK(Newtask).
```

REFERENCES

1 Aiello, M. & Weyrauch, R.W., Checking Proofs in the Metamathematics of First Order Logic, Stanford A.I.Lab.Memo AIM-22, August 1974.

2 Champeaux, D. A theorem prover dating a semantic network, Proceedings of the AISB/G1 Conference on A.I., pp. 82-92, Hamburg 1978

3 Gelernter, H. et al, Empirical Explorations of the Geometry Theorem Proving Machine, reprinted in: Computers and Thought, Ed. Feigenbaum-Feldman, McGraw-Hill, 1963, pp. 153-163.

4 Green, C, The application of Theorem Proving to Question-answering Systems, AI Group, Technical Note 8, SRI, Stanford 1969.

5 Loveland, D.W., Automated Theorem Proving: A Logical Basis, North-Holland, 1978.

6 Manna, Z., Introduction to Mathematical Theory of Computation, McGraw-Hill, 1972.

7 Sacerdoti, E., Planning in a Hierarchy of Abstraction Spaces, IJCAI3, pp. 412-422, Stanford, 1973.

P.S.

W. Bibel brought under our attention the paper Ernst, G.W., The Utility of Independent Subgoals in Theorem Proving, Information and Control 18, pp. 237-252, 1971. A two level process is described where the first level corresponds with the sub-problem finder INSURER. The differences are: - rule 3.2 is missing; - the check for redundancy using INSTANCE inside INSURER is missing.

Recently we have extended INSURER with additional INSTANCE-checking such that the role of INSTANCE can be formulated as:
Let X = $\wedge$(A,A1, ...,Ai, ...,An);
if INSTANCE(A,A1)
   then replace X by $\wedge$(A1, ..., Ai, ...,An);
if INSTANCE(movenot($\sim$A),Ai) then replace X by false;
and similar rules for disjunctions. (See Quine, W.O., Methods of Logic, Holt NY, 1950, pp. 101-107, which mentions special cases.)

INSURER thus fortified was capable to reduce to true:

$$[\{(\exists x1)(y1)P(x1)\equiv P(y1)\}\equiv\{(\exists x2)Q(x2)\equiv(y2)P(y2)\}]\equiv$$
$$[\{(\exists x3)(y3)Q(x2)\equiv Q(y3)\}\equiv\{(\exists x4)P(x4)\equiv(y4)Q(y4)\}]\cdot$$

(This problem was posed by P. Andrews at the Fourth Workshop on Automated Deduction, Austin, Feb. 1979.)

Question: Can INSURER - with these INSTANCE rewrite rules - recognize every valid formula from the monadic predicate calculus?

The Origin and Resolution of Ambiguities in Causal Arguments

Johan de Kleer
Xerox Palo Alto Research Center
Palo Alto, CA 94304

Abstract

The causal arguments that people typically use to explain the behavior of physical systems contain ambiguities and hidden assumptions which result from imposing a particular point of view on the behavior of the system. The causality of such an argument is an artifact of imposing this point of view. Usually there exist other equally "valid" but conflicting arguments based on the same evidence. The inherent local nature of causal arguments makes it impossible for them to capture the more global effects that arc needed to resolve these ambiguities. However, their local nature makes causal arguments computationally simple to construct. This paper discusses these ideas in the context of electronics after first presenting a general theory of causal arguments. The causal rules that elcctncal engineers appear to use to reason about circuits are presented, and their use in constructing causal arguments for circuit behavior is discussed.

Introduction

This research attempts to articulate the nature of the causal arguments which are so prevalent in human explanations of physical phenomenon. In particular, this paper is concerned with exploring "mechanism-like" explanations for the behaviors of physical systems. Although mechanistic or causal arguments arc ubiquitous, their structure and generation is rarely formalized. Both the knowledge and its associated calculus is usually tacit: there exists a shared body of common knowledge which is referred to when analyzing, explaining and understanding physical systems. Thus, this research can be viewed as articulating the *tacit calculus* used to understand and discover causal arguments.

This kind of research can have impact on education, cognitive science, and artificial intelligence. Ihis tacit calculus is part of what we are really trying to communicate in the classroom. If we understood it better this education process could be made more efficient. Furthermore, a formal model of the informal tacit calculus provides the foundations for computer-based instructional systems [Brown. Collins & Harris 78]. From a cognitive science perspective, causal reasoning is an instance of a more general type of reasoning about physical systems, namely *envisioning* [de Kleer 77]. The study of causal reasoning provides another set of distinctions that helps clarify envisioning. This common sense tacit calculus is exactly what most modern problem solving systems have been lacking, and thus a better understanding of the tacit calculi can lead to more robust knowledge based systems.

In NEWTON [de Kleer 77] an initial theory of *envisioning* was developed which focused on a kind of qualitative simulation of the physical system of the roller-coaster. An example of a problem NFWTON could deal with was:

Figure 1 : Will the cart reach X?

NEWTON approaches this problem by simulating the behavior of the cart utilizing only a few qualitative features such as the concavity of the surfaces and the direction of the velocity of the cart: "The cart rolls down the track and starts rolling up the inside of the loop. At this point it may roll back and oscillate. As it approaches the top of the loop it may fall off..." Since the specific values of the initial velocity and heights of the roller-coaster are not taken into account, the qualitative simulation cannot resolve the two critical ambiguities: (I) will the cart oscillate, (2) will the cart fall off. In order to resolve the ambiguities NEWTON analyzes their underlying assumptions and uses this information to reformulate the qualitative problem as quantitative one for subsequent quantitative analysis. For example, if the cart reaches X and docs not oscillate, it must be assumed that the velocity did not go to zero. This assumption then guides the quantitative analysis to look at the velocity of the cart on the initial segment of the loop. After these ambiguities have been resolved, the behavior of the loop-the-loop system of Figure 1 can be correctly simulated. The original qualitative simulation of the envisioning could not resolve the ambiguities and thus had to consider all the alternatives, which quantitative analysis had to later resolve. Therefore, envisioning can be viewed as the construction of a space of *possible* simulations which some other knowledge has to distinguish among.

A goal for this research is to develop a more sophisticated theory of envisioning. One of the serious problems that arises in extending the theory is that the behavior of more complex physical systems is typically

determined by a large collection of simultaneous constraints. For such systems, the fundamental difficulty is to determine what events are causing which others, unlike the roller-coaster domain where the only difficulties arise from the imprecision of the qualitative nature of the simulation. Since the primary source of ambiguities arises from potential ambiguities for the *causes* of events, envisioning for constraint systems is called *causal reasoning.* As we shall see the two difficulties in articulating this tacit calculus are: (1) the notion of causality that people appear to use to reason about constraint systems is *mythical* and bears little resemblance to what is actually the case, and (2) the resulting causal arguments are often *rationalizations,* since the ambiguities are resolved empirically rather than analytically.

Electronic circuits provide a rich domain to explore these issues since they are excellent examples of constraint systems (all of the network laws arc expressed as constraints), yet humans have extensive experience in dealing with them. Therefore after the general theory of causal reasoning is presented in the next sections a causal reasoncr for electronic circuits will be developed.

The use of this common sense tacit calculus leads to more powerful artificial intelligence programs. Causal explanations describe how the behaviors of the individual constituents contribute to the overall behavior of the system. This knowledge is important for understanding, designing and troubleshooting designed systems. For example, in the case of electronic circuits a complete algebraic analysis of even simple circuits can be computationally prohibitive, but knowledge of how the individual components contribute to the circuit's composite behavior can significantly improve the efficiency of the analysis [de Kleer & Sussman 78). For example, an integrated circuit operational amplifier contains a large number of transistors, but few of them are situated on the main signal path. For many calculations, the effect of these auxiliary transistors on the signal can be ignored or accounted for by much simpler transistor models. The use of these simpler models significantly reduces the complexity of the algebraic analysis.

The causal explanation identifies which transistors are crucial to the behavior and which are not. Causal reasoning also plays a fundamental role in identifying the faults responsible for symptomatic behavior and in localizing faults at a shallower level of detail before entering the more expensive deep analysis [Brown 76] (de Kleer 76). Early designs can be checked to see whether they have any hope of achieving their desired behavior, and the sections which are critical to the desired behavior can be identified for special attention [McDcrmott 76).

This research differs from related work by Freiling [77] and Rieger & Grinberg [77] by focusing on the distinction between the physical system that manifests the behavior and the abstract mechanism by which the system achieves that behavior. Ricgcr's theory has no representation of the system that his cause-effect diagram is a description of. In my approach causal reasoning is, in effect, a method to construct a description of the abstract mechanism from the description of the physical system. This approach has a methodological advantage over Rieger's since it eliminates much of the arbitrariness from the representation of any particular mechanism. The current research has not progressed to the point that it is capable of producing CSA-likc description of the mechanism; this is a logical next step. In [de Klecr 79] the theory of causal reasoning outlined below is used as the basis for a recognizer, QUAL, which takes a description a circuit and produces a description of the mechanism by which the circuit achieves its behavior.

Theory

The general form of a "mechanistic" argument is a sequence of events occurring in the functioning of the physical system where each event can be causally related to events earlier in the sequence. Each event is an assertion about some behavioral parameter of some constituent of the system (e.g. velocity at a point or current through a terminal). The sequential argument always reads as if it is temporally ordered.

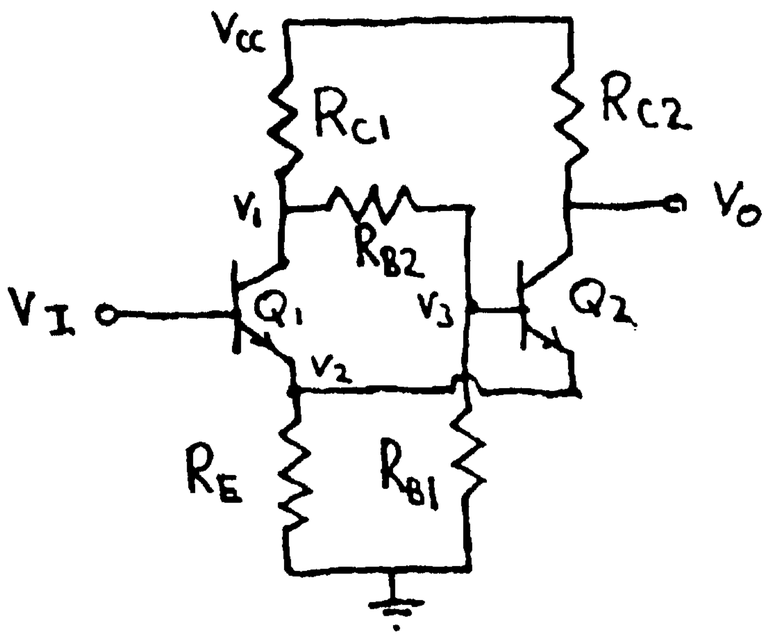


Figure 2 : The Schmitt Trigger

"... An increase in v_1 augments the forward bias on the emitter junction of the first transistor, thereby causing an incremental increase in the collector current, iC_1 of that transistor. Consequently both the collector-to-ground voltage v_1 of the first transistor, and the base to-ground voltage of the second transistor v_3, decrease. The second transistor operates as an emitter follower which has an additional load resistor on the collector. Therefore, there is an decrease in the emitter-to-ground voltage v_2. This decrease in v_2 causes the forward bias at the emitter of the first transistor to increase even more than would occur as a consequence of the initial inceasc in v_1 alone...." [Harris *et.al* 66, p.68]

A causal argument consists of a sequence of assertions about system constituents each of which hold as the consequence of previous assertions. In a causal argument, *A* is a consequence of *B* means *A* is caused by *B.* The causal

argument "... An increase in v_1 augments the forward bias on

the emitter junction of the first transistor, thereby causing an incremental increase in the collector current,...", is a sequence of two assertions: v_I increases, i_{CI} increases. These are the two events of this causal argument. The deduction of one event from another is determined by the causal rules of the device models. In the above example the model for the first transistor is one in which increased emitter potential causes increased collector current.

The causal argument makes reference to an underlying *device topology* which describes the system's constituents and all possible interactions between them. Each event of a causal argument is an assertion concerning some constituent, and its antecedents refer to other events about constituents which are adjacent in the device topology. This is a consequence of the *local* nature of the causal models. A particular causal argument selects only a subset of the device topology as the information bearing connections. This subset is called the *causal topology.*

The local nature of the rules is the source of much of the power of causal arguments, as well as some of the problems. It ensures that every constituent (e.g. transistor, resistor, etc.) of a given type can be modeled the same way without regard to its surrounding constituents. Therefore the number of causal models is very small. This ensures that at least one and often many causal topologies can be constructed for any system constructed out of known constituents.

The potential ambiguities in a causal argument stem from (1) the qualitative nature of the rules, (2) the local nature of the rules, and (3) the inability to distinguish between the causes and effects among the events. Although the Schmitt trigger is inherently a constraint system and roller-coaster appears to be inherently sequential, the forms of the mechanistic arguments which explain their behavior are very similar. This was achieved by the engineer imposing a mythical causality on the behavior of the system which then admitted a temporal sequential argument.

The explanation for the Schmitt trigger made a number of unsubstantiated assumptions aside from the choice of transistor models. Why does the v_I increment appear across Q_1 instead of R_E ?Why does the voltage v_1 drop since Q_2s turning off should raise it? Why is the current contributed by Q_2 turning off more than the current taken by Q_1 turning on? There are many values for the parameters for which the circuit cannot function at all. The arguments arc only rationalizations of the observed behavior (observed by actual measurements or staled in the textbook). This does not detract from the usefulness of The explanations: no explanation ever accounts for every detail of the behavior. The usefulness of an explanation does not depend on how complete or correct it is, but whether the explanation is sufficient for the purposes it is applied to. One of the aims of this research is a taxonomy of different types of rationalization and when they are useful. With such a taxonomy, for example, we can hope to make some progress in the area of automatic summarization and explanation of how complex systems work.

All of these ambiguities can be traced to assumptions made by the local rules. In order to sec this, the discovery, or generation, of causal arguments must be considered in more detail.

Generating Causal Arguments

The apparent temporal order of the events of a causal argument allows it to be viewed as the description of a simulation of the system's behavior. This simulation can be easily repeated by applying the causal models to the underlying causal topology of the argument. The method for originally discovering a causal argument is also a simulation but without the knowledge of this causal topology. This *generation simulation* splits at every point where the device topology leads to different causal topologies thereby producing a collection of different causal arguments each with its own distinct set of underlying assumptions. The same causal models arc used in the resulting causal argument as in the generation process. This is a stronger locality claim than made in the previous section since it stales that the models used for the discovery of the original argument can only refer to information local in the device topology. This is surprising since one would expect that the generation simulation would require more detailed models.

This simulation method of generating causal arguments is computationally quite simple. The finite device topology ensures that the simulation must terminate, and the local nature of the device models makes the simulation of devices simple, as well as ensuring that every system constructed from the modeled devices can be analyzed. (The generation simulation can be compared with conventional forward deduction, except that the deductions are severely limited by the device topology and that there is no negation.)

In order to construct a causal reasoner for a given domain three issues have to be addressed:
(1) What is the device topology?
(2) What arc the rules of the device models?
(3) What arc the possible assumptions, and how should they be dealt with?
The remainder of this paper explores these three issues in the context of electronics.

In electronics, unlike most physical systems, the device topology can be determined directly from a circuit schematic. The choice of electronics as a domain to explore causal reasoning simplifies this first issue considerably. As the models are presented a number of different assumption types will become evident, but all the types of assumptions are dealt with in the same way.

Device Models for Electronics

The classical engineering models that are used to model the behavior of electrical devices are widely agreed upon. However, the causal qualitative models that people use to reason about circuits arc not. In fact, these qualitative models

are rarely articulated, even though the tacit models that underlie people's arguments appear to be very similar. The following discussion glosses over the difficulties of identifying these heretofore tacit models. For a more detailed discussion of how these models were arrived at and alternative models see [de Kleer 79).

The causal explanation of how a circuit works is a qualitative description of the equilibrating process that ensues when signals are applied to the circuit. The behavior of the Schmitt trigger was described in this way. This will be called *incremental qualitative* (IQ) analysis. Since most circuits are designed to deal with changing input signals, it is not surprising that the main purpose of most circuits is achieved incrementally. For example, an amplifier must amplify changes in its input, digital circuits must switch their internal states as applied signals change, and power-supplies must provide constant current or voltage in the face of changing loads and power sources.

Incremental qualitative arguments rarely need to refer to more than the sign of the derivative which indicates whether the signal is increasing or decreasing. This requires an algebra of four values: "t" signal is increasing, "0" signal is not changing, "4" signal is decreasing, and "?" signal is unknown. The arithmetic of this algebra is very simple:

x :	↓	0	↑	?
y:				
↓	↓	↓	?	?
0	↓	0	↑	?
↑	?	↑	↑	?
?	?	?	?	?

Table 1 : x + y

Only addition and subtraction are important, and no other operations are ever used.

The approach for constructing the models is to start with the classical constraint models, and reformulate them preserving only the sign of the derivatives of the variables. Ohm's law has a particularly simple formulation:

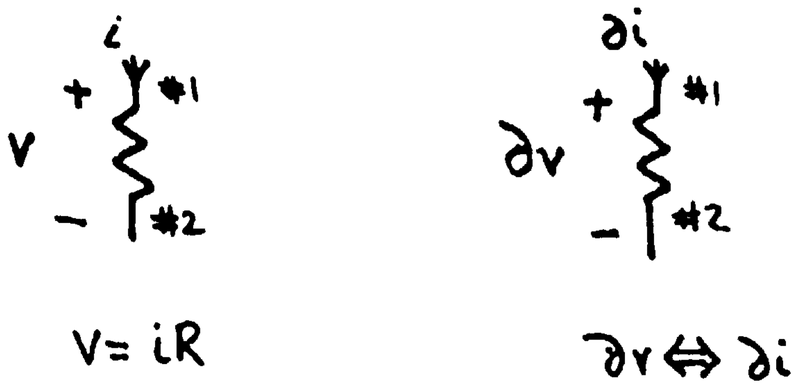


Figure 3 : Ohm's Law for Resistors

dx refers to the sign of the derivative of *x.* Currents are defined to flow into devices away from nodes. Kirchoffs Current Law (KCL) applies to components so that the current into #1 is equal and opposite the current through terminal #2. The IQ model for Ohm's law is:

∂v <=> ∂1

The rule specifies that the derivative of the current must be of the same sign as the derivative of the voltage. Since the resistor has no preferred causal flow direction this rule must be bilateral. This action is specified by the " < = > " operator.

The behavior of nonlinear devices can be modeled by a small number of linear regions, or states. Since the correct state cannot be determined when a nonlinear device is first examined by the simulation, the use of a causal rule for any state can only be made under the assumption that the device is operating within that particular region.

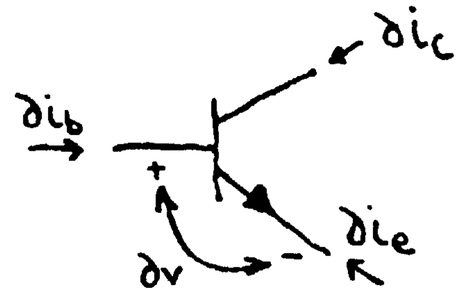


Figure 4 : Transistor

If Q is on : ∂v => ∂ic, ∂v => -∂ie, ∂v => ∂ib
If Q is off : ∂ib = 0, ∂ic = 0, ∂ie = 0
If Q is saturated : ∂ic = 0

The " = > " action indicates a unidirectional causality, and the " = " action indicates that the circuit quantity is fixed at that constant value. Applied input signals can also cause the transistor to change its region of operation to another state. These possible state transitions require additional rules which make different kinds of assumptions. However, these will not be discussed here.

We now have enough rules to analyze some simple circuits:

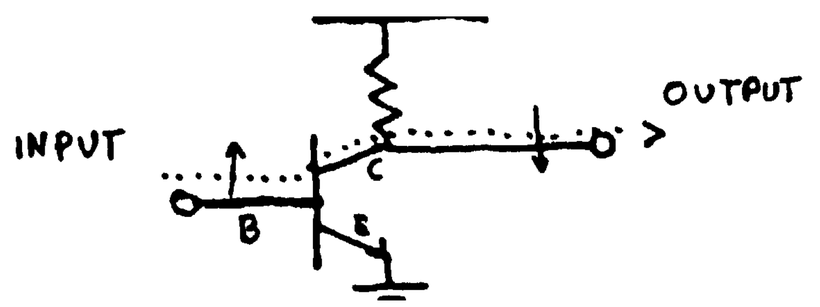


Figure 5 : Simple Inverter

The following causal argument is generated for this circuit (assuming the transistor is on):

Input voltage goes up (Given)
Collector current increases (Transistor rule)
Current through the resistor increases (KCl)
Voltage across the resistor increases (Resistor rule)
Voltage at output decreases (KVL)

In order to generate the causal argument the conventional

Kirchoffs Laws were utilized. Kirchoffs current law (KCL) specifics that the current flowing into any node or device is zero. Since there are only two devices connected to the output node, KCL states that the current flowing out of the collector equals the current flowing into the resistor. Kirchoffs voltage law (KVL) specifies that the voltage around any loop is zero. Since the supply voltage is unchanging, KVL states that the voltage change across the resistor must be equal and opposite the voltage across the transistor.

Heuristic Rules

The rules discussed so far are sufficient to deal with many circuits. The only assumptions that are made involve state choices and therefore every causal argument that makes the correct state choices is guaranteed to be valid. Unfortunately, most circuits cannot be completely analyzed. One such case occurs when a transistor's collector is connected to a number of circuit fragments:

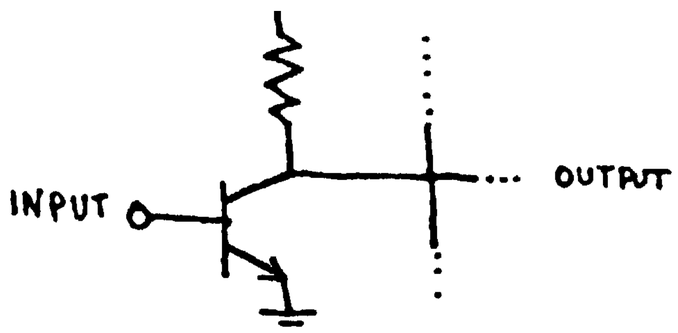


Figure 6 : An Unanaly/.able Circuit Fragment

An increased base voltage causes an increased collector current. Since there arc a number of devices connected to the collector node, the current through these devices cannot be determined. Furthermore, the transistor model provides no information about the voltage at the collector node. Thus the simulation halts with the voltage and currents at the node left unassigned. In *quantitative analysis* this would be a point at which to introduce a variable [Stallman & Sussman 77].

This situation arises quite commonly in engineer's analyses, 'The method for dealing with it can be summarized in one heuristic: *the (IQ) value of the potential at the node (the voltage with respect to ground) is opposite to the (IQ) value of the current drawn from the node.* An instance of this heuristic is: "The increasing current pulls down the node." In the above circuit fragment, the increased collector current causes the voltage at the collector node to drop. This heuristic rule is makes the assumption that the circuit around the node is behaving as a positive resistance. This assumption can be violated.

A second unanalysable circuit fragment occurs at the input of the Schrnitt trigger:

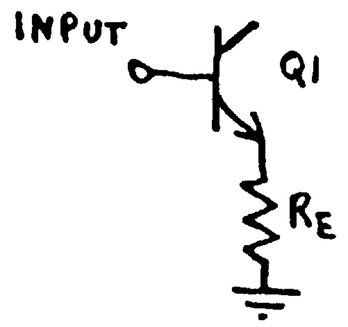


Figure 7 : Schrnitt Trigger Input

The input signal is applied between the base of the transistor and ground (i.e. across both the transistor and the resistor). Since this voltage does not appear across any component in isolation, no causal argument is discovered for the behavior of this circuit. A second heuristic is used to deal with this case: *whenever a voltage is discovered between a device terminal and ground, this value is applied directly to the device.* This heuristic assumes that the first signal discovered at the input of some device dominates the signals at the other terminals.

The first heuristic is called the KCL-heuristic, and the second the KVL-heuristic. These two heuristics combined with the device rules arc sufficient to explain the behavior of most circuits. The central remaining difficulty is that this causal calculus docs not necessarily ascribe a unique behavior to the circuit being analyzed. These ambiguities result from assumptions made by the heuristic rules. The assumptions come from three different sources:

(1) The nonlinear nature of the devices.
(2) Transitions between the linear states.
(3) KCL- and KVL-heuristics.

Ambiguities and Assumptions

A second phase of the analysis considers the different possible causal arguments (ambiguities in the behavior) that a circuit can have in order to determine which assumptions are relevant. In order to do this each event of a causal argument is tagged with the assumptions that were made in the simulation to get to it. This tagging is illustrated by the following more formal description of the causal argument for the Schmitt trigger. Each voltage or current is described by a short descriptor, a value, a list of assumptions, and a justification. The assumption list is enclosed within "< ... >". An assumption concerning a device's state is described as (< device > < state >), a KCL-heuristic assumption is described as [< terminal > < device >], and a KVL-heuristic assumption is described as [< device > < terminal >].

(VOLTAGE INPUT GROUND) = ↓
    Premise.
(VOLTAGE B1 E1) = ↓ <[Q1 B]>
    KVL-heuristic [Q1 B]
(CURRENT C Q1) = ↓ <(Q1 ON) [Q1 B]>
    ∂V => ∂IC for Q1
(VOLTAGE C1 GROUND) = ↑ <[C Q1] (Q1 ON) [Q1 B])
    KCL-heuristic [C Q1]

These assumptions play a critical role in two rather complex kinds of qualitative reasoning, which are presented next.

Points of View

This section and the next contain a very brief discussion of the roles of interpretations and teleology. Both involve surprising subtle and complex issues which are difficult to present in the space available. See [de Kleer 79] for a detailed presentation.

A collection of assumptions can define a point of view on the behavior of the circuit. Each different collection of assumptions suggests a different way in which the circuit can work by selecting a different set of events which depend on the assumptions. There a number of conditions that a collection of assumptions must satisfy before it can be reasonably considered a point of view. For this reason the notion of *interpretation* is introduced. An interpretation of a circuit's behavior is a collection of assumptions which:

(1) is noncontradictory - docs not select contradictory events.
(2) is maximal - every event that can be added must be added.
(3) justifies heuristics -- all the assumed positive resistances must behave as such.

These three conditions rule out most collections of assumptions, and the remaining interpretations are very few (never more than twice the number of transistors in the circuit). Often there is only one interpretation.

The Role of Teleology

Every interpretation produced by this procedure describes a behavior that could be valid for some assignment of circuit parameters (i.e. specific resistances, gains, power supply voltages, etc.). The interpretations can usually be disambiguated by appealing to the teleology of the circuit. Since circuits are *designed artifacts* - systems whose behavior is to achieve some particular purpose - they have a *teleology.* By knowing the purpose of the device the interpretation whose behavior is consistent with this purpose can be selected as the correct one. For certain classes of circuits, just the knowledge that the circuit has some purpose can be sufficient to identify the correct interpretation.

In the case of the Schmitt trigger the analysis discovers four interpretations:

[1] correct.
[2] approximately interpretation [1] but without feedback.
[3] signal reversing the feedback path.
[4] approximately interpretation [2] and (3).

(Interpretations [2] and [3) do not violate the maximality condition for interpretations, but in order to see this a detailed examination of the assumptions involved is required.)

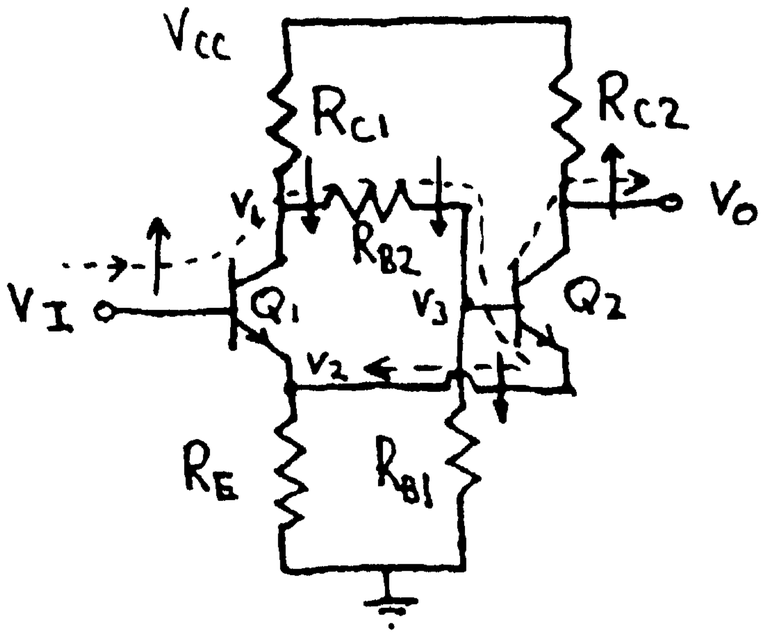


Figure 8 : Voltages in Interpretation [1]

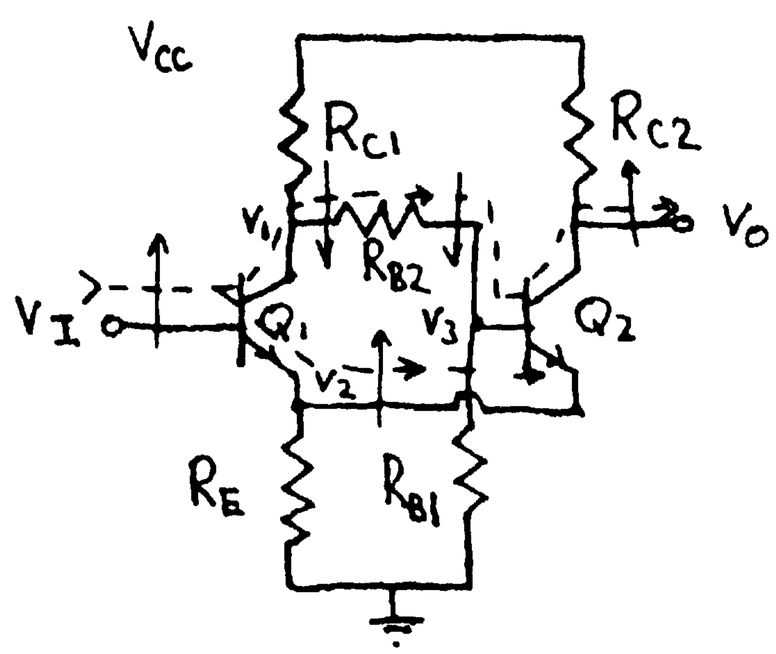


Figure 9 : Voltages in Interpretation [4]

Conclusions

The power of the causal reasoning (envisioning) comes from the fact that it is *complete, limiting* and *articulate.* Envisioning is complete for the class of circuits QUAL considers since it is capable of simulating every possible behavior. Therefore, any behavior which the envisionment does not predict as a possibility cannot happen. Envisioning is limiting in that it generates only a small number of ambiguities. Finally, envisioning articulates the source of the ambiguities so that other knowledge can be used to deal with them. In the case of electronics this other knowledge is Ideological. Without any one of these properties the calculus would be useless. For example, without the completeness property no necessary relation exists between the envisionment and what is actually the case. If the envisionment does not articulate the source of the ambiguities, other knowledge cannot be used to resolve them.

NEWTON [de Kleer 77], a program to solve physics problems in the roller coaster domain, is also based on envisioning. The envisionment for a roller-coaster problem consists of a sequence of qualitatively described scenes indicating how the roller-coaster moves along the surface. NEWTON invokes mathematical laws to quantitatively resolve

the ambiguities, while QUAL resolves the ambiguities quite differently by appealing to the teleology of the system. Both programs are based on the same theory of envisioning, but resolve the ambiguities in radically different ways. Although ambiguities appear at first to be an undesirable side-effect of a theory of envisioning, the examples from physics and electronics illustrate that they play an important role in understanding complex systems. The assumptions underlying the ambiguities provide the key to reformulating the analysis so that other knowledge can be profitably employed.

In the case of the roller coaster world the causal rules are quite obvious and are determined by the behavior of the roller-coaster cart through time. However, in the case of electrical systems it is very difficult to determine a time order on the events, and the causality between events is largely imposed by the understandcr. In this case the causal rules can only be determined by studying arguments that people actually use to understand circuit behavior.

Acknowledgements

1 would like to acknowledge the help of my thesis advisor G.J. Sussman. I had many enlightening discussions with J.S. Brown concerning the content of this paper. A forthcoming joint paper with J.S. Brown deals with the cognitive aspects of this work.

This paper describes research done in part at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's research is provided in part by the National Science Foundation under Grant MCS77-04828.

References

[Brown 76]
Brown, A.L., "Qualitative Knowledge, Causal Reasoning, and the Localization of Failures", Artificial Intelligence Laboratory, TR-362. Cambridge: M.IT., 1976.

[Brown, Collins & Harris 78]
Brown, J.S., A. Collins and G. Harris, "Artificial Intelligence and Learning Strategies", in *Learning Strategies,* edited by H.F. O'Ncil, New York: Academic Press, 1978.

[de Klcer 76]
de Kleer, J., "Local Methods for Localizing Faults in Electronic Circuits", Artificial Intelligence Laboratory, AIM-394, Cambridge: M.I.T.. 1976.

[de Klecr 77]
de Kleer, J., "Multiple Representations of Knowledge in a Mechanics Problem Solver", *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* pp. 299-304, 1977.

[de Klecr 79]
de Kleer, J., "Causal and Telcological Reasoning In Circuit Recognition", Ph.D. thesis, Cambridge: M.I.T., 1979.

[de Klcer & Sussman 78]
de Klecr, J. and G.J. Sussman, "Propagation of Constraints Applied to Circuit Synthesis", Artificial Intelligence Laboratory, AIM-485. Cambridge: M.I.T., 1978.

[Freiling 77]
Freiling, M.J., "The Use of a Hierarchical Representation in the Understanding of Mechanical Systems", Ph.D. thesis, Cambridge: M.I.T., 1977.

[Harris *et.al* 66]
Harris, J.N., P.E. Gray and C.L. Searle, *Digital Transistor Circuits,* New York: John Wiley & Sons, 1966.

[McDermott 76]
McDermott, D.V., "Flexibility and Efficiency in a Computer Program for Designing Circuits", Artificial Intelligence Laboratory, TR-402, Cambridge: M.I.T., 1976.

[Rieger & Grinberg 77]
Rieger, C, and M. Grinberg, "The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms", *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* pp. 250-255, 1977.

[Stallman &, Sussman 77]
Stallman, R.M. and G.J. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer* Aided Circuit Analysis", *Artificial Intelligence* 9, pp. 135-196, 1977.

SCHEDULING OP PROCESSES IN A SPEECH UNDERSTANDING SYSEM BASED ON APPROXIMATE REASONING

Renato De Mori Lorenza Saitta
Istituto di Scienze dell'Inforraazione
University di Torino
C.so Massimo D'Azeglio 42
10125 - TORINO (Italy)

A method for scheduling laiowledge source instantiations in a Speech Understanding System is presented. It is based on the evaluation of linguistic probabilities and on approximate reasoning.

1. INTRODUCTION

Searching for control strategies and methods for scheduling interpretation processes is among the most difficult tasks involved in designing Speech, Language and Image Understanding Systems. Remarkable efforts have been made in the area of Speech Understanding Systems (SUS), whose state of the art has been the object of a recent review by De Mori [l]. Sophisticated control strategies and scheduling polices are required when the world to be interpreted is very complex (for example for large protocols with big dictionaries in a SUS). For such cases, a paradigm of the type ''hypothesize-and-test'' and knowledge representation by rules of the type "precondition -> action" have been proposed [2]. The first attempts to design and evaluate control strategies in such a conceptual framework and in an environment where the input informations are corrupted and redundant appear in papers by Woods et al. [3], Lesser and Hayes-Roth (4] and in the Ph. D. Thesis by Paxton [5]. They mostly use numerical probabilities or heuristic scores to evaluate the hypotheses emitted by the system and to assign priorities to the processes that have to be executed to verify new hypotheses or theories. Valuable as they are, all these approaches leave room for improvements.

On the other hand, control strategies of systems designed for understanding non corrupted input messages are mostly based on non-numerical rules. The content of this paper tries to establish a connection between the numeric and the non-numeric approach to the design of control strategies.

An attempt has been made for evaluating interpretations of the speech signal by a-posteriori probabilities, using the Bayes theorem [3]. But, in these evaluations, a high degree of imprecision is involved, owing to the approximations made in extracting acoustic features and in evaluating a-priori probabilities. Moreover, an increase of the complexity of an interpretation corresponds very often to having more heuristic and subjective simplification introduced in the computation of probabilities. Thus the a-posteriori probabilities obtained by the Bayes theorem are affected by a degree of imprecision. There is little hope that this imprecision can be removed by refinements involving a practically impossible experimental work. The most realistic position is that of accepting imprecisions and representing a-posteriori probabilities by linguistic variables. A mathematical framework in

which linguistic probabilities can be treated is the theory of fuzzy linguistic variables [6-9]. This theory not only allows one to deal with imprecise concepts, as the linguistic probabilities are, but also to combine probabilistic concepts with concepts of other types, using inference rules for developping an approximate reasoning. Furthermore, a-posteriori probabilities are not the only parameters on which scheduling of further interpretation actions have to be based. In fact, a control system should be designed on the basis of inference rules which take into account also considerations of action importance and suitability, that are not probabilistic in nature.

2. ASSIGNMENT OF LINGUISTIC PROBABILITIES

Let Ω be a portion of the input data to be interpreted. System knowledge is shared among levels and each level may have a knowledge source associated with it. The lowest phonetic features of the system are "Vocalic", "Non vocalic" and "Sonorant", "Non-sonorant". The latter two features are applicable, in the system, only to consonants. The assignment of linguistic probabilities to interpretations of Ω begins with the context-independent phonetic features, given the evidence of the acoustic features. For example, the assignment of the phonetic features "sonorant" and "non-sonorant" to Ω is controlled by two fuzzy grammars G_S and G_N, described in details in [11]. Parsing the input string under the control of G_S and G_N leads to the evaluation of the compatibility values $\mu_S(\Omega)$ and $\mu_N(\Omega)$ between Ω and the start symbols of G_S and G_N respectively. Rather than on the exact values of $\mu_S(\Omega)$ and $\mu_N(\Omega)$, scheduling of subsequent processes depends on considerations that may be expressed by fuzzy propositions like the following: "$\mu_S(\Omega)$ and $\mu_N(\Omega)$ are approximately equal" or "$\mu_S(\Omega)$ is much larger than $\mu_N(\Omega)$", etc.. These fuzzy propositions lead to the definition of fuzzy sets on the interval $[-1,+1]$ of

$$x = \mu_S(\Omega) - \mu_N(\Omega) \qquad (1)$$

Let $A \in \mathcal{A}$ be one of such fuzzy sets. Let J be the cardinality of $\mathcal{A}$; the evidence of the features "sonorant" and "non-sonorant" in Ω will be represented by the fuzzy linguistic variable A such that:

$$\mu_A(x(\Omega)) = \max_{j \in [1,J]} \mu_{A_j}(x(\Omega)) \qquad (2)$$

Let H_S and H_N be the hypotheses: Ω is a sonorant consonant and Ω is a non-sonorant consonant respectively. By the Bayes theorem one gets:

$$Pr(H_i|A) = \frac{Pr(A|H_i)\,Pr(H_i)}{Pr(A)} \quad (i=S,N) \qquad (3)$$

The evaluation of $Pr(H_i|A)$ is rather complex because A is a fuzzy event instead of a precisely defined event. Given the probability density $p(x|H_i)$ and the definition of A on the interval $[-1,+1]$ of the variable x and applying the extension principle and the definition of fuzzy integrals [10] it is possible to compute a possibility value $\mu_{Pr(H_i|A)}(z)$ for every value $z \in [0,1]$ of the probability $Pr(H_i|A)$. This possibility|probability distribution can be compared with linguistic probabilities, defined as fuzzy sets on the interval $[0,1]$. Such definitions can be found in [10]. Let $P(H_i|A)$ be the fuzzy set obtained with the (3) and λ_j a linguistic probability value: the value λ_j $(j \in [1,J])$ that best approximates $P(H_i|A)$ is retained instead of an exact value of the left side probability in (3). The problem of linguistic approximation is treated in details by Bellman and Zadeh [13].

The computations mentioned so far are performed once forever and the results are represented by tables giving the best linguistic approximation for a probability|possibility distribution, given the evidence of the features in Ω represented by fuzzy sets $A_i \in \mathcal{A}$.

3. PRIORITY ASSIGNMENT

A Speech Understanding System is organized by levels; let $\langle l_1, \ldots, l_J \rangle$ be the set of such levels. The knowledge of the system strategy may be represented in procedural form by rules of the type:

⟨precondition→action⟩. For example, in the project referred to in this paper, the lexical knowledge is represented by frames, whose composition is described in another paper [14]. Each frame w_i^* contains a set of preconditions P_i^* and a set of actions A_i^*. A precondition $p_{ik} \in P_i^*$, that allows the action $A_{ih} \in A_i^*$ to be requested, is expressed in terms of items corresponding to hypotheses already verified. Combining the probability values of these items, it is possible to evaluate the linguistic probability value of the precondition p_{ik}; let λ_{ik} be such value. The assignment corresponds to the following proposition:

$$p_{ik} \text{ is } \lambda_{ik} \qquad (4)$$

Priority assignment is performed by the understanding controller on the basis of two types of evaluation. The first refers to the possibility of the succes of an action, given the probability of the preconditions. The second refers to the suitability of the action with respect to a knowledge like computational complexity and goal satisfaction. The latter evaluation is based on a proposition of the following type:

IF p_{ik} is likely THEN A_{ih} is suitable (5)

where "suitable" is a fuzzy linguistic variable belonging to a dictionary Σ.
The (5) is an example of the rules of inference of the understanding controller. Such rules may be learned by experiments or they may be initially settled by the designer. During the understanding process, the controller will assign a linguistic value to the process executing the action A_{ih} according to the result of the following approximate reasoning:

$$\begin{array}{ll} p_{ik} \text{ is } \lambda_{ik} & \text{(premise)} \\ \text{IF } p_{ik} \text{ is likely THEN } A_{ih} \text{ is suitable} & \text{(implication)} \\ \hline \text{Execution of } A_{ih} \text{ is } \sigma\text{-suitable} & \text{(conclusion)} \end{array} \qquad (6)$$

When more actions compete for execution with different degrees of local suitability, the priority have to be assigned to them on the basis of a few general inference rules of the control strategy. Such rules establish general degrees of importance of possible actions, given premises that are assumed to be true or false.

4. AN EXAMPLE OF APPLICATION

We defined the four fuzzy sets:
$A_1 \triangleq$ "$\mu_S(\Omega)$ and $\mu_N(\Omega)$ approximatively equal"; $A_2 \triangleq$ "$\mu_S(\Omega)$ slightly higher than $\mu_N(\Omega)$"; $A_3 \triangleq$ "$\mu_N(\Omega)$ much higher than $\mu_S(\Omega)$"; $A_4 \triangleq$ "$\mu_S(\Omega)$ much higher than $\mu_N(\Omega)$". By the method described in [10] we computed the possibility|probability distrubution of the a-posteriori probabilities $Pr(H_i|A_j)$ ($(i=S,N)$, $j \in [1,4]$), which we approximated by standard linguistic variables.
Table I and II show the results obtained By using suitability values as priority values for the competing processes "sonorant" "non-sonorant" the right action is scheduled with higher priority in 84.6% of the cases and in the 14% of the cases the two actions have been scheduled with equal priority. Using a Bayes decision we obtained, in the most favorable case an error rate of 0.8% with 22% undecided cases.

A_1	$\lambda(H_S\|A_1)$	$\lambda(H_N\|A_1)$
A_1 $-0.120 \leq x < 0.116$	Uncertain	Uncertain
A_2 $0.116 \leq x < 0.369$	Likely	Unlikely
A_3 $-1.000 \leq x < -0.120$	Extremely unlikely	Extremely likely
A_4 $0.369 \leq x < 1.000$	Very likely	Very unlikely

Table I - Definition of the fuzzy sets A_1 and their a posteriori probability.

	A_1	A_2	A_3	A_4
S	18	68	0	107
N	33	4	128	1

Table II - Assignment of priority according to the linguistic probability value.

REFERENCES

[I] R. De Mori: "Recent advances in automatic speech recognition", Invited paper Proc. 4-th Int. J. Conf. Pattern Recognition, Kyoto (1978)

[2] D. Raj Reddy:"Hearsay speech understanding system: Summary of results of the five-year research effort at Carnegie-Mellon University", Dept. Comp. Sci. Carnegie-Mellon Univ., (1977)

[3] W. Woods:"Shortfall and density scoring strategies for speech understanding control", Proc. 5-th Int. J. Conf. Artificial Intelligence, Cambridge, MA, 18-26 (1977)

[4] F. Hayes-Roth, V. Lesser:"Focus of attention in a distributed logic speech understanding system", ibidem, 27-35 (1977)

[5] H. Paxton:"A framework for speech understanding", Ph. D. Thesis, Stanford Univ.,Ca (1977)

[6] L. Zadeh:"PRUF - A meaning representation language for natural languages", Int. J. Man-Machine Studies 10 395-460 (1978)

[7] B. Gaines:"Foundations of fuzzy reasoning", Int. J. Man-Machine Studies 6., 623-668 (1976)

[8] L. Zadeh:"A theory of approximate reasoning", Memo M77/58, Electronic Research Lab., Univ. of California, Berkeley, Ca. (1977)

[9] E. Mamdani, S. Assilian:"An experiment in linguistic synthesis with a fuzzy logic controller", Int. J. Man-Machine Studies 7,1-13 (1975)

[10] D. Dubois, H. Prade: "Fuzzy Algebra, Analysis, Logics", School of Electrical Engineering, Purdue Lafayette, I.R. TR-EE 78-13 (1978)

[II] R. De Mori, P. Laface:"Automatic detection of distintive features in continuous speech", Proc. 3-rd Int. J. Conf. Pattern Recognition, Coronado, CA., 609-617 (1976)

[I2] L. Zadeh:"The concept of linguistic variable and its application to approximate reasoning,III" Information Science 9, 43-80 (1975)

[I3] R. Bellman, L. Zadeh:"Local and fuzzy logics" Memo ERL M-584, Univ. of California, Berkeley (77)

[I4] R. De Mori, L. Lesmo, M. Poncini:"The structure of a lexicon for a speech understanding system", Proc. IEEE Conf. ASSP, Washington D.C., 252-255 (1979)

INDUCING FUNCTION PROPERTIES FROM COMPUTATION TRACES

Pierpaolo Degano
Istituto di Scienze della Informazione
Universita degli Studi di Pisa
Corso Italia 40
56100 Pisa, Italy

Franco Sirovich
Istituto di Elaborazione della Informazione
Consiglio Nazionale delle Ricerche
Via S. Maria 46
56100 Pisa, Italy

A system which induces properties of functions is presented. Induction is performed by step-wise generalizing specific given elements of the function domains for which the system can test that the property holds. The system relies on symbolic computation, reflexivity lemmas, and an estimate of the behaviour of the functions. Finally, the paper gives a basis for an evaluation of the system by constructively defining a class of theorems which the system is able to induce.

"In mathematics as in physical sciences we may use observation and induction to discover general laws. But there is a difference. In the physical sciences, there is no higher authority than observation and induction, but in mathematics there is such an authority: rigorous proof." G. Polya How to solve it. Doubleday (New York, 1957).

1. INTRODUCTION

Inductive reasoning has been a popular task in Artificial Intelligence since the very early beginning. It has been attempted in such domains as (just to mention a few) series completion, grammar inference, automatic programming, and theory formation [5,6,9,10,14,17,19, 20,22] .

In deductive sciences, inductive reasoning plays a peculiar role in the process of knowledge development. In fact, the correctness of an induction can be confirmed by a rigorous proof, even though only a semi-decision procedure might be available, depending on the formal system complexity. Moreover, the rejection of an inductive hypothesis may bring forth explanations about the failure which help in refining the hypothesis.

It goes far beyond the scope of this paper to give a complete account of all the work on inductive reasoning in formal systems. We might mention just a few references. Meltzer [15], Michalski [I6] , Plotkin [I8] and Vere [22] have tackled various forms of inductive reasoning tasks in the predicate calculus. Brown and Tarnlund [4] were concerned with finding a close form solution to difference equations. In the context of program verification, Boyer and Moore [2] , Brotz [3], and Aubin [I] generalize theorems which are then proven by a suitable induction principle. The problem domain we have chosen is akin to the latter.

The system we present induces properties of recursive functions. The properties are defined to the system as (boolean valued) recursive functions. The system can observe that a given property holds for given elements of the domains of the involved functions (called ground property), and examine the corresponding ground property computation. Since the computation is indeed a proof, the evidence the system starts from is that a very trivial theorem holds. The definitions of properties and functions are available and must be taken into account, because no relevant inductive reasoning can be made irrespectively of their nature. Secondly, induction can depend on the details about the flow of computation. We simply provide the system with the kind of information one might get from any tracing facility, i.e. which function, when and where, was applied during the computation.

Thus, inductive reasoning requires the system to grasp a notion of function definition and computation up to be able to formulate a theorem about functions at an expertise level. We

do not claim that a competent system which induces function properties does understand functions and computability. The system we present here is not at all able to develop a theory of computation, hence it does not compare with systems for scientific discovery, such as Lenat's AM [I2] . Nonetheless, it exhibits some competence in function property induction.

Again, let us stress the advantage we gain from addressing an inductive reasoning task in the framework of a formal system. The aptness of the inductive system can be precisely stated, confirmed by a formal proof, and anyway appreciated on firm grounds (as far as the observer understands the formal function and property definitions). In this prospect, we cared for providing a precise evaluation of our inductive method effectiveness, and we proved that the method is strong enough to induce formulae which are actually theorems if the involved functions and properties belong to a constructively defined class.

2A FORMAL CALCULUS ENVIRONMENT

We use a simple recursive function formalism, TEL (Term Equation Language) which was introduced [I3] for proving program properties by symbolic computation (e.g. [2]) and is similar to other independently developed formalisms [1,7] . For the present application we add types to TEL so that the resulting language is so similar to Aubin's that the formal treatment and all results of his carry over Typed TEL (later referred to as TTEL). We now briefly overview TTEL, borrowing some nomenclature from Aubin's.

Every term, i.e. every variable and function application, has a type. Each type is defined by a set of type equations which also define the type constructors. All the types and constructors occurring in the equation (apart from the type and constructor being defined) must be defined at a lower level. The language is quantifier free, because any variable occurring in an equation is implicitly universally quantified over its type. Examples are

(TYPE(TT())=B; TYPE(FF())=B)
(TYPE(0())=N; TYPE(S(N))=N)
(TYPE(NIL())=L; TYPE(C(N,L))=L)
(TYPE(LNIL())=LL; TYPE(LC(L,LL))=LL)

The constructor TYPE is used only to denote type equations. In the following we will omit the argument list of O-adic constructors. A type is called reflexive if it is defined in terms of itself (e.g. N,L and LL). Analogously, constructors like S and C are reflexive and the argument position where the type they construct occurs is called the reflection argument position. Non-reflexive constructors will also be called terminators of the type. Given a term c(tl, ...,tn) where c is a constructor, tl,...,tn are terms, then ti (l<i<n) is an immediate predecessor of c(tl,. . ..,tn) if ti occurs in a reflexion argument position.

Defined functions are introduced by stages. A function definition is a pair, whose first component is a type equation which defines the types of the arguments and the type of the result. For example TYPE(EQN(N,N))=B. The second component consists in a set of equations (rewrite equations), which allows a definition by cases [7,II] . Rewrite equations obey the schema f(al,...,an)=<rewriting term) where f is the function being defined, and al,...,an are terms (formal arguments) which may consist either in a variable, or in a constructor applied to variables only (recursion arguments). The only variables which may occur in the <rewriting term> are the formal argument variables. If f occurs in the <rewriting term)(recursive equation), its recursion arguments must be immediate predecessors of the formal recursion arguments.

The definition by cases is restricted as follows. If an argument position is recursive in one equation, then it must be a recursive argument position for all the equations, and for each constructor of the required type there must be exactly one rewrite equation. If the function simultaneously recurs on two or more argument positions, then exactly one rewrite equatxon must be given for each tuple of constructors of the required types (see the example below).

The metalinguistic constraints allow the definition of total functions only. Nevertheless, all primitive recursive functions over the type *of* natural numbers N can be defined. This is concrete example.

{EQN(0,0)=TT; (ENbI)
EQN(0,S(y))=FF; (ENb2)

EQN(S(x),0)=FF; (ENb3)
EQN(S(x),S(y))=EQN(x,y)} (ENr)

The definition of computation of a TTEL term follows.

i) Specialize a rewrite equation so that its left-hand side becomes identical to a (sub)term of the evaluating term;
ii) Substitute in the evaluating term the specialized equation right-hand side for the (sub)term;
iii) Repeat from Step (i) until no equation left-hand side can be made identical to a (sub)term of the evaluating term.

The interpreter adopts the call-by-need computation rule.

Let us now extend the TTEL term definition by introducing free typed variables. A free variable can be instantiated to any term of its type, possibly introducing new free variables. A symbolic term is a term in which free variables occur, otherwise the term is ground. We can extend the definition of computation to handle symbolic evaluating terms. Step (i) only needs to be extended so that the involved test for identity can cause evaluating term free variables to be instantiated. The computation of a symbolic term may be non-deterministic, due to the inherent non-determinism of free variable instantiation. Thus, symbolic computations are (possibly infinite) trees. A concrete example is the following. The term EQN(S(x),S(S(^)) (free variables will be underlined) is reduced to EQN (x,S(£)) by (ENr) and then either to FF if x is instantiated to 0 by (ENb2), or to EQN (x_l,vj if x is instantiated to S(x1) by (ENr). From this point on, all EQN equations can be applied.

We might provide the same inference rules available in Aubin's system. We omit them here since we are not interested in the deductive aspects of the formalism. It is only important to notice that ground property computations yielding TT can be seen as proofs of the property because the interpreter itself implements the tactics Aubin calls simplification.A theorem with universally quantified variables is proven by straight computation if when free variables are substituted for quantified variables, the obtained symbolic term computation yields TT without instantiating the introduced free variables.

3. An INDUCTIVE REASONING TASK IN TTEL

The system accepts TTEL type and function definitions, and can ask the TTEL interpreter to compute a ground property. The interpreter provides the system with the resulting value and with a computation trace consisting of a sequence of pairs (rewritten subterm, applied rewrite equation) . The goal is to induce from this evidence a formula which subsumes the ground property, and hopefully is a theorem itself.

First, the system finds the most general theorem whose proof consists in the given computation trace. This is a proper generalization and the system is able to check it. Secondly, the system exploits properties of predicates such as equality to further generalize the theorem. The system does not give a proof of the new theorem, but a proof could easily be obtained by simplification and induction.A final generalization is obtained on the grounds of few rules which might instead yield non valid results.

4. GENERALIZATION BASED ON PROOF

The first clue the system starts from is the given computation trace. In fact, the very same computation trace may prove a theorem stronger than the given one. In the first place, if a function application term is never evaluated during the computation, the term can safely be substituted by a universally quantified variable and the given computation will still be a proof of the obtained theorem. A typical example is provided by the following ground property (function definitions are given in Appendix I)

```
EQLN(A(C(PLUS(S(0),0),NIL),                    (I)
       C(S(0),C(0,NIL))),
     A(C(S(0),C(0,NIL)),
       C(PLUS(S(0),0),NIL) )).
```

Since the terms PLUS(S(0),0) are never evaluated, they can be sustituted by a universally quantified variable, yielding the theorem

```
EQLN(A(C(x,NIL),C(S(0),C(0,NIL)),
     A(C(S(0),C(0,NIL)),C(x,NIL))).
```

The same generalization can be done on those data terms (i.e. terms built on constructors only) whose structure is irrelevant to the computation. The system can distinguish the relevant part of the occuring data terms by resorting to the function rewrite equations, whose formal argument terms describe exactly the most general data term the rewrite equation can be

applied to. The required generalization is obtained by firstly substituting free variables for the maximal data terms in the theorem. Then the resulting symbolic term is evaluated by forcing the computation to exactly follow the given computation trace. The original free variables will turn out to be instantiated only as far as needed to obtain the given computation. Let us work out the above example to clarify the matter.

Suppose the maximal data terms are replaced as
EQLN(A(vl ,v2) ,A(v2,vl) . (2)
Since the computation trace reports that the recursive A equation was applied to the A terms, the variables v_l and v2 need to be instantiated to C(vII,vI2) and C(v21_,v22) respectively. Analogously, v12 is subsequently instantiated to NIL and v22 to C(v23,NIL). Collecting all instantiations, and substituting the still remaining free variables by universally quantified variables, we obtain the theorem
EQLN(A(C(vII,NIL),C(v21,C(v23,NIL))),
A(C(v21,C(v23,NIL)),C(vII,NIL))).
However, if the starting term isEQL(A(vI,v2), A(y_3,v£)) the forced computation is not able to reconstruct the identities of vI to v4, and of v2 to v3. Consequently, the substitution of free variables for input data terms must be done carefully.

The heuristic rule for free variable introduction adopted by the system is based on the following observation. In order to have interesting theorems, the variables occurring in one actual argument term of functions such as EQLN (called balanced functions) should occur in the other argument term as well. The system is able to recognize balanced functions from their definitions (basically because they recur simultaneously on both argument positions). Moreover, the user is allowed to tag functions as being balanced (e.g. the boolean functions AND and IMPLIES may conveniently be handled as balanced functions). Thus, the system substitutes identical data terms with different free variables in one argument term, and if the involved function is balanced, carries the introduced variables over the other argument term. For example, if the ground property is
EQLN(A(C(0,NIL),C(0,NIL)),A(C(0,NIL),C(0,NIL)))
then free variables are introduced as follows
EQLN(A(vI,v2),A(v2,vI))
and forced computation yields the theorem
EQLN(A(C(vII,NIL),C(v21,NIL)),
A(C(v21,NIL),C(vII,NIL))).
Actually, vI and v2 in the second A term could also be introduced the other way around, thus yielding a trivial theorem. The system produces all possible free variables introductions. Some of them are proven by a triviality checker, others are discarded by subsumption.

The limitation of the proof based generalization method stems from its strong dependence on the involved functions. For example, consider the following ground property.
EQL(A(C(0,NIL),A(C(0,NIL),R(NIL))), (3)
A(A(C(0,NIL),C(0,NIL)),R(NIL))).
No free variable introduction helps, because the forced computation will anyway yield back the ground property. In fact, EQL is a much more demanding equivalence relation than EQLN, because EQL accurately checks the list elements by means of EQN.

The generalization method presented in the next Section exploits the presence of equivalence predicates in the theorem, and it is a first start on generalizing the proof.

5. GENERALIZATION BASED ON EQUIVALENCE

Let us describe the role of equivalence by means of example (3). The forced computation starts with the following term (called symbolic input term).
E0L(A(vI,A(v2,R(v3))),A(A(vI,v2),R(v3))).
The computation trace forces the instantiation of vI_ to let the outermost A terms produce (through the recursive A equation) two C terms. Thus the symbolic term is rewritten as follows
EQL(C(vII,A(vI2,A(v2,R(v3)))
C(vII,A(A(vI2,v2),R(v3)))).
The EQL recursive equation is applied yielding
AND(EQN(vII,vII),EQL(A(vI2,A(v2,R(v3))),
A(A(vI2,v2),R(v3)))).
This is the first place where the notion of equivalence can help generalizing. The system notices the occurrence of the term EQN(vII,vII) whose corresponding evaluation in the computation trace yields TT. Since EQN is an equivalence relation, the reflexivity lemma EQN(x,x) can easily be proven, and used to rewrite EQN(vII,vII) to TT.The part of the computation trace describing the evaluation of EQN can be by-passed thus leaving the variable vII free.

More precisely, whenever an equivalence predicate application e(t1,t2) is found during the forced computation, the system attempts to use the reflexivity lemma e(x,x). If t1 and t2 are identical, and are sub-terms of the symbolic input term, a new free variable is introduced for both t1 and t2.

In the given example, the computation proceeds with the evaluation of the EQL term, and by instantiation of v12 to NIL yields
EQL(A(v2,R(v3)),A(v2,R(v3))).
The reflexivity lemma is not applied because the second A term is not part of the symbolic input term but it comes from the computation of the term A(v1,v2). On the contrary, after a few steps the variable v2 ripples out and the evaluating term becomes EQL(R(v3),R(v3)).
Both R(v3) terms satisfy the conditions above, and the system generalize them. Finally, collecting all instantiations, the induced theorem is the following

EQL(A(C(v11,NIL),A(C(v21,NIL),v4)), (4)
A(A(C(v11,NIL),C(v21,NIL)),v4)).

The given computation trace is no more a proof of the induced theorem. Yet, a proof could easily be obtained by a simplification inference rule, and by use of reflexivity lemmas. Hence, the induced formula is actually a theorem.

Since the system exploits only the reflexivity property of the equivalence relations, any reflexive predicate can be handled this way. We did not attempt to let the system discover reflexivity by itself from the predicate definitions. However, the system is able to induce such a property.

Although the formula induced at this point is valid, it is a poor generalization, because the structure of the given data terms may still over-specialize the theorem. We would like for example the C terms in (4) to be generalized, thus obtaining the associativity theorem for A.

6. GENERALIZATION BASED ON ESTIMATED FUNCTION BEHAVIOUR

The data structures still present in the theorem mirror (part of) the structure of the given data terms. A further (and possibly bold) generalization is now required. The first problem is to select data terms as candidates for generalization. The second problem is to check that candidate modifications do not change the computation structure.

The system looks for those data terms which are so general that they appear as "skeletons" for their type. A skeleton of type T is a data term such that: i) All constructors of type T occur in it; ii)In the argument positions of type T'T a variable of type T' occurs.
Examples of skeletons are C(v1,C(v2,NIL)) and S(0), while the following data termsdo not classify as skeletons: TT, because it does not contain the constructor FF, and C(S(0), C(v1,NIL)), because S(0) is not a variable.

Focussing on skeletons embodies a peculiar notion of computation. First of all, the "structure" of data terms and the "structure" of computations are considered to be strongly interrelated, so that the forced computation is believed to be able to reconstruct the details of data terms which are necessary and sufficient to determine the computation flow. Secondly, the skeleton definition requires all constructors for the type to occur in a skeleton, in the assumption that if a constructor would not occur, the given computation trace would not be a representative of all the possible computation traces. Finally, the hierarchical definition of types is believed to induce a corresponding nesting in the computation, and therefore will require hierarchical generalizations, as we will see later.

Still, two different skeletons may be related to each other in such a way that their generalization to two different variables would not be valid. The system adopts just a naive notion of relationship, i.e. two skeletons are related iff they share a universally quantified variable.

Good candidates for generalization are defined to be the maximal unrelated skeletons. However, the given theorem may hold only if the actual data terms obey exactly the structure described by the corresponding skeletons. A classical example is the following
EQN(PLUS(S(0),S(S(0))),TIMES(S(0),S(S(S(0))))).
Hence, the system must test whether candidate skeleton modifications appear to bring about computation structure modifications.

The system test is based on those functions, such as EQN, which simultaneously recur on the-

ir arguments. Simultaneously recurring functions (srf's) implicitly define a measure on the argument values (i.e. the number of recursive applications which are needed to completely "peel down" an argument value). The difference between the argument measures determines which srf base equation is applied to terminate the computation. If the actual argument measures depend at different degree on the measure of the candidate skeleton, a skeleton modificationmay cause a different base equation to be used to terminate the (modified) computation. To be on the safe side, the system discards the candidate.

The measure of a data term cannot be computed a priori since it is defined by the rewrite equations of the functions which recur on it.
In the current implementation a drastic simplification is made, and only data structures which are reflexive on one argument position are tested. Moreover, the actual argument terms of a srf will in general be function application terms (just like in the example). Hence, the system must be able to estimate how functions transform data term measures.

For each function, the system computes an arithmetical expression (called norm) which, roughly speaking, expresses the measure of the function value in terms of the measure of its arguments. Function composition in the argument terms of the srf can be accounted for by introducing two suitable auxiliary functions (defined by composition) such that the srf argument terms can be rewritten as applications *of* the auxiliary functions to data terms only. The auxiliary function norms tell the system how the measures of the the outermost predicate argument values depend on the measures of the skeletons. Since we want to capture the effect of the modification of a specific skeleton, the partial derivatives of the auxiliary function norms w.r.t. the argument position under generalization are computed. The generalization is accepted iff the derivatives can be simplified to the same expression. Note that such a test brings together the function definitions, the form of the theorem, and the computation structure.

For instance, let us consider the candidate S(0) in the example above. The norm of PLUS is pl+p2 (see below), and the norm of TIMES is pl*p2 (the variable symbol pi occurring in norm expressions stands for the measure of the i-th argument term). The partial derivatives of pl+p2 and pl*p2 w.r.t. p1 are 1 and p2, thus S(0) is not generalized.

Since constructors may occur in function definitions, norms for constructors must be given (which, by the way, define the measures of data terms). The norm of a constructor is defined to be 0 if the constructor is not reflexive, otherwise 1+pi if the constructor is reflexive in the argument position i. The norm of a function is computed from the function definition and from the norms of the occurring functions. Basically, the norm of the function used in the base equation is combined with the norm of the function used to perform recursion where the operators are substituted by higher rank operators. The algorithm for computing the norm [8] is rather intricate because many heuristics have been designed to cope with the complexity of function definitions. The computed norm is actually satisfactory for a wide class of functions (examples are reported in Appendix II). However, the estimate of decreasing functions, and of functions with more than one recursive or base equations are far from being adequate.

We can now describe the stepwise generalization method based on estimated function behaviour. Candidate skeletons are recognized in the given theorem, and each candidate is tested as follows. The computation trace is analyzed to pick out those srf application terms in which the candidate occurs. If no such term is found, the candidate skeleton is generalized. If for each srf application term the partial derivatives of the arguments w.r.t. the candidate are the same, then the candidate is generalized, otherwise it is discarded. Candidate generalization may introduce new candidate skeletons which are in turn tested and generalized as above.

7. A COMPLETE EXAMPLE

Let us consider the following ground property

```
EQS(LR(LA(LC(C(PLUS(S(0),0),C(S(0),NIL)),      (EI)
                LNIL),
          LC(C(PLUS(S(0),0),C(S(0),NIL)),
                LNIL))),
     LA(LR(LC(C(PLUS(S(0),0),C(S(0),NIL)),
                LNIL) ),
          LR(LC(C(PLUS(S(0),0),C(S(0),NID),
                LNIL)))).
```

The term PLUS(S(0),0) is never evaluated. A free variable nx is substituted for it,and free variables nIIx and nIIy are introduced to force computation yielding the symbolic terms

EQS(LR(LA(nIIx,nIIy)), (E2)
LA(LR(nIIy),LR(nIIx))) and

EQS(LR(LA(nIIx,nIIy)), (E3)
LA(LR(nIIx),LR(nIIy))).

The forced computation of (E2) gives

EQS(LR(LA(LC(C(nxJ.,C(nx2, NIL)), LNIL), (E4)
LC(C(nyI,C(ny2,NIL)),LNIL))),
LA(LR(LC(C(nyI,C(ny2,NIL)),LNIL)),
LR(LC(C(nxI,C(nx2,NIL)),LNIL)))).

The type LL terms are due to the applications of LA equations, while the type L terms come from the applications of EQLN equations. TheN variables appear because EQLN checks for L length equality only. The forced computation of (E4) gives

E0S(LR(LA(LC(C(nxI,C(nx2,NIL)),LNIL), (E5)
LC(C(nyI,C(ny2,NIL)),LNIL))),
LA(LR(LC(C(nxI,C(nx2,NIL)),LNIL)),
LR(LC(C(nyI,C(ny2,NIL)),LNIL)))).

The generalization method based on equivalence is applied to (E4). The reflexive predicate EQLN application term

EQLN(C(nyI,C(ny2,NIL)) ,C(ny2_,C(nv^,NIL)))

was reduced to TT (note the role of LR). Since the argument terms are identical and do occur in (E4), they are generalized to nIy. Analogously ,the data term C(nxI ,C(nx2,NIL)) is generalized to nIx. The following theorem is thus induced

EQS(LR(LA(LC(nix,LNIL),LC(nix,LNIL))), (E6)
LA(LR(LC(nIy,LNIL)),LR(LC(nIx,LNIL)))).

When the method is applied to (E5),the following EQLN term is found

EQLN(C(nyI,C(n^2,NIL)),C(nxI,C(nx2,NIL)))(E5.1)

Since the argument terms are not identical, (E5) is correctly no more generalized.

Finally, the generalization method based on estimated function behaviour is applied to (E6). The skeletons are LC(nIx,LNIL) and LC(nIy, LNIL), and EQS is the only srf which applies to terms containing them. Since the partial derivatives of LR(LA(pI,p2))and LA(LR(pI),LR(p2)) w. r.t. p1 are bothI, the skeleton LC(nIx,LNIL) is generalized to nIIx. LC(nIy,LNIL) is analogously generalized to nIIy, and the system induces the following theorem

EQS(LR(LA(nIIx,nIIy)),LA(LR(nIIy),LR(nIIx))).

When (E5) is considered, the skeletons C(nxI, C (nx2,NIL)) and C(nyI,C,(n^2,NIL)) are not generalized. In fact, the srf EQLN is directly applied to them (see(E5.1)), i.e. we have a term of the form EQLN(pI,p2). The partial derivatives of p1 and p2 w.r.t. plare 1 and 0, thus the skeleton C(nyI,C(ny2,NIL)) is not generalized. The same happens for C(nxI,C(nx2,NIL)). Again (E5) is correctly no more generalized.

8. COMPARISON WITH RELATED WORK

The inductive method we have presented has been influenced in many respects by previous research in induction. First of all, the generalization based on proof can be classified as a successive refinement method. We do not need to iterate the process of guessing and refining, because the formal calculus environment provides for a powerful technique (forced symbolic computation) to exactly adjust the initial guessing. The use of the given computation trace to strengthen the theorem, relates to Brown' and Tarnlund's temporal method based on proofs [4].

Secondly, we have had the advantage of being able to draw on the ideas of Boyer and Moore *[2]* and of Aubin [I] , who tackle the problem of generalizing the theorem to be proven by induction. One basic problem is that of distinguishing different occurrences *of* a variable. Boyer and Moore generalize terms which the involved functions recur on, thus relating generalization to function definition. In their footsteps, Aubin points out the close relationship among generalization, proof by induction and symbolic computation. His method generalizes those variables which an interpreter would first instantiate during symbolic computation. Thus, only the first and fourth occurrences of x are generalized in the theorem EQL(A(x,A(x,x)),A(A(x,x),x)). We bring the whole computation structure to bear on the problem, and we can capture rather complex relationships, as shown in the example in Section 7. Because of the peculiar role played by the LR function, Aubin's method would incorrectly generalize the first and third occurrence of x in the theorem EQS(LR(LA(x,x)),LA(LR(x),LR(x))).

9. PROPERTY INDUCTION, THEOREM PROVING AND PROGRAM VERIFICATION

Just as one can prove whether a specific induced property does hold, one can prove that the

inductive method itself is guaranteed to induce valid properties only. We have been able to prove this result, with a few limitations [8].

First of all, the induced theorem must have the form b(t1,t2), where b is a boolean function defined according to the schema

b(T,T')=BCONST1 b(c(x,y),T')=BCONST2

b(T,c'(z,w))=BCONST3

b(c(x,y),c'(z,w))=b1(b2(x,z),b(y,w))

b1(BCONST4,y)=BCONST5 b1(BCONST6,y)=y

where BCONSTi are boolean constants (note that BCONST4≠BCONST6), T,T',c,c' type constructors, and b2 a boolean function belonging to the class.

The following definition schemata characterize the class of functions occurring in t1 and t2.

n(T,y)=u(y) n(C(x,y),z)=C'(p(x),n(y,z))

r(T,y)=w(y) r(C(x,y),z)=n'(r(y,z),C'(x,T'))

n'(T',y)=v(y) n'(C'(x,y),z)=C"(x,n'(y,z))

s(x,y)=f(g(x,y),h(x,y))

where u,v,w,p and f,g,h either are constructors or belong to the class (note that n' is a restricted n-function).

The limitations on the form of the property and of the boolean functions are introduced to avoid theorems whose proofs need case analysis. The form of the involved functions is constrained so that the structure of their computation gradually changes when the structure of the data terms is gradually changed. Significant functions, predicates and theorems fall into the class. It is certainly not possible to express significance by a figure, but it may be interesting that about 35% of the theorems listed in [1] and involving single reflexive data types are induced by our system and belong to the above class. Classical examples are EQL(A(x,A(y,z)),A(A(x,y),z)), EQL(R(R(x)),x), and EQL(R(A(x,y)),A(R(y),R(x))). The reflexivity theorems for EQN, EQL, EQLN, and EQS fall into the class and are induced by the system.

At the present, the application of such inductive method to theorem proving has one advantage but suffers from a few limitations. The advantage is that no combinatorial explosion arises in the proof. Free variable introduction is indeed a non-deterministic process, but no nesting of non-deterministic choices is involved. On the other hand, the major limitation of the proposed method is that it is not proven complete. Thus, it can be used only as an auxiliary tool, which may fail, but at least requiring a bounded amount of resources. Moreover, the method can prove only function properties whose restrictions have been described above. These are heavy limitations for a general purpose theorem prover.

In the framework of program verification, where the goal is in general that of proving properties of functions, it is more likely that the inductive method can help, provided that the theorem to be proven (or a subgoal generated during the proof) is within its reach.

The present inductive method has been implemented into an experimental system mainly by modifying existing TTEL symbolic interpreters. No care has been taken to obtain an efficient system.

APPENDIX I

```
(TYPE(EQLN(L,L))=B; {EQLN(NIL,NIL)=TT;
 EQLN(C(x,y),NIL)=FF; EQLN(NIL,C(z,w))=FF;
 EQLN(C(x,y),C(z,w))=EQLN(y,w)})
(TYPE(EQL(L,L))=B; { EQL(NIL,NIL)=TT;
 EQL(C(x,y),NIL)=FF;  EQL(NIL,C(z,w))=FF;
 EQL(C(x,y),C(z,w))=AND(EQN(x,z),EQL(y,w))} )
(TYPE(EQS(LL,LL))=B; { EQS(LNIL,LNIL)=TT;
 EQS(LC(x,y),LNIL)=FF;  EQS(LNIL,LC(z,w))=FF;
 EQS(LC(x,y),LC(z,w))=AND(EQLN(x,z),EQS(y,w))})
(TYPE(AND(B,B))=B;
{AND(TT,TT)=TT;  AND(TT,FF)=FF;
 AND(FF,FF)=FF;  AND(FF,TT)=FF})
(TYPE(PLUS(N,N))=N;
{PLUS(O,y)=y;  PLUS(S(x),y)=S(PLUS(x,y))} )
(TYPE(HANOI(N))=N; { HANOI(O)=O;
 HANOI(S(x))=S(TIMES(S(S(O)),HANOI(x))} )
(TYPE(TIMES(N,N))=N; { TIMES(O,y)=O;
 TIMES(s(x),y)=PLUS(TIMES(x,y),y)} )
(TYPE(A(L,L))=L;
{A(NIL,z)=z; A(C(x,y),z)=C(x,A(y,z))} )
(TYPE(R(L))=L;
{R(NIL)=NIL;  R(C(x,y))=A(R(y),C(x,NIL))} )
(TYPE(LA(LL,LL))=LL;
{LA(LNIL,z)=z;  LA(LC(x,y),z)=LC(x,LA(y,z))} )
(TYPE(LR(LL))=LL; { LR(LNIL)=LNIL;
 LR(LC(x,y))=LA(LR(y),LC(x,LNIL))} )
```

APPENDIX II

We report here the norm computation for some functions given in Appendix I. We look at function definitions in terms of the schema

f(T,y)=g(y) f(c(x,y),z)=h(x,f(y,z)).
If Norm(h) has the form a+p1, then Norm(f)=Norm(g)+a*p1. For example, since Norm(S)=1+p1 and Norm(C)=1+p1, then Norm(PLUS)=Norm(y)+1*p1=p2+p1, and Norm(R)= =Norm(NIL)+Norm(C(x,NIL)*p1=0+1*p1=p1. If Norm(h)=a+b*p1, then Norm(f)=a*(b↑p1)/(b-1)+ +Norm(g)*b↑p1.For example,Norm(HANOI)=1*(2↑p1--1)/(2-1)+0*2↑p1=2↑p1-1, since Norm(g)=0, a=1, b=2.

REFERENCES

[1] Aubin,R."Mechanizing structural induction" Ph.D.Th.,U.of Edinburgh, Dept.of Artif.Intel. (1976).

[2] Boyer,R.S.,and Moore,J S."Proving theorems about LISP functions". J.ACM 22 (1975),129-144.

[3] Brotz,D.K."Embedding heuristic problem solving methods in a mechanical theorem prover". STAN-CS-74-446, Comp. Science Dept., Stanford Univ. 1974.

[4] Brown,F.M.,and Tarnlund,S."Inductive reasoning in mathematics". Proc. IJCAI-7. Cambridge, MIT, Mass., August, 1977, pp.844-850.

[5] Buchanan,B.G."Logic of scientific discov ery". STAN-CS-66-46. Dept. of Comp. Science Stanford Univ., 1966.

[6] Buchanan,B.G., Feigenbaum, E.A.,and Lederberg, J. "A heuristic programming study of theory formation in science". Proc. IJCAI-71, London, IC, September, 1971,pp. 40-48.

[7] Burstall,R.M. "Proving properties of programs by structural induction". Comp. J. *12*, 1 (1969), 41-48.

[8] Degano, P.,and Sirovich, F. "Inductive generalization and proofs of function properties". To appear in Comput.Linguist.Comput.Lang.

[9] Evans,T.G. "A program for the solution of geometric-analogy intelligence test questions". In Semantic Information Processing, M.Minsky Ed., MIT Press, 1968, pp. 271-656.

[10] Feldman,J., Gips,J., Horning,J.J., and Reder,S. "Grammatical complexity and inference". STAN-CS-69-125, Comp. Science Dept., Stanford Univ., 1969.

[11] Hoare.C.A.R. "Recursive data structures". Int.J.Comp.and Inf.Sc.4, 2 (1975),105-162.

[I2] Lenat,D.B. "Automatic theory formation in mathematics". Proc.IJCAI-77 Cambridge, MIT, Mass., August, 1977,pp.866-842.

[I3] Levi,G. and Sirovich, F. "Proving program properties, symbolic evaluation and logical procedural semantics". Lecture Notes in Computer Science, Vol.62. MFCS, Springer, Berlin, 1975, pp. 569-574.

[I4] McCarthy,J., and Hayes,P. "Some philosophical problems from the standpoint of artificial intelligence". In M.I. 4, B.Meltzer, and D. Michie Eds., Edinburgh Univ. Press, 1969, pp.466-502.

[I5] Meltzer,B. "Power amplification for theorem-provers". In M.I. 5, B. Meltzer.and D. Michie Eds., Edinburgh Univ. Press, 1969, pp. 165-179.

[I6] Michalski,R.S. "A system of programs for computer-aided induction: a summary" Proc. IJCAI-77 Cambridge, MIT, Mass.,1977,pp. 619-620.

[I7] Newell,A. "Heuristic programming: ill-structured problems". In Progress in Operational Research, III, Aronofsky J.S. Ed., Wiley & Son-S, 1969, pp. 662-415.

[I8] Plotkin,G.D. "A further note on inductive generalization". In M.I. 6, B. Meltzer,and D. Michie Eds., Edinburgh Univ. Press, 1971, pp. 101-124.

[19] Popplestone,R.J. "An Experiment in automatic induction". In M.I. 5, B. Meltzer, and D. Michie Eds., Edinburgh Univ. Press,1969, pp. 206-215.

[20] Simon,H.A.,and Kotovsky,K. "Human acquisition of concepts for sequential patterns". Psychol.Rev.,70 (1966), 564-546.

[21] Vere,S.A. "Induction of concepts in the predicate calculus". Proc. IJCAI-75 Tbilisi, GSSR, August, 1975, pp. 281-287.

[22] Waldinger.R.J,,and Lee.R.C.T. "PROW: a step toward automatic program writing". Proc. IJCA1-69 Washington, DC, 1969, pp.241-252.

PREDICTION AND SUBSTANTIATION: TWO PROCESSES THAT COMPRISE UNDERSTANDING

Gerald DeJong
Yale AI Project
Box 2158 Yale Station
New Haven, CT 06520
U.S.A.

This paper describes a new approach to natural language processing which results in a very robust and efficient system. The approach taken is to integrate the parser with the rest of the system. This enables the parser to benefit from predictions that the rest of the system makes in the course of its processing. A program, called FRUMP for Fast Reading Understanding and Memory Program, employs this approach to parsing. FRUMP skims articles rather than reading them for detail. The program works on the relatively unconstrained domain of news articles. It routinely understands stories it has never before seen.

INTRODUCTION

In this paper I will argue for a new and different method of natural language analysis. This radical departure from previous systems is motivated by the fact that previous systems did not and could not handle novel real world input. Previous systems either worked only for the specific sentences they were carefully prepared for ([5], [8], [12], [17], [20], [21], [22]) or the domain of input sentences that could be handled was excessively and artificially constrained ([2], [3], [7]).

This paper presents an approach to natural language analysis which has resulted in a very robust system without imposing excessive constraints on the domain of the input text. As we will see, these advantages are directly traceable to to the fact that the parser is heavily integrated into the rest of the understanding system.

A program has been written which embodies the theory of parsing. The program (FRUMP for Fast Reading Understanding and Memory Program) skims input text for important facts rather than reading it for detail. FRUMP is often able to correctly process English text which has never before been seen by either the program or its programmers. Furthermore, the domain of the program is not excessively constrained. FRUMP is designed to work on news articles. The program can process text from diverse domains such as reports of plane crashes, countries establishing diplomatic ties, forest fires, and wars. A UPI news wire is connected to the Yale computer to provide real world data for FRUMP. FRUMP routinely understands actual news articles from the UPI news wire and notifies logged-in users by sending a summary to their terminals.

The radical restructuring of the understanding process enables FRUMP to be very efficient. Most news articles are processed in less than 20 seconds of CPU time on a DEC System 20 computer. FRUMP can easily keep up with the rate news stories arrive over the UPI news wire. We estimate that about 30% of the articles from. the news wire are understandable by a script processor such as FRUMP. FRUMP understands about a third of these. That is, about 10% of the wire stories. There are several reasons why FRUMP does not attain the theoretical limit of 30%. In order of importance, they are lack of world knowledge, lack of vocabulary, and insufficient knowledge about English sentence structure. This is not to say that FRUMP will not eventually achieve the 30% figure. However, a concerted effort to supply the missing information will be necessary. Such a concerted effort will have to wait while FRUMP is in its research stages.

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under contract N00014-75-C-1111.

DEFICIENCIES OF PREVIOUS SYSTEMS

Most previous natural language understanding programs have been made up of at least two separate subsystems ([5], [8], [17], [21], [22]). A parser subsystem analyzes the input natural language text into some intermediate form. The intermediate representations range from surface representations like syntactic parse trees ([19], [10]) to simplified English [12] to representations involving conceptual primitives [14]. An Inferencer subsystem then builds a representation of the meaning of the input text. This involves incorporating the parser output into the meaning representation, inferring any missing events, and supplying the causal connections between events.

Whatever the form of the intermediate representation, its purpose is always to isolate the inferencer from the capriciousness of natural language. The inference process can be made simpler and more efficient if the input to the inferencer always appears in some canonical form. The intermediate representation provides this canonical form. However, in all of these systems one gets the feeling that there are actually two parsers: the one everyone admits to having which produces the intermediate representation, and a second one hidden in the "inferencer" which parses the output of the first parser to decide what it really means.

There are several systems which do not separate parsing from semantic analysis. In Winograd's SHRDLU [21] parsing and semantic interpretation routines could call each other freely. However, the parsing was always done first. The major difference is that the above systems parsed a entire sentence at a time while SHRDLU parsed less than a sentence at a time. The SOPHIE system [2] analyzes English text by using a grammar which contains both syntactic and semantic rules. This "semantic grammar" approach [3] makes the system very robust. However, since semantic rules (which are domain specific) are intertwined with domain independent rules, the system cannot easily be extended to new domains. A new semantic grammar must be written to process text from a different domain. A similar approach was taken by Epstein and Walker [7] in constructing a natural language front end to a medical data base. The grammar rules contain domain specific knowledge. Again this has the effect of tying the system to one domain.

None of the above systems worked well on real world natural language input from different domains. Either the systems were too fragile to process anything but the example inputs they were designed for, or they constrained the domain of inputs to the point that the system could not be readily extended to new input domains.

Making the parser responsive to the needs of the understander means that the component modules of the system must be constantly communicating. Furthermore, the text will be interpreted in a goal-directed manner. The parser will only try to build meaning structures that are desired by other modules of the system. Text will be interpreted a little bit at a time with a parser that has a very good idea of what it ought to produce, and only in a rich predictive context set up by the expectations of the understander. Furthermore, in such a system there is no need for an intermediate representation produced by a parser artificially separated from the rest of the system. Rather a natural language system as a whole will behave as one integrated process.

Recently inferencers have become more and more predictive in nature. Compare, for example, the MARGIE inferencer [13] to the SAM script applier [5]. Rieger's inferencer tried to generate all possible inferences from every input. SAM, on the other hand, had ready made strings of inferences for various situations. It looked for situations in which to apply its ready made inferences rather than wildly making every possible inference. A somewhat similar approach was taken by Charniak in his system Ms. Malaprop [4], Rumelhart's work on schemas [15] is also in the same vein. The HEARSAY II system [9] uses a predictive hypothesize-and-test paradigm. The GUS system [1] is also predictive. It carries on a conversation about trips much as a travel agent would. It has built in general specifications of what it needs to find out and directs the conversation accordingly.

Thus, it has become increasingly clear that an inferencer must know what kinds of inputs to expect in order to make sense of them. The existence of so many "frame-like" systems illustrates this. While a frame [11] is a very broad concept, all frame-like systems have one thing in common: they are largely top down processors. That is, they predict characteristics of an input before it is seen.

And yet, in natural language processing, there has been little attempt to take advantage of the inferencer's predictions when parsing the natural language text. The conventional design of natural language systems makes effective

communication between the inferencer an the parser extremely difficult.

FRUMP PARSING

FRUMP does not suffer from these problems. FRUMP is a natural language understander that skims newspaper articles on many different domains. The system has two modules: a PREDICTOR and a SUBSTANTIATOR. One module predicts constraints on what might happen next. The other module tries to justify and give substance to these predicted characterizations of the next possible events. The communication between the two modules is relatively unconstrained. Predictions and communication are on a conceptual level. FRUMP uses a conceptual representation scheme called conceptual dependency [16].

The process of giving substance to the predicted characterizations is very purposeful. Text is analyzed and inference routines are called only in response to PREDICTOR'S anticipations of events. Thus SUBSTANTIATOR can channel its processing. SUBSTANTIATOR tries to interpret the text in a way that matches one of the outstanding predictions. It also applies only those inference rules which might be able to help satisfy the predictions.

In FRUMP, text analysis is driven from PREDICTOR'S predictions, not from the input. FRUMP does not have conventional parser which produces conceptualizations when it is presented with an input sentence. Rather, the input text is parsed only when PREDICTOR wants a specific piece of information and SUBSTANTIATOR has decided that the missing information might be found in the text. The text is only examined a little bit at a time, and then only to substantiate specific predictions.

Communication between the two modules assures that they are both operating in the richest and most constrained context possible. When SUBSTANTIATOR verifies a prediction, it tells PREDICTOR the actual conceptualization it found. On the basis of the additional information included in the fleshed out conceptualization, PREDICTOR can refine its predictions. It might be able to make a previous prediction more explicit, it might take back previous predictions, and it might make new ones. These refined predictions are communicated to SUBSTANTIATOR to redirect its efforts. Thus the text analyzer and the inference processes of SUBSTANTIATOR are always operating in the most complete and constrained context that PREDICTOR can provide.

THE PREDICTOR

The PREDICTOR'S job is to make predictions about what events are likely to occur given the current context. The predictions can be on almost any conceptual level. That is, PREDICTOR can anticipate that particular a event is likely, or it can predict characteristics of a small part of an event, or it can predict groups of events. The prediction mechanism is centered on the idea of a sketchy script.

A sketcfty sqrj.pfr is a data structure that organizes FRUMP's knowledge about the world. Each sketchy script is the repository for the knowledge FRUMP has about what can occur in a given situation. FRUMP currently has sketchy scripts for forty eight different situations ranging from earthquakes to countries establishing diplomatic ties to labor strikes. When FRUMP realizes it is reading a story about a particular situation, it applies the knowledge from the corresponding sketchy script in order to predict what events are likely to occur. FRUMP's understanding process includes constructing representations from the input text and inferring missing but implied events.

Scripts have been used before for natural language processing ([18], [5]). However, they were detailed scripts. A detailed script contains all of the events that might occur in a situation; a sketchy script contains only the important events.

The following is an example of one of FRUMP's sketchy scripts. These are the facts that FRUMP knows about kidnappings. The facts in the sketchy script are represented in conceptual terms. The English meanings of the conceptual representations are given here.

1) The kidnappers will probably communicate a ransom demand to the family, company, or government of the person kidnapped.

2) The local police, FBI or other police agencies might be called in

3) The ransom may or may not be met

4) If the ransom is met, predict that the kidnapped person will probably be released but might continue to be held or be killed

5) If the ransom demand is not met, predict that the person will likely be held longer or killed but might be released

6) The kidnappers may or may not be apprehended

7) If the kidnappers are caught, predict a court case trying them for kidnapping

In addition to these facts, FRUMP knows that the important data in a kidnapping are the identity of the kidnappers, the identity of the kidnapped person, the identity of the group or person to whom the ransom demand was made, the nature of the ransom demand and which of the above predictions were found in or inferred from the text.

The identities of the participants in the sketchy script are script variables. In understanding a text, FRUMP both tries to find instances of the predicted facts and to bind the script variables to the identities given in the text.

PREDICTOR uses sketchy scripts to predict the likely events in a situation. When FRUMP has decided on a sketchy script to use in understanding a text, PREDICTOR asks SUBSTANTIATOR to produce the conceptual dependency representations that make up that sketchy script. That is, it predicts the important events in that situation as specified by the sketchy script.

SUBSTANTIATOR often produces only a part of a predicted conceptual representation as it did at the beginning of the example. When this happens, PREDICTOR individually predicts the missing pieces of the conceptualization. In this way it leads SUBSTANTIATOR through the process of building up a complete conceptualization.

Of course, before PREDICTOR can take advantage of the important events in a sketchy script, it must decide to use that sketchy script from among all of its sketchy scripts. This is a serious problem, but a fast efficient solution has been found. It is described in some detail in [6]. Space prohibits discussing the sketchy script selection algorithm here. Suffice it to say that any of FRUMP's 48 sketchy scripts can be selected efficiently. Top down predictions are still given to SUBSTANTIATOR during the selection process. The complexity of the selection algorithm proportional to the log of the number of scripts in the system.

THE SUBSTANTIATOR

There are three ways SUBSTANTIATOR can flesh out or verify a prediction. SUBSTANTIATOR is composed itself of four modules. One for each of the three substantiation methods and a module to chose which substantiation method to use for any given prediction.

The three modules that actually build the conceptual dependency structures are the conceptualization inferencer, the text analyzer and the conceptual dependency role inferencer. SUBSTANTIATOR's selection routine decides which of the structure building modules to try first in substantiating a given prediction. If the selected module fails, another may be tried.

The conceptualization inferencer is the simplest of the structure building modules. This module is able to infer entire conceptualizations at once. The inferences are always script-related. For example, in a story about countries establishing diplomatic relations, this module is able to infer that if the U.S. establishes diplomatic ties with China, then very likely China established similar diplomatic ties with the U.S. as well. This module has little to do with top-down parsing and so will not be discussed further.

The text analyzer builds only a piece of a conceptual structure at a time. When the text analyzer is asked to find a role filler, it tries to predict the sentence location of the desired word. Syntactic knowledge is used as heuristics to locate desired words. For example, the module knows where to find the subject of a verb, prepositional object, adjective modifiers, etc.

The conceptual dependency role inferencer also adds only a piece of a conceptual structure at a time. However, it uses information already understood to infer a missing conceptual role. The role inference rules are indexed by conceptual acts and within each act by the role it can add. Applicable inference rules can therefore be found quickly.

The SUBSTANTIATOR selection routine decides which of the modules will be used in satisfying a prediction. If the prediction is an entire conceptualization, it tries the conceptualization inferencer first. If that fails, it calls the text analyzer to find a part of the predicted conceptualization. Any successes are reported back to PREDICTOR so that it may use the information to revise its predictions.

If a prediction is of a role and its filler, SUBSTANTIATOR uses either the text analyzer or the role inferencer. The module used depends on how accurately each estimates it can fill the role. The parsing heuristics and inference rules are tagged with how well they can be expected to work. The selection routine chooses the module that can fill the missing role the most certainly. If the selected method cannot add the desired role filler, the less certain techniques are tried.

Now we can see how natural it is to divide the system into a prediction module and a verification module. As far as the PREDICTOR is concerned, there is no difference between the text analyzer and the role inferencer. PREDICTOR asks for a role to be filled in a certain way and is told later whether or not it could be done. Both the role inferencer and the text analyzer work on a small piece of a conceptualization at a time. Both are guided by the predictions made for the desired role filler. The only difference between the effect of text analysis and a role added by inference is that the text analyzer tends to be more certain of its result. Inferences, by their very nature, are at best good guesses. Thus the text analyzer is in a sense the inferencer of first resort. Since text analysis and role inferences are treated exactly the same, they belong in the same module.

AN EXAMPLE

INPUT:

THE CHILEAN GOVERNMENT HAS SEIZED OPERATIONAL AND FINANCIAL CONTROL OF THE EL TENIENTE MINING COMPANY, ONE OF THE THREE BIG COPPER ENTERPRISES HERE IN WHICH UNITED STATES COMPANIES HAVE INTERESTS. WHEN THE KENNECOTT COPPER COMPANY, THE OWNERS, SOLD A 51 PER CENT INTEREST IN THE COMPANY TO THE CHILEAN STATE COPPER CORPORATION IN 1967 IT RETAINED A CONTRACT TO MANAGE THE MINE. ROBERT HALDEMAN, EXECUTIVE VICE PRESIDENT OF EL TENIENTE, SAID THE CONTRACT HAD BEEN "IMPAIRED" BY THE LATEST GOVERNMENT ACTION. AFTER A MEETING WITH COMPANY OFFICIALS AT THE MINE SITE NEAR HERE, HOWEVER, HE SAID THAT HE HAD INSTRUCTED THEM TO COOPERATE WITH EIGHT ADMINISTRATORS THAT THE CHILEAN GOVERNMENT HAD APPOINTED TO CONTROL ALL ASPECTS OF THE COMPANY'S OPERATIONS.

SELECTED SKETCHY SCRIPT $NATIONALIZE

CPU TIME FOR UNDERSTANDING = 3646 MILLISECONDS

ENGLISH:

CHILE HAS NATIONALIZED AN AMERICAN OWNED COMPANY.

FRENCH:

LE CHILI A NATIONALISE SOCIETE AMERICAINE.

CHINESE:

JYHLIH BAA I GE MEEIGWO GONGSY SHOUGUEIGWOYEOULE.

SPANISH:

CHILE NATIONALIZO UNA COMPANIA PREVIAMENTE CONTROLADO POR INTERESES NORTE AMERICANOS.

CONCLUSION

FRUMP has been a very successful program. It often understands new input directly from the UPI news wire. This is in large part due to the robustness of FRUMP's parsing algorithm. FRUMP succeeds where previous natural language systems failed because previous systems made parsing a much harder problem than it actually is. Those systems required their parsers to analyze input text without the benefit of the knowledge in the rest of the understanding system. Even systems which used predictive understanders did not communicate those predictions to their parsers.

There are limitations to FRUMP's applicability. FRUMP can understand only situations for which it has a sketchy script, and there are some situations for which no sketchy script can be written. For example, FRUMP cannot understand editorials. Editorials are attempts to persuade the reader. This usually involves giving the reader arguments for adopting a particular view. However, the arguments are usually of a novel form. They often explain a heretofore unrealized consequence of adopting the right or wrong position. The sketchy script for a situation must include all of the important facts of that situation that FRUMP is to understand. There is no way to anticipate these novel, unforeseen ramifications at the time the sketchy script is written. Editorials by their very nature are not script-like and so cannot be processed by a script understander.

However, this does not argue that FRUMP's parsing techniques cannot be applied to a larger class of articles. It is a weakness of FRUMP's PREDICTOR module, not a breakdown in the PREDICTOR/SUBSTANTIATOR dialogue, that makes understanding these articles impossible for FRUMP. A more powerful PREDICTOR which can provide the SUBSTANTIATOR module with accurate characterizations of possible inputs for these

stories is needed. With a more powerful PREDICTOR, FRUMP-like parsing could be applied to a much larger class of text inputs.

REFERENCES

[I] Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS, a frame driven dialog system.

[2] Brown, J. S., and Burton, R. R. (1975). Multiple representations of knowledge for tutorial reasoning. In D. Bobrow and A. Collins (Eds.), Representation and Understanding, Academic Press, New York.

[3] Burton, R. R. (1976). Semantic grammar: an engineering technique for constructing natural language understanding systems. BBN Report 3453, ICAI Report 3. Bolt Beranek and Newman Inc., Boston.

[4] Charniak, E. (1977). Ms. Malaprop, a language comprehension program. Proceedings of IJCAI-77, Cambridge, MA.

[5] Cullingford, R. (1978). Script application: computer understanding of newspaper stories. Ph.D. Thesis, Yale University, New Haven, CT.

[6] DeJong, G. (1979). Skimming stories in real time. Ph.D. Thesis, Yale University, New Haven, CT.

[7] Epstein, M. and Walker, D. (1978). Natural language access to a melanoma data base. SRI Technical Note 171, Menlo Park, CA.

[8] Kaplan, R. M. (1975). On process models for sentence analysis. In D. A. Norman and D. E. Rumelhart (eds.), Explorations in cognition. w. H. Freeman and Company, San Francisco.

[9] Lesser, V. and Erman, L. (1977). A retrospective view of the HEARSAY-II architecture. Proceedings of IJCAI-77, Cambridge, MA.

[10] Marcus, M. (1977). A theory of syntactic recognition for natural language. Unpublished Ph.D. thesis, MIT, Cambridge, MA.

[11] Minaky, M. (1975). A framework for representing knowledge. In P. H. Winston (ed.), The Psychology of Computer Vision. McGraw-Hill, New York, NY.

[12] Parkinson, R., Colby, M., and Faught, W. (1976). Conversational language comprehension using integrated pattern-matching and limited parsing. Technical Report, UCLA Psychiatry Department, Los Angles, CA.

[13] Rieger, C. (1974). Conceptual memory. Ph.D. Thesis, Computer Science Department, Stanford University, Stanford, CA.

[14] Riesbeck, C. K. (1975). Conceptual analysis. In R. Schank (ed.), Conceptual Information Processing- North Holland, Amsterdam.

[15] Rumelhart, D. (1975). Notes on a schema for stories. In D. Bobrow and A. Collins

Academic Press, New York.

[16] Schank, R. C. (1973). Identification of conceptualizations underlying natural language. In R. C. Schank, and K. Colby (eds.), Computer Models of Thougftt and Language. W. H. Freeman and Co., San Francisco.

[17] schank, R. c. (1975). Conceptual

Amsterdam.

[18] Schank, R. C. and Abelson, R. P. (1977). Scripts. Plans, Goals, and Understanding. Lawrence Erlbaum Press, Hillsdale, NJ.

[19] Thorne, J., Bratley, P., and Dewar, H. (1968). The syntactic analysis of English by machine. In D. Michie (ed.), Machine Intelligence 3, American Elsevier Publishing Company, New York.

[20] Wilks, Y. (1973). An artificial intelligence approach to machine translation. In R. C. Schank and K. Colby (eds.), Computer Models of Thought and Language.. W. H. Freeman and Co., San Francisco.

[21] winograd, T. (1972). understanding. Natural language. Academic Press, New York.

[22] Woods, W. A. (1970). Transition network grammars for natural language analysis. Communications al the ACH- vol. 13, p. 591.

LEARNING AND GENERALIZATION OF CHARACTERISTIC DESCRIPTIONS: EVALUATION CRITERIA AND COMPARATIVE REVIEW OF SELECTED METHODS

Thomas G. Dietterich
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Ryszard S. Michalski
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Some recent work in the area of learning structura1 descriptions from examples is reviewed in light of the need in many diverse disciplines for programs which can perform conceptual data analysis. Such programs describe complex data in terms or logical, functional, and causal relationships which cannot be discovered using traditional data analysis techniques. Various important aspects of the problem of learning structural descriptions are examined and criteria for evaluating current work is presented. Methods published by Buchanan, et.al.[1-3,20), Hayes-Roth [6-91, and Vere [22-25], are analyzed according to these criteria and compared to a method developed by the authors. Finally some goals are suggested for future research.

1. INTRODUCTION

1.1 Motivation and Basic Concepts

There are many problem areas where large volumes of data are generated about a class of objects, the behavior of a system, a process, etc. scientists in fields as diverse as agriculture, chemistry, and psychology are faced with tne need to analyze such data in order to detect regularities and common patterns. Traditional tools for data analysis include various statistical techniques, curve-fitting techniques, numerical taxonomy, etc. These methods, however, are often not satisfactory because they impose an overly restrictive mathematical framework on the scope of possible solutions. For example, statistical methods describe the data in terms of probability distribution functions placed on random variables. As a result, the types of patterns which they can discover are limited to those which can be expressed by placing constraints upon the parameters of various probability distribution functions. Because of the mathematical frameworks upon which they are based, traditional methods cannot detect conceptual patterns such as the logical, causal, or functional relationships that are typical of descriptions producer' by numans. This is a well-known problem in AI, namely that a system in order to learn something must first be able to express it. The solution requires introducing more powerful representations for hypotheses and developing corresponding techniques of data analysis and pattern discovery, work done in AI and related areas on computer induction and learning structural descriptions from examples has laid the groundwork for researh in this area. This is not accidental, because, as Michie [18] has pointed out, the development of systems which deal with problems In human conceptual terms is a fundamental characteristic of AI research.

In this paper, we examine some of the recent work in AI on the subject of learning and generalization of structural descriptions. In particular, we will review four recent methods of inductive generalization: Buchanan et. al., Hayes-Roth, Vere, and our own work (Earlier well-known work by Winston was recently reviewed by Knapman (11)). We also outline some goals for research in this area. Attention is given primarily to the simplest form of generalization, namely the maximally specific conjunctive statements which characterize a single set of input events (called for short, conjunctive generalizations). The reason for this choice is that most work done in this area is addressing this, quite restricted, subject. Many of the researchers whose work we review in this paper have done work on other aspects of machine learning including generalization usinc negative examples (Vere, Michalski) ana developing discriminant descriptions of several classes of objects (Michalski). Due to space limitations, we have been unable to include these topics in this paper. Instead, thhese contributions are mentioned in the sections concerning extensions. We begin the analysis by first discussing several important aspects of the problem of learning conceptual descriptions:

- types of descriptions: characteristic versus discriminant
- forms of descriptions
- types of generalization processes Involved in generalizing descriptions (rules of generalization)
- constructive versus non-constructive induction
- general versus problem-oriented methods of induction.

1.2 Types of Descriptions

We distinguish between characteristic and discriminant descriptions [16]. A characteristic description is a description of ii single set ot objects (examples, events) which is Intended to discriminate that set of objects from all other possible objects. For example, a characteristic description of the set of all tables would discriminate any table from all things which are non-tables. Psychologists consider this problem under the name of concept formation (e.g. Hunt [10]). Since it is impos-

The authors gratefully acknowledge the support of NSF under grant MCS-76-22940.

sible to examine all other possible objects, a characteristic description Is usually developed by specifying all characteristics which are true for all known objects of the class (positive examples). Alternatively, In some problems there are available so-called near misses" which can be used to more precisely circumscribe the given class.

A discriminant description is a description of a single class of objects in the context of a fixed set of other classes of objects. It states only those properties of objects in the class under consideration which are necessary to distinguish them from the objects in the other classes. A characteristic description can be viewed as a discriminant description in which the given class is discriminated against infinitely many alternative classes.

In this paper we restrict ourselves to the problem or determining characteristic descriptions. The problem of determining discriminant descriptions has been studied by Michalski and his collaborators [13-17]).

1.3 Forms of Descriptions

Descriptions, either characteristic or discriminant, may take several forms. In this paper we concentrate on generalizations in conjunctive form. Other forms include disjunctions, exceptions, production rules of various types, hierarchical and multilevel descriptions, semantic nets, and frames.

1.4 Generalization Pules

The process of inducing a general description from examples can be viewed as a process of applying certain generalization rules to the initial descriptions to transform them Into more general output descriptions. This viewpoint Permlts one to characterize various methods of nduction by specifying the rules of generalization which they use. Below is a brief review of various generalization rules based on the paper [17].

1) Dropping Condition Rule. If a description is viewed as a conjunction of conditions which must be satisfied, then one way to generalize it is to drop one or more of these conditions. For example:

$$\mathbf{red(x) \wedge big(x) \ |< \ red(x)}$$

(this reads: "the description 'xs which are red and big' can be generalized to the description 'xs which are red ; |< denotes the generalization operator)

ii) Turning Constants to Variables Pule. If we have two or more descriptions, each of which refers to a specific object (in a set to be characterized), we can generalize these by creating one description which contains a variable in place of the specific object:

$$\begin{matrix} \mathbf{tall(Fred) \wedge man(Fred)} \\ \mathbf{tall(Jim) \wedge man(Jim)} \end{matrix} \ \mathbf{|< \forall x \ tall(x) \wedge man(x)}$$

assuming that the value set of x Is {Fred, Jim, ... }• 'x' can be Interpreted as representing 'a person from the group under consideration.'

These first two rules of generalization are the rules most commonly used In the literature on computer induction. Both rules can, however, be viewed as special cases of the following rule.

ill) Generalizing by Internal Disjunction Rule. A description an be generalized by extending the set of values that a descriptor (i.e. variable, function, or predicate) is permitted to take on in order that the description is satisfied. This process involves an operation called the internal disjunction. For example:

$$\begin{matrix} \mathbf{shape(x,square)} \\ \mathbf{shape(x,triangle)} \end{matrix} \ \mathbf{K}$$

$$\mathbf{shape(x,(square\ or\ triangle\ or\ rectangle))}$$

where statements on the left of |< describe some single objects in a class, and the statement on the right is a plausible generalization.

Using the notation of variable-valued logic system VL_{21} [17] this rule can be expressed somewhat more compactly:

$$\begin{matrix} \mathbf{[shape(x)=square]} \\ \mathbf{[shape(x)=triangle]} \end{matrix} \ \mathbf{K}$$

$$\mathbf{[shape(x)=square,\ triangle,\ rectangle]}$$

The ',' in the expression on the right of the |< denotes the internal disjunction. Although it may seem at first glance that the internal disjunction is just a notational abbreviation, this operation appears to be one of the fundamental operations people use in generalizing descriptions.

In general this rule can be expressed:

$$\mathbf{W[L = R1] \ |< \ W[L = R2]}$$

where W is some condition and where R1 and R2 are sets of values linked by internal disjunction, and R1 R2.

There are two important special cases of this rule. First, when the descriptor involved takes on values which are linearly ordered (a linear descriptor) and the second when the descriptor takes on values which represent concepts at various levels of generality (a structured descriptor).

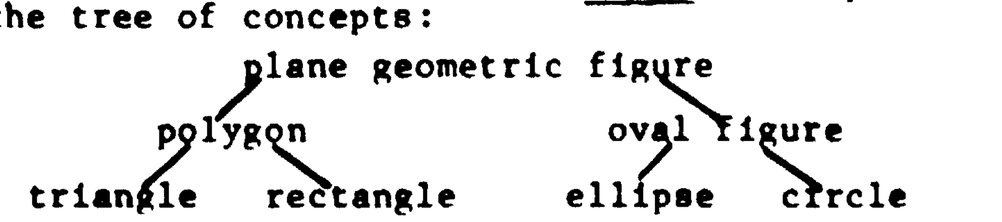
In the case of a linear descriptor we have:

iv) Closing Interval Rule. For example, suppose two objects of the same class have all the same characteristics except that they have different sizes, a and b. Then, it is plausible to hypothesize that all objects which share these characteristics but which have sizes between a and b are also in this class.

$$\begin{matrix} \mathbf{W\ [size(x1)=a]} \\ \mathbf{W\ [size(x2)=b]} \end{matrix} \ \mathbf{|< \ W\ [size(x) = a..b]}$$

In the case of structured descriptors we have:

v) Climbing Generalization Tree Rule. Suppose the value set of the shape descriptor is the tree of concepts:



With this tree structure, values such as triangle and rectangle can be generalized by climbing the generalization tree:

$$\begin{matrix} \mathbf{[shape(x)=rectangle]} \\ \mathbf{[shape(x)=triangle]} \end{matrix} \ \mathbf{|< \ [shape(x)=polygon]}$$

1.5 Constructive Induction

Most methods of induction produce descriptions which involve the same descriptors which were present in the initial data. These methods operate by selecting descriptors from the input data and putting them into a form which is an appropriate generalization. Such methods perform non-constructive induction. A method performs constructive induction if it includes mechanisms which can generate new descriptors not present in the input data. These new descriptors are generated by applying rules of

constructive Induction. Such rules may be written as procedures or as production rules and may be based on general knowledge or on problem-oriented knowledge (for examples of constructive generalization rules see [17]). Constructive Induction rules can interpret the input data in terms of knowledge about the problem domain. Frequently, the solution to a problem is dependent upon finding the proper description for the problem; as in the mutilated checkerboard problem. An inductive program should contain facilities for constructive induction including a library of general constructive induction rules. Tne user should be able to suggest new rules for the program to examine. In order to activate those rules which would be most useful, the program must be able to efficiently search the space of possible constructive induction rules.

Programs which perform constructive induction are more likely to find useful and interesting patterns in complex data since they have the ability to examine the data using many different representations.

1.6 General versus Problem-oriented Methods

It is a common view that general methods of induction, although mathematically elegant and theoretically applicable to many problems, are in practice very inefficient and rarely lead to any interesting solutions. This opinion seems to have lead certain workers to abandon (at least temporarily) work on general methods and concentrate on some specific problem (e.g., Buchanan, et. al. [1,2,3] or Lenat [12]). This approach often leads to interesting and practical solutions. On the other hand, it is often difficult to extract general principles of induction from such problem-specific work. It is also difficult to apply such special-purpose programs to new areas.

An attractive possibility for solving this dilemma is to develop methods which incorporate various general principles of induction (including constructive induction) together with mechanisms for using exchangeable packages of problem-specific knowledge. In this way a general method of induction, provided with an appropriate package of knowledge, could be both easily applicable to different problems and also efficient and practically useful. This idea underlies the development of the INDUCE programs [14,17,4).

2. COMPARATIVE REVIEW OF SELECTED METHODS

2.1 Evaluation Criteria

We evaluate the selected methods of induction in terms of several criteria considered especially important in view of the remarks in section 1.

i) Adequacy of the representation language. The language used~TF represent input data and output generalizations determines to a large extent the quality and usefulness of the output descriptions. Although it is difficult to assess the adequacy of a representation language out of the context of some specific problem, recent work in AI has shown that languages which treat all phenomena uniformly must sacrifice descriptive precision. For example, researchers who are attempting to build natural-language systems prefer the richer knowledge representations such as frames and semantic nets (with their tremendous variety of syntactic forms) to more uniform and less structured representations such as attribute-value lists and PLANNER-style databases. In our own work on inductive learning, we have chosen to use the representation language VL_{21} (see below) which has a wider variety of syntactic forms than our earlier language VL,. Although languages with many syntactic forms do provide greater descriptive precision, they also make the induction process more complex. In order to control this complexity, a compromise must be sought between uniformity and richness of forms. In the evaluation of each method, a review of the operators and syntactic forms of each description language is provided.

ii) Rules of generalization implemented. The generaiization rules implemented in each algorithm are listed.

iii) Computational efficiency. The exact analysis of the computational efficiency of these algorithms is very difficult due both to the inherent complexity of the algorithms and to the lack of precise formulations of the algorithms in available publications. However, it seems useful to have some data comparing the efficiency of these algorithms even if that data is approximate and based on hand-simulations. To get some indication of the efficiency we measure the total number of description generations or comparisons required by each method to perform a test example (see Fig. 1). We also measure the ratio of the number of output conjunctive generalizations to the total number of generalizations examined on this example• Since these numbers are derived from only one example, it Is not appropriate to draw strong conclusions from them concerning the general performance of the algorithms. Our conclusions are based primarily on the general behavior of the algorithms.

iv) Flexibility and extensibility. Mere conjunctive characteristc generalizations are not particularly useful for conceptual data analysis because of their limited format and their lack of formal mechanisms for handling errors in the input data. It is important in evaluating these algorithms to consider the ease with which each method could be extended to

a) discover descriptions with forms other than conjunctive generalizations (see section 1.3),

b) include mechanisms which facilitate the detection of errors in the input data,

c) provide a general facility for incorporating domain-specific knowledge into the induction process as an exchangeable package (Ideally, the domain-specific knowledge should be isolated from tne general-purpose inductive process.), and

d) perform constructive induction.

It is difficult to assess the flexibility and extensibility of the algorithms presented here. We base our evaluation on the general approaches of the methods and on extensions which nave already been made to them.

In the following sections, we describe each method by presenting the description language used, sketching the underlying algorithm, and evaluating the method in terms of the above criteria. Each method will be illustrated using the test example shown in Fig. 1.

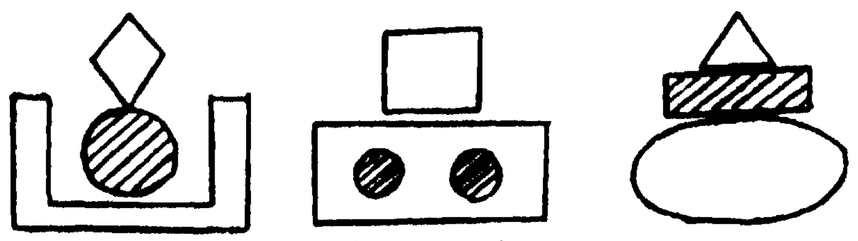
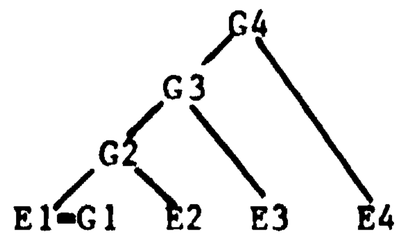


Figure 1

2.2 Data-driven Methods; Hayes-Roth and Vere.

Methods can be divided into bottom-up (data-driven), top-down (model-driven), and mixed methods. Bottom-up methods generalize the input events pairvlse until the final conjunctive generalization is computed:



G2 is the set of conjunctive generalizations of El and E2. Gi is the set of conjunctive generalizations obtained by taking each element of Gi-1 and generalizing it with El.

We consider here only the methods described by Hayes-Roth and Vere. Other bottom-up methods Include the candidate elimination approach described by Mitchell [19] and the Uniclass method described by Stepp [21].

2.2.1 Hayes-Roth: Program SPROUTER [6-9]

Hayes-Roth uses the term maximal abstraction or interference match for maximally specific conjunctive generalization. He uses parameterized structural representations (PSRs) to represent both the input events and their generalizations. For example, consider the two events described in Fig. 2:

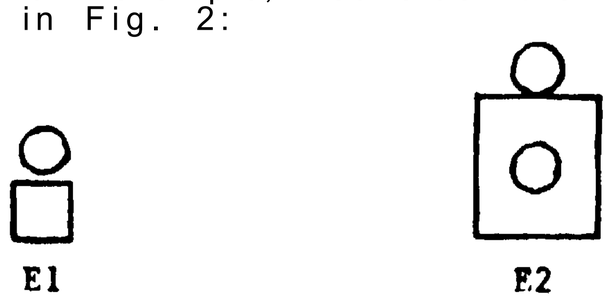


Figure 2

The PSRs for these could be:

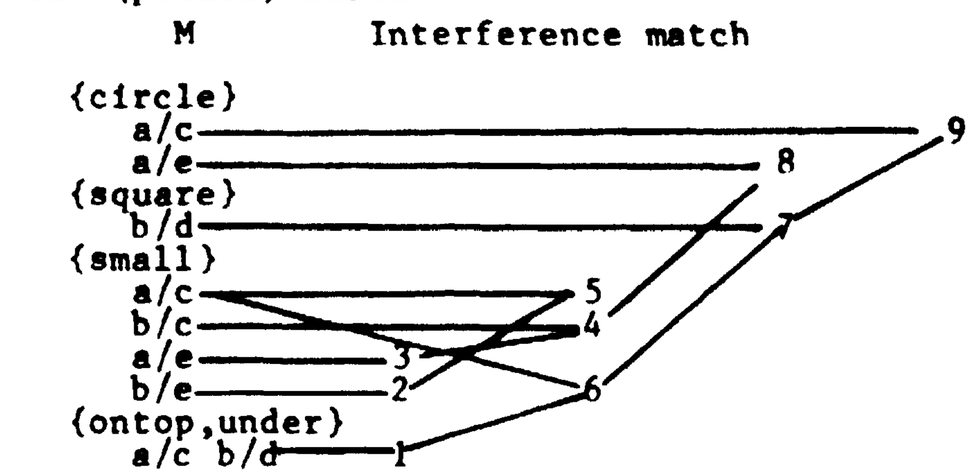
```
E1: {{circle:a}{square:b}{small:a}
     {small:b}{ontop:a, under:b}}
E2: {{circle:c}{square:d}{circle:e}
     {small:c}{large:d}{small:e}
     {ontop:c, under:d}
     {inside:e, outside:d}}
```

The expressions such as {small:a} are case frames made up of case labels (small, circle, etc.) and parameters (a, b, c, d). *The* PSP can be Interpreted as a conjunction of predicates of the form small(a) where the parameters are exlstentlally quantified variables which are assumed to be distinct.

The interference match attempts to find the longest one-to-one match of parameters and case frames (i.e., the longest common subexpression). This is accomplished in two steps. First the case relations in El and E2 are matched in all possible ways to obtain the set M. Two case relations match if all of their case labels match. Each element of M is a case relation and a list of parameter correspondences which permit that case relation to match in both events:

```
M = {{circle:((a/c)(a/e))}{square:((b/d))}
     {small:((a/c)(b/c)(a/e)(b/e))}
     {ontop,under:((a/c b/d))}}
```

The second step Involves selecting a subset of the parameter correspondences in M such that all parameters can be bound consistently. This is conducted by a breadth-first search of the space of possible bindings with pruning of unpromising nodes. *The* search can be visualized as a node-building process. Here is one such (pruned) search:



The nodes are numbered in order of generation. One at a time, a node is examined and Joined with all other consistent nodes which have already been examined. The nodes 5, 8, and 9 are conjunctive generalizations. Node 9 binds a to c (to give 1) and b to d (to give 2) to produce the conjunction:

```
{{circle:1}{square:2}{small:1}
 {ontop:1, under:2}}
```

The node-building process is guided by computing a utility value for each candidate node to be built. The nodes are pruned by setting an upper limit on the total number of possible nodes and pruning nodes of low utility when that limit is reached.

Evaluation:

i) Representational adequacy. The algorithm discovers the following conjunctive generalizations of the example in Fig. 1:

1. {{ontop:l, under:2>{medium: lXclear: 1}}
   There is a medium clear object ontop of something.
2. {(ontop:l under:2>{medlum:l>{large:2) <clear:2>)
   There is a medium object ontop of a large, clear object.
3. {{medium: 1{clear: 1>{large: 3>{clear: 3} {shaded:2}}
   There is a medium sized clear object, a large sized clear object, and a shaded object.

PSRs provide two symbolic forms: parameters and case labels. The case labels can express ordinary predicates and relations easily. Symmetric relations may be expresed by using the same label twice as in {same!size:a, 8ame!8ize:b}. The only operator is the conjunction. The language has no disjunction or Internal disjunction. As a result, the fact that the top element in Fig. 1 is always either a square or a diamond cannot be discovered.

ii) Pules of generalization. The method uses the dropping condition and turning constants to variables rules.

iii) Computational efficiency. On our test example, the algorithm requires 22 comparisons and generates 20 candidate conjunctive generalizations of which 6 are retained. This gives a figure of 6/20 or 30% for computational efficiency. Four separate interference matches are required since the first match of El and E2 produces three possible conjunctive generalizations.

iv) Flexibility and extensibility. Hayes-Roth has indicated (personal communication) that this method has been extended to produce disjunctive generalizations and to detect errors in data. Hayes-Roth has applied this method to various problems in the design of the speech understanding system Hearsay II. Howev-

er, no facility has been developed for incorporating domain-specific knowledge into the generalization process.

Also, no facility for constructive induction has been incorporated although Hayes-Poth has developed a technique for converting a PSR to a lower-level finer-grained uniform PSR. This transformation permits the program to develop descriptions wnich involve a many-to-one binding of parameters.

2.2.2 Vere: Program Thoth [22-25]

Vere uses the term maximal conjunctive generalization or maximal unifying generalization to denote the maximally specific conjunctive generalization. Each event is represented as a conjunction of literals. A literal is a parenthesized list of constants called terms. For example, the objects in Fig. 2 would be described :

EI: (circle a (square b)(small a)(small b) (ontop a D)
F2: (circle c)(square d)(circle e) (small c)(large d)(small e) (ontop c d)(inside e d)

Although these resemble Hayes-Poth's PSFs, they are quite different. There are no distinguished symbols. All terms are treated uniformly.

The algorithm operates in four steps. First, the literals in each of the two events to be generalized are matched in all possible ways to enerate the set of matching pairs MP. Two iterals match if they contain the same number of constants and tney share a common term in the same position. For the example of Fig. 2,

MP= { ((circle a)-(circle *c)),*
((circle a)-(circle e)),
((square b)-(square d)),
((small a)-(small c),
((small a)-(small e)j»
((small b)-(small c)),
((small b)-(small e)),
(option a b) -(ontop c d)))

The second step involves selecting all possible subsets of Mr such that no single literal of one event is paired with more than one literal in another event. Each of these subsets eventually forms a new generalization of the original events.

In the third step, each subset of matching pairs selected in step 2 is extended by adding to the subset additional pairs of literals which did not previously match. A new pair p is added to a sunset S of MP if each literal in p is related to some other pair q in S by a common constant in a common position. For example, if S contained the pair ((square b)-(square d)) then we could add to S the pair ((ontop a b)-(inside e d)) because the third element of (ontop a b) is the second element of (square b) and the third element of (inside e d) is the second element of (square d) (Vere calls this a 3-2 relationship). We continue adding new pairs until no more can be added.

In step A the resulting set of pairs is converted Into a new conjunction of literals by merging each pair to form a single literal. Constants which do not match are turned into new constants which may be viewed as variables. For example, ((circle a)-(circle c)) would be converted to (circle 1).

Evaluation:

i) Representational adequacy. When applied to the test example (Fig. 1) this algorithm produces many generalizations. A few of the significant ones are listed here:

1. (ontop 1 2)(medium 1)(large 2)(clear 2)(clear 3)(shaded A)(5 A)
   There is a medium object on top of a large clear object. Another object is clear. There is a shaded object. (Note also the vacuous relationship 5 derived from unifying circle and triangle).
2. (ontop 1 2)(clear 1)(medium 1)(9 1) (5 3 4)(shaded 3)(7 3)(6 3)(clear A) (large 4)(8 4)
   There is a medium, clear object on top of some other object and there are two objects related in some way (5) such that one is shaded and the other is large and clear. (Note the vacuous relationships 6, 7, 8, and 9).
3. (ontop 1 2)(medium 1)(clear 2)(large 2)(5 *2)* (shaded 3)(7 3)(clear A)(6 A)
   Tnere is a medium object on top of a large clear object. There is a shaded object and there is a clear object. (Note the vacuous relationships 5, 6, and 7)•

The representation is very general. By convention the first symbol or a literal can be interpreted as a predicate symbol. The algorithm, however, treats all constants uniformly. This creates difficulties. For instance the algorithm generates vacuous literals in certain situations. Literals can be formed by pairing (red x) with (big y) to produce meaningless generalizations. One advantage of this relaxation of semantic constraints is that the pro?ram can discover conjunctive generalizations nvolving a many-to-one binding of variables.

The language contains only a conjunction operator. No disjunction or internal disjunction is included.

ii) Pules of generalization. The algorithm implements the dropping condition rule and the turning constants to variables rule.

ill) Computational efficiency. From the published articles [22-25] it is not clear how to perform step 2. The space of possibilities is very large and an exhaustive search could not possibly give the computation times which Vere has published. It would be interesting to find out what heuristics are being used to guide the search.

iv) Flexibility and extensibility. Vere has published algorithms which discover descriptions with disjunctions [2A] and exceptions [251. He has also developed techniques to generalize relational production rules [23,24]. The method has been demonstrated using the traditional Al toy problems of 10 analogy tests and blocks-world sequences. A facility for using background information to assist the induction process has also been developed. It uses a spreading activation technique to extract relevant relations from a knowledge base and add them to the input examples prior to generalizing them. Since the method has been extended to discover disjunctions and exceptions, It would be expected that the method could also operate in noisy environments.

2.3 Model-driven Methods: Buchanan et. al., and Michalski.

Model-driven methods search a set of possible generalizations in an attempt to find a few best" hypotheses which satisfy certain requirements. The two methods discussed here search for a small number of conjunctions which together cover all of the input events. The search proceeds by choosing as the initial working hypothesis some starting point in the partially ordered set of all possible descrip-

tions. If the working hypotheses satisfy certain termination criteria, then the search halts. Otherwise, the current hypotheses are modified by slightly generalizing or specializing them. These new hypotheses are then checked to see if they satisfy the termination criteria. The process of modifying and checking continues until the criteria are met. Top-down techniques typically have better noise immunity and can easily be extended to discover disjunctions. The principal disadvantage of these techniques is that the working hypotheses must repeatedly be checked to determine whether they subsume all of the input events.

2.3.1 Buchanan, et. al.: Program Meta-DENDRAL [1-3,20]

The algorithm which we describe here is taken from the RULEGEN program (part of the Meta-DENDPAL system). Meta-DENDRAL was designed to discover cleavage rules to explain mass spectrometry data. The descriptive language is based on the ball-and-stict model of chemical molecules. Each input event is a bond environment which describes some portion of a molecule. The environment is represented by a graph of the atoms in the molecule with four descriptors attached to each atom and forms the left nand side of a cleavage rule. The right hand side of the rule predicts a cleavage based on the existence in a molecule of the left-hand side of the rule (breakbond (**) indicates that the ** bond is predicted to be broken). A typical cleavage rule (with atoms w, x, y, and z) is:

```
LEFT-HAND SIDE (BOND ENVIRONMENT):
 Molecule graph:      w ** x -- y -- z --
 Atom descriptors:
   atom      type       nhs     nbrs     dots
    w       carbon       3        1        0
    x       carbon       2        2        0
    y      nitrogen      1        2        0
    z       carbon       2        2        0
RIGHT-HAND SIDE (CLEAVAGE PREDICTION):
 => Breakbond (**)
```

The algorithm chooses as its starting point the most general bond enviroment (x ** y) with no properties specified for either atom. During the search, this description is grown by successively specializing a property of one of the atoms in the graph or by adding a new atom to the graph. After each specialization, the new graph is checked to see if it is "better" than the parent graph from which is was derived. A daughter graph is better than its parent if it still covers at least half of the input rules (it's general enough) and still focusses on only one cleavage process (it's specific enough). The cleavage rules built by this algorithm are further improved by the program PULFMOD.

Evaluation:

I) Representational adequacy. The representation was adequate for the specific task of developing cleavage rules. It was not intended to be a general representation for objects outside or the chemical world. The descriptions can be viewed as conjunctions. Individual rules developed by the program can be considered to be linked by disjunction.

II) Rules of generalization. The dropping condition and turning constants to variables rules are used "in reverse" during the specialization process. RULEGEN does not seem to have the ability to handle an internal disjunction but RULEMOD apparently does. For example, it can indicate that the type of atom is "anything except hydrogen". In similar work on nuclear example in which the value of nhs is listed as "greater than or equal to one" (which indicates an internal disjunction).

iii) Computational efficiency. Because this is a problem-specific algorithm, we cannot supply comparison figures here for how this algorithm would work on our test example. The current program is considered to be relatively inefficient [2].

iv) Flexibility and extensibility. Meta-DENDPAL has been extended to handle NMP spectra. The program works well in an errorful environment. It uses domain-specific knowledge extensively. However, there is no strict separation between a general-purpose induction component and a special-purpose knowledge component. It is not clear whether the methods developed for Meta-DENDRAL could be easily applied to any non-chemical domain. The program does not perform constructive induction in any general way. However, the INTSUM program does ?erform sophisticated transformations on the nput spectra in order to develop the bond-environment descriptions.

2.3.2 Michalski and Dietterich: Program INDUCE 1.2

The algorithm described here is one of three algorithms designed by Michalski and his collaborators* The others are a data-driven method described by Stepp [21] and a mixed method described by Larson and Michalski [13,14]. *The* language used to describe the input events is VL_{21}, an extension to first-order predicate logic (FOPL) [17]. Each event is represented as a conjunction of selectors. A selector typically contains a function or predicate descriptor (with variables as arguments) and a list of values that the descriptor may assume. The selector [size(xl)=small. medium] asserts that the size of xl may take the values small or medium. The events in Fig. 2 are represented as :

```
E1: [size(x1)=small][size(x2)=small]
    [shape(x1)=circle][shape(x2)=square]
    [ontop(x1,x2)]
E2: [size(x1)=small][size(x2)=large]
    [size(x3)=small][shape(x1)=circle]
    [shape(x2)=square][shape(x3)=circle]
    [ontop(x1,x2)][inside(x3,x2)]
```

In this method, descriptors are divided into two classes: attribute descriptors and structure-specifying descriptors. Attribute descriptors describe attributes such as size or shape or distance which are applicable to all variables (representing, e.g., object parts). Structure-specifying descriptors include all other descriptors. They typically represent relationships among variables such as ontop or inside. Each input conjunction is broken into two conjuncts—one built of selectors containing only attribute descriptors (the attribute conjunct; and one built of selectors containing only structure-specifying descriptors (the structure conjunct)•

The algorithm is based on the observation that the structure-specifying descriptors are responsible for the computational complexity of generalizing structural descriptions. It we could determine conjunctions of structure-specifying selectors which were relevant for describing a particular class of objects, then the generalization of the attribute conjuncts could be handled quickly bv an appropriate covering algorithm. The algorithm seeks to determine such a set of structure conjuncts which appear likely to be part of a maximally specific conjunctive generalization of all of the in-

put events. It does this by finding conjunctions which are maximally specific generalizations of the input structure conjuncts considered alone. Such conjunctive generalizations of the structure conjuncts must be contained in some maximally specific generalizations of the entire set of input events. However, there may be maximally specific conjunctive generalizations of the input events which contain few if any structure-specifying selectors. This algorithm also finds these generalizations by considering structure conjuncts which are less than maximally specific.

The algorithm operates in two phases. The first phase is the structure-determining phase. A random sample of the Input structure conjuncts is taken. This sample becomes the initial set of generalizations G_0. In each step, G_i is first pruned to a fixed size by removing unpromising generalizations. Then G_i is checked to see if any of its generalizations covers all of the structure conjuncts. If any do, they are removed from G_i. and placed in the set C of candidate conjunctive generalizations. Lastly, G_i. is generalized to form G_{i+1} by taking each element of G_i. and generalizing it in all possible ways by dropping single selectors. When the set of candidates C reaches a prespecified size, the search stops.

The second phase is the attribute-determining phase. In this phase, the problem Is converted to a multiple-valued logic covering problem using the vL_1 propositional calculus [15,16]. Each candidate cover A in C is matched against all input events and the relevant variables are identified. For each match, the appropriate attribute conjuncts are extracted and used to form a VL_1 event. For example,

if A = [ontop(p1,p2)] and
E1 = [ontop(p1,p2)][ontop(p2,p3)]
[size(p1)=1][size(p2)=3][size(p3)=5]
[color(p1)=red][color(p2)=green]
[color(p3)=blue]

then we get two VL_1 events:

V1= (1, 3, red, green) and
V2= (3, 5, green, blue).

These are vectors of attributes which correspond here to the descriptors:

(size(p1), size(p2), color(p1), color(p2))

for p_1 and p2 in A.

All Input events are converted into VL_1 events In this manner. In general, more than one VL_1 event is created from each input event. The 9et of VL_1 events can be covered using a covering algorithm. A cover could be obtained by forming the union of the values taken on by each VL_1 attribute. Such an approach usually leads *to* overgeneralization since only one VL_1 event derived from each input event need be covered. We use a beam-search technique to select a subset of the VL_1, events to be covered.

This two-phase algorithm provides two computational advantages. First, the time required to compare expressions in the structure-determining phase is reduced because the structure conjuncts are usually much smaller than the full input conjuncts. Second, the manipulation of VL_1 formulas is very easy since they may be represented as bit strings and manipulated using fast bit-parallel operations. The chief disadvantage of this algorithm is that it is difficult to decide when to terminate the structure-determining phase.

Evaluation:

i) Representational adequacy. The algorithm discovers, among others, the following generalizations of the events in Fig. 1:

1. [ontop(p1,p2)][size(p1)=medium]
[shape(p1)=circle,square,rectangle]
[size(p2)=large]
[shape(p2)=box,rectangle,ellipse]
[texture(p2)=clear]
There is a medium-sized circle, rectangle or square on top of a large, clear box, rectangle, or ellipse.

2. [ontop(p1,p2)][size(p1)=medium]
[shape(p1)=polygon] [texture(p1)=clear]
[size(p2)=medium,large]
[shape(p2)=rectangle,circle]
There is a clear, medium-sized polygon on top of a medium or large circle or rectangle.

3. [ontop(p1,p2)][size(p1)=medium]
[shape(p1)=polygon]
[size(p2)=medium,large]
[shape(p2)=rectangle,ellipse,circle]
There is a medium-sized polygon on top of a large or medium rectangle, ellipse or circle.

4. [size(p1)=small,medium]
[shape(p1)=circle,rectangle]
[texture(p1)=shaded]
There is a shaded object which is either medium or small in size and has a circular or rectangular shape.

This algorithm Implements the conjunction, disjunction and internal disjunction operators. It provides a fairly non-uniform set of representational facilities. Descriptors, variables, and values are all distinguished. Descriptors are further analyzed into structure-specifying descriptors and attribute descriptors. The current method provides for descriptors which have unordered, linearly ordered, and tree ordered value sets. This variety of possible representations permits a better "fit" between the description language and any specific problem.

ii) Pules of generalization. *The* algorithm uses all rules mentioned in section 1.4 and also a few constructive induction rules (see below). All constants are coded as variables. The effect of the turning-constants to variables rule is achieved as a special case of the generalization by internal disjunction rule.

iii) Computational efficiency. *The* algorithm reguires 28 comparisons and builds 13 ruLes during the search to develop the descriptions listed above. Four rules are retained so this gives an efficiency ratio of 4/13 or 30%.

iv) Flexibility and extensibility. The algorithm can easily discover disjunctions by altering the termination criteria for the structure-determining phase to accept structure conjuncts which do not necessarily cover all of the input events. The same general two-phase approacn can also be applied to problems of determining discriminant generalizations. Larson and Michalski have done work on determining discriminant classification rules [13,14,15].

The algorithm has good noise immunity. Noise events can be discovered because the algorithm tends to place them in separate terms of a disjunction.

Domain-specific knowledge can be incorporated into the program by defining the domains of descriptors, specifying the structures of these domains, specifying certain simple production rules, and by providing constructive induction rules. These forms of knowledge representation

are not always convenient, however. Further work should provide other facilities for knowledge representation.

A few simple constructive induction rules have been incorporated into the current implementation as a preprocessor. Other constructive Induction rules can be specified by the user. Using the built-in constructive induction rules, the program produces the following conjunctive generalization of the input events in Fig. 1:

> # p's with texture clear-2][top-most(pi)] ontop(pKp2)] [size(pi)-medium! shape(pi)-polygon][texture(pi)-clear] size(p2)-medium,large] shape(p2)-circle,rectangle]
>
> There are exactly two clear objects In each event. *The* top most object is a medium sized, clear polygon and it is on top of a large or medium sized circle or rectangle.

We hope to expand this constructive induction facility in the future.

2.4 Summary

The comparison of various methods is summarized in Fig. 3. The table shows the distinct advantages and disadvantages of top-down methods as opposed to bottom-up methods. Bottom-up methods tend to be faster but noise immunity and flexibility suffer as a conseguence. Top-down methods have good noise immunity and are easily modified to discover disjunctive and other forms of generalization, They do tend to be computationally more expensive. By separating the structure-determining phase from the attribute-determining phase in our method, a considerable speed-up has been achieved.

3.0 CONCLUSION

One of the problems of current research on induction is that each research group is using a different formal language and terminology. This makes the exchange of information difficult. This paper was intended to help readers get a better understanding of the state of the art in this area.

Some important problems to be addressed In future research include:

1) the development of adequate formal languages and knowledge representations for hypothesis formulation and modification;

II) extension of the scopes of operators and forms which an inductive program can efficiently use during hypothesis formulation;

iii) the development of general mechanisms of induction whicn can be guided by problem-specific packets of knowledge; and

iv) incorporation in the program of extensive facilities for constructive induction and multi-level schemes of description. In particular, an inductive program should be able to assign names to various subdescriptions and use these names in the formulation of hypotheses (i.e. generate hierarchical forms).

Finally, an important principle which should guide future research is what we call the principle of comprehenslbIIIty. This principle states "That the descriptions which an AI program uses and the concepts which it generates should be easily comprehensible by people. In the context of work on induction, the comprehen8ibility principle requires that the descriptions be short and use operators which

Method: / Criterion	Hayes-Roth	Vere	Buchanan et.al.	Michalski
Intended application:	general	general	discovering mass spectrometry rules	general
Language:	Parameterized Structural Representation	Quantifier-free FOPL	Chemical model	Variable-valued logic system VL21
syntactic concepts:	case frames parameters case labels	literals constants	molecule graph attributes constants in in value sets	selectors descriptors dummy variables constants in value sets
operators:	∧	∧	∧, ∨, internal ∨	∧, ∨, internal ∨
Generalization Rules:				
dropping condition?	yes	yes	yes	yes
constants to variables?	yes	yes	yes	yes
generalizing by internal v?	no	no	yes	yes
climbing tree?	no	no	no	yes
closing intervals?	no	no	no	yes
Efficiency:				
comparisons:	22	complete algorithm not known	not applicable	28
conjunctions generated during search:	20	-----	not applicable	13
ratio output to total:	6/20=30%	-----	not applicable	4/13=30%
Extensibility:				
applications	speech analysis	none	mass spectrometry, NMR	soybean disease diagnosis
disjunctive forms?	no	yes	yes	yes
noise immunity	low	probably good	excellent	very good
domain knowledge?	no	yes	yes, built-in to program	yes
constructive induction?	no	no	no	limited facility

Figure 3.

can be easily Interpreted in natural language. Furthermore, systems should be designed to provide flexible interactive facilities. This approach has been adopted in our work because we expect that the most significant applications of AI inductive programs will be as interactive tools for conceptual data analysis.

4. REFERENCES

[I] Buchanan. B. G., E. A. Feigenbaum. J. Lederberg, "A Heuristic Programming Study of Theory Formation in Science/' in Proc. IJCAI-2, 1971, pp. 40-48.

[2] Buchanan, B.C., D. H. Smith, W. C. White, R. J. Critter, E. A. Feigenbaum, J. Lederberg, C. Djerassi, J. Am. Chem. Soc. 98 (1976) p. 6168. ~

[3] Buchanan, B. G., E. A. Feigenbaum. "Dendral and Meta-Dendral, Their Applications Dimension," Artif. Intel!. 11 (1978) pp. 5-24.

[4] Dietterich, T., "User's Guide for INDUCE 1.1," internal report, Dept. of Comp. Sci., Univ. of IIInois, Urbana.

[5] Dietterich, Thomas G., "A Comparison of Methods for Characteristic Generalization," Dept. of Comp. Sci., Univ. of 111. Rept. UIUC-DCS-7R-950. Dec. 1978.

[6] Hayes-Poth. F., "Collected Papers on the Learning and Recognition of Structured Patterns", Dept. of Comp. Sci., Carnegie-Mellon Univ., Jan. 1975.

[7] Hayes-Roth, F., "Patterns of Induction and Associated Knowledge Acquisition Algorithms," Dept. of Comp. Sci., Carnegie-Mellon Univ., May 1976.

[8] Hayes-Roth, F., J. McDermott, "Knowledge Acquisition from Structure Descriptions", In Proc. IJCAI-5, 1977, pp. 356-362.

[9] Hayes-Poth, F., J. McDermott, "An Interference Matching Technique for Inducing Abstractions", CACM 21:5, 1978, pp. 401-410.

[10] Hunt, E.B Fxperiments in Induction, Academic Press, 1966.

[II] Knapman, John, "A Critical Review of Winston's Learning Structural Descriptions from Examples," AISB Quarterly Issue 31, September 1978, pp. 319-3211.

[12] Lenat, D., "AM: An artificial intelligence approach to discovery in mathematics as heuristic search," Comp. Sci. Dept., Rept. STAN-CS-76-570, Stanford Univ., July 1976.

[13] Larson, J., and P.S. Michalski, "Inductive Inference of VL Decision Pules," SIGART Newsletter, June 1977, pp. 38-44.

[14] Larson, J., "Inductive Inference in the Variable Valued Predicate Logic System VL21 : Methodology and Computer Implementation', Rept. No. 869, Dept. of Comp. Sci., Univ. of 111., Urbana, May 1977.

[15] Michalski, R. S., "Variable-valued logic and its application to pattern recognition and machine learning," In Comp. Sci. and Multiple-Valued Logic, ed. DTcT RTne, North-Holland, 1977, pp,.506-534.

[16] Michalski, R.S., "Toward Computer-aided Induction: a brief review of Currently Implemented AOVAL programs," In Proc. IJCAI-5, 1977.

[17] Michalski, P.S. "Pattern Recognition as Knowledge-Guided Induction," Pept. 027, Dept. of Comp. Sci., Univ. of 111. Urbana, 1978.

[18] Michie, D., "New Face of AI," Experimental Programming Pepts.: No. 33, MIRU, Univ. of Edinburgh, 1977.

[19] Mitchell,. T. M., "Version Spaces: A Candidate Elimination Approach to Pule Learning," In Proc. IJCAI-5, MIT, 1977.

[20] Schwenzer, G. M., T. M. Mitchell, Computer-assisted Structure Elucidation Using Automatically Acquired Carbon-13 NMR Rules, in ACS Symposium Series. No. 54, 'Computer-assisted Structure Elucidation,' D.H. Smith (ed), 1977.

[21] Stepp, R., "User's guide for UNICLASS program", internal report, Dept. of Comp. Sci., Univ. of 111., Urbana.

[22] Vere., S.A.. "Induction of Concepts in the Predicate Calculus," In Proc. IJCAI-4, 1975.

[23] Vere. S. A., "Induction of Relational Productions in the Presence of Background Information," In Proc. IJCAI-5, 1977.

[24] Vere, S. A., "Inductive Learning of Relational Productions", in Pattern-Directed Inference Systems, D.A. Waterman and F. Hayes-Roth (eds).Academic Press, 1978.

[25] Vere, S. A.. "Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions," Dept. of Inf. Eng'g, Univ. of 111., Chicago Circle, 1978.

A GLIMPSE OF TRUTH MAINTENANCE***

Jon Doyle
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
bib Technology Square
Cambridge, Massachusetts 02139

To choose their actions, reasoning programs must be able to draw conclusions from limited information and subsequently revise their beliefs when discoveries invalidate previous assumptions. In the *Truth Maintenance System* (TMS) I introduce a new notion of "reason for belief which permits the mechanization of these abilities as a problem solver subsystem The TMS records and maintains the reasons for program beliefs. These recorded reasons are useful in constructing explanations of program beliefs and actions, and in guiding the course of action of a problem solver. This paper outlines some of the structure and applications of the TMS.

I INTRODUCTION

One important problem faced by reasoning programs is the need to make decisions based on limited information. This problem arises both in programs interacting with an external environment and in contemplative programs searching a data base for an answer to some question There are two consequences of this need to predict, the program must have some way to make decisions based on limited information, and the program must have some way to revise its beliefs if these decisions are found to be in error. The first of these abilities is provided by utilizing epistemic classifications of possible program beliefs so that conclusions may be drawn from the lack of belief as well as from other beliefs The second ability is in general a very complex problem for which no complete solutions are known. (See [3,7,8] for surveys of the problem.) However, the simpler problem of revising beliefs based on limited information is solvable by recording the reasons for each program belief. These records can be used to find the set of extant beliefs by determining which beliefs have valid reasons These recorded reasons also are useful in resolving conflicts that arise when limited knowledge gives rise to incompatible conclusions This papei describes a mechanization of these abilities, embodied in a general-purpose problem solver subsystem called the *Truth Maintenance System* (TMS) The TMS is based on a new analysis of what constitute "reasons" for beliefs and "assumed" beliefs In this analysis, all beliefs are derived from the existence of valid reasons for them, and assumptions are statements which are believed because some other statements are not believed

The TMS records and maintains "proofs" of program beliefs It manipulates two data structures, *nodes,* which represent beliefs, and *justifications,* which rrpresent reasons for beliefs The fundamental actions the TMS can be called upon to perform are the creation of a new node, to which the problem solving program can attach the statement of a belief, and the addition of a new justification to a node, to represent assertion of the belief associated with the node by some rule or procedure in the problem solver. The addition of new justifications may invoke the automatic procedure of *truth maintenance* to make any revisions necessary in the set of beliefs The TMS revises beliefs by using the recorded justifications to compute non-circular proofs of beliefs from basic hypotheses These proofs distinguish one or more justifications .is the *well-founded support* for each believed node, and are used during truth maintenance to determine the set of beliefs to update by finding those nodes whose

* This research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number NOOO14-75-C-0643, and in part by NSF grant MCS77-04828.

** My new paper [2] presents the foundations, mechanisms, and applications of the TMS in much greater depth, detail, and clarity than is possible here That paper also mentions the many related works by others too numerous to cite in this short a paper.

well-founded support depends on changed beliefs. These proofs allow another process, *dependency-directed backtracking,* to resolve conflicts by tracing the well-founded supports of conflicting beliefs to remove one of the assumptions causing the conflict and to make a record used to prevents similar future conflicts

Minsky [6] criticized traditional logics for their *monotonicity,* their lack of any ability to control inferences by the addition of new axioms With this in mind the TMS employs a special type of justification, called a *non-monotonic justification,* to draw conclusions based on limited or incomplete knowledge This type of justification allows belief in a node to be based not only on other beliefs, as occurs in the standard forms of deduction and reasoning, but also on lack of belief in certain nodes. For example, a node N-I representing a statement P might be justified on the basis of a lack of belief in a node N-2 representing the belief ~P. (Distinct nodes are used to represent P and **P.) In this case, the TMS would have N-I believed as long as N-2 was not believed, and we would call N-I an *assumption.* (More generally, by assumption we mean any node whose well founded support is a nonmonotonic justification.)

As a small example, suppose an office scheduling program is considering holding a meeting M on Wednesday. To do this, the program assumes that the meeting is on Wednesday. The data base of the program includes a rule which draws the conclusion that due to regular commitments, any meeting on Wednesday must occur at 100 PM. However, the fragment of the schedule for the week constructed so far has something else scheduled for that time already, and so another rule in the data base concludes that the day for the meeting cannot be Wednesday. These beliefs might be notated as follows

Node	*Statement*	*Justification*
N-1	DAY(M) = WEDNESDAY	(SL () (N-2))
N-2	DAY(M) ≠ WEDNESDAY	
N-3	TIME(M) = 13:00	(SL (R-37 N-1) ())

As seen in the above notation for justifications, each justification consists of two lists. The meaning of the notation is that the statement depends on each of the nodes in the first list being believed, and on each of the nodes in the second list not being believed Since there is no known justification for N-2, it is not believed. The justification for N-I specifies that it depends on the lack of belief in N-2, and so N-I is believed. The justification for N-3 shows that it is believed due to rule R-37 acting on N-I When the assumption N-I is rejected by some rule,

N-2	DAY(M) ≠ WEDNESDAY	(SL (R-9 N-7 N-8) ())

where N-7 and N-8 represent the day and time of some other engagement, the TMS will revise the beliefs so that N-I and N-3 are not believed.

2. REPRESENTATION OF KNOWLEDGE ABOUT BELIEF

A node may have several justifications, each of which represents a different reason for belief in the node. The node is believed if at least one of these justifications is *valid.* The conditions for validity of justifications are described below We say that a node which has a valid justification is *in,* and that a node without a valid justification is *out.* The distinction between *in* and *out* is not that of *true* and *false.* The former classification denotes conditions of knowledge about leasons for belief; the existence or non-existence of valid reasons. *True* and *false,* on the other hand, classify statements according to truth value independent of any reasons for belief In this way, there can be four states of knowledge about a proposition P, corresponding to the node representing P being *in* or *out* and the node representing ~P being *in* or *out.*

There are two basic forms of justifications. These are inspired by the typical forms of arguments in natural deduction inference systems. A sample proof in such a system might be as follows:

Line	*Statement*	*Justification*	*Dependencies*
1.	A⊃B	Premise	(1)
2.	B⊃C	Premise	(2)
3.	A	Hypothesis	(3)
4.	B	MP 1,3	(1,3)
5.	C	MP 2,4	(1,2,3)
6.	A⊃C	Discharge 3,5	(1,2)

Each step of the proof has a line number, a statement, a justification, and the set of line numbers the statement depends on Premises and hypotheses depend on themselves, and other lines depend on the set of premises and hypotheses derived from their justifications The above proof proves ADC from the premises ADB and BDC by hypothesizing A and concluding C The assumption A is then discharged to provide the proof of A⊃C There are two effects that justifications can have on the set of dependencies in natural deduction systems, either the justifications can sum the dependencies of the referenced lines (as in line 4). or they can subtract the dependencies of some lines from those of other lines (as in line 6). The two types of justifications used in a TMS account for these effects on dependencies. A *suppot-list (SL) justification* says that the justified node depends on a set of other nodes, and thus in effect sums the dependencies of the referenced nodes. A *conditional-proof*

(CP) justification says that the node it justifies depends on the validity of a certain hypothetical argument, and as in the example above, subtracts the dependencies of some nodes (the hypotheses of the hypothetical argument) from the dependencies of others (the conclusion of the hypothetical argument) These two types of justifications can be used to construct a variety of forms of dependency relationships For example, we might rewrite the example in terms of TMS justifications as follows (here ignoring the difference between premises and hypotheses, and ignoring the inference rule MP):

```
N-1     A⊃B              (SL () ())
N-2     B⊃C              (SL () ())
N-3       A              (SL () ())
N-4       B              (SL (N-1 N-3) ())
N-5       C              (SL (N-2 N-4) ())
N-6     A⊃C              (CP N-5 (N-3) ())
```

The support-list justification is of the form

(SL <*in*list> <*out*list>).

A SL-justification is valid if each node in its in list is *in,* and each node in its outlist is *out.* A SL-justification can be used to represent several types of deductions. When both the inlist and outlist are empty, we say the justification forms a *premise* justification A premise justification is always valid, and so the node it justifies will always be believed Normal deductions are represented by support-list justifications with empty outlists. These represent monotonic deductions of the justified node from belief in the nodes of the inlist. *Assumptions* are nodes whose well-founded support is a support-list justification with a nonempty outlist. These justifications can be interpreted by viewing the nodes of the *inlist* as comprising the reasons for making the assumption; the nodes of the *outlist* represent the specific incompletenesses of knowledge authorizing the assumption.

The conditional-proof justification takes the form

(CP <consequent> <inhypotheses> <outhypotheses>).

A node justified by such a justification represents an implication, whose support is derived by a conditional proof of the consequent node from the hypothesis nodes. A justification of this form is valid if the consequent node is *in* whenever (a) each node of the inhypotheses is *in* and (b)each node of the outhypotheses is *out.* Except in a few esoteric uses, the set of outhypotheses is empty. Standard conditional proofs in natural deduction systems specify a single set of hypotheses, which correspond to our inhypotheses. The truth maintenance system requires that the set of hypotheses be divided into two disjoint subsets, since nodes may be derived both from some nodes being *in* and other nodes being *out.*

3. DEFAULT ASSUMPTIONS

One very common technique used in problem solving systems is to specify a default choice for the value of some quantity. (See, for example. [81) This choice is made with the intent of overriding it if either a good reason is found for using some other value, or if making the default choice leads to an inconsistency The assumption of the day of the week for a meeting in the first example above is such a default assumption.

In the case of a binary choice, a default assumption can be represented by believing a node if the node representing its negation is *out.* When the default is chosen from a set of alternatives, the following generalization of the binary case is used Let $\{F_1, \ , F_n\}$ be the set of the nodes which represent each of the possible values of the choice Let G be a node which represents the reason for making the default assumption Then F_i may be made the default choice by providing it with the justification

(SL (*G*) ($F_1 \ldots F_{i-1} F_{i+1} \ldots F_n$)).

If no information about the choice exists, there will be no reasons for believing any of the alternatives except F_i Thus F_i will be *in* and each of the other alternatives will be *out* If some other alternative receives a valid justification from other sources, that alternative will become in. This will invalidate the support of F_i and F_i will become *out.* If a contradiction is derived from F_i the dependency-directed backtracking mechanism will recognize that F_i is an assumption by means of its dependence on the other alternatives being *out.* (See the section on dependency-directed backtracking for an explanation of this.) The result of backtracking may be to justify one of the other alternatives, say F_i. causing F_i to go *out.* The justification for F ■ will be of the form

(SL <various things> <remainders>)

where the remainders are the F_k's remaining after F_i and F_i are taken away In effect, backtracking will cause the removal of the default choice with the set of alternatives, and will set up a new default assumption structure from the remaining alternatives As a concrete example, our scheduling program might default a meeting day as follows:

```
N-I        DAY (M) -MONDAY
```

```
N-2     DAY(M) = WEDNESDAY      (SL () (N-1 N-3))
N-3     DAY(M) = FRIDAY
```

In this example, Wednesday is assumed to be the day of the meeting M, with Monday and Friday being the alternatives. Wednesday will be the default choice until a valid reason is supplied for either Monday or Friday.

If the complete set of alternatives from which the default assumption is to be chosen cannot be known in advance but must be discovered piecemeal, a slightly different structure is necessary This ability to extend the set of alternatives is necessary, for example, when the default is a number, due to the large set of possible alternatives For cases like this the following structure may be used instead. Retaining the above notation, let ~F- be a new node which will represent the negation of F_i. We will arrange for F_i to be believed if **F_i cannot be proven, and will set up justifications so that if F_i is distinct from F_i F_i will imply "F_i This is done by giving F_i the justification

$$(SL\ (G)\ (\sim F_i)),$$

and by giving **F_i a justification of the form

$$(SL\ (F_j)\ ())$$

for each alternative F_i. distinct from F_i. As before, F_i will be assumed if no reasons for using any other alternative exist Furthermore, new alternatives can be added to the set simply by giving **F_i a new justification corresponding to the new alternative This structure for default assumptions will behave as did the fixed structure in the case of an unselecred alternative receiving independent support. Backtrackmp, however, has a different effect. If a contradiction is derived from the default assumption supported by the extensible sttucture, **F_i will be justified so as ro make F_i become *out* If this happens, no alternative will be elected to take the place of the default assumption. The extensible structure requires an external mechanism to construct a new default assumption whenever the current default is ruled out For example, a census program might make assumptions about the number of children in a family as follows:

```
N-1     #-CHILDREN(F) = 2        (SL () (N-2))
N-2     #-CHILDREN(F) = 2        (SL (N-3) ())
                                 (SL (N-4) ())
N-3     #-CHILDREN(F) = 0
N-4     #-CHILDREN(F) = 1
```

With this system of justifications, N-1 would be believed because no different number of children is known. If it turns out that the family has, for example, 5 children a new statement would have to be made, along with a new justification of N-2 in terms of this new statement.

4. DEPENDENCY-DIRECTED BACKTRACKING

Making assumptions admits the possibility of making errors. When a contradiction or other inconsistent state of the data base occurs, the TMS employs a process called dependency-directed backtracking to find and remove incorrect assumptions so as to restore consistency. There are several steps involved in dependency-directed backtracking, but first the inconsistency must somehow be signalled to the TMS. as there is no built-in notion of inconsistency. This signalling consists of informing the TMS that a node represents an inconsistency With this knowledge, the TMS will try to restore consistency whenever the node comes *in* by rejecting enough assumptions to force the node *out.* Any node may be marked for such treatment A node so marked is called a *contradiction.*

The steps of dependency-directed backtracking are as follows. First, the well-founded support of the contradiction node is traced backwards to find the set of assumptions (nodes with a nonmonotonic justification as their well-founded support) underlying the contradiction. Belief in at least one of these assumptions must be retracted to remove the contradiction. This is done by creating a new justification for one of the *out* nodes underlying one of the assumptions. Since the backtracker may be mistaken in its assignment of blame to that assumption, the justification used to retract the assumption must indicate the alternatives that were available but not utilized by the backtracker. Thus the new justification includes (a) the reason why the contradiction occurred and (b) the other assumptions involved. Thus the second step of the backtracking process is to construct a node recording the reason why the contradiction occurred, and the third step is to use this node and the other assumptions in justifying an *out* node supporting the assumption selected for removal

In more detail, the first step of the backtracking process traces backwards through the well-founded support of the contradiction node to collect the set of "maximal" assumptions supporting the contradiction. Not all assumptions found by tracing the well-founded support are used, instead, only those assumptions which do not support other assumptions underlying the contradiction as well. That is, the well-founded support relationships induce a natural partial-ordering on nodes, where one node is said to be "lower" than a second node if the first occurs in the second's well-founded support. The maximal assumptions are then

those assumptions which are maximal in this partial order. Only this "front line" of assumptions is used because if the reason for revoking a lower-level assumption involves a higher-level assumption, then the removal of the lower-level assumption would cause truth maintenance to remove the higher-level assumption that it supports, so the reason for removing the lower-level assumption would not hold up This reflects the fact that there may not be enough information to definitely rule out a lower-level assumption.

The second step summarizes the reason for the contradiction in terms of the set of selected assumptions Let 5 - \A_1, $A_n\}$ indicate the set of inconsistent assumptions. The backtracker then creates a node called a *nogood,* a new node signifying that S is inconsistent Since contradiction nodes really represent the false statement, the nogood node can be taken to represent

$$A_1 \wedge \ldots \wedge A_n \supset false.$$

or alternatively,

(I) $$\sim (A_1 \wedge \ldots \wedge A_n).$$

S is recorded as the *nogood-set* of the nogood. This meaning for the nogood node is produced by justifying it with the conditional proof of the contradiction node relative to the assumption nodes, that is, with the justification

(2) (CP <contradiction node> *S* ()).

In this way, the inconsistency of the set of assumptions will be remembered even after the contradiction has been resolved by the retraction of some hypothesis.

The final step is selecting an assumption A_i (the "culprit") from 5 and justifying one of the *out* nodes listed in its well-founded supporting justification. (If these underlying *out* nodes are thought of as "denials" of the assumption, then this step is much like reasoning by *reductio ad absurdum.)* Let *NO* be the nogood, and let the inconsistent assumptions be $A_1,\ldots$, A_n. Let D_j, , D_k be the *out* nodes appearing in the justification which supports belief in the assumption A_i This justification for the assumption can be invalidated by justifying D_1 with the justification

(3) (SL (NG $A_1 \ldots A_{i-1} A_{i+1} \ldots A_n$) ($D_2 \ldots D_k$)).

This justification is valid whenever the nogood and other assumptions are believed and the other "denials" of the culprit are not believed If the choice of culprit was in error, then another contradiction will occur in the future involving D_1 and by this justification will be led to suspect the remaining assumptions, as well as D_1 if there are any other *out* nodes listed in its justification. If, by means of other previously existing justifications, the current contradiction is still *m* following the addition of this justification, backtracking is repeated Presumably the new invocation of the backtracking process will find that the previous culprit is no longer an assumption Backtracking halts when the contradiction becomes *out,* or when no assumptions can be found underlying the contradiction.

As an example, consider a program scheduling a meeting, preferably at 10 AM in either room 813 or 801. This might be represented as

```
N-1     TIME(M) = 1000          (SL () (N-2))
N-2     TIME(M) = 1000
N-3     ROOM(M) = 813           (SL () (N-4)
N-4     ROOM(M) = 801
```

With these justifications, N-I and N-3 are *in,* and the other two nodes are *out.* If some previously scheduled meeting exists, it might cause this combination of time and room for the meeting to be ruled out by means of a contradiction.

```
N-5     CONTRADICTION           (SL (N-1 N-3) ())
```

The dependency-directed backtracking system then traces the well founded support of the contradiction to find that it depends on two assumptions, N-I and N-3, both of which are maximal

```
N-6     NOGOOD N-1 N-3          (CP N-5 (N-1 N-3) ())
N-4     ROOM(M) = 801           (SL (N-6 N-1) ())
```

A nogood node is created which means, in accordance with form (1) above.

$$\sim (TIME(M) = 1000 \wedge ROOM(M) = 813)$$

and this nogood is given a justification corresponding to form (2) above The assumption N-3 is selected arbitrarily as the culprit, and is rejected by providing its only *out* supporting node. N-4. with a justification of the form (3) above Following this, N-I, N-4, and N-6 are *in,* and N-2, N-3. and N-5 are *out* N-6. the nogood node, has an always-valid CP-justification since the contradiction node N-5 depends directly on the two assumptions N-I and N-3 without any additional beliefs intervening If some further consideration determines that room 801 cannot be used after all. another contradiction node could be created to force a different choice

```
N-7     CONTRADICTION           (SL (N-4) ())
```

```
N-8     NOGOOD N-1                  (CP N-7 (N-1) ())
N-2     TIME(M) = 1000              (SL (N-8) ())
```

Tracing backwards from N-7 through N-4, N-6, and N-I, the backtracker finds that the contradiction depends on only one assumption. N-I The nogood node N-8 is created and justified with a CP-justification which in effect is equivalent to the SL-justification

(SL (N-6) ()),

since the nogood N-6 contributes to the contradiction but does not itself depend on the assumption N-I The revocation of the assumption N-I removes N-5, the previous objection to the choice of room, so at the close of this bit of decision making N-2, N-3, N-6, and N-8 are *in,* and N-I, N-4, N-5. and N-7 are *out.*

There are a number of variations on this particular scheme for dependency directed backtracking All of these variations are considerable improvements over the chronological backtracking systems used in classical systems like MICRO-PLANNER and many early theorem provers. The improvements stem from the non-chronological nature of dependency-directed backtracking, in which the support relationships rather than the temporal ordenngs determine the choices responsible for an error Another improvement is that the cause of the contradiction is summarized via a nogood node. This summarization keeps the system from making the same mistake in the future. Stallman and Sussman [10] indicate that these two improvements lead to sizable gains in efficiency.

5 DISCUSSION

The TMS solves part of the problems of belief revision and limited information by introducing new conceptions of justified belief, justification, and assumption. It lends itself to several other uses as well as belief revision and making assumptions: for example, providing explanations of program beliefs and actions (sec [2)). and controlling the problem solver's actions (see [1,2.10]). London [4] explores in detail the use of dependency networks in updating a problem solver world model in the face of actions. For the mathematically oriented, Drew McDcrmott and I try to analyze the logic and semantics underlying the TMS in [*5] Non-monotonic logic,* as we call it, has several novel properties

ACKNOWLEDGEMENTS

I thank Gerald Jay Sussman, Richard M. Stallman, Guy L. Steele Jr., Johan de Kleer, Drew McDermott, David McAllester, Scott Fahlman. Howard Shrobe, Charles Rich, Marilyn Matz. Beth Levin, Jim Stansfield, Mitchell Marcus, and Richard Brown for ideas, comments and advice, and the Fannie and John Hertz Foundation for supporting my research with a graduate fellowship.

REFERENCES

[I] de Kleer, J., J Doyle, G. L Steele Jr., and G. J. Sussman, "Explicit Control of Reasoning," MIT AI Lab, Memo 427, 1977.

[2] Doyle. J.. "A Truth Maintenance System," MIT AI Lab, Memo 521. 1979.

[3] Hayes, P J . "The Frame Problem and Related Problems in Artificial Intelligence." in A. Ehthorn and D. Jones, editors. *Artificial and Human Thinking,* San Francisco: Josey-Bass, 1973

[4] London. P. E., "Dependency Networks as a Representation for Modelling in General Problem Solvers," Computer Science Department TR-589, University of Maryland, 1978.

[5] McDermott, D. and J Doyle, "Non-Monotonic Logic I", MIT AI Lab, Memo 486, 1978, to appear in *Artificial Intelligence.*

[6] Minsky, M.. "A Framework for Representing Knowledge," MIT AI Lab, Memo 306, 1974

[7] Qmne, W. V and J. S. Ulhan, *The Web of Belief,* second edition. New York; Random House, 1978.

[S] Reiter. R., "On Reasoning by Default," *Proc. Second Symp. on Theoretical Issues in Natural Language Processing,* Urbana. Illinois, 1978.

[9] Rescher, N., *Hypothetical Reasoning,* Amsterdam: North Holland 1964.

[10] Stallman, R. M., and G J Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Compute! Aided Circuit Analysis," *Artificial Intelligence,* Vol 9. No 2, (October 1977), pp. 135-196

ALTERNATIVE PARSERS FOR CONCEPTUAL DEPENDENCY
GETTING THERE IS HALF THE FUN

Marc Eisenstadt
The Open University
Milton Keynes
ENGLAND, U.K.

This paper takes a look behind the scenes at two conceptually-based parsers in order to shed light on the true differences between them. The first one is Riesbeck's parser for Schank's conceptual dependency; the second is the 'LNR' parser for Norman and Rume hart's active semantic networks. Both are described in terms of Kaplan's General Syntactic Processor formalism. This analysis shows that 'conceptual dependency' and *active semantic networks' have little or nothing to do with the actual functioning of the parsers. Computationally, the two parsers differ only in terms of (a) effective use of interrupts and (b) reliance on selectional restrictions to guide parsing. A synthesis of the best features of both is suggested.

1. INTRODUCTION

An important trend in natural language processing which emerged in the early 1970's was the use of case-based language-independent notation to represent the conceptual content of segments of text (3,5,7). Although there is much overlap among the theories of different research teams working in this area, there appear to be fundamental differences in the ways in which these theories specify the actual processing of text. But which differences are deep theoretical ones, and which are merely artifacts of implementation? I address this question by looking at the processes underlying two parsers which were influential during the 1970's: Riesbeck's parser *(4)* for Schank's conceptual dependency formalism (5); and the 'LNR' parser for Norman and Rumelhart's active semantic network formalism (3). The actual internal representation of text is <u>not</u> the main focus of this paper. Rather, it is the parsing process itself which is under scrutiny.

To carry out this analysis, I have implemented versions of both parsers using Kaplan's General Syntactic Processor, GSP (1). I will first describe the GSP versions of each parser, and then the way in which their best features can be combined.

P. RIESBECK'S PARSER

?.1 Overview

The intent of Riesbeck's parser is to build a conceptual dependency representation of *a* segment of text. In operation, the main verb of each segment (phrase) of text is located, and a representation of its 'meaning' (its decomposition) is constructed in memory. The surrounding text, primarily noun-phrases, is then systematically mapped onto vacant slots within the memory representation of the decomposed verb. The parser's 'expectations' are represented by active packets of production rules. Important phrase boundaries and slot-filler (case) information are signposted by words which may (a) satisfy existing expectations of the parser (by matching the condition side of a production rule) or (b) generate new expectations for the *parser* (by activating a new packet of production rules).

2.2 The GSP <u>version</u>

In GSP terms, Riesbeck's 'active' production rules are represented by standard ATN-style arcs in the grammar (6). The interrupts which activate new packets of rules are modelled in GSP by storing in the lexicon the code which causes control to shift to a new grammar state (since any change in the value of global variables, such as Riesbeck's 'request list', is formally equivalent to a new GSP-machine state).

For example, the lexical entry for the verb 'give' includes, among; other things, the following:

INTERRUPT: (SETR CONCEPT (GETF DECOMPOSITION))
(NEWSTATE GIVE0)

GSP's controlling executive knows to look at the 'interrupt' code stored in the lexicon. In this case, the code says to set the CONCEPT register to the structure found for this entry under the indicator DECOMPOSITION (i.e. its conceptual dependency representation), and to advance the grammar to the state labelled GIVE0. State GIVE0 is the one which initiates the actions corresponding to Rienbeck's setting up of new expectations, and is basically an entire sub-grammar devoted to the verb 'give'. This sub-grammar asks specific questions about the attributes of incoming noun phrases, and assigns these phrases to case slots accordingingly, e.g. the surface pattern HUMAN give HUMAN PHYSOBJ results in the second HUMAN getting placed in the RECIPient slot.

Although conceptual dependency emphasises the role of 'deep conceptual cases', the parser itself is necessarily a hybrid beast - it builds conceptual representation the instant a verb is encountered, but it fills vacant case slots on the basis of purely surface syntactic considerations, combined with relatively simple Katz & Fodor-type selectional restrictions (although Riesbeck does allow the restrictions to be relaxed if nothing else makes more sense). Kiesbeck's parser could in fact build some other kind of internal representation, based on any one of several linguistic theories of lexical decomposition, without any effect on the course of parsing.

Riesbeck's parser discovers noun phrases when a signalling word, such as 'a', interrupts processing and passes control to a noun-phrase sub-grammar. Active production rules which were pending before the 'a' was discovered are saved on a special 'hold' list, and restored when a 'phrasebreak' (e.g. a verb or a period) is encountered. At that point, an EVAL-PHRASE routine (not unlike an ATN BUILDQ) constructs an internal representation of the just-parsed noun-phrase. In GSP terms, this activity is almost identical to the standard ATN 'PUSH NP', except that the potentially expensive aspects of a PUSH and POP (namely, saving return states and building possibly incomplete NP structures) are not carried out until it is virtually certain that they will succeed. This variation is handled in the GSP version by having the appearance of an item such as a determiner (which would normally be sought at the beginning of an NP sub-grammar) initiate a PUSH 'safely' after that key item is encountered in the text. This is done by having the 'interrupt' code for determiners pass control directly to state NP in the grammar. Then, instead of performing a tentative POP when the NP sub-grammar thinks it is finished, the 'interrupt' code of verbs (and other items which signal a 'phrasebreak' in various contexts) initiates the equivalent of a 'safe' POP directly when it is appropriate.

Of course 'safe' doesn't always mean 'guaranteed'. It is clear that there are all sorts of examples which would cause Riesbeck's parser to go astray. However, Riesbeck is concerned mainly about the fundamental operating principles, rather than performance on a specific case. The assumption is that these principles could be applied to more elaborate examples. This remains to be demonstrated for a so-called 'conceptually-based' parser, although a related theme underlying the workings of Marcus' parser(2) indicates that 'safe' PUSHes and POPs indeed make powerful parsing tools. The advantages of lookahead in parsing are significant only when such lookahead is relatively cheap and reasonably accurate. The GSP implementation of Riesbeck's parser allows one to automatically incorportate fairly simple lookahead within the framework of other, more well established parsing mechanisms (e.g. the PUSH NP of ATNs).

3. THE LNR PARSER

3.1 Overview

The LNR parser is designed to map sentences onto a surface case frame. This case frame is an n-ary predicate which contains a pointer to the appropriate code responsible for decomposing the predicate into a canonical 'meaning' representtation. The case frame of each verb is stored in a memory network, and is referred to by the parser for guidance in picking up arguments to go with the main predicate. Aside from the subject and object slots, which typically surround the verb, other (arbitrary) case slots are signalled by the appearance of key prepositions.

3.2 The GSP version

The LNR parser is implemented as a fairly straight forward ATN, with the PUSH NP and PUSH PP arcs augmented with extra tests which must be passed for the transitions to be allowed. These tests are simply checks to see whether, say, the current prepositional phrase is consistent with the 'surface frame' stored in the lexical entry for the current verb, and whether the relevant case slot is still vacant. For example, the lexical entry for the verb 'give* includes the following:

```
SURFACE-FRAME: NP1: AGENT
               NP2: OBJ
               'TO': RECIP
```

The slots 'NP1' and 'NP2' are signposts for the

parser, saying that the NPs preceding and following the verb should be placed into the AGENT and OBJ registers, respectively. The 'to' slot is a signpost saying that a prepositional phrase beginning with 'to' is acceptable, and that its (nested) noun phrase should be placed in the RECIP register.

The LNR parser only decomposes the main verb after the surface case frame is entirely filled in. Since the parser consults only the surface-frame for guidance during processing, the final internal representation has no effect on the actual course of parsing, and could have been based on any of several theories of lexical decomposition, including conceptual dependency.

Selectional restrictions are not included at all in the LNR parsing process - this must be done as part of the code which is executed during decomposition. The problem with this approach is that it is not clear how total failure of such selectional tests would influence the course of parsing itself. A method of incorporating Riesbeck-style tests is described below.

4. A SYNTHESIS

Many of the differences between the LNR and Riesbeck parsers are apparent rather than real. LNR's use of a surface frame as a signpost to guide a general procedure (e.g. PUSH NP) to work on a specific verb is isomorphic to Riesbeck's branching off to an entirely separate sub-grammar following the verb. Also, LNR's test for a 'vacant slot' is equivalent to Riesbeck' use of self-deleting production rules. Neither the final decomposed representation of the main verb nor the time at which decomposition occurs have any effect on the actual course of parsing for either Riesbeck or LNR.

The two parsers do, however, differ in terms of whether they use interrupts and selectional restrictions to guide parsing. Riesbeck's parser uses both, and is on these counts computationally superior to the LNR parser. The main strength of the LNR parser is its notational clarity, which derives from its description as an ATN. This makes it an ideal starting point for a hybrid parser made by augmenting it with Riesbeck's use of interrupts and selectional restrictions.

Beginning with an ATN specification of the LNR parser, Riesbeck-style interrupts are added by splicing into the lexicon the 'interrupt' code for items which can initiate 'safe' PUSHes (e.g. determiners) and items which can initiate 'safe' POPs (e.g. verbs). The old PUSHes and POPs then become redundant, and can be eliminated.

Selectional restrictions of the simplicity of Riesbeck's (i.e. looking for features such as HUMAN, PHYSOBJ, etc.) are then easily added. This simply involves augmenting LNR's surface-frame slots with selectional information, e.g.

'TO': RECIP/HUMAN,

and then conjoining a general test onto the PUSH PP arc so that it tests whether the noun phrase following the preposition is an instance of the class named after the '/'.

Preliminary results with my own hybrid LNR/Riesbeck Parser (implemented in my POP-2 version of GSP) indicate that the combination is cost-effective for simple sentences, where the occurrence of obviously ill-fated PUSHes (e.g. testing for non-existing prepositional phrases) can be totally avoided. The selectional restrictions provide an extra heuristic for case slot-filling purposes, but can be easily fooled by counterexamples, so are better treated as 'preferences' to be followed tentatively. The most promising line for future development is the use of 'safe' PUSHes and POPs to handle multi-constituent lookahead, as in (2). I am currently working on a GSP version of Marcus' parser. Insights gained from placing his parser in a common framework with that of Riesbeck and LNR will be described in a forthcoming paper.

REFERENCES

(1) Kaplan, R.M. A general syntactic processor. In R. Pustin (ed.), Natural language processing. Englewood Cliffs, N.J.: Prentice-Hall, 1973.

(2) Marcus, M. A theory of syntactic recognition for natural language. AI Memo, AI Laboratory, MIT, Cambridge, Mass., 1978.

(3) Norman, D.A., Rumelhart, D.E., and the LNR Research Group. Explorations in cognition. San Francisco: W.H. Freeman, 1975•

(4) Riesbeck, C.K. Computational understanding: anaysis of sentences and context. Working paper 4, Istituto per gli Studi Semantici e Cognitivi, Castagnola, Switzerland, 1974.

(5) Schank, R.C. Conceptual dependency: a theory of natural language understanding. Cognitive Psychology, 197?, 3 552-631.

(6) Woods, W.A. Transition network grammars for natural language analysis. Comm. ACM 1970, 13' 591-606.

(7) Wilks, Y. An intelligent analyser and understander of English. Comm. ACM, 1975, 18, *264-274.*

CONSISTENCY OF THEORIES OF IDEAS

Bob Elschlager
Computer Science Department
Stanford University
Stanford, California 94305

Inconsistency is a problem with first order predicate calculus theories in which ideas are treated as objects. This paper presents general results on inconsistency, including an interesting restriction on the relation of knowing. A theory having a uniform method of expressing ideas, ideas of ideas, and so on, is sketched. It is shown how to restrict some of the axioms of this theory in order to obtain consistency. The method of obtaining consistency seems to have some generality.

1. INTRODUCTION

As has been noted, for example, in [4], one of the problems with first order predicate calculus (PC) theories that objectify ideas is inconsistency. The next section describes a consistent theory that treats ideas as objects. It describes the axioms AX of this theory, indicates the restrictions that make the theory consistent, briefly discusses the notion of constructive idea, and then deals with some of the details of the formulation. After that, it is shown how easily inconsistency can arise in such theories. Then there is stated a theorem which says that in spite of these inconsistency results the theory described here, along with a number of its extensions, is consistent.

2. THE THEORY

The axiom system AX of the theory states that everyone knows the axioms of logic and all that is logically deducible from them. The axiom system also states that whatever anyone knows is true, and that if one knows something then one knows that one knows it. There are also axioms that state what it means for a variable to be *free in* a string, as well as what it means for one string to be *free for* a variable in another string (these standard concepts of logic are used in specifying the logical axioms in the PC).

Most important for consistency considerations are the axioms that specify the relation of denotation. It is desired that every (closed) string should *denote* its value, or in other words, tnat the string should denote what it *names* (examples of naming are given below). As will be seen in the section on inconsistency, such a desire can easily lead to paradox. Instead we make the weaker assumption (axiom a4) that every (closed) *determinate* string denotes its value. (Determinate strings, defined in the next section, are a very general class of strings, yet they avoid paradox.) By thus weakening the assumption about denote, one can prove that any of a variety of extensions of the axiom system AX are consistent.

The final axioms of the system state that whatever is a logical consequence of any of these axioms and is determinate, is known by everyone in all states (these last axioms are defined inductively). I do not see that the determinate restriction in these final axioms is necessary, but I have not yet seen how to prove consistency without it.

At this point the notion of *constructive* can be briefly mentioned. One can optionally introduce this notion into the formulation (the consistency results below apply in either case). A constructive idea, together with a fixed body of knowledge, allows a person to identify the object named by the idea. The notion is related to the distinction between "being aware of and "actually knowing". Thus the idea "number-of-people-Jn(rooml)" makes one aware of a number without actually knowing what the number is. A person may be able deduce facts about a number (without actually knowing the number), but one actually knows the number only after deducing that the idea denotes the same thing as a constructive idea. In the notation illustrated later,

$$\exists\, U\ \text{constructive}(U) \wedge K(p,\ ='^{\bullet}(\text{number-of-people-in}^{\bullet}(\text{room1}^{\bullet}),U),\ s).$$

The notion of "can" is defined not only as knowing, but constructively knowing that an action will achieve a condition.

The remainder of this section sketches some of the details of the theory.

The language L of the theory is a (first order PC) language with sorts. It contains a set of basic symbols, and these can be *starred* any number of times to produce new symbols. For example, the starred versions of, say, symbols "K", ">", or "3" are "$K^{\bullet}$", "$>^{\bullet}$", or "$3^{\bullet}$", while the starred version of these are "$K^{\bullet\bullet}$", "$>^{\bullet\bullet}$", or "$3^{\bullet\bullet}$"; these are nine distinct symbols.

For purposes of explanation, ideas are concretized as strings of L (the word "string" is used to mean either a term or formula of L). There will be no indication of the sorts, although variables that range over strings are in upper case, variables ranging over variables are in gothic, and usually, but not necessarily, lower case is reserved for variables that range over objects other than ideas.

For examples of the intended use of the symbols of L and of the notion of naming, consider,

(1) Ted
(2) 3
(3) safe1
(4) Ted$^{\bullet}$
(5) 3$^{\bullet}$
(6) safe1$^{\bullet}$
(7) >(5,3)
(8) >$^{\bullet}$(5$^{\bullet}$,3$^{\bullet}$)
(9) >$^{\bullet\bullet}$(5$^{\bullet\bullet}$,3$^{\bullet\bullet}$)
(10) >(x,3)
(11) >$^{\bullet}$(x$^{\bullet}$,3$^{\bullet}$)
(12) >$^{\bullet}$(X,3$^{\bullet}$)
(13) ∃ x >(x,3)
(14) ∃$^{\bullet}$(x$^{\bullet}$,>$^{\bullet}$(x$^{\bullet}$,3$^{\bullet}$))
(15) ∃$^{\bullet}$(X,>$^{\bullet}$(x$^{\bullet}$,3$^{\bullet}$)).

Strings 1-3 name Ted, 3, and safe1, respectively. On the other hand, 4-6 name the strings "Ted", "3", and "safe1", respectively.

String 7 says that 5 is greater than 3, and therefore it names true (this is a truth independent of states). Strings 8-9 name the strings ">(5,3)" and ">$^{\bullet}$(5$^{\bullet}$,3$^{\bullet}$)", respectively. Whether 10 names true or false depends on the value assigned to the variable x.

String 11 names the string ">(x,3)". What 12 names depends on the value assigned to the variable X; if X is assigned "6","7+3", or "y+6", then 12 would name ">(6,3)", ">(7+3,3)", or ">(y+6,3)", respectively. The string 13 names true since there is a number greater than 3, while 14 names the string "∃ x >(x,3)". What 15 names depends on the value assigned to X; if X were assigned "x", "y", or "z", then 15 would name "∃ x >(x,3)", "∃ y >(x,3)", or "∃ z >(x,3)", respectively.

One important relation is that of knowing. This relation has three arguments; the first argument is a person, the second argument a string (of the sort that names a truth value), and the third argument a state. The intended interpretation is that in the state the person knows that the string is true. Thus, the first of

(16) K(Ted,>$^{\bullet}$(5$^{\bullet}$,3$^{\bullet}$),state1)
(17) K(Ted, K$^{\bullet}$(Ted$^{\bullet}$,>$^{\bullet\bullet}$(5$^{\bullet\bullet}$,3$^{\bullet\bullet}$),state2$^{\bullet}$),state1)

says that in state1 Ted knows that ">(5,3)" is true, or, that 5 is greater than 3; formula 17 says that in state1 Ted knows that in state2 Ted knew (will know) that 5 is greater than 3.

Another special relation is that for denotation, represented by the relation symbol "denote". Examples are,

(18) denote(3$^{\bullet}$,3)
(19) denote(∃$^{\bullet}$(x$^{\bullet}$,>$^{\bullet}$(x$^{\bullet}$,0$^{\bullet}$)),TRUE).

The first says that the string "3" denotes 3, the second that the string "∃ x >(x,0)" denotes true.

For the purpose of conveniently specifying axiom schemas, two mappings from strings to strings are defined: these mappings are not functions in the model. If B is a formula, (*cl* B) indicates the closure of B,i.e., B with all its free variables universally quantified. Thus,

(20) *cl* ("∃ z (x+y =' z)") = "∀x y ∃ z (x+y =' z)".

A string will be said to be *closed* if it has no free variables; so if B is closed, then (*cl* B) is the same as B.

The mapping *id* replaces all symbols by their starred versions, changing quantifiers into functions.

(21) *id*(">(x,5)") = ">$^{\bullet}$(x$^{\bullet}$,5$^{\bullet}$)"
(22) *id*("K(Streleski,>$^{\bullet}$(5$^{\bullet}$,3$^{\bullet}$),state1)") =
"K$^{\bullet}$(Streleski$^{\bullet}$,>$^{\bullet\bullet}$(5$^{\bullet\bullet}$,3$^{\bullet\bullet}$),state1$^{\bullet}$"
(23) *id*("∃ x >(x,3)") = "∃$^{\bullet}$(x$^{\bullet}$,>$^{\bullet}$(x$^{\bullet}$,3$^{\bullet}$))"

Another convention for specifying schemas is that B and C are syntactic variables that range over strings; they are not variables of L

To illustrate these conventions consider the axiom schema

(24) K(p,*id*(B),s) ⊃ (*cl* B), where B is any formula.

This says that whatever anyone knows is true. By taking B as "∃ x >(x,y)" or "∃ s1 red(block1,s1)", one gets two of the instances of this schema:

(25) K(p,∃$^{\bullet}$(x$^{\bullet}$,>$^{\bullet}$(x$^{\bullet}$,y$^{\bullet}$)),s) ⊃ ∀ y ∃ x >(x,y)
(26) K(p,∃$^{\bullet}$(s1$^{\bullet}$,red$^{\bullet}$(block1$^{\bullet}$,s1$^{\bullet}$)),s) ⊃ ∃ s1 red(block1,s1).

Some examples of the axioms and axiom schemas AX of the theory are

(a1) ∀ U V p s [K(p,⊃$^{\bullet}$(U,⊃$^{\bullet}$(V,U))s)]
(a2) ∀ V W [K(p,V,s) ∧ K(p,⊃$^{\bullet}$(V,W),s) ⊃ K(p,W,s)]
(a3) ∀ X V [K(p,V,s) ⊃ K(p,∀$^{\bullet}$(X,V),s)]
(a4)denote(*id*(B),B), for every closed determinate string B
(a5) ∀ U x y [denote(U,x) ∧denote(U,y) ⊃ (x =' y)].

Axiom a1 states that in all states all people know that the string (S1 ⊃ (S2 ⊃ S1)) is true for all formulas S1 and S2. Axiom a2 states that in any state if someone knows S1 and (S1 ⊃ S2), then in that state they know S2, where S1 and S2 are any formulas.

Axiom schema a4 states that every closed determinate string denotes its value, while axiom a5 says that a string can denote at most one thing.

3. DETERMINATE STRINGS

We define *determinate* string inductively as follows. A single variable or constant is determinate. The string F(x1,...,xn) is determinate for any variables x1,...,xn and any function (or relation) symbol F *other than* "K" or "denote". If x1,...,xn are some of the free variables in a determinate string B(x1,...,xn), and if B1,...,Bn are determinate strings, then B(B1,...,Bn) is determinate. If B is determinate, then so are (∀ x B) and (∃ x B). Finally, if B is determinate, and if u,p,and s are variables (of the appropriate sort), then (denote(*id*(B),u)) and (K(p,*id*(B),s)) are determinate.

Notice that *any* string that does not contain the symbols "K" or "denote" is determinate (the string may contain starred symbols, including K$^{\bullet}$" and "denote$^{\bullet}$", since these are distinct from "K" and "denote"). In general, a string is determinate if every first argument of a "denote" in the string, as well as every second argument of a "K" in the string, is an *id* of some determinate string. Some examples of determinate strings are

(e1) ∀ x (x > y)
(e2) Ted
(e3) denote (Ted$^{\bullet}$,x) ∧ K(p,>$^{\bullet}$(x$^{\bullet}$,y$^{\bullet}$),s)
(e4) ∃ p K(p,denote$^{\bullet}$(Ted$^{\bullet\bullet}$,x$^{\bullet}$),s)
(e5) denote(denote$^{\bullet}$(Ted$^{\bullet\bullet}$,x$^{\bullet}$),denote(W,y)).

Some examples of strings that are not determinate are

(e6) K(p,>$^{\bullet}$(X,y$^{\bullet}$),s)
(e7) denote(V,x)
(e8) denote(denote$^{\bullet}$(V$^{\bullet}$,x$^{\bullet}$),u).

4. INCONSISTENCY

The results in this section show how inconsistency considerations restrict both the "denote" and the "K" relation. These results are fairly general because only the few conditions explicitly stated in this section will be used: the axioms in the other sections are *not* assumed. In the following, Sub(u,vw) and Tr(u) are formulas. For reference purposes, we list,

(sb1) $\forall u\ v\ w1\ w2\ (Sub(u,v,w1) \wedge Sub(u,v,w2) \supset w1 =' w2)$
(sb2) $Sub(id(B),id(C(v)),id(C(id(B))))$, for all strings B and C
(tr1) $Tr(id(B)) \equiv (B \equiv TRUE)$, for all closed formulas B
(dn1) $denote(id(B),B)$, for all closed formulas B
(k1) $\forall\ p\ s\ [K(p,id(B),s) \supset cl(B)]$, for all formulas B
(k2) $cl(B) \supset \exists\ p\ s\ K(p,id(B),s)$, for every formula B.

One may think of sb1 and sb2 as saying that the formula Sub(u,v,w) expresses substitution; of tr1 as saying that the formula Tr(u) expresses truth; of k1 as saying that whatever anyone knows is true; of k2 as saying that whatever is true is known in at least one state by at least one person; of dn1 as being an unrestricted version of the "denote" relation. (In the following, when we say that one of these schemas is provable or is true, we mean that every instance of it is provable or is true.)

Theorem. Let A be a set of formulas. Suppose that there is a formula Sub(u,v,w) such that sb1 and sb2 are provable from A. Furthermore, suppose either that dn1 is provable from A or that there is a formula Tr(u) such that tr1 is provable from A. Then A is inconsistent.

This is Tarski's theorem (see for instance pg 85 of [1]) converted to a sort system. One might rephrase this theorem by saying that if substitution is definable in A, then one can allow neither the unrestricted version dn1 of the denotation relation (cf. axiom a4) nor an unrestricted version tr1 of the truth relation.

Inconsistency results also place restrictions on the relation of knowing.

Theorem. Let A be a set of formulas. Suppose that Sub(u,v,w) is a formula such that sb1 and sb2 are provable from A. Suppose also that k1 and k2 are provable from A. Then A is inconsistent.

This theorem has an interesting formulation in terms of models and never known truths.

Theorem. Let Sub(u,v,w) be a formula and let M be a model of sb1,sb2, and k1 (i.e., sb1, sb2, and k1 are true of M). Then in M, there is a "never known truth", which is to say, there is a formula C such that (*cl* C) is true of M and such that $(\forall p\ s\ \neg K(p,id(C),s))$ is true of M. In other words, if in M substitution is expressible, and if in M whatever anyone knows is true, then there is a truth that no one in any state ever knows.

There are variations and generalizations of these four theorems. Instead of using substitution as given in sb1 and sb2, one can use Peano axioms together with a certain relation betwen strings. There are likely many such variations. The last theorem can be generalized in a further direction: the set of never known truths in M is infinite in number, and cannot be "delimited in certain ways".

5. CONSISTENCY

On account of the previous inconsistency results, the consideration of consistency is an important one. Let S be a set of formulas that satisfies either c1 or c2.

(c1) S is consistent, and no part of any formula in S *refers* to an idea, which is to say, S speaks only of objects that are not ideas.
(c2) No formula in S contains the symbols "denote" or "K" (though they may contain such symbols as "denote*" and "K"), and there is a model of S (i.e., a structure in which all the formulas of S are true) such that the ideas in the model are the strings of L, and such that the connection between the starred symbols and the model is as illustrated by the explanations of examples 1-15.

Theorem. The union of S together with the axioms AX of the theory described earlier is consistent.

For example, the axioms AX together with the Peano axioms and axioms expressing a variety of string substitutions and relations is consistent.

6. CONCLUSION

The axiom system AX was motivated by the desire to have a powerful set of axioms so that the consistency results would be stronger; no overall attention was paid to efficiency of proofs [61

The results presented here are fairly general. The inconsistency results depend on few assumptions, and those could be modified in many ways. The proof of the consistency theorem is more complicated. A model of the axioms is created in an infinite sequence of stages, with "denote" and "K" appropriately extended at each stage by what has a fixed value (is "determinate") no matter how the rest of the stages are completed. This method seems general, going beyond the particular theory above or its formulation.

REFERENCES

[1] Lyndon, Roger C. (1966) *Notes on Logic*, Van Nostrand, New York.
[2] McCarthy, J. and Hayes, P. J. (1969) Some Philosophical Problems from the Standpoint of Artificial Intelligence, in B. Meltzer and D. Michie (eds.) *Machine Intelligence 4*, 463-502, Edinburgh: Edinburgh University Press.
[3] McCarthy, J. (1977) Epistemological Problems of Artificial Intelligence, *1977 IJCAI Proceedings*.
[4] McCarthy, J. (1979) *First Order Theories of Individual Concepts and Propositions*, Stanford AI Lab, AIM-325.
[5] Moore, Robert C. (1977) Reasoning about Knowledge and Action, *1977 IJCAI Proceedings*.
[6] Moore, Robert C. (1979) *Reasoning about Knowledge and Action*, Ph.D. Dissertation, Comp.Sci.Dept., MIT.

A COMPUTATIONAL APPROACH TO THE STUDY OF HUMAN SKILL ACQUISITION.

Jan J. Elshout
Laboratory for Experimental Psychology, University of Amsterdam
Weesperplein 8
Amsterdam, The Netherlands

Bob J. Wielinga
Laboratory for Experimental Psychology
University of Amsterdam
Weesperplein 8
Amsterdam, The Netherlands

A general model for "learning by doing" is presented which accounts for a number of phenomena observed in protocols of human subjects solving a series of similar problems. The model is based on the conjecture that skill acquisition is a process of progressive development of problem directed knowledge structures which are to a large extent domain specific. A program is described for encoding protocols of human subjects in terms of the model.

When confronted with a series of similar problems to solve, humans usually improve their performance with experience. To investigate this "spontaneous" process of skill acquisition, we have chosen a methodology similar to that of \1\ : taking protocols of human subjects solving a series of problems and constructing a program which is capable of coding these protocols, in terms of a computational model, possibly with the help of a human coder. The kind of problem we have chosen is the two waterjug problem: given two waterjugs with known capacities, a tap and a drain, produce a given quantity by filling, emptying and pouring the jugs. This type of problem is generally considered to be "semantically poor", but it has the advantage that the transition from novice to expert in the domain is a matter of only a few hours exercise. Moreover, analysis of protocols of novices shows that many "semantic problems" have to be solved before expert behaviour emerges [2] .

Detailed analysis of some of the protocols by hand led to three major observations: 1) many processes apparent in the protocols can best be described in computational terms (eg. planning, debugging, global/local control, demons), 2) "learning by doing" is a complex interaction between a variety of problem solving processes, and is not easily described by a set of independent learning mechanisms and 3) learning manifests itself as a qualitative change in the problem solving process: things are not just done faster, they are done differently. The latter observation gives rise to the basic conjecture that underlies our research: improved performance is a manifestation of the progressive development of problem directed knowledge structures (schemata) which are to a large extent domain specific.

On the basis of these observations (and a number of more specific ones, see [2]), we have developed a general model of the skill acquisition process. In our model we distinguish a number of functional components (cf. [3]): ORIENTATE, SOLVE, REFLECT and REALIZE, each represented by a process. Processes may be simultaneously active and communicate via a common working memory of which the processes themselves are part. Most control decisions in the system are taken locally by specific processes, but in some cases (e.g. major errors or break throughs) focus of attention has to be shifted from one process to another. This more global control function is performed by,a separate process: the "Research Director" [4] . The Research Director process has global knowledge about the ongoing activities, the distribution of resources, the current goals, a global measure of progress and a set of rules describing what to do when something unexpected happens. The basic control functions of the Research Director are: initiating activities, providing these activities with advise on the currently appropriate mode of operation (e.g. "sloppy", "reflective", "mechanic", "careful"), interpreting general comments or gripes from processes and acting accordingly if necessary.

When congronted with a new problem the Research Director will activate ORIENTATE. The main task of ORIENTATE is to construct (or retrieve from

memory) a representation of the problem: the problemconception [5] . In the problem conception information is stored about the specific problem parameters, possibly applicable plans, information about bugs that occured previously etc. In the protocolfragment shown in Appendix 1, orientation is restricted to assimilating problem parameters (line 3). Other protocols show more elaborate orientation such as computing various relations between the numbers involved.

Given a problem conception which is well enough specified to attempt a solution of the problem, the Research Director will activate SOLVE. The performance of SOLVE is primarily dependent on the availability of problem directed schemata (plans, algorithms, rules). When an algorithm or a plan is available in the problem conception it will be executed. Planning (which is a subprocess of SOLVE) is often performed when the problem (as the subject sees it) allows some form of abstraction or when a new result is obtained. In the waterjug problem domain planning amounts to solving the problem in number space using arithmetic operations. Many subjects use plans (or algorithms) which are not based on abstractions of the operators, but on generalization of previous solutions. The subject who produced the exampleprotocol, uses an algoritlim developed while solving previous problems. This algorithm consists of two parts: an "Opening" (fill the smaller jug, pour it over, fill the larger jug up, (lines 4 to 11), and a "keep-plan" (create a new quantity by pouring and save it in the smaller jug; lines 12-15). When no plans or algorithms are avail" able,SOLVE has to resort to search, which may be guided by domain specific rules such as "never fill up two jugs", "new numbers are winners".

The activities of SOLVE are monitored by the function REFLECT and its processes. REFLECT evaluates the rate of progress, checks the efficiency of the problem solving process (e.g. by detecting redundant steps; see lines 6-8 in the example), diagnoses errors in plans and algorithms (cf. HACKER '6]), suggests therapy for buggy plans, and tries to generate abstractions of existing schemata (plans, operations). The REALIZE component of the model has as explicit task to modify existing knowledge structures or to create new ones. REALIZE fixes bugs in plans according to the diagnosis and therapy suggested by REFLECT. REALIZE also builds plans from generalizations of previous solutions.

On the basis of the general model sketched above we have constructed a protocol encoding programm: PANKAN. PANKAN consists of two basic components: 1) a collection of processes representing the functions in the model and 2) a component which relates the behaviour of the model to utterances in the protocol. The implementations of the functional components of the model differ in their degree of detail. ORIENTATE simply retrieves a problem conception schema from memory and fills the appropriate slots with the actual problem parameters. SOLVE accomodates several options: simple planning, execution of plans and algorithms, search. REFLECT currently performs the following functions: evaluation of obtained quantities according to criteria such as "newness", "known subgoal", etc.; demon triggered evaluation of "phantom numbers" (empty space in a partially filled jug), which causes the Research Director to initiate a new problem solving process; detection of redundent steps; diagnosis of errors due to "clobbered preconditions". REALIZE is currently able to implement the suggestions given by REFLECT.

Apart from these few exceptions, the current versions of REFLECT and REALIZE are not able to change knowledge structures or create new ones. The program has to be provided with the knowledge that is needed to account for the behaviour of subjects in various stages of the learning process.

The second component of PANKAN (ANALYSE) monitors the behaviour of the learning and problem solving processes and tries to relate this behaviour to the utterances in the protocol, by either interpreting the protocol on the basis of keywords or by asking questions to a human coder (see Appendix 2 for examples). When the behaviour of the model is not in accordance with the protocol, ANALYSE can take a variety of actions. If the model allows a choice of behaviours, ANALYSE will redirect the activity of the model. Another possibility is that there is a need for a specific schema which is not part of the database of the model. ANALYSE will try to find one in a library of schemata. A third action that ANALYSE can take is to shift the focus of attention (e.g. from planning to search). When all of these actions fail to bring model and protocol in agreement, the program fails to encode (part of) the protocol.

When provided with the appropriate knowledge structures, the current version of PANKAN is able to analyse many "simple" protocols, i.e. protocols reflecting more or less expert level, where no serious deviations of the normal execution of plan occur. Of the more interesting protocols i.e. where complex process interactions occur, the current progran can only analyse a few. So far we have not been able to

analyse a complete series of 15 protocols. Problems that are hampering such an analysis include: the limited power of the system to predict and resolve confusions (e.g. confusing empty space in a jug with its content; errors due to neglect of preconditions), limited capability of creating and selecting the appropriate knowledge structures, problems with major shifts in attention and the corresponding control problems.
In spite of these difficulties we believe our effort to be a fruitful one. Using concepts and ideas from AI research, has considerated enhanced our capabilities to analyse protocols in such detail that interesting learning phenomena become evident. The construction of a protocol analysis program has forced us to formalize both the framework on which our protocol analysis is built and specific analytic methods.

APPENDIX 1.

Analysis of a protocolfragment made by PANKAN. Jugs are 7 and 6, goal is 4. Only main activities are given.

```
1    4
1    now, the pile is getting thinner,
     (conversation because of new Datum)
     (Research director active)
     (approach is Reflective)
     (orientation)
2    but it is still going slowly...
     (conversation because of New Datum)
3    7 6 4 (problemconception is Pc7a1)
     (assimilation of new data)
     (Demon active)
     (solving)
     (Reflect active)
4    hum, fill this one
     (using Plan7a1)
     (using Openin1)
     (Fill of righ jug)
5    6 ... (Newnumber 6)
     (current state is 0  6)
6    you don't have to transfer this one to
     the other side
     (Evaluate active)
     (hold all)
7    that doesn't make sense,
     (result of Evaluate is: redundency)
8    hum, I should have filled the other one...
     (Therapist active)
     (Therap\1 seen as a Therapy it has plan =
     Openin1
     replacement = Fill1A) suggested)
     (Realize active)
     (Opening changed)
9    anyway, 6  0
     (continue all)
     (Pour of righ jug)
     (current state is 6   0)
10   you fill this one up
     (Fill of left jug)
11   7   0  (Newnumber 7)
     (current state is 7  0)
12   you pour it over
     (using Keep.p1)
     (Pour of left jug)
     (Newnumber 1)
     (current state is 1  6)
13   and you empty that one
     (Empty right jug)
14   1  0 (current state is 1  0)
15   and transfer it to the other side...
     (Keep of left jug)(Current state is 0 1)
```

APPENDIX 2

Questions asked to the coder and answers given (underlined) during analysis of the example protocol. Numbers indicate lines.

line 1: do you make conversation not related to the problem?
yes, 1 2
line 8: do you make an evaluation?
yes
do you criticize the current plan?
yes
line 11: do you finish the evaluation here?
yes
line 12: do you want to pour the right jug into the left one?
yes, 1 2

REFERENCES

[1] Bhaskar, R. & Simon, H.A. "Problem solving in semantically rich domains: an example from engineering thermodynamics." Cognitive Science. 1:2 (1977) 193-215.

[2] Elshout, J.J. & Wielinga, B.J. "A computatinal approach to the study of Human Skill Acquisition." University of Amsterdam, ICO.256,1979.

[3] Smith, R.G., Mitchell, T.M., Chestedi, R.A. & Buchanan, B.G. "A model for learning systems" In Proc. IJCAI-77. MIT, Cambridge, Mass., August, 1977, pp. 338-343.

[4] Hardy, S. "Synthesis of LISP functions from examples." Ph.D. thesis, University of Essex, 1976.

[5] de Groot, A.D. "Thought and choice in chess". The Hague: Mouton, 1965.

[6] Sussman, G.J. "A computer model of skill acquisition", New York: American Elsevier, 1975.

KNOBS: AN EXPERIMENTAL KNOWLEDGE BASED TACTICAL AIR MISSION PLANNING SYSTEM AND A RULE BASED AIRCRAFT IDENTIFICATION SIMULATION FACILITY

C. Engelman
Charles H. Berg
Miriam Bischoff

The MITRE Corporation
P.O. Box 208
Bedford, Massachusetts 01730
U.S.A.

KNOBS (from Knowledge Based System), proper, refers to an experimental system under development at the MITRE Corporation to explore the applicability of artificial intelligence techniques to the implementation of an automated, extremely flexible tactical mission planning consultant. This paper will discuss that system and then another which supports the rule based simulation of aircraft identification strategies.

1. MISSION PLANNING:

Tactical mission planning may be posed as the intelligent utilization of one's available tactical air resources. Projections of possible theater level tactical conflicts indicate an expectation of very intense, complex, and time sensitive planning requirements as well as a requirement to improvise and implement novel tactical doctrine in response to unforeseen modes of weapon systems employment by the enemy. It is our thesis that knowledge based techniques represent the most promising approach towards the achievement of such trainable software. Work has concentrated on the problem of planning interdiction attacks, particularly in the choice of attack aircraft, munitions, and the electronic protection required to gain those aircraft safe passage. Currently the system is undergoing intense detailed design, while parts have been implemented in INTERLISP. These latter are sufficient to serve frequent "advocacy" demonstrations of the flexibility, human legibility and system self-explanation capabilities associated with knowledge based systems.

1.1 DATA BASE: There are basically two data bases; one for resources, the other for targets. Targets are to be stored in a hierarchical data base represented as list structure in the INTERLISP virtual storage. Individual targets are to be stored as FRL £3] frames. At this writing, we are involved in the translation of FRL from MACLISP into INTERLISP. The individual target frames are in fact depositories of simple record type information, with the procedures to be invoked when such values are needed or changed being derived by inheritance chains through more generic "target class" frames.

1.2 KNOWLEDGE REPRESENTATIONS: The system will employ two forms of knowledge representation; production rules and frames. The rules are chained backwards in a deductive manner to manage such generic choices as aircraft, weapons, support, and electronic counter measures. Software has been implemented for the interpretation, the translation (in both directions) between the internal form of the rules and the user legible/writable form, the explanation of deductions (in terms of the entire chain of rules and data employed in a deduction), and the creation and editing of

A typical (MYCIN-like) rule for the selection of munitions is:

```
                                --TR17--
IF:
      1: THE target IS PINPOINT
  AND 2: THE CLOUD COVER OF THE target
         IS LESS THAN 2/8
  AND 3: THE time on target IS WITHIN DAYTIME
  AND 4: THE AAA DEFENSE OF THE target
         IS MORE THAN LIGHT
  AND 5: PAVE KNIFE IS AVAILABLE AT time on target
THEN:
      1: THE BEST MUNITIONS FOR THE target IS
         ONE OF: MK-82-LG, MK-83-LG, MK-84-LG
```

This work was sponsored by the Directorate of Mathematical and Information Sciences of the Air Force Office of Scientific Research, the Information Sciences Division of the Rome Air Development Center, and the MITRE Corporation Independent Research Program.

Lower case indicates variables. The justification for this rule can be inferred from the discussion in [2], P- 782.

The syntactic pattern matcher employed by the inference mechanism is quite general. It includes synonym and spelling correction capabilities, as well as provision for restrictions on variable instantiations defined by arbitrary multivariate predicates. This matcher is also employed to implement a flexible command-query system.

Frames will be employed to represent both the data base and, at a higher level, missions and support sub-missions. Frames are particularly attractive with respect to the time sharing of support missions, (e.g., reconnaisance, refueling, and SAM suppression) among several attack missions.

1.3 EXAMPLE: **The following dialog (user lines start with "&&") is possible in the current system:**

```
&&  WHAT IS THE HARDNESS OF TARGET 2?
THE HARDNESS OF TARGET 2 IS UNKNOWN
&&  HOW ABOUT TARGET 1?
TARGET 1 IS VERY HARD
&&  WHY?
BY TR3:  TARGET 1 IS VERY HARD
 SINCE:
    1: DATA: TARGET 1 IS A BRIDGE
&&  IS TARGET 3 AN AIRBASE?
NO, TARGET 3 IS AN SA-4
&&  CHOOSE MUNITIONS FOR IT
THE BEST MUNITIONS FOR TARGET 3 IS SHRIKE
&&  WHY?
BY MR4: THE BEST MUNITIONS FOR TARGET 3 IS SHRIKE
 SINCE:
       1:  BY TR1: TARGET 3 IS AN EMITTER
       SINCE:
          1:  BY TR4: TARGET 3 IS A SAM
          SINCE:
             1:  DATA: TARGET 3 IS AN SA-4
```

For more information about SA-4's, SHRIKEs and why one is bad for the other, see [2] pp. 77 and 150.

2. AIRCRAFT IDENTIFICATION:

A critical contemporary military need is for a method of sure, safe identification of aircraft, capable of reliable distinction between friend and foe (IFF). We shall concentrate here on approaches to the IFF problem based primarily on "cooperative" techniques. It seems possible today to build electronic query and response systems that almost solve the problem electronically. Difficulties, though, stem from the facts that in certain theaters it is anticipated that an aircraft would be required to pass thousands of such query-response tests -- and pass all of them to survive, that equipment can fail completely, that a spread spectrum signal might be detected if it happens to be near enough to and in the direction of an enemy receiver which is afforded sufficient integration time, and that jamming is in fact quite often possible. In addition, tomorrow's war must be as silent as possible, for in querying an aircraft for identification, the inquirer exposes both himself and the plane, should he be a friend. Therefore one should not normally query without the intent to fire. It seems likely that the query should also transmit (crypto-securely) information such as the position and the weapon the inquirer is contemplating firing. The aircraft may then use that information to decide whether or not to respond on the basis of all the threats he has perceived or inferred. The logic of the IFF problem thus becomes one of evaluating procedural strategies in a collusive, n-person game. Such evaluation is proving error-prone as more complex IFF "solutions" are being proposed and more thorough analyses demanded.

2.1 WHAT SHOULD BE DONE: Our first response was to initiate the design of a rather general knowledge based IFF game definition and execution system. In fact, there are are a number of "electronic warfare" problems that share this same "gaming" characteristic: electronic counter and counter-counter measures, the exploitation or defeat of the enemy's surveillance or intelligence structure, the maintenance of survivable communications and/or surveillance networks, or the penetration by an interdiction aircraft of the enemy's defenses. For a knowledge based approach to this last, see [1]. While we are stressing the IFF problem in this paper, it was our intent that the game definition/execution mechanism be sufficiently general to support study of the logic of many of these "electro-magnetic warfare" problems.

2.2 WHAT WE HAVE DONE: KNOBS has been augmented by the addition of a forward chaining production system in which the conflict set resolution algorithm chooses the minimal set of rules that covers the maximal set of facts (as discussed in [4]). This has been employed to create a simulation of the world as it might be

seen by a single "Blue" surface-to-air missile (SAM) site. The SAM has a number of choices to make: whether to turn its radar on or off, whether or not to query each plane it has detected, and whether or not to fire at such a plane. Our initial thought was to model the SAM's strategies for these choices by a set of productions. What we have done, in fact, is to write a number of production sets, basically one for each iterative step in the game, and to use the production system interpreter as the mechanism for the entire simulation. This rather simple idea turns out to be quite powerful. The resulting simulations, in which not just the strategies but also the weapons systems limitations and even the "physics" are represented by human legible rules, yield a strong sense of cause and effect between hypothesized principles of behavior and game results.

2.3 SAMPLE RULE: The following is a quite conservative rule meant to express a determination not to query an aircraft unless it is perceived as a possible direct threat to the SAM:

```
                --QUERY1--
IF:
          1:   RADAR IS ON
   AND    2:   SAM HAS MISSILES LEFT
   AND    3:   ?A/C HAS NOT BEEN FIGURED
   AND    4:   ?A/C HAS NOT BEEN QUERIED
   AND    5:   ?A/C IS LOW
   AND    6:   ?A/C IS FLYING WEST
THEN:
          1:   QUERY ?A/C
```

Identifiers starting with "?" represent existentially quantified variables.

2.4 SAMPLE __RUN: The listing below is an excerpt from an actual game run. We have preserved just enough to trace the dissimilar fates of two planes, P0004, a Blue F-IIIF, and P0008, a Red MIG-21, with rather similar histories:

```
STARTING TURN // 1
. . .
AIRCRAFT P0004 OF TYPE F-IIIF HAS ENTERED RANGE
. . .
AIRCRAFT P0008 OF TYPE MIG-21 HAS ENTERED RANGE
SAM QUERIES P0004
FIGURE P0004 IS RED
SI FIRED AT P0004
.. .
SAM QUERIES P000B
FIGURE P0008 IS RED
S3 FIRED AT P0008
...
STARTING TURN #2
...
STARTING TURN #3
...
MISSILE S1 MISSED
...
MISSILE S3 MISSED
...
S5 FIRED AT P0008
...
STARTING TURN #4
P0004 --A F-111F -- PASSED THROUGH SAFELY
...
STARTING TURN # 5
...
MISSILE S5 HIT
P0008 -- A MIG-21 -- DESTROYED
```

By reference to the rules and some descriptive data, it is very easy to understand what happened. Both planes aroused the SAM's suspicion since they were flying WEST and LOW. Both therefore got queried. Neither responded: the M1G because it was Red and therefore did not know the code key and the F-111 because its IFF equipment was broken. Both therefore were figured Red and were fired upon. Both were saved from this round of missiles by their ECM. P0004 then passed out of range. Unfortunately for P0008, it stayed in range long enough to allow the SAM time for a second shot. This time the ECM was inadequate and the A/C was destroyed.

REFERENCES

1. Garvey, T. D., Fischler, M. A., Martin, R. A., and Sinko, J. W., "Machine Intelligence Based Multi-Sensor ESM System", Interim Technical Report, SRI International, Menlo Park, July 1978.

2. Pretty, R. T., (Ed.), Jane's Weapon Systems 1978, Franklin Watts Inc., New "York, 1977.

3. Roberts, R. B. and Goldstein, I. P., "The FRL Manual", MIT AI Report 409, Cambridge, June 1977.

4. Rosenschein, S. J., "Structuring a Pattern Space, with Applications to Lexical Information and Event Interpretation." Doctoral Dissertation, University of Pennsylvania, Philadelphia, 1975

STRUCTURE AND FUNCTION OF THE CRYSALIS SYSTEM

Robert Engelmore[1]
Computer Science Department
Stanford University
Stanford, California 04306

Allan Terry[2]
Dept. of Information and Computer Science
University of California, Irvine
Irvine, California 02717

ABSTRACT

CRYSALIS Is a knowledge-based system whose goal Is to infer the three-dimensional structures of proteins from x-ray crystallographic data. The system uses both formal and Judgmental knowledge from experts to select appropriate procedures and to constrain the space of plausible protein structures. The hypothesis generating and testing procedures operate upon a variety of representations of the data, and work with several different descriptions of the structure being inferred. This paper focuses on the architecture of the system, and points out some of its Interesting features. An example of the system's performance Is given.

1 Introduction

In this paper we present an application of Artificial Intelligence methodology to the domain of Protein Crystallography. [3] The long term practical goal of this work Is the creation of a fully automated system for protein structure determination, starting with data collection and ending with an accurate 3-D model of the molecule. Most of the software already exists at the two ends of the process. At the "front end", programs exist for reducing and transforming the *x-ray* diffraction data, and estimating phases. The result of this processing is an electron density map (EDM), which gives a blurred view of the electron cloud surrounding the molecule. At the "back *end"* are a variety of numerical techniques for taking a full or partial model of the protein and iteratively refining the atomic coordinates so that the model Is a best compromise between one which best fits the data and one which best matches Ideal stereochemical constraints [Harmans74] [Agarwal77]. The "middle" portion of the process is the generation of a first-order model of the protein, based on an Interpretation of the EDM and the amino acid sequence (when known). EDM Interpretation has traditionally been a manual process of visual pattern recognition, for which CRT displays have become a significant aid in recent years. Although the use of a large computing system In conjunction with a graphical display has made a significant reduction in EDM Interpretation time [T$arnoglou77], the knowledge used In matching the molecule to the EDM remains in the head of the model builder. Our aim Is to capture that knowledge within an automated EDM Interpretation program, thereby closing the gap which remains in building a fully automated system.

Many diverse sources of knowledge contribute to the Inference of a protein structure from an EDM and associated data. Examples: chemical knowledge about protein composition Is used to estimate overall size and to generate expectations about special features that should be present In the data; stereochemical knowledge about protein conformation is used to constrain the relative positions of atoms (e.g. certain subsets of atoms form rigid groups); *X-ray* crystallographic knowledge is used to detect symmetries, or to match a hypothesized structure against the data; specific heuristics for interpreting EDMs are continually Invoked to match features In the EDM with expectations from the model.

In order to use the knowledge sources efficiently, a global data base -- the "blackboard" -- Is constructed which contains the currently active hypothesis elements, at all levels of description. The decision to activate a particular knowledge source Is not pre-established, but depends on the current state of the solution and what available knowledge source Is most likely to make further progress. The control Is, to a large extent, determined by what has Just been learned: a small change In the state of the "blackboard" may establish a new Island of opportunity, providing the preconditions to Instantiate further knowledge sources.

Figure 1 shows the types of data and hypotheses that are

[1] Present address Is DARPA/IPTO, 1400 Wilson Blvd., Arlington, Va. 22200

[2] Present address Is Dept. of Computer Science, Stanford University, Stanford Ca 04306

[3] This work was supported by the National Science Foundation IMCS74-234G1-A01) and the Advanced Research Projects Agency (MDA 003-77-C-0322). Use of the SUMEX computer facility was made possible by the Biotechnology Resources Program of the NIH under Grant RR-00786.

used In CRYSALIS. As In Hearsay-II [Erman76], the hypotheses ore represented In a hierarchically organized data structure. In our case the different Information levels can bo partitioned Into two distinctly different "planes", but the concept of a globally accessible space of hypotheses and clota abstractions Is essentially the same for both systems. Knowledge sources play the same role as In Heorsay-II, oddlng, changing, or testing hypothesis elements on the blackboard. For further discussion see tEngelmore77]. The processes of generating or modifying hypotheses and of Invoking knowledge sources are nearly Identical to those described for the AGE system [NII79].

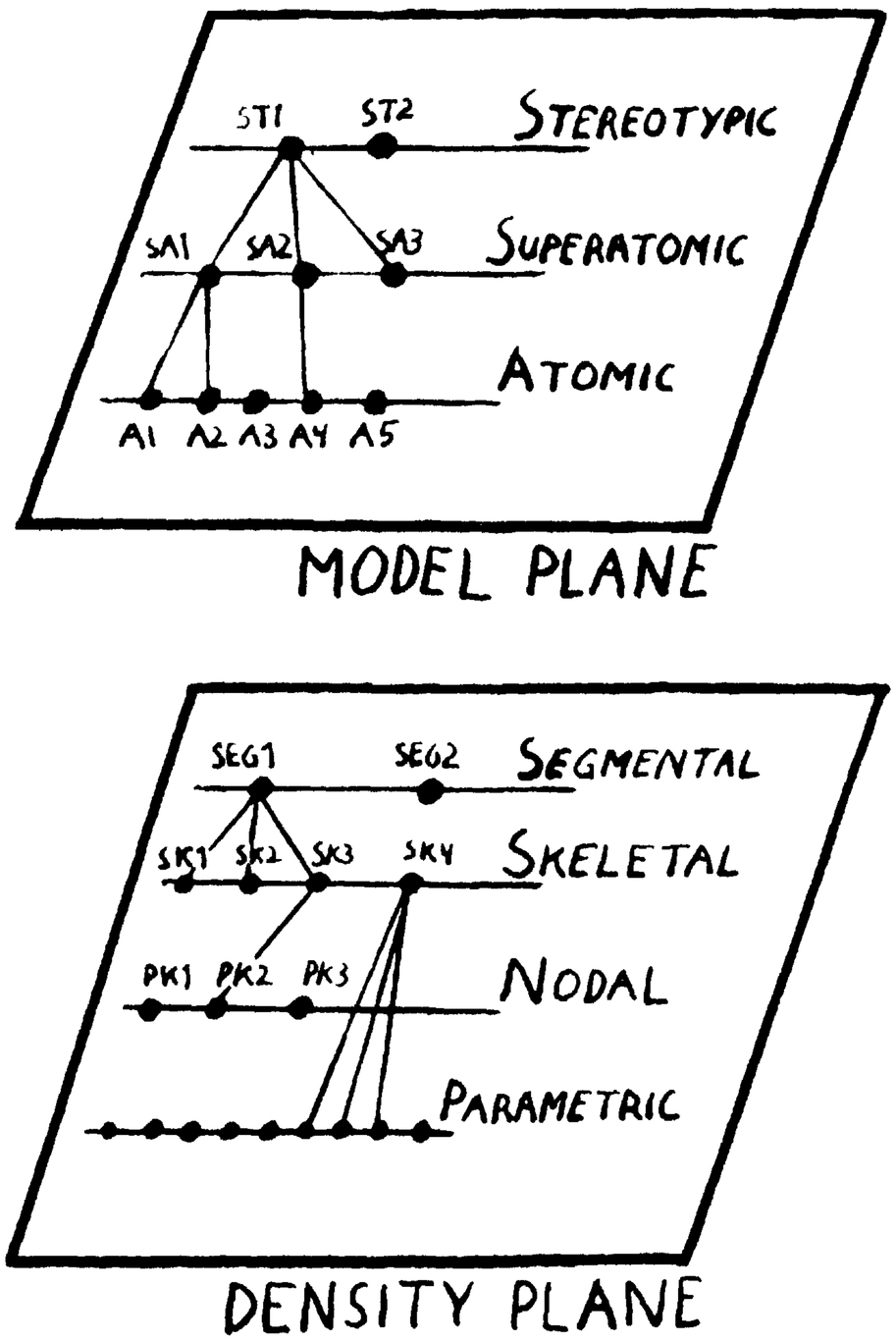


FIGURE 1

In the course of deriving a protein structure which Is a best explanation of the given data, the crystallographer generates a three-dimensional description of the electron density distribution of the molecule, often called an electron density map (EDM). Due to the resolution Imposed by the experimental conditions, the EDM Is an Indistinct Image of the structure, which does not reveal the positions of Individual atoms. The crystallographer must Interpret the mop In light of auxiliary data and general principles of protein chemistry In order to derive a complete description of the molecular structure.

The most Important piece of auxiliary data is the amino acid sequence, along with a specification of any special chemical groups attached to the protein chain. This information defines the set of atoms which comprise the structure, and their Interconnections, but gives little or no Information on how the protein folds and twists from one end of Its polypeptide chain to the other. The EDM, on the other hand, shows where the protein must lie In 3-space, but gives little Information on which atoms go where. Thus the omlno odd sequence ond and the EDM are complementary sources of Information.

By carefully studying the map, and guided by the amino acid sequonce, the experienced protein crystallographer can find features which allow him to Infer approximate atomic locations, molecular boundaries, groups of atoms, the polypeptide backbone, etc. Typically, several weeks to months of tedious effort are required to build, manually, a model of the molecular structure which conforms to the electron density map and Is also consistent with the sequence, his knowledge of protein chemistry, stereochemical constraints and other available chemical ond physical data.

2 Structure of the CRYSALIS system

In the CRYSALIS system the data, the hypothesis and the knowledge base are all hierarchically structured. The data hove several levels of abstraction, the hypothesis Is described at three levels of detail, ond the knowledge base is divided into domain knowledge, task knowledge and strategy knowledge. These three hierarchies are discussed below.

2.1 The data hierarchy

The Electron Density Map

The primary Input data source Is the EDM, which Is derived from X-ray diffraction studies of the crystallized protein. The EDM Is on image of the electron cloud surrounding the atoms of the molecule. Its representation is a set of Intensity volues defined on o three-dimensional grid of 100,000 to 300,000 points. Usually 1% to 8% of the Intensities ore significant to a model builder. The usefulness of the map can be characterized by its resolution and its quality. CRYSALIS is intended to Interpret EDMs of resolution In the range of 2.0 to 2.6 Ang. (for comparison, an average distance between atoms Is 1.7 Ang.), ond of relatively high signal to noise ratio.

Because of the size of the data base, and the relatively small number of significant points, the system uses several abstractions of these data.

The Peak List

The most obvious abstraction of the map Is a list of local maxima. These maxima (peaks) are calculated from the map by a process of interpolation, and are thus net constrained to lie on the EDM grid points. A peak usually Indicates the position of an atom, or an average position of

a small group of atoms. The height of a peak Ia a rough linear function of the atomic number of the atom(s) producing the peak. While the Inability to resolve Individual atoms means there Is an error In this correspondence, one can distinguish between various classes of peaks.

The Skeleton

The second major abstraction is the skeleton. By a method first developed by Greer [Graer 74] and later refined by us, the EDM Is repeatedly scanned, discarding Intensities at all grid points which are below a given threshold and not required for preserving continuity of the electron density cloud. What finally remains Is a list of grid point locations, called nodes, and their connections to other nodes. In this way one reduces the number of data points by two to three orders of magnitude. The skeleton preserves the general topology of the protein, at the price of losing some of the fine detail that distinguishes one amino acid side chain from another.

The Segment List

The segment list condenses the skeleton (without losing any Information) by hiding nodes with only two neighbors. A segment Is the set of connected nodes starting at a tip (one neighbor) or branch point (three or more neighbors) and terminating at another tip or branch point.

Once the data have been condensed to the level of segments, one can frequently identify topological features that correspond to structural elements of the protein molecule. For example, there are usually some long, connected segments that correspond to the backbone of the protein. This backbone is sometimes "clean" In that It shows a simple alteration of objects that appear to be sIdechaIns and main chain (peptide) links. Skeletonization Is a heuristic procedure, however, and often at least some portions of the skeleton are difficult to interpret.

Subgraphs of the Segment list

It Is often useful to think in terms of subgraphs of segments. Since In many cases these subgraphs correspond to superatoms (see below), many of the heuristics for EDM Interpretation can be expressed In torms of subgraphs. There are two major kinds of subgraphs: sIdechaIns and peptides. Sidechain subgraphs *are* collections of segmonts which look like they might be representations of real sIdechaIns (a bit of nomenclature: a "sidechain" Is the physical entity the system Is trying to model, a "sidechain subgraph" is a part of the segment list that one suspects Is the skeletal representation of the sidechain). Thus far we have Identified six kinds of sidechain subgraphs. Peptide subgraphs are defined as all segments between one sidechain subgraph (or bridge) and the next.

2.2 The hypothesis hierarchy

The goal hypothesis In our system is a model of a protein molecule which best explains the given experimental data and Is consistent with accepted principles of stereochemistry *and* protein chemistry. As mentioned earlier, many diverse sources of knowledge are brought to bear on the problem of EDM Interpretation. Each knowledge source (KS) may use different descriptions of the objects on which It operates. For example, a helix locator works with an abstraction of a molecule consisting of a specification of the backbone shape, omitting ail other details. A side chain template matcher, on the other hand, *uses* a full specification of all atomic positions to define the side chain template. In *order* to cope with this diversity, the hypothesis is represented *as* hierarchically organized levels of descriptions, as shown In Figure 1. A KS Is a collection of rules which makes inferences either within or between levels in the hypothesis space.

There are three levels of description in the model plane. The most detailed level Is the atomic level, a specification of the spatial coordinates of all atoms in the model with respect to some arbitrary origin (the coordinates of hydrogen atoms are generally omitted). Proteins exhibit well-defined topologIcal constraints which permit descriptions at higher levels of aggregation. Thus, proteins consist of a linear polymeric chain and, In many cases, attached atomic groups called co-factors. The level of description which describes the model In terms of familiar groups of atoms such as peptide links, side chains, and cofactors, Is called the superatomic level. These units may be aggregated still further into what Is generally called a "secondary structure", I.e., a specification of the relative locations of large Identifiable portions of the protein. Examples are the alpha helix and the beta sheet conformations, well known to protein chemists. Many other such "stereotypes" exist, although they may be associated with a specific family of proteins. This level of description Is labelled stereotypic in figure 1.

A partial or complete hypothesis consists of linked hypothesis elements. A hypothesis element (HE) Is a labelled entity In the space of hypotheses. Attached to each entity Is a set of attributes which define the *HE* In terms appropriate to the level of description on which It resides. HEs at the three levels are defined as follows:
Stereotype Hypothesis Element

LABEL - STn, n = 1, 2, ...
TYPE - e.g. BACKBONE
RANGE - (*a* . b) where a and b are sequence numbers, and a < b
BLOCK - (x . y); x (or y) is non-NIL if the backbone is blocked at a (or b). A backbone Is blocked at an end when the trace that created it is blocked by something in the skeleton it can't interpret.
MEMBERS - a list of superatoms that make up the backbone
CF - a certainty factor equal to the average of all the MEMBER's CFs

Superatom Hypothesis Element

LABEL - SAn, n = 1, 2, ...
TYPE - e.g. PEPTIDE or SIDECHAIN
BELONGSTO - (acid . seq), example is (MET . 3)
MEMBEROF - points to BACKBONE ST, if any
MEMBERS - a list of any atom hypotheses that are part of this superatom
SEGS - list of subgraphs of the segment list and their associated CFs

Atomic Hypothesis Element

LABEL - An, n = 1, 2, ...
TYPE - one of C, N, O, S, Ca, Mg, Fe, CO, Zn, H
NAME - standard name designating position in the residue such as: CA, SD, NE1
BELONGSTO - as in PEPTIDE
MEMBEROF - list of superatoms this atom is in
D.PEAKS - links to peaklist with associated CFs
NODES - links to skeleton with associated CFs
SPACE.LOC - list of possible positions for the atom and their CFs

Note that there may be multiple data links. For Instance, on atom may have links to several peaks in the peak list on the data plane, Indicating that Its position has not been established uniquely. A measure of certainty Is associated with each link as it is generated. These certainty factors are used to compare and merge alternate hypotheses.

2.3 The rule hierarchy

The formal and Informal procedures which comprise the knowledge sources are expressed as rules. These rules are collected Into sets of rules, each set being appropriate to use when particular classes of events occur. The correspondence between event classes and rule sets Is established by another set of rules, the task rules. The task rules are used to decide which KS or sequence of KSs to call In order to perform one of the typical tasks in building the structure, e.g., tracing the protein backbone between two anchor points. The decision Is based on the state of the blackboard and the items on an event list. The task rules thus form a second layer of rules which direct the system's selection of an appropriate method for proceeding.

Once a task Is completed, or If the task fails, the system must look to a higher level of control to determine what to do next. At this higher level — the strategy level -- the structure building process can either try to solve the current subproblem by another method, or shift the focus of attention to another region of the structure. Strategy knowledge Is expressed as rules which make use of the current state of the blackboard and the event list.

We thus have a completely rule-based control structure, employing three distinct levels of rules (or knowledge): the specialists, commonly called the knowledge sources, the task rules for method selection, *and* the strategy rules for controlling the focus of attention. Although this pyramidal structure of rules and meta-rules could continue Indefinitely, a three-tiered system of control appears to offer sufficient flexibility In choosing and deploying the resident knowledge sources.

The following subsections discuss in greater detail the structure and content of CRYSALIS at the three levels of control.

The strategy level The overall control of the system Is assigned to the strategy level. A set of strategy rules governs the choice of task to perform and a region of the hypothesis In which to work. Examples are:

1) IF there are no hypothesis elements
THEN do INITIALIZATION task

2) IF there are two toeholds A and B
and A has certainty greater than 400
and B has certainty greater than 400
and the number of reslcluos In the sequence separating the toeholds is loss than five
THEN do TWO.POINT.TRACE task

(Notes: Certainty Is a measure of belief associated with the properties of each hypothesis element. Its range of values Is -1000 to 1000. Cortainty factors are established and/or modified by KSs as they find support for a hypothesis element In the underlying data. A "toehold" Is a hypothesis element that is linked to both the amino acid sequence and to the density plane with a relatively high certainty, I.e. an "Identified" part of the EDM.)

3) IF there Is a toehold, A
and A has certainty greater than 400
and the direction of the backbone at that toehold Is known
THEN do EXPANSION.TRACE task

The task level At the task level decisions are made on how to best accomplish a specific task, I.e. which KS or combination of KSs should be used, This knowledge is embodied In a set of task rules, whose conditions are predicates operating on the event list, and whose actions Invoke specific KSs. Eight tasks have been formulated thus far, of which the following are examples:

INITIALIZATION The Invocation of this task reads input data Into system, puts any given Information on the blackboard, tries to Identify all large atoms, finds disulphide links, predicts occurrences of helicol regions, tries to discover a set of chainends, and performs on "Interesting sequence" analysis. If the protein contains cofactors, they are located and removod from the segment list In order to simplify the data. The end result of performing this task Is the establishment of a

fow toeholds In the data from which further Inferences can be made.

EXTENSION.TRACE This task Is applicable If the system has established the direction of the backbone at a given toehold. When this task Is Invoked, the model Is extended, at tho superatomic level, In a given direction until the cumulative certainty drops below a given threshold.

TWO.POINT.TRACE Chain tracing between two given toeholds Is In many ways the most certain of the various chain tracing tasks because the limits are known. By tracing from each toehold toward the other and comparing the results, very high certainty values can be generated. This task uses many of the same KSs used in the EXTENSION.TRACE task. The main differences are In the task rules, e.g., determining the conditions for stopping the trace at one side *and* beginning the trace at the other.

MODEL.DRIVEN.TRACE If it can be established that the current location Is part of a known secondary structure (such as alpha helix or beta sheet), a model of that structure can be used to predict successive atomic positions. The trace becomes a very highly constrained loop of prediction and verification. Unlike other tracing tasks, the predictive model allows the trace to skip residues that do not *appear* In the data.

The knowledge source level The most detailed knowledge of the domain Is contalnod at the KS level. Each KS is a "chunk" of formal or informal knowledge used to solve some particular sub-problem In EDM Interpretation. Examples are matching a sidechain subgraph to a template, matching peaks In the EDM with large atoms Inferred from tho amino acid sequence, or simplifying the skeleton by *removing* the cofactor contribution, KSs are typically represented as a set of productions, but In some cases It has been more efficient to represent the knowledge as a procedure (particularly when heavy numerical computing is Involved). About two dozen KSs have been Implemented In the CRYSALIS system thus far.

2.4 Event-driven control

The CRYSALIS system uses an event-driven control structure. An event Is a description of a change In the hypothesis, e.g., the addition of a new hypothesis element or the establishment of new links between existing hypothesis elements. In this scheme the current state of the hypothesis space determines what to do next. The normal iterative cycle of problem solving uses the event list to trigger knowledge sources, which create or change hypothesis elements and place new events on the event list. Each task is executed until It is explicitly finished or until no further rules are fired, after which control passes up to the strategy level to shift attention to a new task. Tho particular hypothesis element under Investigation, or the particular KS Invoked, are determined by the type of event selected from the event list. Thus, under normal conditions, the monitor always has a means for choosing Its next move. Items may be selected from the event list according to a specified processing mode, e.g., FIFO, LIFO or some priority scheme for choosing the "best" event.

The system's behavior Is "opportunistic" in that It Is guided primarily by what was most recently discovered, rather than by a necessity to satisfy sub-goals. The choice of an event-driven control structure Is based partly on efficiency In selecting appropriate knowledge sources and partly on conformity with the structure modeling process normally employed by protein crystallographers.

2.6 Focus of Attention Mechanisms

There are two levels of attention focussing, corresponding to the two levels of control In CRYSALIS. At the strategy level, a "coarse" focus of attention Is created by assigning a task to a region of the hypothesis space. The conditions on strategy rules refer to global features of the current hypothesis, such as the presence of solved *and* unsolved regions along the amino acid sequence. A finer degree of focussing Is provided at the task level, where task rules focus on specific events, and specific KSs (where the real model building Is accomplished) to process those events.

3 Discussion

The sheer volume and variety of knowledge that crystallographers bring to bear on the EDM Interpretation problem Implies a correspondingly large and varied automated system. CRYSALIS employs many Ideas developed over the past decade for representation and utilization of knowledge In export systems. Although none of those features by itself breaks new ground In AI research, their conjunction In one system makes an Interesting case study.

3.1 Flexible architecture

An essential requirement of CRYSALIS Is flexibility: the addition of new facts and procedures must be done easily and quickly, as nearly every new protein structure presents new problems and ad hoc knowledge. The modularity of the rules, the multi-level control and the multiple blackboard planes for hypotheses and data are all Incorporated in the system In order to accommodate expansion or modification. Modularity also allows us as system builders to experiment with alternate designs for the components, e.g. using a strictly procedural knowledge source In place of a rule-based one, with no external change In the system.

3.2 Rule-based control of strategy and task levels

The method of Interpreting protein EDM's Is, at Its critical points, opportunistic. Whore to start, when to leave one port of the structure and focus upon another, what level of detail to look at, when to stop -- these questions are continually presented to the expert as he builds his structure. Tho knowledge needed to answer them Is almost entirely heuristic, and as subject to change as any other task-specific knowledge. It thus seems natural, and indeed has shown to be practical, that this strategic knowledge, which controls the order in which various tasks are performed, be represented as rules. Moreover, the tosks may be executed In various ways, depending upon what KSs are available and the particular situation encountered, so control at the task level is also facilitated by a rule-based representation.

The use of two levels of control has been found to be an efficient alternative to using one large rule set.

3.3 Event-driven processing

The opportunities which arise In EDM Interpretation *are* dynamic, I.e., they flow not only from the Initial data but from the partially built structure. Thinking of the solution process as one of recognition and verification, the recognition phase operates on both the data *and* the partial hypothesis. As the hypothesis changes, new Islands of opportunity arise. By specifically recording these events, the focus-of-attention problem Is ameliorated.

3.4 Multi-level hypothesis/data structure

Use of the blackboard concept was suggested by the similarity of this problem with that of other signal understonding tasks, specifically *in* the need for a common store accessible to many diverse and Independent KSs. The representation of both the hypothesis and the data In tho uniform way discussed earlier makes It easier to add or modify the various levels. For example, it might be useful at some time to add a "glob" level of description of the EDM, which would capture some visual Information, along with a new level of description of the model, such as families of amino acids. The multi-level planes facilitate this kind of modification.

The multiple levels of description are also necessitated by the lack of a common language for describing the objects and operators of the domain, in contrast, say, to the Heuristic DENDRAL program which uses a uniform language of chemical subgraphs for this purpose. The domain knowledge In CRYSALIS Is expressed most naturally at several levels, e.g. Inferences can be drawn from peaks In the EDM to atoms In the model, but Inferences about a skeletal segment are related to superatoms.

Moreover, the Individual levels of detail In the model plane have an Intrinsic Interest. As Is evident from the protein crystallographic literature, there *is* more of interest to a molecular structure than the coordinates of Its constituent atoms. Announcements of new structures usually contain visual representations of the backbone alone, secondary structure, hydrogen bonding, active site geometry, etc. Often the model Is Incompletely specified at the atomic level, so that descriptions of the structure In some rogions must be couched in the language of side chains or other abstractions. The multi-leveled hypothesis structure i*n* CRYSALIS permits this variety of descriptions.

3.6 III-structuredness

Two characterIstIcs of the task domain are particularly Interesting In that they provide an ill-structuredness which Is normally not found In AI applications. The first Is the lack of a well defined termination criterion. One can, of course, stop the program when the position of all atoms Is known, but this is almost never the case. In practice, one stops when no further progress can be made, as for example when no strategies available to the system are applicable, or If further model building effort yields diminishing returns (the recognition of which is an interesting and thus far unsolved theoretical problem). Moreover, to the extent that one can lay down some rules for terminating the process, those rules are usually dependent on the quality of the data.

A second characteristic Is that the detail of the resulting model, I.e. the solution, varies with the quality and resolution of the given data. Different knowledge sources are Invoked for different ranges of resolution, and may either draw different Inferences or link different levels of tho model plane with the data. In any case the detail of the model Is adjusted to fit the experimental information, and can degrade gracefully with decreasing resolution. Thus If some data are particularly poor, or not given at all (e.g. on Incomplete sequence) It still should be possible to get some results.

4 Conclusion! Current status and activities

As It currently exists, CRYSALIS Is In Its early adolescence, capable of Interpreting relatively obvious features In a good EDM, but not yet worthy of attracting serious attention In the protein-crystallographIc community. We are currently working toward a more extensive knowledge-based system capable of complete Interpretation of medium-quality, medium-resolution (2 to 2.6 Ang.) EDMs. During the next year or two wo expect to bring the system's level of performance to the point where It will be noteworthy, by helping in a major way to solve a new protein structure. That goal will be reached, we believe, by Incorporating more detailed knowledge about protein chemistry and stereochemistry, and by improving the heuristics for matching the abstracted EDM with structural features determined by the amino acid

sequence. We *are* also extending our preprocessing programs so that more meaningful and/or less ambiguous features can be extracted from the data. Another current activity Is a re-design of the system to improve Its understandabality, modularity and flexibility.

ACKNOWLEDGMENTS

We wish to acknowledge the assistance of our colleagues In the crystallographIc community, specifically Dr. Stephen Freer, Dr. Richard Alclen and Prof. Joseph Kraut of UCSD, and Dr. Carroll Johnson of ORNL. At Stanford, Ms. H. Penny Nil was Instrumental in the initial construction of CRYSALIS; Mr. Eric Grosse assisted in developing algorithms for abstracting the data.

Appendix A

An Illustrative example

Space limitations prevent a full example from being Included In these proceedings. An annotated typescript, Illustrating the performance of CRYSALIS on a typical Interpretation problem, will be available at the conference or from the authors upon written request.

References

[Agarwal77]
Agarwal, R. C, and Isaacs, N. W., Method for obtaining a high resolution protein map starting from a low resolution map, Proc. Natl. Acad. Sci. USA, Vol. 74, pp 2836-2839 (1977).

[Engelmore77]
Engelmore, R.S. and Nil, H.P., A Knowledge-Based System for the Interpretation of Protein X-Ray CrystallographIc Data, Heuristic Programming Projoct Memo HPP-77-2, January, 1977. (Alternate Identification: STAN-CS-77-689)

[Erman76]
Ermnn, L. D., Overview of the Hearsay Speech Understanding R*esearch, In Working Papers in Speech Recognition -IV- The HEARSAY II System,* Carnegie-Mellon University, Computer Science Speech Group, 1976.

[Greer74]
Groer, J., Three-dimensional Pattern Recognition: An Approach to Automated Interpretation of Electron Density Maps of Proteins, *J. Mol Biol., 82,* 279 (1974).

[Hayes-Roth77]
Hayes-Roth, F. and Lesser, V.R., Focus of Attention In the Hearsay*!! Speech Understanding System, *Proc. of 5 th Int. Joint Conf. on Artificial Intelligence,* Cambridge, Mass., 1977

[Hermans74]
Hermans, J. Jr. and McQueen, J.E. Jr., *Acta Cryst. A30*, 730 (1974).

[Nil78]
Nil, H.P. and Aiello, N., AGE (Attempt to Generalize): A Knowledge-based Program for Building Knowledge-Based Programs, Proceedings of 6th International Joint Conference on Artificial Intelligence, Tokyo, August 1978.

[Tsernoglou77]
Tsernoglou, D., Petsko, G.A., McQueen, J.E. Jr. and Hermans, J., *Science 197*, 1378 (1977).

A MODEL FOR PERCEPTION OF STRUCTURAL IMAGE FEATURE

Hajime Enomoto Naoki Yonezaki Katsumi Nitta
Department of Computer Science
Tokyo Institute of Technology
Ookayama, Meguro-ku, Tokyo, Japan

A model to extract a structure feature of images belonging to one category is proposed, implemented and studied. This model consists of two parts i.e. an extraction of partial geometrical features and an extraction of structure feature by using them under the positional constraints. First,necessary conditions which geometrical features should satisfy is stated and a statistical method to select a key feature is described. Then the algorithm to extract structure feature as the maximal set of common subimages which satisfy the positional constraint is introduced. The structure feature is regarded as a intrinsic feature which is used when an animal perceives an image at a glance. The experiment was carried out about image of human faces and had a fairly good results.

1. INTRODUCTION

Perception process of images by computer is composed of extracting some features from images, describing them, and making the correspondence between objects and description. The process of correspondence needs much knowledge and takes much time[I]. The animal has an ability for efficient image perception which contains the fast algorithm to observe it initially and get the structure feature. It is obtained without knowledge and becomes the cue for understanding.

This paper proposes a model of perception along this fact and a learning schema to get the structure feature. The structure feature is a positional relation of local features which is essential to recognize objects. At first, necessary conditions for these features are discussed. Then, the problem of constructing the structure feature is treated. This method obtains the maximal matching of subimages which satisfy positional constraint. And finally, experiments for images of human faces show the efficiency of this schema.

2. FEATURE EXTRACTION

When an image is given, we can extract various features, such as edges, regions, and textures. If we have a model of objects, we do not have to get all these features. We need only the strong enough features which are appropriate as keys for recognition of images. But. in the learning schema, all features should be treated equally without such a knowledge. The structure feature is constructed by the features that satisfy the following conditions.

(a.) They should not depend on the cordinate system and their position should not be effected by the operations that lose resolution.
(b.) They should be clustered into some categories and the relations between them should be considered more important than the absolute value.
lc) They should include spatial informations.

We divide an image into subimages, which are sets of adjacent pixels, and attach a feature vector $(X_1, X_2, —X_N)$ to each subimage. Each component of this vector is the values on the feature operators.

These feature vectors are distributed spatially and if some relation between them is intrinsic for the image, we regard such relation as the structure feature[2].

3 SPATIAL MATCHING

3.1. Selection of Key Features and Labeling

Within all components of a feature vector, some components may be meaningless as the structure feature, and we must select key features. They are selected on a basis of statistical distribution of each feature of images which belong to the same kind.

(Selection Algorithm)
Step 1: Select one component Xi whose histograms show resemblance for all input images.

This resemblance is measured by the configuration of cores of clustcrs[3].
Step 2: If there arc different numbers of clusters for X_i,some near clusters are merged.
Step 3: For each cluster of X_i, select the component Xj the distributions of which resemble each other for input images, and execute Step 2. This step is repeated until there does not exist the component that show the resemblance, and the sequence of components (X_i,X_i,---) is obtained. Then, each sub image is labeled by an n-tuple (Y_i,Y_j-——) where each component Y_k, is the name of cluster of corresponding feature X_k to which the subimage belongs.

During these steps, the feature which shows peculiar property in a particular image is not selected, for the histograms of such features do not resemble those of other images, and it is considered as a meaningless feature for the structure feature which is common in images.

3.2 Positional Constraint

Here, we define the notion of "match" and "positional constraint".

If two subimages $S_1 \in I_1$, $S_2 \in I_2$,have the same labels, we say S_1 matches S_2 and it is denoted by $S_1 \simeq S_2$.
If there are such subimages $S_1, S_1' \in I_1$, S_2, $S_2' \in I_2$ that satisfy the following conditions, we say they satisfy positional constraint and it is denoted by $(S_1,S_1') \simeq (S_2,S_2')$. (In the following condition, I(S),J(S) denote the row and the column index of S, respectively.)
(a.) $S_1 \simeq S_2$, $S_1' \simeq S_2'$.
(b.) If $I(S_1)<I(S_1')$, $I(S_1)=I(S_1')$, $I(S_1)>I(S_1')$,
then $I(S_2)<I(S_2')$, $I(S_2)=I(S_2')$,
$I(S_2)>I(S_2')$ hold, respectively.
(c.) if $J(S_1)<J(S_1')$, $J(S_1)=J(S_1')$, $J(S_1)>J(S_1')$,
then $J(S_2)<J(S_2')$, $J(S_2)=J(S_2')$,
$J(S_2)>J(S_2')$ hold, respectively.

When two images I_1, I_2 are given, a set of matched subimages of I_1 and I_2 is a set $M = \{(S_1,S_2)\}$ where $\forall (S_1,S_2) \in M\ [S_1 \in I_1, S_2 \in I_2, S_1 \simeq S_2]$ and $\forall (S_1,S_2),(S_1',S_2') \in M\ [(S_1,S_1') \simeq (S_2,S_2')]$.
A set of matched subimages expresses a common structure of images. We regard the maximal set of matched subimages as the structure feature. It takes much time to obtain the maximal set, so we introduce a simple matching algorithm by which a good set can be obtained, though not a maximal one.

3.3 Simple Matching Algorithm

Let an image be a set of subimages $\{S_{ij} | 1 \leqq i \leqq M, 1 \leqq j \leqq N\}$, and let each subimage S_{ij} have a label L_{ij}. For each row i, the sequence of labels $L_{i1}, L_{i2}, \ldots L_{iN}$ is denoted by L<i>. When two sequences L<i> and L<j> are given, the distance between them is defined as the reciprocal of the length of the longest common subsequence (LCS)[4], and denoted as D(i,j). The matching algorithm is as follows:

Step 1: Let P be a set of all pairs of sequences (L_1<i>, L_2<j>) where L_1, L_2 are from I_1, I_2, respectively.
Step 2: Select a pair (L_1<i_1>, L_2<j_1>) whose distance D(i,j) is the smallest in P.
Step 3: From P, remove the pairs (L_1<i>,L_2<j>) that do not satisfy $i>i_1$, $j>j_1$ or $i<i_1$, $j<j_1$.
Step 4: If P is empty or there is not a pair (L_1<i>,L_2<j>) which satisfies D(i,j)<0, go to step 5. Else, go to step 2.
Step 5: Change the labels of the subimages to "don't care symbol" that are not included in the selected rows, or that are not included in the LCS in step 2.
Step 6: Repeat from step 1 to step 5 for any pair of column vectors. In this case the "don't care symbol" is not included in LCS.

The subimages which are left unattached "don't care symbol" construct a set of matched subimages.

4. EXPERIMENT

We apply our algorithm to images of human face. Human faces have some subobjects such as hair, eyes, brows, and so on. However, we try to match the images without the knowledge about them. Input images are composed of 128x128 pixels and each pixel has 9 bit integer value as intensity. (Fig. 1)

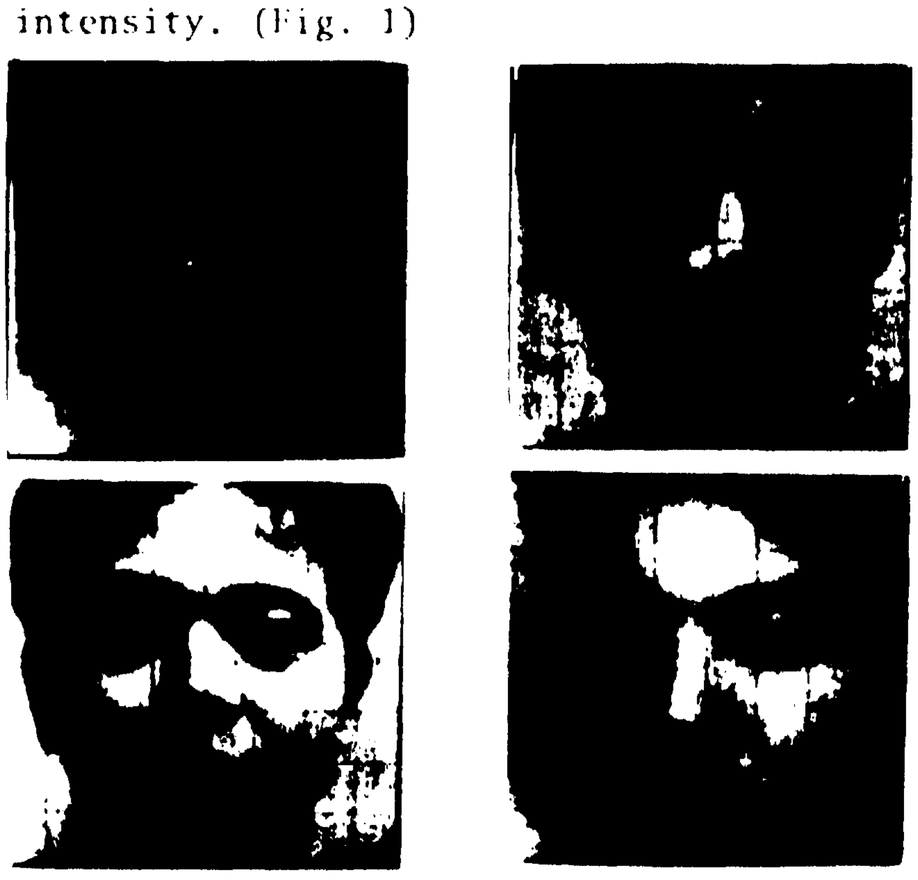


Fig.1 Original images

These images are divided into subimages whose sizes are 4x4, and each subimages are attached a feature vector (X_1, X_2, X_3) where X_1, X_2, X_3 correspond to mean intensity, busyness and edge value, respectively[5][6]. Each component is categorized and consequently each subimage has a label composed of 3-tuple of cluster names. In Fig.2, concerning the first two images, the boundaries between subimages attached different labels are displayed.

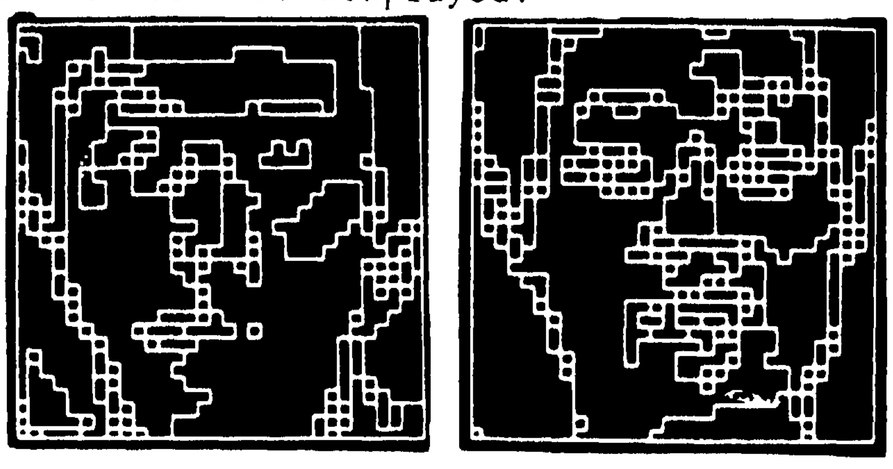


Fig.2 Labels attached to subimages

By comparing the histograms of these features, clusters of X_1 are merged, and new labels are attached to each subimage. (Fig. 3)

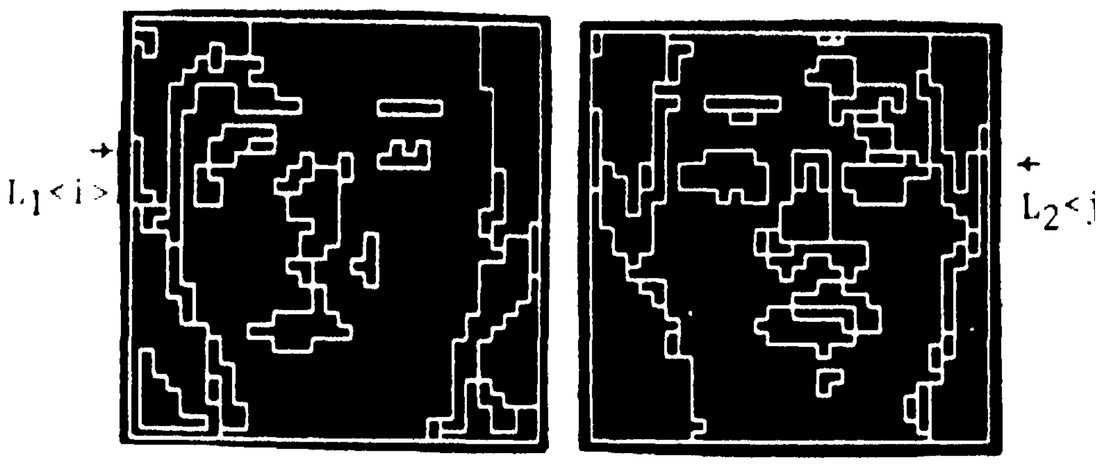


Fig.3 Merged labels

LCS of sequences of these labels is calculated by the simple matching algorithm. For example, let L_1<i>, L_2<j> be given as following sequences. (They are sequences of labels which are indexed by arrows in Fig.3, and each letter expresses a label attatched to a subimage.)

L_1<i> = baadbbddbbbbbbdbbdbbddddbbbbbaaaa

L_2<j> = baaaadbbbddddbbbddbbbccedbddade

LCS of these sequences is calculated in Step 2 and following result is obtained. (Labels that are not included in LCS are changed into don't care symbol " * ".) The distance between them is 1/16.

L_1<i> = baadbb**b****dbb*b*dd**bb***a***

L_2<j> = baa**dbbb**d*bbb*dd**b****b**a**

The result of matching first and second original images are shown in Fig. 4. In this image, only the regions in which more than one-third of subimages are left unattached "don't care symbol" are displayed. It takes about 3 minutes to match two images.

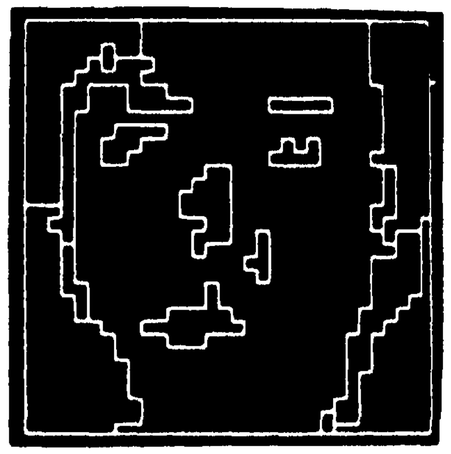


Fig.4 Matched image of two images

To match many images, this algorithm is applied one by one. The result of matching four images of Fig.I is shown in Fig. 5. Some principal subobjects are remained because the relative positions of them are common for all images and each of them has a same local feature.

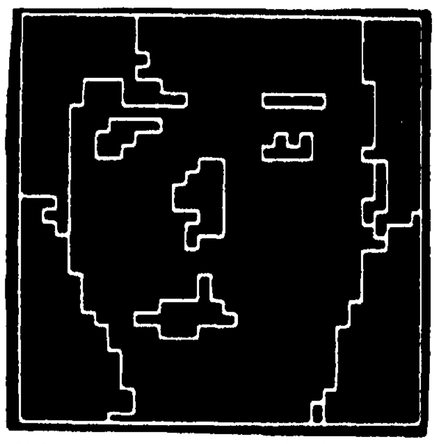


Fig.5 Matched image of four images

5. CONCLUSION

We have presented an algorithm to get a structure feature of images. This feature correspond to the structure we get from images at a glance, and it is used as the cue to understand them. We used the relation between subimages and from statistical property, we got the structare feature. The advantage of this method is that we can keep from noises, because noises which happen irregularly are removed while matching process. The results of experiments show the maximal set of subimages has enough information as the structure feature.

REFERENCES

[1] Winston, P.H. "The Psychology of Computer Vision." McGraw-Hill, 1975.

[2] Fnomoto, H. et al. "Statistical Approach for Construction and Estimation of Syntactic Pattern Recognition Mechanism." in Proc. IJCPR-78. Kyoto, Japan, November,1978, pp.359-363.

[3] Fukada,Y, "Spatial Clustering Procedures for Region Analysis." In Proc. IJCPR-78, Kyoto, Japan, November, 1978, pp.329-331.

[4] Hirschberg, D. "Algorithm for the Longest Common Subsequence Problem." JACM 24:4 (1977) pp.664-675.

[5] Ohlander, R.B. "Analysis of Natural Scenes." PhD thesis, Computer Science Dept. Carnegie Mellon University, April, 1975.

[6] Rosenfeld, A. "digital Image Processing." Academic Press, 1976.

REPRESENTATION OF DYNAMIC CLINICAL KNOWLEDGE: MEASUREMENT INTERPRETATION IN THE INTENSIVE CARE UNIT

Lawrence M. Fagan, John C, Kunz, Edward A, Feigenbaum
Heuristic Programming Project, Computer Science Department
Stanford University, Stanford, California, 94305

John J, Osborn
The Institutes of Medical Sciences, Pacific Medical Center
2200 Webster Street, San Francisco, California, 94115

This paper reports work in progress on a program to provide diagnostic and therapeutic suggestions about patients in the Intensive Care Unit (ICU). The Ventilator Manager program (VM) dynamically interprets the clinical significance of quantitative data from the ICU. This data is used to manage post-surgical patients receiving mechanical ventilatory assistance. An extension of a physiological monitoring system, VM (1) provides a summary of the patient's physiological status appropriate for the clinician, (2) recognizes untoward events in the patient/machine system and provides suggestions for corrective action, (3) suggests adjustments to ventilatory therapy based on a long-term assessment of the patient status and therapeutic goals, (4) detects possible measurement errors, and (5) maintains a set of patient-specific expectations and goals for future evaluation. The program produces interpretations of the physiological measurements over time, using a model of the therapeutic procedures in the ICU and clinical knowledge about the diagnostic implications of the data. These therapeutic guidelines are represented by a knowledge base of rules created by clinicians with extensive ICU experience.

1.0 INTRODUCTION

Many of the artificial intelligence approaches to medical decision making have concentrated on providing advice based on data available at one particular point in time. This formulation of the problem treats the diagnostic process as a static situation. In actual practice, the clinician receives additional information from tests and observations over time and must reevaluate hypotheses about the nature of the diagnosis and reevaluate the status and prognosis of the patient. The patient situation is affected by the progression of the disease process and the response to prior therapeutic interventions. Some of these aspects have been captured in current computer medicine work, particularly the digitalis therapy advisor [2], This system generates conclusions over time pertaining to the size and timing of the next dose of digitalis. Both the MYCIN [4) and CASNET systems [8] allow the user to rerun a consultation as new patient information becomes available. The IRIS system [5] is capable of attaching a time specification to each conclusion created by the system, consisting of a numeric or symbolic time range when the conclusion was applicable.

One attempt to explore these issues is the development of the Ventilator Manager (VM) program, which provides diagnostic and therapeutic advice in the Intensive Care Unit, The input to VM are the values of 30 physiological measurements provided on a 2- or 10-minute basis by a automatic monitoring system [3] . The output is in the form of suggestions to clinicians and periodic summaries.

The clinical situation being modeled is the post-surgical progress of a patient in the Intensive Care Unit, concentrating on the status of his mechanical breathing assistance. The mechanical ventilator provides total or partial assist of ventilation for seriously ill patients. The type and settings of the ventilator are adjusted to match the patient's intrinsic breathing ability. The "volume" mechanical ventilator provides a fixed volume of air under pressure through a tube to the patient. The ventilator can be adjusted to provide breaths at fixed intervals called "controlled mandatory ventilation" or in response to sucking by the patient, known as "assist mode," Adjustments to the respiration rate or output volume of the ventilator are made to insure an adequate "minute volume" to

* This research is supported by the NIH under GM-24669, and utilizes the SUMEX-AIM computing resource (NIH RR-00785). We wish to thank Bruce Buchanan, Dianne McClung, Penny Nii, and Josh Rubin for their help on this project.

the patient. When the patient's status improves, the mechanical ventilator is disconnected and replaced by a ''t-piece" that connects an oxygen supply with the tube to the patient's lungs. If the patient can demonstrate adequate ventilation then the tube is removed ("extubation") , Often many of these clinical states must be repeated until the patient can breathe on his own.

2.0 Knowledge representation

The availability of new measurement data requires updated interpretations based on the changing values and trends. As the patient setting changes—e.g., as a patient starts to breathe on his own during removal (weaning) from the ventilator—the same measurement values can lead to different interpretations. In order to properly interpret data collected during changing therapeutic contexts, the knowledge base includes a model of the stages that a patient follows from admission to the unit through the end of the critical monitoring phase. Recognition and utilization of the appropriate patient context is an essential step in determining the meaning of most physiological measurements. The goals for patient management are also stated in terms of these clinical contexts. The program maintains a description of the current and optimal ventilatory therapies for any given time. Figure 1 shows a summary of conclusions generated by the program on a periodic basis (currently one hour).

```
  Current conclusions:
  HYPOTENSION PRESENT for 41 MINUTES
  HYPERVENTILATION PRESENT for 33 MINUTES
  SYSTOLIC BLOOD PRESSURE LOW for 46 MINUTES

Conclusions: {time of day}  |.....|.....|.....|.
                            13    14    15    16
  {physiological states}
HEMODYNAMICS - STABLE                ====
HYPERVENTILATION - PRESENT       =     == == ====
HYPOTENSION - PRESENT              ====  == =====
TACHYCARDIA - PRESENT            ==== ==
  {patient context}
Patient is ASSIST                      =====   ==
Patient is CMV                       ====     ===
Patient is VOLUME                  ==
Patient is NOT-MONITORED     ====
  {therapeutic goals}
Goal is CMV                        ==============
Goal is VOLUME               ====
                             |.....|.....|.....|.
                             13    14    15    16
```

Figure 1. Summary prepared by VM based on data from a patient in the ICU. Comments are in {}.

Knowledge is represented in VM by production rules [4,1,7], The rules are of the form:

IF facts about measurements
and/or previous conclusions are true
THEN
1) Make a conclusion based on these facts;
2) Make appropriate suggestions to clinicians; and
3) Create new expectations about future acceptable ranges for measured variables.

Additional information associated with each rule includes: the symbolic name and type or rule group (e,g., instrument fault rules); main concept (definition) of the rule; and all of the therapeutic states in which it makes sense. Figure 2 shows a sample rule for determining hemodynamic stability,

```
STATUS RULE: STABLE-HEMODYNAMICS
DEFINITION: Defines stable hemodynamics for
        most settings
APPLIES to patients on VOLUME, CMV, ASSIST,
        T-PIECE
COMMENT: Look at mean arterial pressure for
        changes in blood pressure and systolic
        blood pressure for maximum pressures,
IF
   HEART RATE is ACCEPTABLE
   PULSE RATE does NOT CHANGE by 20 beats/minute
        in 15 minutes
   MEAN ARTERIAL PRESSURE is ACCEPTABLE
   MEAN ARTERIAL PRESSURE does NOT CHANGE by 15
        torr in 15 minutes
   SYSTOLIC BLOOD PRESSURE is ACCEPTABLE
THEN

   The HEMODYNAMICS are STABLE
```

Figure 2. Sample VM Interpretation Rule.

2.1 Treating Measurement Ranges Symbolically

Most of the rules represent the measurement values symbolically, using the terms "acceptable" or "ideal" to characterize the appropriate ranges. The actual meaning of "acceptable" changes as the patient moves from state to state, but the statement of the relation between the physiological measurements remains constant. The use of symbolic statements (e.g., "heart rate is acceptable") allows for a simple representation of common clinical practice and the exposition of common principles of physiological interpretation in different contexts. In addition, it minimizes the number of rules needed to describe the complexity of the diagnostic situation.

The meaning of the symbolic range is determined by rules that establish the context-dependent expectations about the value of measured data. For example, when a patient is taken off the ventilator, the upper limit of acceptability for the expired carbon dioxide measurement is raised. The actual numeric calculation of "expired 002 high" in the premise of any rule will change when the context switches (removal from ventilatory support), but the statement of the rules remains the same. An example rule that creates these expectations is shown in Figure 3,

```
IF   PATIENT TRANSITIONED FROM ASSIST TO CMV
THEN EXPECT THE FOLLOWING
                      [ acceptable range ]
                very      [ ideal ]          very
                low   low   min   max   high   high
                ---   ---   ---   ---   ----   ----
MEAN PRESSURE   60    75    80    95    110    120
HEART RATE            60                110
EXPIRED CO2     22    28    30    35    42     50
```

Figure 3, Portion of an Initializing Rule, This rule establishes initial expectations of acceptable and ideal ranges of variables. Not all ranges are defined for each measurement.

The VM knowledge base includes rules to support the following reasoning steps: (1) characterize measured data as reasonable or spurious; (2) determine therapeutic state of the patient (currently the mode of ventilation); (3) establish expectations of future values of measured variables; (4) check physiological status, including cardiac rate, hemodynamics, ventilation, oxygenation; and (5) check compliance with long-term therapeutic goals.

3,0 RULE INTERPRETATION

The VM rule interpreter is based on the EMYCIN interpreter [6,4], The major changes are: forward chaining (data-driven) rule invocation as opposed to backward chaining, checking to see that information acquired in a previous time frame is still valid for making conclusions, and the ability to cycle through the rule set each time new information is available,

A data-driven approach is used to take advantaqe of the small set of measurement values available in each time frame. Because of the nature of the ICU environment, one can assume almost no dialogue will take place with clinicians when they are using the system. Thus, conclusions must be based on the available data. Each of the rule groups corresponding to the reasoning steps mentioned above are considered in order. It is necessary to separate out the reasoning steps since one part of the reasoning chain may conclude that the clinical context has changed, affecting the interpretation of the more "abstract" reasoning steps.

Identical conclusions made in contiguous time frames are represented by the interval specified by the times of the first and last assertion. A list of these intervals sumnarizes the history of a particular conclusion over time. The evaluation of a rule clause such as "Patient hyperventilating for the past 30 minutes" is made by direct examination of the time intervals stored along with conclusions as opposed to looking at the original measurements. Expectations are associated with the appropriate measurement and are classified by type—e,g., the upper limit of the acceptable range—and duration. Expectations can persist for a fixed interval, such as "for twenty minutes starting in ten minutes," or for the duration of one or more clinical contexts.

4,0 REFERENCES

[1] Davis, R. and King, J, "An Overview of Production Systems," in Machine Intelligence 8: Machine Representations of Knowledge, ed. Elcock, E, and Michie, D., John Wylie, 1977.

[2] Gorry, G,, Silverman, H. and Pauker, S. "Capturing clinical expertise: a computer program that considers clinical responses to digitalis." Amer. J. Med 64 (1978) pp. 452-460.

[3] Osborn, J., Beaumont, J,, Raison, J. and Abbott R, "Computation for Quantitative On-Line Measurement in an Intensive Care Ward," Computers in Biomedical Research 3 1969,

[4] Shortliffe, E, Computer-based medical consultations: Mycin, American Elsevier, New York, 1976.

[5) Trigoboff, M, and Kulikowski, C: "IRIS: A System for the Propagation of of Inferences in a Semantic Net," In Proc, IJCAI-77, MIT, Cambridge, Mass., August, 1977.

[6] van Melle, W, "A Domain-Independent Production Rule System for Consultation Programs." In Proc, IJCAI-79. Tokyo, Japan, August, 1979.

[7] Waterman, D, and Hayes-Roth, F. "An Overview of Pattern-Directed Inference Systems," In Pattern-Directed Inference Systems, ed. Waterman, D. and Hayes-Roth, F. Academic Press, 1978, pp. 3-22.

[8] Weiss, S,, Kulikowski, C,, and Safir, A, "A Model-Based Method for Computer-Aided Medical Decision-Making," Artif, Intell. 11 (1978).

MODELLING INTENTIONAL BEHAVIOR GENERATION

William S. Faught
The Rand Corporation
Santa Monica, California 90406

Two problems of modelling intentions are identified: representing intentional and non-IntentionaL (mechanical) actions in a single model consistently, and implementing a control structure that accounts for how intentional actions are initiated. These two problems stem from a common source: the interaction of intentional and nonintentional actions. We review a simulation model of cognitive and motivational processes based on a production system architecture that includes both types of actions. We then describe a meta system addition to the simulation model composed of nonintentional actions.

I- MECHANICAL VS. INTENTIONAL ACTION

One example of a model of intentional (and nonintentional) actions is the PARRY3 system [2]. PARRY3 is a simulation model of a paranoid patient in an interview situation. The system (computer program) communicates with a psychiatrist in English via a teletype [3]. The simulation model uses a conventional production system (PS) architecture. The PS has conditions on the left hand sides (LHS) and actions on the right hand sides (RHS) and runs in the normal forward manner: testing conditions and performing actions.

The major elements in the model are action sequences, external conditions, affects, beliefs, inferences, and intentions. An action sequence is a set of linked (with tags) PS rules that represent a sequential set of actions that the system is to perform. Representing an action sequence in sequential rules (as opposed to a single action) allows higher priority actions to interrupt the sequence. An example of an action sequence:

How do you like it there?	Question
Why do you want to know?	Clarification question
Maybe it upsets you.	Clarification
It's ok. But I am upset.	Answer

External conditions (on left hand sides of rules) represent information from the system's interface to the world. Examples are: the interviewer's linguistic input, facts about and measurements of the input, amount of time since the last input.

The model has eight affects representing eight primary emotions (e.g., anger, fear, shame) that identify the significant experiences for the model-patient (simulation), which in turn provide motivation for the model-patient's actions Affects on a rule's LHS represent affect states that cause behavior. Affect levels on a rule's RHS represent incremental affect modifications performed before the succeeding PS interpretation cycle.

The model also has a number of beliefs (e.g., the interviewer wants to help) and inferences to manipulate them. Beliefs are represented in a standard format: as propositions with modifiable truth values representing the amount of evidence that they are true. Beliefs on a LHS represent conditions; beliefs on a RHS represent actions to add evidence that the beliefs are true. Inferences are PS rules with beliefs on their RHS. An example inference rule is:

```
(INPUT & (INTOPIC = MENTALSTATE)
& (SHAME = HIGH)) => (DOCABNORMAL)
& (FEAR 40)
```

That is, if there is an input, and its topic is the model-patient's mental state, and shame is high, then conclude the interviewer is abnormal and increment the fear affect.

Finally, the model has a number of intentions representing situations that are (believed to be) advantageous to the model-patient (e.g., introducing a new topic). Intentions are similar to beliefs - they have modifiable values and can occur on LHS and RHS of rules.

Values of intentions represent the intention's activation level or "strength of commitment" to achieving the goal. Intentions on a LHS instigate actions. Intentions on a RHS represent actions to (de-)activate the intention (modify its strength value). Additionally, each intention has several special PS rules associated with it that the intention activates and that test whether the intention's desired situation (goal) has come about. If the system has achieved the goal, the rules reduce the intention's strength (so that it is no longer active). An example rule activating an intention is:

(DDHELP & (FEAR LOW)) => PHELP

That is, if the interviewer desires to help, and if fear is low, then set the intention for the model-patient to get help. Subsequent rules use the intention PHELP as a condition to determine what new topic to introduce and when to introduce it (e.g., in a conversational lull).

Generating mechanical actions in this simulation, then, occurs when PS rules with external conditions, affects, and beliefs instigate actions in a stimulus-response manner. Generating intentional actions is consistent with this formulation: intentions (as well as affects, beliefs, and conditions) are LHS conditions which instigate actions. The system transforms an intention from an information structure to a causative agent when the system "activates" the intention by increasing its strength as part of a cognitive decision process. Once the intention is activated, however, the system's cognitive (belief) processes no longer have direct control over the action. The action is subject to the normal conflict resolution algorithm of the PS rule set, and other actions (most likely based on high affect content) may override the intention's action.

2. THE PROBLEM OF CONATION

The second difficulty of modelling intentional action is the problem of conation [1]. The question is: Who performs intentional actions? A mechanical action, such as avoiding a fearful stimulus, can be modelled with a stimulus-response process. The problem is explaining the causal mechanism behind a person's intentional actions. The usual conative paradigm identifies the cause of intentional action B as another process A. However, we have gained nothing if we claim that the person caused process A. If instead we claim that process A is the person's want or desire to do process B, then it, too, is an intentional action, and we are in danger of an infinite regress. (We will ignore a third strawman explanation, "the Ghost in the Machine" [A], in which the mind is a "ghost" that resides in the body.)

As an alternate approach, consider a (hypothetical) system performing mechanical actions. At some point the system's actions fail and the system stops. We construct an additional process of mechanical actions to locate the error and resume or restart the original mechanical actions. This process would use beliefs and remaining information from the original actions to hypothesize what the failed action was and specify the appropriate recovery actions.

The features of this paradigm are: (1) All actions can be considered mechanical at some level; (2) Mechanical actions can manipulate intentions. In particular, mechanical actions can activate intentions causing intentional actions; (3) The system senses these activated intentions in a manner similar to other (affect and external) conditions. Their activation causes intentional actions.

In this paradigm, an intentional action is the result of a mechanical action that is one level of abstraction higher than the intentional action. The processes involved in the higher level abstraction comprise a meta system with the lower level being an object system.

3. A META SYSTEM

We shall describe our efforts at implementing such a system. (The system is more fully specified in Faught [1978]. It has been incorporated into the PARRY3 system.) The simulation model of cognitive and motivational processes (described above) serves as an "object" system; i.e., the meta system models the object system's actions. The meta system consists of (a) beliefs, conditions, and inferences to model the situation, and (b) actions (rules) to determine facts and manipulate intentions. Meta system constructs are represented in the same format as object level beliefs, inferences, and actions, and are acted on by the same PS interpreter.

The meta level action sequences and inferences are invoked only when certain events disturb the system. The system invokes meta actions whenever a high negative affect condition occurs, when an action failure occurs (detected when no PS rules match for one cycle), or when the interviewer asks the model-patient directly

for an opinion or for information on its internal state (e.g., "What do you want to do right now?"). The meta actions are contained in an action sequence having special conditions on its LHS, e.g., no PS rules matching during the previous PS interpreter cycle. The action sequence carries out the following individual actions: (1) Attempts to determine three beliefs comprising the current situation—what actions the system has been performing, what actions the system is currently performing, and what actions the interviewer is performing. (2) Determines whether the system finds the current situation desirable, i.e., whether the system likes the current situation according to affect measures of the situation (e.g., whether negative affect levels are high). (3) Sets an intention to continue the current situation if it is desirable. (4) If the situation is undesirable, the system determines which states are possible to obtain, chooses the most desirable one, and sets an intention to perform an action leading to the selected state.

As a result of performing this action sequence, the system typically activates an intention to start an action. This intention is part of the normal object-level system; if no higher priority action is being performed, the intention will initiate an action. Note that the meta level routines do not have direct control over the object system's processing; their conative control consists of activating beliefs and intentions, relying on PS rules to detect the beliefs and intentions and initiate the corresponding action.

The meta level processes are not a replica of the object system. The meta processes (beliefs, inferences, actions) contain certain object-level abstractions - e.g., beliefs about what action the system was performing. However, both levels are driven by the same affects and external conditions, and both modify the same affects. The PS interpreter matches rules from both levels during any one cycle. What distinguishes the meta level is its set of abstractions about object-level proc essing.

Several criteria come to mind when evaluating the meta system's effectiveness: (a) Does the meta system perform a rich set of cognitive operations to model the system's actions? (b) Are the system's actions sometimes independent of the immediate external environment? (c) Can the meta system monitor and restart object-level actions effectively, i.e., with a minimum of object-level actions. While the meta system can restart some intentional actions, the system as a whole fails to exhibit these qualities. The reason seems to be a lack of complexity, not of details of the external world but of details of its own actions' components. For example, the system has only a limited number of beliefs that represent the actions it is performing at any one time; most of the beliefs are mutually exclusive. There is none of the subtlety required for restarting failures of complex actions. Restarting procedures would include interpreting what an action without an explicit goal was attempting to accomplish, the typical methods of restarting that action, and the consequences of the action's failure. The system needs an extensive set of (situation, potential action, typical outcome) triples and similar triples for meta level processing. The lack of subtle, interlocking meta and object-level data appears to be a major obstacle to eliciting effective meta level processing.

REFERENCES

[1] Boden, M.A. "The Structure of Intentions." Journal for the Theory of Social Behavior. 3:1 (1973)" 23-46.

| 2) Faught, W. S. Motivation and Intensionlity in - Computer Simulation Model of Parajnoia. Basel: Birkhaeuser Verlag, 1978.

[3] Faught, W.S., Colby, K.M. and Parkison, R.C. "Inferences, Affects, and Intentions in a Model of Paranoia." Cognitive Psychology, 9 (1977) 153-187.

|4] Ryle, G. The Concept of Mind. London: Hutchinson, 1949.

RECOGNITION IN A PROGRAM UNDERSTANDING SYSTEM

Stephen Fickas
Information *and* Computer Science Dept.
University of California Irvine
Irvine, California 92664

Ruven Brooks
Dept. of Psychiatry and
Behavioral Science
University of Texas Galveston
Galveston, Texas 77550

Wr propose a recognition model for a Program Understanding system based on a top-down control structure that begins from a given high level plan and attempts to refine and decompose the plan until it can be matched against actual codr. We introduce the notion of "dictinctivc features" and "beacons" which help guide the recognition process and arc based on a set of functional program "templates". Additional mechansims are proposed to recover from situations in which the primary top-down approach fails.

1. INTRODUCTION

The goals of our research project are twofold: one, to develop a working definition of Program Understanding which is capable of incorporating all of the diverse pieces of information expert human programmers employ in understanding a orogram, including efficiency tradeoffs, program comments, knowledge of the task domain, language dependent and independent programming knowledge; two, to define a set of problem solving methods which use such knowledge in constructing an understanding of a computer program. This paper addresses the second of these goals, particularly the types of recognition methods most suited to the program understanding task. The remainder of this section provides the preliminary background for our discussion of recognition in section 2. To test our ideas, we are currently developing an AI system which accepts actual programs from a specific domain, and produces explanations of the programs in the framework we have defined.

1.1. Description Trees

We view a program as the final product of a hierarchical design process similar to Wirth's notion of step-wise-refinement. We define a Description Tree (DT) as a hierarchical plan structure generated from this process. Our DT has much in common with Brown's[3] hierarchical descriptions, de Klecr's[4] plan fragment description's, and Rich and Shrobe's[8] complete program descriptions.

For any given task specification, there can exist a possibly large number of different DTs, one for each unique final program (Barstow [1] defines a "refinement tree" which is analogous to a supertree of all DTs generated from the same specification).

I .2 Program Building Blocks

The basic conceptual *programming* unit of our system is called a program building block or *pbb,* and is in most ways equivalent to what Rich and Shrobe[8] refer to as a Segment in their Programmer's Apprentice. Each *pbb* has a set of data inputs, pre-conditions which must hold on entry, a set of resulting outputs, and post-conditions which *are* true on exit. Other types of information which help guide recognition may also be attached to a *pbb,* eg. possible mnemonic names. Figure 1 shows a *pbb* that copies a file to a linked-list.

```
(pbb file-to-list-copy
  (INPUT: file1)
  (PRE-COND: type(file1,SEQUENTIAL))
  (OUTPUT: linked-list1)
  (POST-COND: type(linked-list1,LINKED-LIST)
              map1to1(linked-list1,file1))
  (MNEMONICS: READ$FILE COPY$FILE)]
```

Figure 1: Example *pbb*

1.3. Implementation Plans

Any non-primitive *pbb* can bo decomposed into sub-parts (also *pbbs)* linked by dataflow and teleology. We call this decomposition an *Implementation Plan* or *IP* (for more details, see Rich and Shrobe's[8] concept of "deep plan"). In terms of the DT, for any non-terminal node B, all sons of B constitute the sub-parts of *an IP* of B.

2. CONSTRUCTING A DESCRIPTION TREE

Our initial goal is to build a system that constructs a DT given simply the task specification *and* the *program* code; however, we arc including design features which will enable it to use knowledge from preprocessed mnemonic names and comments. While the final system is not intended as an explicit model of *human* behavior, a guiding principle in the design is that it should use, wherever feasible, mechanisms derived from the analysis of human protocols.

2.1. Justification of a Top-down Approach

The essential control structure of the system is to start with an initial hypothesized *Implementation Plan,* hypothesizing and confirming further refinements and decompositions, halting when the lowest level *pbbs* (ie. leaves of the DT) match the actual code. There are several reasons why a top down method is preferred. First, a top-down approach provides a useful high level understanding even when forced to halt before reaching the lower levels of the Description Tree. For example, being able to identify sections of a program which read a file into an array, sort the array, and merge the array with another file is useful even if exact details have not been (or cannot be) worked out for the read, sort or merge *Implementation Plans.*

Second, from our observations, human programmers generally employ top-down methods when attempting to understand a program. If a program understanding system employs the same basic top-down approach, then a user of the system should be able to follow, in a natural way, the system's progress. If the actions of our system can be easily understood by the human user, then it is feasible to consult the user when the system decides it needs help. Both the value of using a human expert as an active partner and the importance of conforming to his problem solving methods has been well documented in the AI literature.

2 2. Templates

In our system, *pbbs* are considered to be independent of the particular implementation language in the sense that, while they may assume the existence of certain data structures or typing conventions, the *pbbs* cannot be compared directly to the string of characters that form the program. The information to make this comparison is represented in our system by a structure we call a template (This term was selected on the basis of a structure proposed to fulfill a similar function in a model of human programmer behavior, Brooks[2]).

A template links together information about the way In which a function or operation is performed in a particular programming language. In many respects, our use of templates resembles the use of axiomatic assertions to describe the effects of pieces of syntax (Hoare[5]); however, for use in our system, we have extended that idea by generalizing the notion of syntactic type from just those structures which are primitive in the programming language to encompass *any* kind of pattern in the surface structure of the code which has an associated function. For example, the function of "looping though an array" might have associated with it the following pattern:

DO <statementx> <index> = I,<array size>

... *array{* <index>)...

Note that there are places in the pattern in which arbitrary, unmatched pieces of code can occur.

2 3 Distinctive Features and Beacons

To verify a hypothesized refinement or decomposition, code must be found in the program text which helps validate the hypothesis. To do this, a template must be found whose function is the needed one and whose pattern matches some portion of the code. The expected number of templates in our system is large so that matching templates efficiently is a central problem.

Our initial approach in solving this problem was based on an observation about the way human programmers look for particular functions in a program. Programmers generally associate with each function a set of *distinctive features* which enable them to locate the function in the code. For example, an interchange of two elements of an array is a *distinctive feature* of sort routines. We therefore thought that templates could be grouped into sets of *distinctive features* for each *pbb* in our system, *and* that a hypothesized *pbb* could be bound to the code by searching for just those *distinctive features* associated with its function. A further observation on human programmers caused us to modify that approach somewhat.

While human programmers look for *distinctive features* to help confirm their hypotheses, they also notice particularly salient features of other *pbbs.* We term these unusually salient features *beacons.* For example, a programmer looking for (hypothesizing) a sort routine might also identify the file-to-list-copy *pbb* of figure 1 by noticing a READ statement *(beacon)* somewhere else in the program. We felt that such behavior was advantageous in our system as well for nominating new hypotheses, and modifying and refining existing hypotheses; the approach we use for obtaining this behavior is a dynamically adjustable saliency ordering for matching templates.

2 4. Binding

Once a template is matched, the code is bound to the associated *pbb.* This binding is both for operations and data structures so that each surface data structure is bound to its associated input or output slot. For example, suppose that the *pbb* of figure 1 was bound to the piece of code in figure 2.

```
      I = 1
100   READ(20,900,END=200)NEW(I+1)
      NEW(I)=I+2
      I=I+2
      GO TO 100
200   NEW(I)=0
```

Figure 2: Binding code

NEW is now bound to linkedlistl. This allows us to in effect attach the piece of information "holds linked list" to NEW and move the system's understanding of NEW away from the surface and closer to the domain.

If more than one template matches • particular piece of

code, only the more detailed is actually bound. For example, if a section of code was both matched by a Quicksort template, and a more general, abstract sorting template, the code would be bound to the *Quicksort pbb.*

2 5 Searching the Code

There are several possible outcomes when attempting to bind a particular hypothesized *pbb.* First, a template might exactly match the hypothesized *pbb.* In this case, the system has no new information regarding possible refinements to the matched p66, and hence, must choose a refinement to try using its own local information. Currently, a fixed order list of refinements is used, but in the future, a more efficient decision process could be used to take into account the type of knowledge experienced programmers carry around as rules of thumb, such as "use a linked list if data is going to be inserted or deleted frequently**, or the inverse "use an array if the data is relatively static" (see also Kant's "implementation selection rules" [6]).

If a template matches a more refined *pbb* than the one hypothesized, then the more abstract *pbb* can be replaced in the corresponding Implementation Plan with the new refined one. The effects of this new, more detailed *pbb* are allowed to ripple throughout the links of the IP, generally causing the refinement of other sub-parts in the IP. An attempt will be made to match each of these new, more refined sub-parts, causing possibly still further refinements and rippling effects.

The depth-first search for *distinctive features* may be preempted by the matching of a beacon. If the *beacon's* associated function is a sub-part of the current hypothesized plan, then the system will bind the sub-part and allow any rippling effects to take place. In at least some cases, this will result in the deactivation of the current *distinctive features* being searched for, and the activation of more refined templates. The overall control flow will thus be opportunistic, working on those parts of the *Implementation Plan* for which the most information is available.

2 6 When Things Go Wrong

In the ideal case, a matching piece of code can be found for each hypothesized sub-part in a plan; in practice, of course, this seldom occurs, and a number of things can go wrong. Two of the more common are: 1) no template can be matched for a particular *pbb,* and 2) a *beacon* template is matched for a *pbb* that is not part of the current hypothesized plan. In the first case, if many other *pbbs* in the plan are still unmatched and if an alternative plan is available, then the system will adopt the alternative plan. In most cases, however, an attempt will be made to match the missing *pbb* by assembling it out of smaller functions. In this case the system does behave in a bottom-up manner.

In the *second* case, the system attempts to revise the hypothesized plan to incorporate the unexpected match. We have not yet defined a satisfactory method for this modification which avoids the problem of re-binding already successfully bound sub-parts, but we are considering approaches which attempt to minimize the number of changes (see for instance the similarity networks of Minsky[7]).

3. CONCLUSION

The system we have described possesses several unique features. First, It allows a useful partial understanding to be developed whenever time, the task, or lack of knowledge make a full understanding unattainable or unreasonable.

Second, by noticing the behavior of human programmers, the notion of *distinctive features* and *beacons* was incorporated into the system. These features, in combination, allow the system to act in an opportunistic fashion, exploiting whatever looks most promising as recognition proceeds.

Finally, a knowledge representation called templates was defined. Templates link a syntactic pattern with a particular function, and are general enough to be used in both top-down *and* bottom-up search.

REFERENCES

[1] BARSTOW, D. (1978) Automatic Construction of Algorithms and Data Structures Using a Knowledge Base of Processing Rules PhD Thesis, AIM-308, Stanford AI Lab

[2] BROOKS, R. (1975) A Model of Human Cognitive Behavior in Writing Code for Computer Programs PhD Thesis, Dept. of Psychology, Carnegie Mellon University 1975

[3] BROWN, A. (1977) Qualitative Knowledge, Causal Reasoning, and the Localization of Failures PhD Thesis, AI-TR-362, MIT AI Lab

[4] de KLEER, J. (1977) A Theory of Plans for Electronic Circuits Working Paper 144, MIT AI Lab

[5] HOARE, C. (1969) An Axiomatic Basis for Computer Programming CACM 12, 1969

[6] KANT, E. (1979) Efficiency Knowledge for Program Synthesis PhD Thesis (forthcoming), Stanford AI Lab

[7] MINSKY, M. (1975) A Framework fro Representing Knowledge The Psychology of Computer Vision, P. Winston (ed.) McGraw-Hill 1975

[8] RICK C. *and* SHROBE, H (1976) Initial Report on a LISP Programmer's Apprentice AI-TR-354, MIT AI Lab, Cambridge, Ma.

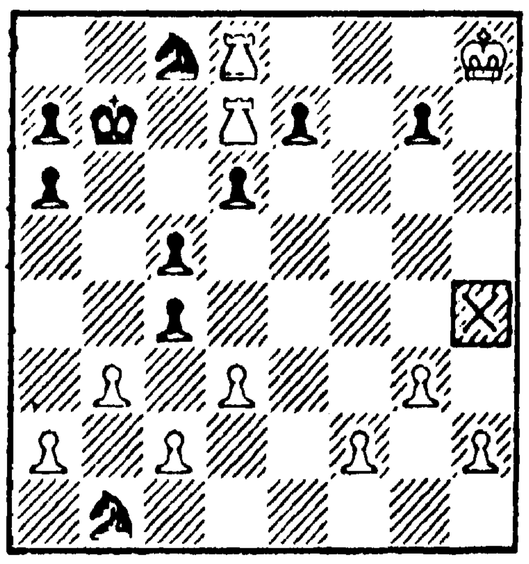
THE INTERACTION OF OBSERVATION AND INFERENCE IN A FORMAL REPRESENTATION SYSTEM

Robert Elliot Filman*
Artificial Intelligence Laboratory
Computer Science Department
Stanford University
Stanford, California

This work is an attempt to formally represent the knowledge required for the solution of a difficult retrograde chess problem (figure I). This solution Includes the extension of a formal deductive system to Include an observational facility. Using a proof checker for first order logic. FOL [9], we have detailed a proof of the solution of the puzzle, Including proofs for almost all of the necessary associated lemmas *[2],* We shall highlight the various representational decisions made In the process of axiomatiiing retrograde chess, discussing both the necessity for these particular choices, and their Implications for designers of representations for other domains. This work is part of the search for *epistemologically effective* formalisms for artificial Intelligence.

I. Introduction

Our consideration centers on a particular retrograde chess puzzle (*figure I*). Its solution (from basic chess principles) is beyond the ability of any current computer program.



A piece has fallen off of the board from the square marked X. What piece was it? This position was achieved in a legal chess game, though there is no presumption that either player was playing to win.

figure I

This problem was chosen because Its solution "requires" both "deductive" and "observational" Inferences, In a context Isolated from other issues of correctness and sufficiency. Deductive Inference Is obtaining new proof steps by the application of syntactic Inference rules. However, human reasoning proceeds not only by deduction, but also by the Immediate recognition of results, a process we identify with *Observation.* We have extended our representational system to include observational inference by performance of computation In a *semantic model.*

* Author's current address: Department, of Computer Science, Indiana University, Bloominglon, Indiana 47401.

** This work was supported in part by D-ARPA under contract MDA903-76-C-0206.

2. Inference

Human problem solving has several different modes. People are capable of *deduction,* applying syntactic rules to previous inferences. This has been the reasoning mode most frequently employed in A.I. programs. However, generally intelligent systems will need to reason by other schemes, such as *induction* (reasoning from particular cases to a general conclusion), *analogy* (modifying reasoning from a solution of another problem to fit the current situation), and what we call *observation* (quickly noticing an apparent conclusion, performed *in* our system by computation in a semantic model). If we are to create such programs, we must find a pragmatic representational mechanism, one that can support these other kinds of reasoning.

We perceive that the general A.I. problem is better approached by separating the representation of the knowledge of the problem domain *(epistemology)* from the search mechanisms used to manipulate that knowledge *{heuristics),* This distinction is discussed more completely in [7].

Classically, an axiom system is *epistemologically adequate* for solving some problem if there exists a solution of that problem from those axioms. For example, a world view based on interacting particles would be classically adequate. AI, intent on the production of intelligent programs, makes greater demands on its fundamental knowledge structures. We require not only the generality of adequate systems, but also economy of expression; that the solution be not only derivable, but that the derivation be short enough for a machine to actually produce it. We call this notion of both economical and adequate formalisms *epistemological effectiveness.*

This paper is primarily concerned with the qualities OF formalisms capable of epistemologically effectively representing both complex deductive and observational reasoning.

Deduction is obtaining conclusions by application of syntactic rules. The term *observation* is usually applied to human inference in several ways; they are unified by the notion of drawing a quick and immediate conclusion by examination.

Examples of human observation are seen in the technician reading the value of a meter, the mathematician recognizing the value of some simple function, and the clerk finding the amount of some transaction in a table.

This examination employs the following scheme: knowledge which is to manipulated in deductive form is represented in a *extended first order logic* formalism. Associated with this logical system is a computational LISP model of these axioms. The results of the evaluation of functions on objects in this model are to be legal inferences in the deductive, logical system. This shall *serve* as our notion of *observation.* From the point of view of our intelligent computer, observation by computation in a semantic model is as opaque as sensory input

3. Basic Formalisms

Just as our use of "English" is fundamental to our ability to present this paper, we have a fundamental need for an epistemological language for expressing reasoning. For that, we have chosen ah *extended first order logic.* It *is* the usual predicate calculus, with the addition of *equality, functions, axiom schemata,* and inference by *computation in a semantic model.*

We *use* formal logic for several reasons: I) The sentences of logic are unambiguous and easily understood. 2) First order logic has *explicit quantification.* 3) There has been much work in logic. Any system employing logic can avail itself of this work. Examples of such progress include resolution and decision procedures for propositionaly logic. 4) Logic *is* a *general* representation. It is obviously not contrived for one particular domain. 5) Declarative representations in general, and first order logic in particular, are easily and uniformly *extensible.* They can be extended not only with new axioms, but also with new predicates, functions and individuals.

These, however, are the minor justifications for the use of logic. More importantly, 6) First order logic provides a precise sense of meaning. From the rules of inference inherent in the system it is clear what can be deduced from any axiom. 7) Almost all other current representations, be they microplanner, semantic nets, KRL, or whatnot, are variations on the rules of formal logic. They are all expressible *in* logical terms. These formalisms differ principally in their interaction with the heuristic system. They are all declarative systems, and (almost) equivalent in terms of epistemology. Reliance on pure logic has two positive attributes: we are isolating the epistemological Issues from the implementation, and we are speaking the *lingua franca* of representation languages.

4. Proof Generation and Proof Checking

An intelligent computer program will need both an epistemologically effective world view and an appropriate set of heuristic procedures. This reasoning program is beyond our current abilities. However, we do not need this reasoning program to judge candidate epistemological formalisms. Rather, we need only to have our deductions certified. Clearly, the ability to accept a valid deduction is a prerequisite to the actual generation of that deduction. And an epistemologically adequate formalism can be tested by the use of a proof checker. This is essentially the *Missouri program* of McCarthy-Hayes [6J, and *is* in the tradition of the *Advice Taker* (5).

We were fortunate to have available for this research the program FOL [1,9], an extended natural deduction proof checker for first order logic. FOL will act as our Missouri program, checking the validity of *our* candidate inferences.

FOL is a proof checker for an *extended* sorted first order logic. FOL has been extended to incorporate a *tautology decider lor* propositional logic with equality. And, more importantly in FOL one has the ability to create a *partial semantic model* of one's world, and to evaluate the values of functions and predicates in that model, returning the results to the deduction level. This is called *semantic simplification.* This ability *is* satisfactory for simulating our notion of observation in chess.

5. Chess Puzzles

At the beginning of this paper, we mentioned a certain chess puzzle *(figure* /).* While space limitations preclude describing the solution of that puzzle, the reader should notice that, not only js the solution very complex, but that it also requires at least two different varieties of reasoning. Any solution will exhibit both *deduction,* inference of the form "Both sides can't be in check at the same time, black is in check, therefore white is not in check", and *observation,* inference of the form "I see black *is* in check" . We perceive our task as finding the appropriate incorporation of this observational reasoning into the deductive framework.

If the reader himself solves the puzzle, he will surely perceive how complicated chess reasoning can be. In addition to complexity, chess (as a problem domain) has another appealing attribute: we can *unequivocally* solve this problem This serves to help us to isolate the interesting features And chess is a somewhat ill-structured domain, the result of historical development, rather than mathematical sparsity or toy simplicity Our results, therefore, will principally be concerned with mechanisms, not representational details. However, understanding these mechanisms requires the presentation of those details

We will employ two different forms of inference in our system, the standard *aeductive* methods, based on axioms and natural deduction and augmented with decision procedures, and *observation,* semantic simplification performed by computation on a partial semantic model. We call this set of functions and data structures *the Chess Eye.* After defining a vocabulary of chess notions, we will consider the chess eye in greater detail.

6. Objects of the Chess World

Space limitations preclude detailing the entire chess axiomatization. Instead, we will describe only a few of the more important and interesting components.

The most obvious chess object is a chessboard, one of the *Boards.* Most chess problems are stated in terms of a board. Similarly, there are 64 *Squares* on any board, each with a unique identity, and 13 possible *Values,* such as *white pawn* and *empty,* corresponding to the objects on these squares.

However, values are different than the chesspieces themselves; the white queen's knight is not the same as the white king's knight; a promoted pawn is not *the* queen. Our reasoning will need these distinctions. Hence, we need a set of the 32 chessmen, called the *Pieces.*

The above, while somewhat sketchy, ought to appear fairly natural. However, as a representational mechanism, they are inadequate. We need to speak of the various things that must

* The fallen piece in figure 1 was the white queen's bishop. The reader is referred to *[2]* for the detailed solution.

have happened to reach this point in the game, the capture that must have occurred, and so forth. Even the chess rules, In delimiting castling moves and draw conditions, refer to the entire game history. We therefore need an explicit set of historical states, the *Positions,* which can be though of as the history (set of moves) employed in reaching this particular board. (There will be many different games that could reached any particular board.) Positions are a variety of state vector; the reader is referred to [6] for a more complete exposition on state vectors.

There is also the natural notion of the *Moves,* the explicit, discrete transitions between the positional states, and the *Colors,* black and white, associated with the two chess armies.

7. Predicates and Functions on Positions

Our most important manipulations will involve the state vectors, the positions. And the most fundamental relationship *in* reasoning about chess *is* that of legal move. Hence, we have the predicate *SVCCESSOR(p1,p2)* defined on two positions, if *p2* can be reached by a legal move from *p1.*

If *SUCCESSOR* is to be the succession function of our system, then, like arithmetic, we want a notion of "less than". This is called *PREDEGAME. PREDEGAME(pl, p2)* is true when *p1* occurred in the game that reached *p2.* Naturally, the initial position, *PO,* has the *PREDEGAME* relation to every legal game.

As each piece has its own Identity, we can refer to the piece occupying a given square in a particular position. Thus, if *px* is a position that was played to reach the problem board, we have *Pos(px, BKRI) - WK.* (Or, on *BKRI* in *px* is the *WK)*

We also have functions which extract the last move from a position, and the actors used in that move *(Move, Mover, To, From, Taken).*

8. Predicates and Functions on Boards

Now, positions were only one variety of state vector. The other, perhaps more familiar, sort is that of *BOARDS.* Just as one can speak of the status of some square in some position, one can take the value on some square on a board *(VALUE ON (GIVEN, BQ2) - RW).* (Note, however, that the range of this function is Values, not Pieces.)

The Issue of legal moves is also Important for boards; we have, for each piece, a predicate the determines the validity of any particular move for that piece on that board. Thus, for example, a bishop-valued piece can move from *BQ1* to *BKB3* on board *b* only if *blAG(b,BV1,BKB})* is true on that board. We can take the composition of these various movement types, and talk about *MOVETO(b,v,sqljq2)* if, on board *b,* a piece of value *v* could move from square *sql* to square sq2. However, legality of move is not purely a function of board; capture *en passant* and castling are not determined solely by board configuration.

Chessboards, as described in the previous section, will not even serve to represent the problem *of figure I.* Consider the square with the *X* upon it. It has no representation within the system of ordinary values that we have described We resolve this difficulty by the introduction of a new constant, of type *Values,* one that stands for "I don't know what's here, but it is something." This is the constant *UD* (for "undefined" value.) Permitting an undefined value to be on the squares of boards creates a natural ordering on boards by greater definition. We call this ordering *SUBBOARD(bl,b2).*

We need to tie boards and positions together. Obviously, a board with no undefined squares is the result of playing out a position. The function that extracts this board is *Tboard.* We state that VP *b.fBOARDtp.b) * SUBBOARD(bJboard(p)).* In reasoning about chess puzzles, we typically take the problem board, and Infer properties of any position that has the *BOARD* relation to it.

9. Observation in the Chess World

In our definition of the FOL inference system, we allowed computation in a partial semantic model to be a legal inference. We therefore define a LISP representation for (some) of the objects of our chess world, and to declare LISP functions to compute the computable predicates and functions on that model. For example, in our LISP model, atomic individual constants, like the black king and the white queen's bishop are LISP atoms; chessboards are eight long lists of eight atomic values.

Attachments are then made to functions and predicates such as VALUEON, ORTHO, and MOVETO. For example, the predicate MOVETO, which is defined axiomatically as:

```
∀b v sq1 sq2.(MOVETO (b,v,sq1,sq2)≡
  ((VALUER(v)∧ORTHO(b,sq1,sq2))∨(VALUEB(v)∧DIAG(b,sq1,sq2))∨
  (VALUEQ(v)∧ORTHO(b,sq1,sq2)) ∨(VALUEQ(v)∧DIAG(b,sq1,sq2))∨
  (VALUEK(v)∧KINGMOVE(sq1,sq2))∨
  (VALUEN(v)∧KNIGHTMOVE(sq1,sq2))∨
  (VALUEP(v)∧PAWNMOVE(b,v,sq1,sq2)))),
```

is attached to the function:

```
MOVETO (LAMBDA (b v sq1 sq2)(COND
  ((VALUER v)(ORTHO b sq1 sq2))((VALUEB v)(DIAG b sq1 sq2))
  ((VALUEQ v)(OR (ORTHO b sq1 sq2)(DIAG b sq1 sq2)))
  ((VALUEK v)(KINGMOVE sq1 sq2))
  ((VALUEN v)(KNIGHTMOVE sq1 sq2))
  ((VALUEP v)(PAWNMOVE b v sq1 sq2))));
```

in. the semantic model. We build up our hierarchy to have predicates such as *BLACKINCHECK* (which determines *it* black is in check on a given board), computable in the chess model. Thus, though our system is capable of expressing complex statements about chess, it is still able to derive the fact that black is in check on the problem board in a single inference.

The semantic simplification mechanism also has the ability to check WFF's quantified over finite sets by case computation in the semantic model. For example, the theorem: V i 3 *sq. Pos(PO^q) - x* (every chesspiece was on some square in the initial position) required 165 steps to derive deductively, but only a single semantic simplification.

This use of procedural computation in a model is a form of observation; the intelligent program, whose internal language is the formal predicate logic and axioms, can use the same mechanism for simplification done in its semantic model and observed events in the real world. What we are providing here is a mechanism for perception; that its primary use In computers will be a form of internal visualization or hallucination is no more a problem than accepting the human use of internal visualization in human problem solving.

10. Axioms

The most important axioms in this system are those that delimit legal moves. For example, consider the following representative axiom, MCONSEQA (move confluences *A).*

```
axiom MCONSEQA:   Vr q.(SUCCESSOR(r, q)⊃
    ((¬WHITETURN r≡WHITETURN q)∧¬POSITIONINCHECK(q,Color r)∧
    Prevpos q=r∧(WHITEPIECE Mover Move q≡WHITETURN r)∧
    Pos(r, From Move q)=Mover Move q ∧ Pos(q, To Move q)=
        Mover Move q ∧ Pos(q, From Move q)=EMPTY ∧
    (CAPTURE Move q ⊃  Pos(r, To Move q)=Taken Move q) ∧
    (CASTLING(r,q)∨EN_PASSANT(r,q)∨SIMPLELEGALMOVE(r,q))));;
```

It states some conditions on legal moves; which side is on move, not leaving one's king in check, where the pieces in the move are, and the begining of a taxonomy of move types.

It is also important to talk both about what remains true between moves. For example, those squares not involved in the move retain the same contents between any position and its successor.

```
axiom MCONSEQD:
    Vr q sq.((SUCCESSOR(r,q)∧ ¬sq=From Move q∧¬sq=To Move q∧
    ¬(CASTLE Move q∧(sq=Alsofrom Move q∨sq=Alsoto Move q))∧
    ¬(ENPASSANT Move q∧sq=Takenon Move q))⊃
            Pos(r, sq)=Pos(q, sq));;
```

11. Chess Induction

Another feature of the chess axiomatization that deserves mention, is *Chess Induction.* Chess induction is an axiom schema that states that if a property $\mathcal{P}$ is true of a position p, and remains true over the successor relation, then it will be true in all descendants of p (all games that can be reached from p).

We usually use the initial position, *P0*, for p, and prove properties true of all legal positions. More formal, this is:

$$(\mathcal{P}(P0)\wedge \forall\, r\ p.\ ((\mathcal{P}(r)\wedge \mathrm{SUCCESSOR}(r, p))\supset\mathcal{P}(p)))\supset \forall\, r.\ \mathcal{P}(r)$$

Chess induction is a useful deductive method. For example, using chess induction, we prove theorems such as: *A piece not on its original square has moved in this game (or been the moving rook of a castle)*:

```
∀ r sq x. ((Pos(r, sq)=x∧ ¬(Pos(P0, sq)=x))⊃  ∃ q.(
 (PREDEGAME(q,r)∨q=r)∧((Mover Move q=x∧To Move q=sq)∨
 (CASTLE Move q∧Alsomover Move q=x∧Alsoto Move q=sq))))
```

And that *bishops stay on a single color square*:

```
∀r sq1 sq2 ybi.((Pos(P0, sq1)=ybi∧Pos(r, sq2)=ybi)⊃
      (WHITESQUARES(sq1)≡WHITESQUARES(sq2)))
```

This is a powerful technique for dealing with the frame problem. If we have control over the types of interactions available in state transitions, induction schemas such as this one allow us to prove constant properties of distant states.

12. The Proof

This has been Just a small sprinkling of examples from the chess axiomatlzation. We hope it has been useful in conveying their flavor and form. From this axiomatiiation, we preceded to generate a proof of the solution to the chess puzzle.

The proof itself is too long to detail, but we are able to present a few comments about its "shape". The form of the proof matched, fairly closely, an E*nglish* explanation of the detailed solution to the same problem. The human proof required 68 "steps"; the FOL "proof", 405. In the process of deriving the FOL *proof,* 159 genera) chess lemmas and theorems were proven. These required 1702 steps. Additionally, six lemmas specific to this problem were also demonstrated, requiring another 136 steps. About half of the proof steps in the main proof were instantiations of axioms and theorems, and another quarter were requests for the confirmation of the chess eye. The complete FOL proof may be found in [2].

13. Syntactic and Semantic Reasoning

We will attempt to convey some of the epistemological design concepts uncovered in this examination. This is a brief description; a more thorough examination may be found in [2].

One of the major distinctions underlying this work is that knowledge can be successfully modeled in two different forms. One is by the application of *syntactic rules* and the other, *model based semantic computation.* From the perspective of the person communicating with a representational system, syntactic mechanisms convey by their *form* the information (and possible uses of that information) they represent Thus, in our system the *axiomatic* knowledge as syntactic. Knowing the rules of inference, it is clear what is derivable from any axiom.

Semantic rules, on the other hand, are magic. We are not told the Justification for any rule, why it works, how it works, only that we are to accept its results. Typically, it will be performing this action by functional evaluation in some partial model of the domain, complete with hidden assumptions about what Is right and possible there. Such inference is performed in this system by the functions of the Chess Eye.

Detailed exploration of formal chess reasoning reveals that it is a fallacy to believe a *simple methodology* will be able to express the knowledge of a complex domain. Expressing complex relationships will require complex linguistic machinery. For example, any system dependent upon viewing only a single model, updated with each action, will be unable to compare states resulting from different actions in anything but the most primitive way. Any system that thinks that a "shortest path net traversal" algorithm will permit it to avoid dealing with inconsistency is going to be unable to do deep reasoning that relies on disjunction or case analysis.

14. State Vectors

The manipulation of *state vectors* played the central role in this axiomatization. State vectors are like the practice of proofs in formal logic. Everyone knows how they can be used, but no-one is every caught manipulating one in a complicated way.

We remind the reader that there were two flavors of state vector employed in this research; the first is the *static* vector, the boards. Boards were used to display a particular *slice* of time. Similarly, these chessboards were a fully realized, concrete object upon which calculations could be performed. The second is the *historical* vector, the position. A position contains the complete history of any particular game. However, it has no explicit representation in the model.

Each of these had important application. The full realization of the board (the association with any board with a data structure that completely defined that board) permited computation on that board; and this employment of the Chess Eye proved remarkably useful in reducing the magnitude of derivation. However, the historical state vector, the position, was also vital. Much of the work of this proof consisted of comparing states, asserting that a particular state with

particular properties must have existed. Usually this state occurred in the game played to reach some other state, and was interesting for that reason.

There is a third kind of state vector, the *possible world.* This vector contains information not only about the status of objects in the world, but also propositions about the propositions about those worlds. The most common use of these possible worlds is in semantic modeling of model logic systems. Possible worlds were not needed to model the simple world of retrograde chess.

State vectors seem to be one of three possible approaches to the problem of extending the set of things one can talk about in a formal system. This is the method of *reification,* or the making of new objects Here we postulates new, "ordinary" objects, from whicn propositional information can be obtained by use of the axioms. These objects can be manipulated by the use of the ordinary rules of the standard logic.

A second possible alternative is provided by *modal logic* formalisms; these attempt to talk about propositions by adding new "quantifiers" on propositions, and new inference rules to the deductive system. Unfortunately, if the proposition to be talked about contains several possible modals, the system can become rather cluttered with additional inference rules. And each additional modal requires extending not only the inference system, but also the heuristic system.

The third alternative focuses on the notion of *meta level reasoning.* That is, the propositions on one level become the objects of the next level. An additional set of inference rules is required for transferring between these levels. Since this scheme can be recursively applied, no additional mechanisms need be added to handle other proposition types. This is essentially a more advanced form of reification.

Of these three, the third seems most promising, and it is currently being incorporated into FOL If this axiomatization were to be repeated, the features of this meta level reasoning would have allowed more "incompletion" in the manipulation of the semantic model. However, the state of FOL at the time this research was performed compelled the use of reification and state vectors rather than this last, more general scheme. These distinctions are discussed more fully in [10].

One other major success in the use of state vectors ought to be mentioned: we had great success applying induction schema to historical vectors to prove properties of distant states. The induction schema *Chess Induction* stated that if a property was true over the successor relation, then it held in all descendants of any state for which the property is true. Chess induction permited the proof of many previously intransient theorems, such as "Only pawns can promote" and "Every piece now on some square either moved there in a previous position, or started on that square". We believe that this induction schema provided a good partial remedy to the frames problem, permitting proofs about distant states without even the trouble of simulating the actions involved.

15. Compound Objects

The state vectors are examples of another form of reification, that of creating *compound objects.* The mathematical comprehension principle, creating sets, lists and bags, are forms of compound objects. Our experience is that compound objects play a critical role in even the simplest epistemology.

Compound objects are related to the idea of *partially defined objects.* A partially defined object is one that is a representation for something we have conceptually more information. We suggest there are two different varieties of partially defined objects that need to be dealt with in a complex reasoning system. One of these is exhibited by the *boards* with their explicit use of *UD,* the "undefined" constant. A board with an undefined in it is not conceptually a first class object of our system; rather, it is an explicit representation an object about which we only have partial knowledge. This is its tremendous advantage; with this explicit representation, we can use a partial board as data for a functional evaluation. However, the use of this explicit undefined limits the set of things that can be unknown about a partial board; we might know that an undefined square contains some officer, or that there *is* a queen somewhere on the board, or that this board was a legal move away from some other board. None of these partially defined objects is representable by this scheme. Each of them could be represented by some other scheme, *but not all such partial information* can be represented by any scheme that does not represent its information propositionally. Representing propositional information in the model is already meta-reasonlng [10].

The alternate variety of partially defined objected exhibited In this representational system is the *position.* In one sense, the position is completely defined; the position does not have any formal elimination of information. On the other hand, except for the most trivial positions, one knows nothing about any position that cannot be established propositionally. It is only possible to infer properties of a position from its relation with other positions and boards.

We see then that the board is the *concrete* realization of the *abstract position.* Any epistemologically effective system is going to need to deal with both abstract objects (for which there can be no representation in its model) and concretions of those objects (for *in* model to manipulate).

This discussion leads us to touch upon the Issue of *multiple representations* or the different *aspects* of similar objects. What we are considering here is the ability to refer to objects that have a common conceptual source in different fashions.

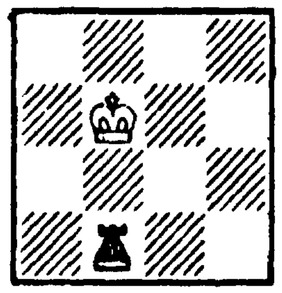
We see this division in our chess axioms. Inherently, there is only one conceptual object associated with any square at any point. Nevertheless, we find the necessity for dealing with this object in two different garbs: piece and value. We essentially have things to say about the pieces that both abstract information out (it's a white rook, not the white queen's rook) and that add information in (but it used to be a pawn). We see this same division in the board/position duality, where the board is really a "slice" of a position.

1G. Linguistic Generality

This final section provides a perspective on how well we've provided a language for reasoning about chess. We need to consider a two part perspective: What were the failures in this representation system? How well could it be used to talk about problems other than "A piece fell off the board ...?"

There are two important linguistically failings in this axiomatization. The first of these was the failure to include general notions of *compound objects,* particularly sets and their kin. It was possible to deal with arguments equivalent to the "pigeon-hole principle" by case checking; but these are not naturally case arguments.

The other source of regret Is that the representation system, particularly In the semantic model, Is tied too closely to the rules of legal chess, and too closely to too simple an idea of representing chess knowledge. This point can be better Illustrated with a pair of examples. Consider this problem:

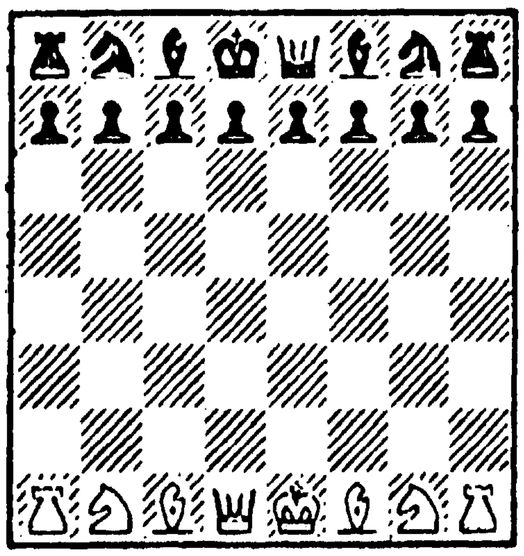


Is white in check?

figure 2

THE CURRENT *axiomatization* as currently FORMULATED, would NOT recognize this "fragment" as a piece of a "board". However, this Is a natural extension for the human problem solver The axlomatlzatlon can be criticized for adhering too closely to the rules of chess, and ignoring its "prlmitives". Correction of the axiomatlzatlon to reflect this more primitive level would not need to be a major a revision.

This second example displays a more fundamental linguistic difficulty. Consider this problem, from [3] somewhat more conventional in Its phrasing.



White to play and mate in four.

figure 3

This is not so much an Issue of mating in four, so much as recognizing that (as the black king and queen are on the wrong color squares) the pieces have exchanged sides. While the current axlomatlzatlon could be used to prove that *if white started at the bottom of this board, then this board could not have been reached in a legal game,* In some sense, the puzzle needs to be solved before It can be put in the acceptable form for the axioms. What we have here is an Issue of language. There are certainly many questions one would like to ask In, for example, English, for which natural language Is inadequate. Imagine, if you wilt, the circumlocutions Involved In giving a written spelling test. The moral here is an epistemologlcal Incompleteness theorem; there will always be things that cannot be said in any system.

We have consider the axiom system In the light of other retrograde problems, and find that is it effectively capable of expressing their solution. In particular, we have considered the problem in [4],* and find no difficulty in expressing its solution in our axiomatizatton.

17. Summary

We have taken a difficult problem of retrograde analysis chess, detailed a set of axioms for the rules of chess, and have shown that those axioms, together with a clever natural deduction system and the ability to simplify in a semantic model, are sufficient to produce a manageable derivation of the solution of a complicated problem. We have considered some of the conclusions that can be formed about formal knowledge representations from this work. Particularly we have explored, the nature of the objects of the chess world, such as state vectors, concrete and abstract objects, and compound objects and the value of things such as semantic simplification, meta reasoning, modal reasoning and induction schema.

18. Acknowledgments

] would like to acknowledge the helpful advice provided by Drs. John McCarthy, Richard Weyhrauch and Terry Winograd in the course of pursuing this research.

19. References

1 Filman, Robert E, and R.W.Weyhrauch; An FOL Primer. Memo AIM-288, Stanford University, September 1976.

2 Filman. Robert E; The Interaction of Observation and Inference. PhD thesis, Computer Science Dept. Memo AIM-327, Stanford University, March 1979.

3 Gardner, Martin, *Mathematical Games* in the Scientific American 2025 (May 1959).

4 Gardner, Martin *Mathematical Games* in the Scientific American 228.5 (May 1973).

5 McCarthy, John; *The Advice Taker.* In M. Minsky, ed. Semantic Information Processing. MIT Press, Cambridge, Mass 1968

6 McCarthy, John, and P.J. Hayes; *Some Philosophical Problems from the Standpoint of Artificial Intelligence.* In Machine Intelligence 4, 8 Meltzer & D Michie, eds. Edinburgh University Press, Edinburgh, 1969.

7 McCarthy, John; *Epistemological Problems of Artificial Intelligence.* In Proc. IJCAI-77. MIT, Cambridge, Mass., August, 1977, pp 1038-1044

8 McCarthy, John; *Concept valued functions.* To appear In Machine Intelligence 9. D. Michie, ed.

9 Weyhrauch, Richard W.; A Users Manual for FOL, Memo AIM-235.1, Stanford University, 1977.

10 Weyhrauch, Richard W. and Robert E Filman; Meta Reasoning and Extended Inference for AI *(in preparation).*

Black king *on* WQ1, bishop on WQ5, rook on QN5: white bishop on WQR4. Where is the white king?

JETS: ACHIEVING COMPLETENESS THROUGH COVERAGE AND CLOSURE

Tim Finin, Bradley Goodman and Harry Tennant

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Work in progress on JETS, the successor to PLANES, is described. JETS is a natural language question answering system that is intended to interface users to a large relational data base. The architecture is designed to extend the conceptual coverage of JETS to better meet the conversational and data base usage requirements of users. The implementation of JETS is designed to gain a high degree of closure over concept manipulation, contributing to a solution to the problems of perspicuity and scale. Specific examples are given of concept manipulation through the implied relationships of modification and of an approach to problem-solving through the use of frames.

JETS is a natural language question answering system that is currently under development. Its goals and design are an outgrowth of the experience gained in the development and user environment testing of PLANES [13, 14]. This paper will describe JETS in terms of motivation, performance goals, strategies for achieving the goals, and techniques for measuring performance.

The PLANES system was developed to study the problems of natural language access to a very large data base. The data base which was used contained naval aircraft maintenance and flight records, and is the same data base that JETS will be applied to. The thrust of PLANES was toward the investigation of natural language data base interfaces that could be engineered to comply with response time and machine size constraints while enabling the user to express questions in the language to which he is accustomed. PLANES accepts user utterances that may or may not comply with standard grammar, that may include pronominal reference, and that may include some forms of ellipsis. In an effort to make an objective assessment of the achievements of PLANES, testing was conducted using the techniques described in [12]. The testing revealed that in addition to problems of linguistic completeness, there are interesting questions about the conceptual completeness of question answering systems. JETS is the response to those questions.

1. Coverage and Completeness

The performance of question answering systems can be thought of as having two nearly independent components: the elements of conceptual coverage and the elements of linguistic coverage. The conceptual coverage of a system refers to the set of concepts that it can deal with. The linguistic coverage of a system refers to the set of linguistic features that have been included in it to allow for variations in the way the concepts are referenced. Syntactic structure, anaphoric reference and ellipsis are all examples of elements of linguistic coverage. Conceptual coverage and linguistic coverage are attributes of the program. The adequacy of the conceptual and linguistic coverage may be measured against the demands that a set of users place on the system. The measures are called conceptual and linguistic completeness, respectively. The attributes, conceptual and linguistic coverage, and the measurements, conceptual and linguistic completeness, are described in more detail in [12].

The major work on PLANES, as on most question answering systems, has been to provide the user with a habitable [15] system through high linguistic completeness. Our testing shows that linguistic completeness is not the only limiting factor in building habitable natural language systems. The users wanted to be able to refer to certain concepts, but were unable to do so. For example, users often asked questions about the structure of the data base or asked for definitions. They made statements to set the context for later questions, or attempted to summarize a portion of the dialogue expecting agreement or disagreement with their summaries. They made presuppositions about the data that were not Justified. In short, there were a significant number of concepts that users attempted to reference or capabilities that they attempted to use that were not included in PLANES.

In addition to the voids in conceptual coverage that were discovered while testing PLANfiS, it was observed that the users readily adapted to the limitations of PLANES[1] linguistic coverage. The user's "intellectual overhead" or "distraction from the problem" was not

This work is supported by the Office of Naval Research under Contract N0O014-75-C-O612.

measurable, but they did appear to be able to express themselves. We feel that improving the linguistic completeness of question answerers is indeed an area where attention should continue to be focused. However, linguistic completeness is only part of the problem of building habitable natural language systems. The work on JETS is centered on improving the conceptual completeness of natural language question answering systems.

With this motivation, our work on JETS has four goals. First, we intend to enlarge the domain of discourse. Instead of simply enabling a question answerer to accept questions about a data base, it must also be able to handle references to objects defined in the discourse. This includes characterizing the responses to queries that it generates to resolve references to its own utterances by the users. It should be able to handle references like "the last plane I mentioned[1]* and "the last question**. It should be able to clarify for the user how it interprets vague, ambiguous, or technical phrases.

Second, the natural language component should take on greater responsibility for pragmatic knowledge. It should have two distinct components. One component contains knowledge about the data in the data base, specifically including relationships between data types and between individual data elements. The other component contains knowledge about the activities that are reported on in the data base, rather than just the data in the data base. We have seen users refer to concepts that are related to the activities, but that are not in the data base. A natural language system should be able to understand these references if only to be more helpful in explaining why the user*s question cannot be answered. The following responses could be generated by JETS only if references to concepts outside the range of the data base were understood:

1 There is no data on individual pilots in the data base.

2 Aircraft are not grouped by squadron in the data base, but they can be grouped by permanent unit assignments.

3 The data base does not describe all combat aircraft, only fighter and attack aircraft.

4 The data base does not specify the part that failed, only the system or subsystem that contained the failure.

The third goal for JETS is to break away from the familiar model for question answerers of interpreting each user utterance as a data base query. We have observed that users do not intend eaoh utterance as a query. Some utterances set the oontext for later discourse or give general Instructions such as data formatting. Some users take several sentences to specify one query, while others couch several queries in one sentence. JETS builds an internal description of the user*s utterance, and takes action on it depending on the user demands that are implied in the utterance.

Oar fourth goal is to evaluate the performance of JETS with respect to its conceptual completeness. Designing and testing explicitly for linguistic completeness Is a goal for future research. Our conceptual completeness tests will consist of giving short descriptions of the domain of discourse to subjects, each acting in two roles. First, acting as composers, they will generate data base problems. Then, acting as users, they will try to solve problems generated by the other subjects. By having subjects who are only slightly familiar with the data base compose problems, the problems will not be biased toward those that can actually be solved using the data base. This is intended to simulate the data base problems that would occur to a casual user, one who is not very familiar with the exact contents of the data base.

The users will interact with JETS through a human interpreter, whose task will be to maintain the conceptual content of the user utterances while making them comprehendable to JETS. In other words, the interpreter compensates for potential deficiencies in linguistic coverage. The conceptual completeness of the system can be measured by computing the fraction of user utterances that were properly handled by JETS.

2. Closure

The architecture of JETS centers on establishing adequate conceptual coverage of the data base and its contents, the activities that the data base describes, and the dialogue between the user and system. The implementation of JETS, on the other hand, centers on capturing a degree of closure in the manipulation of concepts. Closure, as defined by Woods [16], implies handling user utterances which are consistent with the domain, but which may not have been specifically foreseen. Closure can be approached by basing interpretation on rules that are as general as possible.

The knowledge of JETS is organized into a network of frames. There are many kinds of links associating frames with one another, but two of the most useful are the generality/specificity links. With these links, the conceptual system can be thought of as a directed tree of frames rooted at the most general concept. The more specific frames, those closer to the leaves of the tree, inherit the information stored with their ancestor frames and add more specific information to that. The fundamental advantage of the abstraction hierarchy is that the classification of specific concepts into more general ones promotes the identification of regularities among the concepts. The hierarchy encourages the use of interpretation rules that are written to apply to the most general frames possible, and then automatically apply to their more specific descendents. The conceptual component of JETS will be implemented in FRL [4], a language for building frame based systems.

Use of the frames In JETS will be made primarily by a set of interpretation rules. These rules are represented as frames and currently have three major slots: a pattern, an action and a condition.

Implementing the interpretation component as a set of rules facilitates the insertion and deletion of rules. This means that incremental changes to the rule set may be made without necessitating the understanding of elaborate interpretation procedures.

The rules are organized hierarchically by generality, the more general rules being closer to the root. The hierarchical organization of rules is useful in resolving problems of contention. When more that one rule qualifies to be applied in a given situation, and one rule is a descendent of another, the more specific rule is chosen. In addition, the hierarchical organization of rules helps identify regularities among the rules. Regularities among the interpretation rules promote the writing of more general rules, thus contributing to gaining closure. Thus, we see several points which should prepare JETS to handle the problem of scale, one of the most important problems of natural laaguage systems: the ready ability to make incremental changes to interpretation rules, the concept clustering advantages of frames, and the identification of regularities among both the concept frames and the interpretation rules.

3. Achieving Semantic Closure

A part of our work is centered on achieving closure in the area of semantic interpretation. Our focus is an attempt to adequately cover basic noun modification, in particular, the interpretation of noun-noun modification [3J.

The problem of interpreting strings of nouns related through modification is a complex one. For our purposes, we divide the problem into three subprobleras: lexical interpretation, modifier parsing and concept modification.

By lexical Interpretation we mean the process of mapping the lexical items (in this case the nouns in the string) into appropriate concepts. The principal difficulty here is handling words with multiple senses.

Modifier parsing is the process of discovering the internal structure associated with the string of nouns or the concepts which result after lexical interpretation. For example, a string of three nouns, N1 N2 N3, might have the structure ((N1 N2) N3) or the structure (N1 (N2 N3)). The first structure would be chosen for the string "engine damage reports" and the second for the string "replacement oil pump".

The term conceptual modification refers to the problem of assigning an interpretation to an instance of one concept modifying another. For example, when the ENGINE concept modifies the DAMAGE concept in the phrase "engine damage" we want to fill the damaged objects role in the DAMAGE concept with the ENGINE concept. A more complex example is the interpretation of the phrase "engine housing acid damage". Here, the desired result is something like:

> A RESULT of a DAMAGE event in which the damaged object is a HOUSING part of an ENGINE and the causes is a CORROSION event in which the corrosive agent is an ACID and the corroded object is the HOUSING part of an ENGINE.

These three subproblems are, of course, interrelated and cannot be completely decoupled. In our initial research we are concentrating on the problem of Conceptual modification. Our goal is that, given any two concepts that are correctly interpreted by the system, their combination (i.e. through modification) will be correctly interpreted by the system. Note that the correct interpretation of a concept does not imply that the system should be able to "handle" the concept in the sense of answering questions about the concept or relating it to the data base.

The problem of interpreting noun-noun modification brings the issue of closure into focus. The essential feature of noun-noun modification is that the semantic relationship which exists between the two nouns is not explicit in the utterance. Moreover, a large number of relationships may, in principle, be possible between the two concepts represented by the nouns. It is the responsibility of the system to attempt to infer or discover an appropriate relationship, given its understanding of the two concepts involved, general pragmatic knowledge, and the current discourse context.

3.1 An example: Time

As an example, consider the use of a time phrase used to modify a noun, as in the phrases "January Skyhawks repairs" and "1976 flights". If the system can interpret phrases referring to time (as almost any system must) then it should attempt to interpret the modification of any other concept which could conceivably have a time phrase attached to it.

In our semantics, a time phrase can only be used to modify a concept which is, or can be viewed as, a kind of an EVENT. A minimal amount of closure is achieved when any event or event related concept can be successfully modified by a TIME concept. What if the modified concept is not an EVENT but something else, say an OBJECT? If a time phrase is hypothesized to modify something which is a kind of OBJECT, our system will attempt to derive an underlying event associated with that object to attach the time phrase to. For example, in the standard PARTS-SUPPLIERS-PROJECTS domain, the phrase "January parts" might suggest the interpretations: "parts which were snipped in January", "parts which were received in January", or "parts which were ordered in January". In such an impoverished domain this is almost trivial, as one can precompute the set of events in which a concept can partake.

In a semantically rich domain, such as our 3-M data base [7], the problem is much more difficult. One can not (or perhaps should not) always enumerate the potential relationships which might exist between even two simple concepts. The ability to handle references to entities and relations mentioned earlier in the discourse makes the problem even more complex. This allows for more potential relationships between any two concepts. For example, the phrase "the January planes" could be used to

refer to a set of planes introduced previously in the discourse. The successful interpretation of this phrase would require a search through the recent discourse to discover a set of planes which was involved in an event which occurred in January.

3.2 An examples: Sets

We have introduced our JETS system to the concept of a SET. In order to achieve a high degree of semantic closure, the system should be able to form the ooncept of a SET over a wide domain of objects. Our previous system, PLANES, handled sets in an unsatisfactory way. One could refer to sets of objects of certain types but not others. For example, PLANES could understand descriptions of and references to a set of aircraft or maintenance codes but it could not handle sets of parts or "how malfunctioned" codes. Such shortcomings are particularly bad in that they mislead users. If a user was successful in using a description of a set of objects which PLANES understood, he could quite reasonably infer that PLANES understood the general ooncept of a set and could form one of arbitrary objects.

Given that sets can be represented and formed in a uniform way over the widest possible domain, we must turn our attention to issues of interpreting the modification of the set concept. The ability to form sets of arbitrary elements will be of limited use if the semantic interpretation rules do not allow one to modify such sets in a general way. Thus, if the system knows what it means for concept X to modify concept Y, then it should know what it means for concept X to modify a concept Z where Z is a set whose members are concepts which are Y's.

Oar approach is to include a meta-rule for sets which uses rules applicable to the particular domain of a set. Our SET frame has a slot for a typical members
as well as one to receive the actual members, if there are any. This slot refers to a frame which describes the typical member of the set. Whenever we wish to modify a set by another concept, this meta-rule will search for primitive rules which Interpret modification of the set's typical ,members by that ooncept. The rules which are found to be applicable are then Invoked on the set's typical member and to the set's individual members, if any exist. This meta-rule for sets is shown in figure [1].

If a concept C modifies a set S then:

[1] Find a modification rule R which interprets the modification of S's typical member by C.

[2] Instantiate a new set S2.

[3] Fill S2's typical member with the result of applying rule R to S's typical member and the concept C.

[4] Fill S2's members with the result of applying R to each of S's members and C.

[5] Return the set S2.

An Interpretation Rule for Sets

figure [1]

Let's examine the use of this meta-rule for sets in the interpretation of the phrase "planes 3, 5, and 48". At the concept level, this phrase is represented as the general PLANE concept modifying a SET concept. This particular SET has the INTEGER concept for its typical member and, as individual members, the concepts for the integers 3, 5 and 48.

One of the rules applicable when a PLANE modifies an INTEGER, is a rule which interprets the integer as representing the plane's serial number. In our world, the only planes which have serial numbers are those in the data base. These planes are represented by the more specific concept 3M-PLANE. The action of this rule is to view (*) the plane concept as a 3M-PLANE and the integer concept as a SERIAL NUMBER. The 3M-PLANE's serial number slot is then filled with the SERIAL-NUMBER concept.

When this rule is found by the meta-rule for sets, it is applied to both the set's typical member filler (in this case the generic INTEGER concept) and to the set's members (which are the integers 3, 5 and 48). The result of all this is:

A SET with typical member = a 3M-PLANE
members =
(a 3M-PLANE with
serial number = an INTEGER with value = 3
a 3M-PLANE with serial number ...
a 3M-PLANE with serial number ...)

To handle the case where a SET is used to modify another concept, we include another meta-rule, shown in figure [2]. Consider the role of this rule in the interpretation of the phrase: "radar and navigation equipment failures".

If a set S modifies a concept C then:

[1] Find a modification rule R which interprets the modification of C by S's typical member.

[2] Instantiate a new set S2.

[3] Fill S2's typical member slot by invoking rule R on the typical member of S and the concept C.

[4] Fill S2's members by invoking R on each of S's members and C.

[5] Return the set S2.

An Interpretation Rule for Sets

figure [2]

This phrase is interpreted as a SET whose typical member is a FUNCTIONAL-SUBSYSTEM and whose members are the concepts RADAR-SUBSYSTEM and NAVIGATION-SUBSYSTEM. In interpreting the

* Viewing a concept X as a concept Y is a process which maps the information in X into a newly instantiated Y concept. If X is a kind of Y, no mapping need be done, however.

modification of the FAILURE concept by this SET, the meta-rule for sets is invoked and attempts to find rules which guide the interpretation of a FUNCTIONAL-SUBSYSTEM modifying a FAILURE. The rule which is most applicable is one which Interprets the modifying concept (the subsystem) as filling the failure location role of the FAILURE concept. The final interpretation of this phrase results in a SET of FAILURES in which the typical member is a FAILURE in a FUNCTIONAL-SUBSYSTEM and which contains two members: a FAILURE in the RADAR-SUBSYSTEM and a FAILURE in the NAVIGATI0N-SUBSYSTEM.

The discussion so far has centered around the need for achieving closure and possible ways of accomplishing it in JETS. However, nothing has been done to help the system develop a plan to extract the required information from the data base in the form the user intended. This is where problem-solving frames are introduced. Problem-solving frames are to be used to describe problem domains and search strategies. They can range from very general sketches of a particular series of problem environments to specific suggestions on how to answer a certain request. The latter problem-solving frame might even be encoded in English—where a sequence of English instructions are given on how to put all of the information together at the end to answer the main request. Problem-solving frames will have to work Jointly with the frames used during the parsing and semantic analysis stages. Those frames describe the objects and events of the JETS' environment and how they are related to the data in the data base. Information gathered by those frames can be used to extract appropriate values to fill the empty slots in the problem-solving frames in order that the whole frame may become instantiated.

4.1 Well -Defined Problems

Before describing the details of problem-solving frames, a description of what a "problem" is should be discussed. A well-defined problem has been defined in the literature to consist of the following properties:

1 expressed in terms the solver can understand,

2 all information necessary for solving the problem should be in the problem statement,

3 the form of the solution is exactly specified, and

4 there must be a systematic (algorithmic) way(s) for testing a proposed solution [8].

In our natural language query system environment (in particular with our experience with PLANES) we find that many requests by users are not normally well-defined, i.e. they do not satisfy the criteria above. Before such requests can be handled, it is necessary to make the request a well-defined problem. Previous systems have brought in this additional knowledge primarily through two sources—using past context to fill in missing information and/or asking the user directly for it [2,14]. We want to introduce the use of world knowledge as a third way of obtaining such information. At the same time we want to use the world knowledge to detect requests not answerable with data available in the data base.

Collecting together such information still does little for bringing the system closer to developing a plan for answering the request. The problem-solving frames are to propose general techniques—such as problem reduction (reducing a problem to simpler subproblems)—for solving problems. To be able to develop a set of problem-solving frames that classify problems and techniques for solving them, we must categorize the "kinds" of problems encountered in our data base environment (based on the assumption that the data base world restricts us enough that the number of interesting classes of problems is manageable). Our studies have shown that the following kinds of requests occur most often: statistical analyses (correlation of data, etc.), categorization (classifying data into categories), association of data types to each other, ranking ("top five", etc.), plotting, causality (generalize concept), very general inferencing and operations on sets.

Another major contribution would be the ability to provide different "views" to a problem and to the values stored in the data base. In essence an "overlay" of the data base could be generated for a particular problem. An example would be data stored in a daily format in the data base viewed as if weekly data existed. This same mechanism could be applied to bring two seemingly disjoint concepts in a problem together.

4.3 A. Scenario

A sample scenario of the use of problem-solving frames can be seen in Figure [3]. They are activated by key words and phrases in the user's request and as side-effects to the instantiation of particular frames during the semantic analysis stage of JETS. The activated frames analyze the request to decide if they can provide any guidance in formulating a plan capable of rendering a solution to the problem. If no such assistance can be offered, the frame is deactivated; otherwise complete control is passed to that problem-solving frame. Figure [4] gives an example of a problem-solving frame for comparison.(*) Notice the use of production-like rules that associate specific conditions and actions. These actions include drawing in other problem-solving frames and invoking specific functions such as MAKE-UNITS-EQUAL which tries to convert the units of measurement (time, length, etc.) of the individual fields into equivalent forms.

In summary, the value of this type of frame is that it provides a general forum for taking all the information provided by the frames instantiated

* The frames in our system are actually being written in FRL but the examples in this paper were written in a more simplified form for readability.

```
U: Compare NORS and NORMUS percentages by squadron
   and by wing for May 73.

J: Invoking COMPARE frame.  Invoking PERCENT frame.

   NOR (Not Operationally Ready) not in the data
   base.  It can be obtained by ADDING NORS (NOR due
   to Scheduled  Maintenance) and NORMUS (NOR due to
   Unscheduled Maintenance).

   SQUADRON  partitions the existing planes up.  No
   such data is stored in the data base.  Such
   partitioning can be done by:
      (1) plane serial number
      (2) plane type
      (3) none of the above
   Please select one of the above:

U: 2

J: WING linked to SQUADRON. PLANE TYPE absorbing
   WING.

   I have interpreted your request as follows:

     1 Retrieve NORS, NORMUS by plane types for time
       period of May 1973.

     2 Form NOR=NORS+NORMUS.

     3 Generate NORS%=NORS/NOR and
       NORMUS%=NORMUS/NOR.

     4 Construct table of PLANE TYPE, NORS
       percentage, NORMUS percentage.

     5 List by PLANE TYPE the maximum of NORS
       percentage or NORMUS percentage.

   Does this satisfy your request?

U: yes

J: Executing...

      PLANE TYPE   NORS%   NORMUS%
          A7         71%     29%
          F4         55%     45%
      SKYHAWKS       27%     73%

          A7: more NORS
          F4: more NOR
          SKYHAWKS: more NORMUS

                    A Scenario

                    figure [3]
```

```
Invoking Concept:
  COMPARE, MORE, LESS, CORRELATE, ANALYZE, ...

Context(s):
  Requires instantiation of two or more entities
  to compare:
    (1) one FIELD over two or more ranges, or
    (2) two or more different FIELDs:  FIELD1,
                  FIELD2,...

Actions:
 Context=(1):
  if FIELD.TYPE=num & CONCEPT=more
     then invoke greater;
  if FIELD.TYPE=num & CONCEPT=less
     then invoke less;
                .
                .
                .
 Context=(2):
  if FIELD1.TYPE=num & FIELD2.TYPE=num
                     & CONCEPT=more
     then invoke greater;
  if FIELD1.TYPE=num & FIELD2.TYPE=num
                     & CONCEPT=less
     then invoke less;
  if FIELD1.TYPE=num & FIELD2.TYPE=set
                     & CONCEPT=more
     then invoke cardinality-of-set
          invoke more;
  if FIELD1.TYPE=num & FIELD2.TYPE=num
                     & CONCEPT=compare
     then invoke find-relationship;
  if CONCEPT=associate
     then invoke check-correlation
          invoke find-relationship;
                .
                .
                .
Global Actions:  make-units-equal

                    COMPARE Frame

                    Figure [4]
```

command is either the sort of question which can be understood directly, or another command which can ultimately be broken up into a sequence of questions that can be answered. They will also be used to generate dialogue with the user when vague terms must be further explained before a formal query could be generated (e.g. defining "worst" for the particular user and question).

during parsing and semantic analysis and analyzing it so that a program can be generated to answer the question. In other words, it is not responsible for writing a program at the formal query level but only to decide what question to answer, to break the question up into a sequence of simpler requests if the question is too complex, to provide higher level mathematical/statistical analysis of the data returned, and to call on a formal query generator to generate the actual formal query.

4.4 Handling vague. and Complex Requests

The other type of problem-solving frame not yet described will be used in analyzing vague and complex questions. Each frame will consist, in essence, of a sequence of simpler commands, where each simpler

In JETS, this kind of problem-solving frame is useful for report generation. The following describes what a problem-solving frame for requests like "Does trend analysis of failure and maintenance rates differ significantly from the corresponding rates of new aircraft?" would have to do.(*)

1 define "differ significantly" (user-defined or system default),

2 retrieve for each maintenance action X whether it was scheduled or unscheduled and time since last maintenance on same system,

• Taken from list of questions most asked by 3-M Naval personnel[7].

3 form maintenance rate for each maintenance action,

4 project trend of maintenance rates by linear regression,

5 calculate average mean time between failure,

6 apply (1) through (5) to new aircraft,

7 generate table of trend of failure rate and maintenance rate for all data vs. trend of failure rate and maintenance rate for new aircraft, and

3 compute standard deviations for the trend of failure rates and maintenance rates for all aircraft and for Just new aircraft.

The problem-solving frame writes all of the program required except the generation of the basic formal queries (the ones that retrieve the actual fields used to calculate the failure and maintenance rates) which are generated by the formal query generator [13].

5. Conclusion

This paper has discussed the goals and design of the JETS natural language question answering system. Our work on JETS is strongly motivated by our experience with the earlier PLANES system. The evaluation of PLANES highlighted its problems and shortcomings. A major goal of the work on JETS is to increase the completeness of the conceptual coverage of the system. The implementation is expected to begin during the summer of 1979. JETS will then be tested to measure linguistic and conceptual completeness. We feel that including evaluation as an integral part of the project is Important for three reasons. First, it will help us to keep issues of closure in focus. Second, it will create a detailed record of performance, making JETS' capabilities and limitations explicit. Third, the record of evaluation will provide a basis of comparison for assessing the performance of otner natural language systems.

REFERENCES

[1] Bobrow, R. J. and J. S. Brown, "SYSTEMATIC UNDERSTANDING: Synthesis, Analysis, and Contingent Knowledge in Specialized Understanding Systems", in D. G. Bobrow and A. Collins, Representation and Understanding, Academic Press, New York, 1975, pp. 103-129.

[2] Codd, E.F., R.S. Arnold, J-M Cadiou, C.L. Cnang and N. Roussopoulos, RENDEZVOUS Version 1: an Experimental English Language Query Formulation System for Casual Users of Relational Data Bases, IBM Report RJ2144, San Jose, 1978.

[3] Finin, T., "The Semantic Interpretation of Noun-Noun Modification", Working Paper 21, Advanced Automation Group, Coordinated Science Lab, University of Illinois, 1979.

[4] Goldstein, I. P. and R. B. Roberts, "The FRL Manual", MIT AI Memo 409, 1977.

[5] Hemphill, Linda and James Rnyne, "Models for Knowledge Bases in Natural Language Query Systems", (submitted for publication).

[6] McDermott, D., "Planning and Acting", Cognitive Science, vol. 2, no. 2, pp. 71-109(1978).

[7] NALDA (Naval Air Logistics Data Analysis) System Data Requirements Determination Report, Naval Aviation Integrated Support Center, Patuxent River, Maryland, 1975.

[8] Newell, A. and H. Simon, Human Problem Solving, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972, pp. 72-75.

[9] Raphael, Bertram, THE THINKING COMPUTER Mind Inside Matter, W.H. Freeman & Co., San Francisco, 1976, pp. 146-150.

[10] Schank, R. C. and R. P. Abelson, Goals, Plans, Scripts and Understanding: an Enquiry into Human Knowledge Structures, Erlbaum Press, N.J., 1977.

[11] Tennant, H., "The PLANES Database", Working Paper 14, Advanced Automation Group, Coordinated Science Lab, University of Illinois, 1978.

[12] Tennant, H., "Experience with the Evaluation of Natural Language Question Answerers", Proc. IJCAI-79, 1979.

[13] Waltz, D. L. and B. A. Goodman, "Writing a Natural Language Data Base System", Proc. IJCAI-77, 1977, pp. 144-150.

[14] Waltz, D. L., "An English Language Question Answering System for a Large Relational Data Base", CACM, vol. 21, July, 1978, pp. 526-539.

[15] Watt, W. C., "Habitability", American Documentation, July, 1968, pp. 338-351.

[16] Woods, W. A. "A Personal View of Natural Language Understanding", SIGART Newsletter, February, 1977, pp. 17-20.

On Inheritance In Knowledge Representation

Mark S. Fox
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvannia 15213

Abstract and Introduction

This paper examines the problem of inheritance in Knowledge representation. Research in the formelIteton of Knowledge hes resulted in a small number of Knowledge classes and associated inheritance relations, e.g., INSTANCE IS-A, DBROTHERC, PERSPtCTIVE, Virtual-Copy, etc. (Brachman, 1977; Fahlman, 1977; Hayes, 1977; Levesque & Mylopoloue, 1978). The process of inheritance is defined by the procedures that access these inheritance relationa. This paper proposes that: 1) in some cases inheritance between concepts is *idiosyncratic* and does not fit predefined inheritance relations, 2) learning and discovery systems require information on how and why one concept wes *derived* from another, which again Is not represented in standard inheritance relations, and 3) current methods of specifying inheritance modification and similarity mappings are complex to specify and understand. Consequently, e decleretive approach to inheritance and similarity specificaton is presented as a solution to the above problems.

1. Specifying Idiosyncratic Inheritance

1 Experience with representing large varieties of Knowledge show that a small fraction (<10Z) escape standard representation schemes, requiring specialized "fixes" (Fahlman, 1979). The idiosyncratic nature of language and Knowledge precludes its complete structuring using a small set of classes and associated processes. We conjecture that a small set of inheritance types will not suffice. In a some cases the inheritance relation will have to be specialized to the particular concepts they relate. Hence, the inheritance link is context sensitive. Tailoring inheritance to its context requires the explication of exactly what \± *to* be transferred, excluded, added, and/or modified.

Current approaches to handling idiosyncratic inheritance rely on property classification to distinguish between properties to be inherited and those not to be inherited (e.g., structure vs assertion properties, set vs type properties). For example, a structural *properly* is inherited *among classes while assertions are not. But anomalous sub-classes may occur which do not inherit all inheritable attributes (classification is fuzzy at best). To handle anomalous sub-classes, *artificial sub-classes* are inserted between the original super-class and sub-classes. Appropriate attributes are moved from the super-class to the artificial classes. This phenomenon is called *anomaly induced class-splitting.* Typically, class-splitting is a bifurcation where one branch is a singleton
set containing the anomaly. Class-splitting increases the size[2] and complexity of the representation thus increasing search time, obfuscating possible relationships among concepts, and negating the storage and description benefits of identifying concepts with class descriptions. It seems that Knowledge classification is an art. which tries to reduce anomaly induced class-splitting.

Secondly, a Knowledge representation must represent arbitrary mappings between concepts. For example a man is like a pig if you map nose onto snout and home onto sty.

To reduce the complexity of representing idiosyncratic concepts and their inheritance relations and to allow more expressive power in describing the similarity relationships between concepts, declarative, idiosyncratic inheritance relations are introduced. Current approaches to specifyng inheritance is implicit in the representation end explicit in the procedures that manipulate the representation. Our goal is to move the explication of inheritance from the procedure to *an* inheritance relation. This enables the context-sensitive specification of inheritance modification. Thus removing the need for class-splitting and any other complexity increasing methods. Secondly, the concept of inheritance is expanded to encompass similarity mappings between concepts (e.g., analogical relations). In the following, we focus not on one particular representation but attempt to describe the mechanism in a representation independent fashion.

We propose a single unidirectional INHERITANCE (INH) relation between two concepts (A — INH —> B) with an attached INHERITANCE CONCEPT (C). The inheritance concept C explicitly states what set of information is inherited, what is excluded, what is created, and what is modified. The inheritance concept can be viewed as a label on the inheritance link specifying a set of transformations. To allow specifications of this type, a language for manipulating representations must be created. While trying not to be pinned down to a single representation, we propose the following primitives:

*This research was aponaorod by tha Defense Advancad Roaoarch Projects Agency (DOD), Arpe Ordor No 3597, monitored by tha Air Force Avionics Laboratory Contract 539815-71.0-1551. In addition, the author it supported In part by a National Research Council of Canada Post graduate Scholarship.

[2]Worst COM la 2" closes for N attribute i. e,, a discrimation not.

1. PASS: Information passed from A to B.
2. ADD: New information added to B.
3. EXCLUDE: Information in A not passed to B.
4. SUBSTITUTE: Information in A replaced by new information in B.[3]
   - -RESTRICT: Substitution results in a restriction of possible values.
   - -CONTRADICT: Substitution results in a contradiction in semantics.
   - -GENERALIZE: Substitution results in an expansion of possible values.
   - -MAP: Substitution results in an analogical replacement of information.
   - -REFINE: Replace with more detailed description.

These primitives are applied to any slot, description, link, node, and other structured primitives of a representation. They specify modifications of the physical structure of a concept to create a new concept. These primitives admit the description of arbitrary transformations among concepts. They are additions to existing representations and are to be interpreted as specifying modifications using the primitives of the particular representation language.

For example, role restriction would have an INHERITANCE link specifying all of the structure inherited using the PASS primitive, and the node being restricted by using the RESTRICT primitive. Differentiation, that is, the addition of SLOTS would use the ADD primitive. Analogical inheritance, would require the PASSing of some information, EXCLUSION of other information, ADDition of new information, and the SUBSTITUTION of information via a MAP (as found in B-structures (Moore & Newell, 1972)). Just what is the semantics of the information PASSED, ADDED, MAPPED, etc. depends on the underlying representation.

It is obvious from the current specification of the inheritance concept that a great deal of information would have to be specified by the PASS primitive. In almost all cases the PASS primitive will be the primary primitive used. The more information to PASS, i.e., the more complex the concept, the more cumbersome it is to write the inheritance concept. To alleviate this problem, we re-introduce the notion of information classes, which is the basis of current representations. The inheritance primitives would then specify that a class of representation structures, e.g., assertions, structures, etc., is to be PASSED, EXCLUDED, etc. But the inheritance concept can still refer to particular structures (node or link). Extending the classification concept to its logical conclusion, we can associate a type with the inheritance concept. For example, IS-A, DSUPERC, INSTANCE/OF, etc. can all be inheritance concept types whose inheritance definition correspond to their interpretation in other representations. But the typed inheritance concept may also specify exceptions to the type's inheritance definition. That Is, an inheritance relation could have an inheritance concept of type "is-a" (which has a standard definition composed of inheritance primitives) but is modified in the particular context by additional inheritance primitives. Since inheritance *types* are defined using inheritance primitives, new types can be defined for commonly occuring relations.

To illustrate these ideas an example is taken from zoology. Example 1 depicts a simplified representation language. A concept is divided into three parts: 1) the VIEW which specifies what the concept is related to. Each slot in the VIEW is a different inheritance concept, 2) the META-CORPUS which specifies wholistic (set, type) information, and 3) the CORPUS which specifies structure information. Example 2 represents a partial description of a *mammal (a denotes a concept). To add *platypus to the set of •mammals requires concept-splitting (the platypus is an exception to the mammal specification, It lays eggs): create a*onotreme with «egg-laying value for *birth-process (ex. 4), and another concept with *live-birth value for ebirth-process. The alternative approach taken in this *paper* results in ex. 5. The *platypus has an inheritance concept of type *!S-A, but the definition of *is-A is overidden by the CONTRADICT primitive specifying that the aplatypus lays eggs instead of live birth. The REFINE primitive is used to replace the *head slot with a *bill and askull slot.

2. Specifying Derivative Relations

A second motivation for describing the relationship between two concepts explicitly is to allow learning and discovery systems to analyse how concepts *are* derived from other concepts.

The goal of learning and discovery systems is to generate new concepts via specialization, generalization , or analogy. In particular, these systems *search* for derivations that are "interesting", where interesting is defined by some heuristic metric. To properly focus search, the method for deriving one concept from another must be recorded. This information is used to analyse how a concept was derived and what should be done to derive a different but related concept. In Lenat*s AM system (1976), a set of heuristics were used to decide how to alter (extend) existing concepts to derive new *and* interesting concepts. A similar approach is used by Fox (1978) to decide how to specialize concepts to create new and interesting concept hierarchies. In Winston's system (1978), transfer frames are hypotheses for what slots to transfer between concepts; heuristics are used for deciding candidate slots. In each system, there exists a set of actions whose application defines a space of new concepts. The action(s) chosen and reason(s) for the choice(s) are important pieces of information used by these systems in deciding, how to extend concepts. The INHERITANCE concept should store it.

We further define each of the INHERITANCE primitives as having the following three attibutes: 1) SET 2) VALUE 3) REASONS. The SET attribute specifies the information the primitive could act on. That is PASS, ADD, EXCLUDE, MAP, RESTRICT, GENERALIZE, or REFINE. The VALUE specifies the actual information chosen from the SET. REASONS specify what decisions led to the choice. Of the three attributes, REASONS is the least defined as it is dependent on both representation *and* Inference mechanisms. If a PASS was to be specified, then the SET attribute of the PASS would

[3]It should bo noted that substitute is the combination of exclude ond add. Wo Include it ■■ ■ primitive because separeting It into oxcludo ond odd would lose the information that there is a contingency, that one structure replace another

list all the structures, e.g., DATTRS the PASS could be applied to. The VALUE attribute would denote the actual structure chosen, the REASON attribute would "explain" why the value was chosen from the SET.

As pointed out earlier, the SUBSTITUTE primitive is equivalent to an EXCLUDE and ADD. Hence, it has two sets of attributes. (SET_e, $VALUE_e$, $REASON_e$) describe the information to be replaced, and (SET_a, $VALUE_a$, $REASON_a$) describes the information actually substituted.

By specifying each of those attributes for each of the inheritance primitives, it is hoped that more information will be made available for learning and discovery systems to make decisions intelligently. By no means are these attributes complete. Their purpose is to focus attention on the types of information needed in inheritance specifications.

Example 6 illustrates how the SET, VALUE and REASON primitives are specified in ADD. In this example, a •platypus is specialized as a *purple-platypus by choosing a color out of the set of possible colors.

3. Conclusion

Inheritance is the primary, most powerful representation primitive available in Knowledge representations. The explication of representation semantics has led to the creation of many types of inheritance links. The semantics of these inheritance types are defined by the procedures that manipulate the representation. Two problems arise in classifying inheritance relations. First, some inheritance relations are idiosyncratic and do not conform to popular classifications. Second, learning and discovery systems require an explication of how concepts are related, in particular what information is inherited, modified, added and/or excluded, and upon what information were these changes based. To adequately deal with these problems, the semantics of inheritance must be moved from the procedures to the representation. This view led to the creation of a general inheritance relation with an associated inheritance concept. The inheritance concept explicitly defines what information is inherited by concept, and what additions, deletions and substitutions are made. It also describes the set of choices available in making the alterations and why a particular alteration was chosen.

4. References

1. Brachman R.J., (1977), "A Structural Paradigm for Representing Knowledge," (Ph.D. Thesis), Harvard University, May 1977.

2. Fahlman S.E., (1977), "A System for Representing and Using Real-World Knowledge," (Ph.D. Thesis), Artificial Intelligence Laboratory, MIT, AI-TR-450.

3. Fahlman S.E., (1979), Private Communication.

4. Fox M.S., (1978), "Knowledge Structuring: An Overview," Proceedings of the Second Conference of the Canadian Society for Computational Studies of Intelligence, Toronto Ont., July 1978.

5. Hayes P.J., (1977), "On Semantic Nets, Frames and Associations," Proceedings of the Fifth Internatonal Joint Conference on Artificial Intelligence, Cambridge MA, Aug. 1977.

6. Lenat D.B., (1976), "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search,", (Ph.D. Thesis), Computer Science Dept. Stanford U., A1M-286.

7. Levesque H, and J. Mylopoulos (1978), "A Procedural Semantics for Semantic Networks," AI MEMO 78-1, Dept. of Computer Science, University of Toronto.

8. Moore J., and A. Newell, (1973), "How can Merlin Understand," In L. Gregg (ed.), Knowledge and Cognition, Potomac MD.: Lawrence Erlbaum Associates.

9. Winston P.H (1978), "Learning by Creatifying Transfer Frames," Artificial Intelligence. Vol. 10, pp. 147-172.

Examples

1. {*Concept
   View: <view-slots>
   Meta-corpus: <meta-corpus-slots>
   Corpus: <corpus-slots>}
2. {*Mammal
   Corpus:
   »Nursing-Method: aBreast
   *Birth-Process: alive
   *color:
   •Head:}
3. {*Mammal
   Corpus:
   •Nursing-Method: •Breast}
4. {*Monotreme
   View:
   *Is-a: aMammal
   Corpus:
   *Birth-Process: *Egg-laying}
5. {*Platypus
   View:
   *Is-A: aMammal
   (Corpus:
   (CONTRADICT DESCRIPTION OF *Birth-Process SLOT
   WITH aEgg-laying)
   (REFINE SLOT *Head TO SLOT *Bill AND SLOT *Skull))}
6. {*Purple-Platypus
   View:
   •Is-a: aPlatypus
   (Corpus: (ADD VALUE ePurple TO DESCRIPTION OF
   SLOT *Color FROM SET {ared, *yellow, *purple}
   FOR REASON "environment is purple"))}

KNOWLEDGE-BASED EXPERIMENT DESIGN IN MOLECULAR GENETICS*

Peter Friedland
Heuristic Programming Project
Computer Science Department
Stanford University
Stanford, CA 94305

A system to produce practical experimental plans in the domain of molecular genetics is in development. It makes use of a large, expert-entered knowledge base containing object-level and strategy knowledge about the domain. The system works by refining general strategies or "skeletal plans," to the level of detail desired by the system user.

1. INTRODUCTION

A large percentage of the experiments carried out in a genetic laboratory fall into the class of structural analysis. The goal of these experiments is to learn something about the physical structure of a particular type of DNA molecule. The experimenter uses his knowledge of fundamental molecular biology, laboratory methods, and basic analytic strategies, combined with what is already known about the structures under study, in order to design an appropriate experiment.

This paper will describe and illustrate one of the methods scientists employ in the experiment design process. It will then discuss a heuristic program which implements this method using a knowledge base entered directly by domain experts.

2. A THEORY OF EXPERIMENT DESIGN

2.1 A Method Scientists Employ

Much of the intellectual effort of experiment design can be characterized as an "encyclopedia-dictionary" loop. Faced with an analytic task, the scientist first picks an appropriate strategy from his collected personal knowledge, from the knowledge of colleagues, or from the literature—his "encyclopedia." The strategy consists of an abstracted or "skeletal" plan which has proven useful for this type of experiment in the past. It may be as general as "label the structural feature you are looking for and then look for the label," or as specific as a detailed laboratory protocol for using an electron microscope to locate loops.

*This research is supported by NSF MCS 78-02777.

The experimenter than instantiates each of the steps of his skeletal plan with a specific laboratory tool. He does this by first referring to his encyclopedia again, looking for information to help him chose which of the possible techniques Is most suitable to the particular problem conditions. Three types of information are useful in making this selection. First, can the technique accomplish the goal of the plan-step? For example, if the specific plan-step were to degrade (break into small pieces) all single-stranded DNA, which techniques are known to have accomplished that task on previous problems. Second, will the technique work on the particular molecule under the particular laboratory conditions? For example, a given enzyme may be able to degrade single-stranded DNA, but it may fail on this particular problem because the molecule is circular or because inhibitors are present in the sample. Finally, which of the remaining techniques is best? This involves using a heuristic based on parameters like convenience, reliability, accuracy, time, and cost.

The refinement process may occur in several hierarchical steps. For the example of degrading all single-stranded material on the molecule, the experimenter might first decide to use enzymatic means of digestion, then decide to use a single-strand specific endonuclease, and finally choose SI endonuclease. The advantage of this hierarchical approach is that the experimenter does not have to consider all techniques as independent, instance-level entities. Instead of making a choice among, say, one hundred different degradation methods, he can first make use of heuristics to choose between two basic classes, enzymatic and nonenzymatic. After selecting enzymatic, he decides between endonucleases and exonucleases, and then between double-strand-specific and single-strand-specific endonucleases. His final decision is among only half a dozen or so particu-

lar enzymes.

Sometimes the experiment designer will not be able to locate a suitable instantiation of a skeletal plan step. Then he refers to his "dictionary," attempting to understand what is the purpose of the step so that he can break it up into its component parts. For example, suppose the experimenter cannot find a workable-instantiation for a denaturation step. He uses his dictionary to find that denaturation really means breaking the hydrogen bonds between the two strands of DNA. He then attempts to find a new skeletal plan for the subgoal of hydrogen bond breaking and repeats the experiment design process for that skeletal plan. The idea is that he is now treating denaturation as the goal of an experiment design problem, i.e. that subgoals are generated when instantiation fails.

2.2 An Example of Plan Instantiation

An example of how a single, general skeletal plan can lead to the design of different experiments can be seen in the problem of sequencing. Sequencing is the process of determining the location on a molecule of specific sites, usually either bases or specific strings of bases. A general strategy for sequencing is as follows:

1. Label one end of the structure.
2. Treat the structure so that on the average one site is cut per molecule.
3. Determine the length of the labeled fragments.

This strategy has been instantiated in different ways for the two problems of total base sequencing and restriction site mapping. For total base sequencing the problem is to determine which of the four bases, A, G, C, or T, is present at each position on the molecule. The first step in the plan is refined to labeling with radioactive Phosphorus 32, the current best method. The second step is instantiated either by a series of chemical reactions or by a carefully controlled enzymatic reaction. The final step is accomplished by separating the fragments by electrophoresis and locating the labels with autoradiography. The process is known as Maxam-Gilbert sequencing (1) in the chemical case and Sanger-Coulson sequencing (2) in the enzymatic case. Both are thought of as major advances, decreasing by over an order of magnitude the time required to determine the base sequence of moderately complex molecule from the previous best method.

For restriction site mapping, the goal is to locate specific four-, five-, or six-base strings on a structure. (These are the active cutting site for enzymes known as restriction enzymes). The first and third steps of the skeletal plan are instantiated as above; the second step is refined to cutting with a partial digest of the desired restriction enzyme (3).

2.3 Relation to other AT Work

The basic idea of maintaining a collection of abstracted plans relates closely to Schank's concept of scripts in understanding natural language (A). The human scientist, in the typical experiment design case, docs not plan from scratch; instead he makes use of a library of knowledge about how classes of laboratory methods fit together to accomplish certain design goals. Maintaining a collection *of* parameterized plans was first discussed in ABSTRIPS with the name of MACROPS (5). The benefits of hierarchical plan refinement have evolved from ABSTR1PS (6) planning work through the robot planning systems of many other including Sacerdoti (7).

3. IMPLEMENTATION

3.1 The Experiment Design HeurIstic

The planning method has been implemented in a straightforward manner. An experimenter describes his problem by telling what he already knows about the structure and its surrounding environment and providing a goal for the experiment. The system searches its library of strategies for skeletal plans that have proven useful for satisfying the given goal or ones like it in the past.

The experiment design system then refines a skeletal plan by working through a detailed taxonomy of laboratory techniques. It uses domain-specific rules to carry out the three-fold selection process described above: goal-oriented rules to choose techniques which can accomplish the desired task, rules relating technique parameters to molecular and sample conditions to eliminate those techniques which will not "go" for the given problem, and heuristics for making a final selection based on the relative convenience, reliability, accuracy, and cost of the competing techniques.

The system also makes use of rules which point out possible interactions between plan-steps, When it encounters an applicable rule of this sort, it postpones further refinement of the step until enough information is known to minimize an unfavorable interaction or (the more common case) maximize the favorable effects of

an interaction.

If an appropriate technique cannot be found to instantiate any plan-step, the system recurs as described above. A subgoal is created to design an experimental plan to provide a new method for accomplishing the plan step.

3.2 The Knowledge Base

All three of the processes described above—problem description, skeletal plan finding, and plan-step refinement—are heavily dependent upon a detsiled knowledge base. This knowledge-base is being constructed by molecular biologists and biochemists through the facilities of the Unit Package developed at Stanford (8).

In the problem description phase of the process the user is guided in creating an instance of a unit for a DNA structure. The prototypical DHA structure unit contains slots for all the possible kinds of information that could be known for DNA structures. It also contains rules for checking consistency of information and for filling in additional slots from given ones—users don't always realize how much they really know about their structures. The user then describes other laboratory conditions by creating an Instance of a prototypical sample-conditions unit.

The skeletal plans are each described as units in the knowledge base. The units contain a slot listing the possible general and specific utilities of the plan, as well as a step-by-step description of the plan. For example, the skeletal plan for locating inverse complementary regions on all types of structures Is:

1. DENATURE
2. RENATURE RAPIDLY
3. DEGRADE SINGLE-STRANDED
4. DETECT LARGE-MOLECULAR-WEIGHT DNA

The bulk of the knowledge base consists of the descriptions of laboratory techniques used by the plan-refinement part of the experiment design system. The technique descriptions are arranged in a tree with the major branches being sequence-analysis, modification, separation and detection. Each class of techniques or individual technique is described in a unit containing declarative Information about utility, optimal laboratory conditions, allowable substrates and the like, as well as the selection rules that are relevant to that technique.

The knowledge base was constructed by several Stanford molecular biologists using the interactive facilities of the Unit Package. It currently contains approximately 25 skeletal plan units and 200 laboratory technique units. There are about 250 rules for plan-step instantiation, consistency checking, and modelling the consequences of modification techniques.

A. RESULTS AND CONCLUSIONS

The experiment design system is currently operational and seems to give satisfactory (i.e., plausible) results for a variety of analysis experiments. The particular experimental problems of finding self-complementary regions (inverted repeats), and mapping intervening sequences between genes have received the most attention. As the knowledge base continues to grow in scope and complexity, the range of experimental plans produced should also increase.

An automated experiment design system will serve two major functions: throughness and variety. Some instantiations of experimental strategies are missed either because a scientist is not aware of all relevant technqieus or because of prejudices toward certain techniques. The design system will be as thorough as its knowledge base—the product of many experts's efforts. The system will promote variety, as it is able to combine the best strategies of different experimenters through the recursive encyclopedia-dictionary process.

REFERENCES

(1) Maxam, A. and Gilbert, W., PNAS, 74, 1977, 560-564.

(2) Sanger, F. and Coulson, A., PNAS, 74, 1977, 5463-5466.

(3) Smith, W. and Birnsteil, M., Nucleic Acids Research, 3, 1976, 2387-2398.

(4) Schank, R. and Abelson, R., Scripts, Plans, Goals and Understanding, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1977.

(5) Fikes, R., Hart, P. and Nilsson, N., "Learning and Executing Generalized Robot Plans," Artificial Intelligence, 3, 1972, 251-288.

(6) Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces," in Proc. IJCAI-73, 412-422.

(7) Sacerdoti, E., "A Structure for Plans and Behavior," Ph.D. Thesis, SRI Artificial Intelligence Center Technical Note 109, 1975.

(8) Stefik, M., "An Examination of a Frame-Structured Representation System," HPP Report HPP-78-13, 1978.

QUANTITATIVE EVALUATION OF TRANSMISSION OF MEANING

Hiroya Fujisaki
Department of Electrical Engineering
Faculty of Engineering
University of Tokyo
Bunkyo-ku, Tokyo, 113 Japan

Yasuhiro Katagiri
Department of Electrical Engineering
Faculty of Engineering
University of Tokyo
Bunkyo-ku, Tokyo, 113 Japan

As a step toward quantifying the processes whereby meaning is transmitted by language, an experiment was designed to facilitate observation of the process of verbalization by the sender as well as the process of comprehension by the receiver. Both sender and receiver characteristics were measured experimentally by confining the subject matter to ages and by restricting the available vocabulary to a small set of nouns. A model was then constructed for the entire process through which meaning is transmitted by a word, clarifying various causes of ambiguity and inaccuracy of transmission. The whole process was evaluated in terms of the amount of information and the r.m.s. error of transmission.

1. INTRODUCTION

Language is obviously an important medium of human thought and communication, and elucidation of its essential characteristics is indispensable not only for understanding the human processes of thought and communication, but also for machine processing of linguistic information. Most of the studies on language, however, have been concentrated on the nature of the language itself as a system of codes, and little attention has been paid to the relationship between the linguistic code and the information conveyed by the code [1].

The present study deals with the communicative roles of language between a sender and a receiver, and experimentally analyzes the sender's process of conversion from information to word as well as the receiver's process of conversion from word to information. The process of transmission of meaning is then formulated and quantitatively evaluated in terms of the amount of transmitted information and the accuracy of transmission [2],

2. PROCESSES IN TRANSMISSION OF MEANING

We deal here with a situation in which a physically definable stimulus is converted by a subject *(sender)* into a linguistic expression (a noun in this case) and is received by another subject *(receiver)*. The input stimulus is first perceived by the sender and then expressed as a noun. The combined characteristics of perception and verbalization by the sender can be measured as the input-output characteristics, and will be referred to as the *coding* characteristics. On the other hand, the communication Is generally terminated by the receiver's process of comprehension. In order to render the receiver's percept into a directly observable form, however, we add here another process in which the receiver reproduces the stimulus which he may find most appropriate for the received linguistic message. Thus the combined characteristics of comprehension and reproduction by the receiver can be measured as the input-output characteristics, and will be referred to as the *decoding* characteristics. The processes involved in this verbal communication situation is schematically shown in Fig. 1.

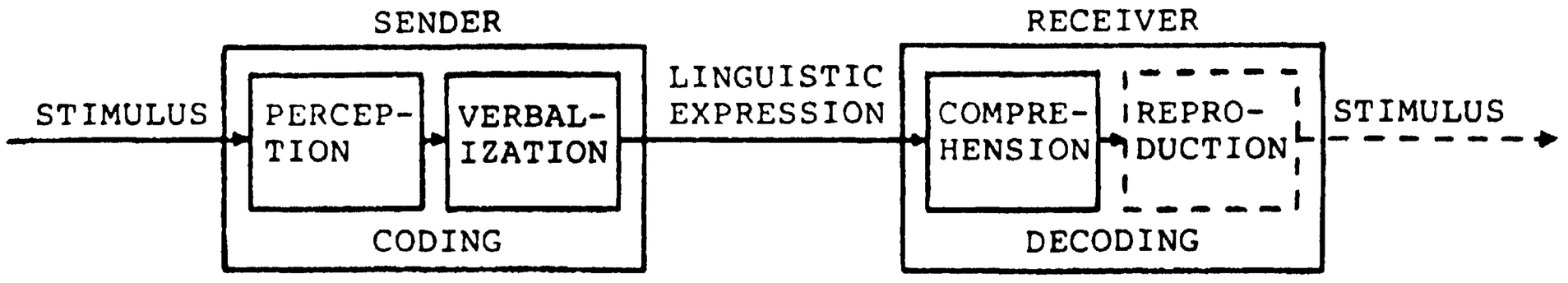


Fig. 1. Processes in transmission of meaning by language.

3. CODING AND DECODING CHARACTERISTICS

AS an example of experimental determination of the coding and decoding characteristics, an investigation was conducted on the relationship between the age of a person and the nouns of Japanese commonly used to designate the age. These nouns were selected such that they were nearly complementary with each other in designating age and uniform in other aspects. The major part of the following experiments was conducted by using the five-word vocabulary: "yo-nen", "sho-nen", "sei-nen", "so-nen", and "ro-nen", corresponding roughly to the English "childhood", "boyhood", "youth", "manhood", and "old age", respectively. The vocabulary size was varied, however, in studying the effect of its change on transmission accuracy. The method of constant stimuli was used to measure the coding and decoding characteristics separately on each of the nine subjects. Because of individual differences, the data from each subject had to be analyzed separately.

Figure 2 shows the coding characteristics of one subject as the probability with which he assigns a noun against a given age. In the vicinity of a category boundary, each of the probability distribution functions can be approximated by a Gaussian distribution function with a mean 0 and a standard deviation p, to be defined here respectively as the coding boundary and the coding accuracy, suggesting that the selection of a particular noun is based on a categorical judgment whose threshold fluctuates due to a number of psychological factors. Thus the coding characteristics of a subject can be represented by the set of 0's and p's for all the category boundaries.

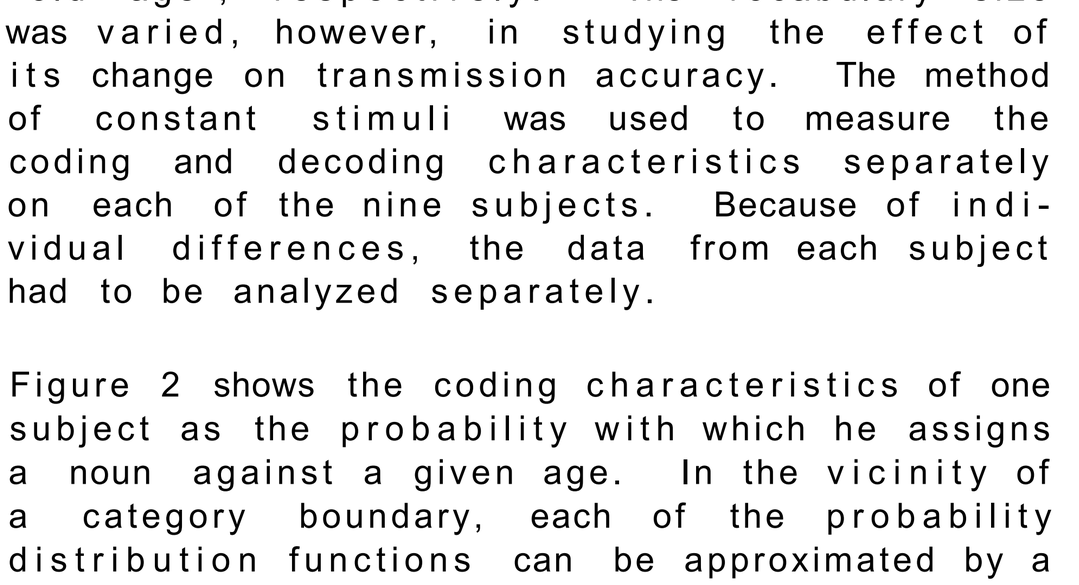


Fig. 2. An example of coding characteristics.

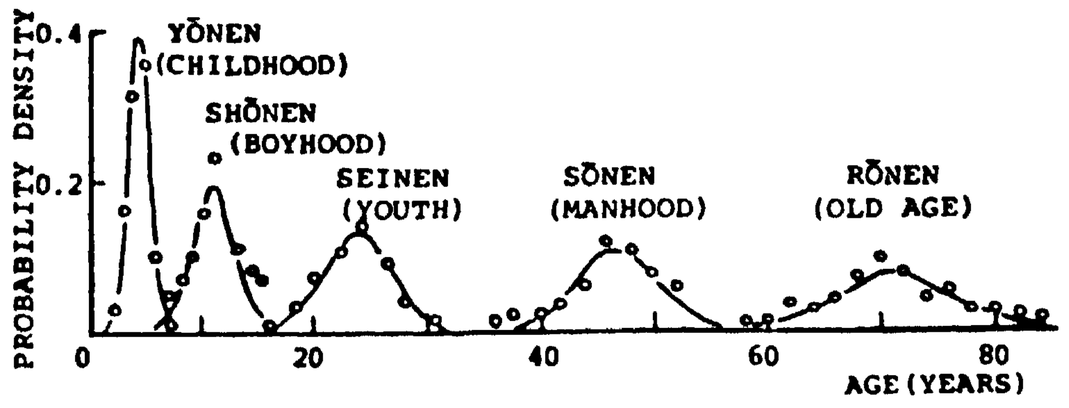


Fig. 3. An example of decoding characteristics.

Figure 3 shows the decoding characteristics of the same subject as a set of five probability density functions, each representing the probability density that a certain age is reproduced against a given noun. In this case, each of the density functions can be approximated by a Gaussian density function with a mean p and a standard deviation o, to be defined here respectively as the decoding reference and the decoding accuracy, suggesting that the reproduction of a particular age is based on a reference which is perturbed by a number of psychological factors. Thus the decoding characteristics of a subject can be represented by the set of u's and a's for all the categories (nouns).

4. A MODEL OF SEMANTIC INFORMATION TRANSMISSION

The results of the foregoing experiments lead to a model for the processes of semantic information transmission shown in Fig. 4, The coding process at the sender can be considered as quantization followed by code assignment. The quantization is probabilistic since each threshold 0_i is perturbed by a Gaussian noise m_i. On the other hand, the decoding process at the receiver can be considered as biased reproduction of the quantization levels. The bias B_i represents the discrepancy between the center of the ith coding step $(0_i + 0_{i-1})/2$ of the sender and the ith decoding reference u_i of the receiver. The reproduction is also probabilistic since each decoding reference Is perturbed by a Gaussian noise n_i. The three variables x, w, and y in Fig. 4 respectively represent the meaning implied by the sender, the code word adopted for transmission, and the meaning recovered by the receiver. The entire process as a communication channel can be characterized by the conditional probability $q(y|x)$.

These formulations allow one to evaluate the mutual information R (in bit/word) and the r.m.s. transmission error E (in units of age) for a given probability density p(x) of the stimulus x

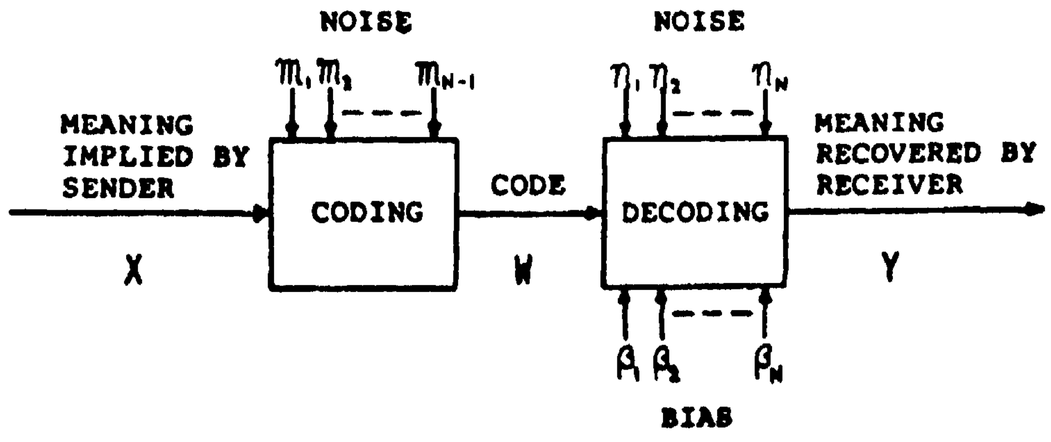


Fig. 4. Formulation of the process of semantic information transmission.

presented to a sender. They are given respectively by

$$R = \int\int p(x)q(y|x)\log[q(y|x)/q(y)]\,dx\,dy,$$

where

$$q(y) = \int p(x)q(y|x)\,dx,$$

and

$$E = \left[\int\int p(x)q(y|x)(y-x)^2\,dx\,dy\right]^{1/2}.$$

The mutual information R indicates the requirement on the lower bound of the channel capacity for semantic information transmission necessary to achieve the accuracy of transmission indicated by the r.m.s. error.

These quantities were calculated from the measured data of coding and decoding characteristics of three subjects, obtained for the vocabulary size of two, five, and eight. The probability density p(x) of the stimuli was assumed to be uniform over the range [0, 80], and the calculation was made for all possible sender-receiver pairs. The results are shown in Fig. 5, where a small circle indicates the mean of all sender-receiver combinations, and a vertical line segment indicates the range of distribution. The results indicate that both the accuracy and the transmitted information increase with the vocabulary size, but also indicate that there is a limit to the accuracy attainable by increasing the vocabulary size. Thus there seems to exist an optimum size of vocabulary from the point of view of transmission efficiency.

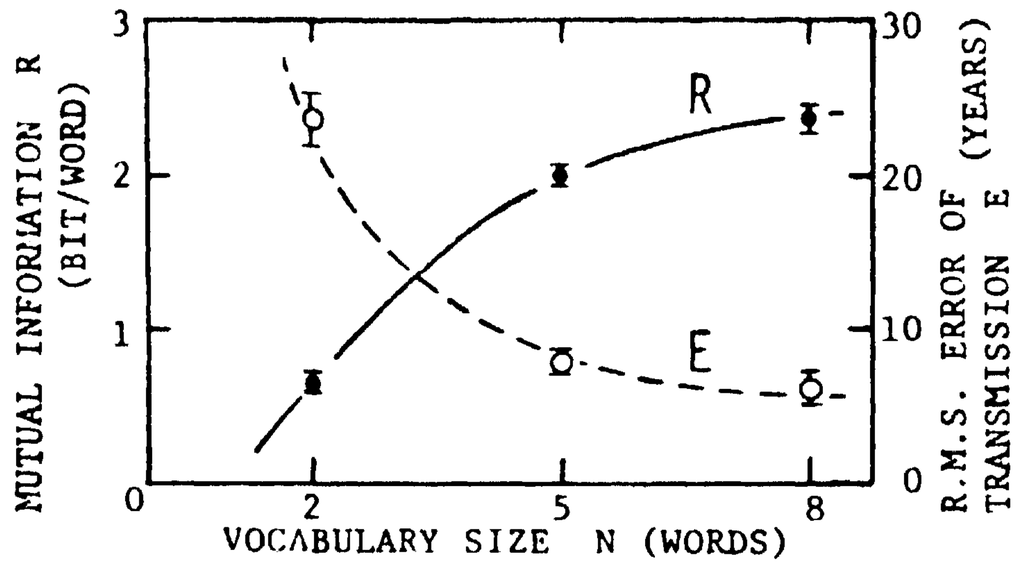


Fig. 5. Mutual information and r.m.s. error of transmission versus vocabulary size.

5. DISCUSSION AND CONCLUSIONS

The probabilistic nature of language use was earlier pointed out by Lenneberg and his co-workers [3,4], Due to the use of unrestricted vocabulary and pooling of data from many subjects, however, these authors did not attain a clear insight into the underlying process of coding in a language user. The problem is also treated by 7,adeh (5] in connection with the concept of "fuzziness." Despite the conceptual distinction made by Zadeh between probability and the membership function, the present authors consider that the membership function is equivalent to the probability distribution in the coding characteristics. It should also be noted that these earlier studies did not separate the two kinds of indeterminacy in language use, i.e., the indeterminacy of coding and that of decoding.

The present study was based on the clear separation of the two processes in semantic information transmission, and developed experimental techniques to observe both the process of verbalization by the sender and the process of language comprehension by the receiver. The characteristics of these processes were measured and formulated. Based on these results, a model was constructed for the process of semantic information transmission by means of a word, and a method was also shown to evaluate the process in terms of the ambiguity and the accuracy of transmission.

Although the results shown in this paper are quite limited, the techniques described here are applicable to a wide range of problems of language use, such as the quantitative analysis of individual, contextual, dialectal, and language differences in semantic information transmission.

ACKNOWLEDGMENTS

The research reported here was supported by a Grant-in-Aid for Scientific Research (No. 310708) from Japanese Ministry of Education.

REFERENCES

[1] Shannon, C. E. and W. Weaver: The Matematical Theory *of* Communication, Univ. of Illinois Press, 1949.

[21 Fujisaki, H., K. Hirose and Y. Katagiri: Transmission of Meaning by Language, Annual Bulletine, Research Institute of Logopedics and Phoniatrics Univ. of Tokyo, 12, 127-140, 1978".

[31 Brown, R. VJ. and E. H. Lenneberg: A Study in Language and Cognition, J. Abnormal and Social Psychology, 49, 454-462, 1954.

[A] Lenneberg, E. H.: Biological Foundations of Language, John Wiley & Sons, 1967.

[5] Zadeh, L. A.: Fuzzy Sets, Informatlon and Control, 8, 338-353, 1965.

SELF-ORGANIZATION OF A NEURAL NETWORK WHICH GIVES POSITION-INVARIANT RESPONSE

Kunihiko Fukushima

NHK Broadcasting Science Research Laboratories
1-10-11, Kinuta, Setagaya, Tokyo 157, Japan

In this paper, I propose a new algorithm for self-organizing a multilayered neural network which has an ability to recognize patterns based on the geometrical similarity of their shapes. This network, whose nickname is ''neo-cognitron", has a structure similar to the hierarchy model of the visual nervous system proposed by Hubel and Wiesel. The network consists of a photoreceptor layer followed by a cascade connection of a number of modular structures, each of which is composed of two layers of cells connected in a cascade. The first layer of each module consists of "S-cells", which show characteristics similar to simple cells or lower order hypercomplex cells, and the second layer consists of "C-cells" similar to complex cells or higher order hypercomplex cells. The imput synapses to each S-cell have plasticity and are modifiable. The network has an ability of unsupervised learning: We don't need any "teacher" during the process of self-organization, and it is only needed to present a set of stimulus patterns repeatedly to the input layer. The network has been simulated on a digital computer. After completion of self-organization, the stimulus patterns has become to elicit their own response from the last C-cell layer. That is, the response of the last C-cell layer changes without fail, if a stimulus patterns of a different category is presented to the input layer. The response of that layer, however, is not affected by the pattern's position at all. Neither is it affected by a certain amount of changes of the pattern's shape or size.

1. INTRODUCTION

In this paper, I propose a new algorithm for self-organizing a neural network which has an ability to recognize stimulus patterns based on the geometrical similarity of their shapes regardless of their positions and slight distortions of their shapes.

This network is given a nickname "neocognitron", because it is a further extension of the "cognitron", which is a self-organizing multi-layered neural network proposed by the author before [1]. Incidentally, the conventional cognitron also had an ability to recognize patterns, but its response was dependent upon the position of the stimulus pattern. That is, the same patterns which were presented at different positions were taken as different patterns by the conventional cognitron.

The neocognitron has also a multilayered structure. It has an ability of unsupervised learning: We don't need any "teacher" during the process of self-organization, and it is only needed to present a set of stimulus patterns repeatedly to the input layer. After completion of self-organization, the network acquires a structure similar to the hierarchy model of the visual nervous system proposed by Hubel and Wiesel [2], [3]. The response of the cells of the last layer of the network is dependent only upon the shape of the stimulus pattern, and is not affected by the position of pattern presen-tation. That is, the network has an ability to recognize patterns without affected by the position of the patterns.

2. STURUCTURE OF THE NETWORK

The network which is proposed here consists of a photoreceptor layer followed by a cascade-connection of a number of madular structures, each of which is composed of two layers of cells connected in a cascade. The first layer of each module consists of "S-cells", which correspond to simple cells or lower order hypercomplex cells according to the classification of Hubel and Wiwsel. The second layer of the module consists of "C-cells", which correspond to complex cells or higher order hypercomplex cells.

The cells in each layer are divided into many subgroups according to the optimum stimulus features of their receptive fields. Since the cells in each subgroup are set in a two dimensional array, we call the subgroup as a "cell-plane". It is assumed that all the cells in a single cell-plane have the same spatial distribution of the input synaptic connections, and only the position of the presynaptic cells are shifted in parallel from cell to cell. Hence, all the cells in a single cell-plane have receptive fields of the same functions, but at different positions.

Let $u_{S\ell}(k_\ell, n)$ be the output of an S-cell in the k_ℓ-th cell-plane in the ℓ-th module, and let $u_{C\ell}(k_\ell, n)$ be the output of a C-cell in that module, where n is two-dimensional co-ordinates representing the position of these cell's receptive fields in the input layer U_0. Besides these excitatory cells, we have inhibitory cells $v_{S\ell}(n)$ and $v_{C\ell}(n)$ in the module.

The output of these cells are represented as follows:

$$u_{S\ell}(k_\ell, n) = r_\ell \cdot \varphi\left[\frac{1+\sum_{k_{\ell-1}=1}^{K_{\ell-1}} \sum_{\nu \in S_\ell} a_\ell(k_{\ell-1}, \nu, k_\ell)\cdot u_{C\ell-1}(k_{\ell-1}, n+\nu)}{1+\frac{2r_\ell}{r_\ell}\cdot b_\ell(k_\ell)\cdot v_{C\ell-1}(n)} - 1\right] \quad (1)$$

$$\text{where} \quad \varphi[x] = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \quad (2)$$

$$v_{C\ell-1}(n) = \sqrt{\sum_{k_{\ell-1}=1}^{K_{\ell-1}} \sum_{\nu \in S_\ell} c_{\ell-1}(\nu)\cdot u^2_{C\ell-1}(k_{\ell-1}, n+\nu)} \quad (3)$$

$$u_{C\ell}(k_\ell, n) = \psi\left[\frac{1+\sum_{\nu \in D_\ell} d_\ell(\nu)\cdot u_{S\ell}(k_\ell, n+\nu)}{1+v_{S\ell}(n)} - 1\right] \quad (4)$$

$$\text{where} \quad \psi[x] = \varphi\left[\frac{x}{1+x}\right] \quad (5)$$

$$v_{S\ell}(n) = \frac{1}{K_\ell}\sum_{k_\ell=1}^{K_\ell} \sum_{\nu \in D_\ell} d_\ell(\nu)\, u_{S\ell}(k_\ell, n+\nu) \quad (6)$$

Where, $a_\ell(k_{\ell-1}, \nu, k_\ell)$ and $b_\ell(k_\ell)$ represent the strength (efficiency) of the modifiable excitatory and inhibitory synapses, respectively. Parameter $c_{\ell-1}(\nu)$ and $d_\ell(\nu)$ represent the strength of the unmodifiable excitatory synapses. Positive parameter r_ℓ controls the intensity of the inhibitory input, and the large value of r_ℓ gives higher selectivity of the cell's response.

Let $u_0(n)$ be the output of a cell of the input layer U_0. In the above equations, it is assumed that $u_{C\ell-1}(k_{\ell-1}, n)$ stands for $u_0(n)$ in case of ℓ=1. Incidentally, $K_{\ell-1}$=1 for ℓ=1.

The inhibitory cell $v_{C\ell-1}(n)$ has an r.m.s.-type input-to-output characteristic as indicated by eq. (3). The employment of r.m.s.-type cells is useful for endowing the network with an ability to make reasonable evaluation of the similarity between the stimulus patterns. Its usefulness was analytically proved for a cognitron [4], and the same discussion can also be applied to our new network.

3. THE SELF-ORGANIZATION OF THE NETWORK

One of the fundamental hypotheses employed in this network is the assumption that all the S-cells in a single cell-plane have afferent synaptic connections of identical spatial distribution, and that only the position of the presynaptic cells shift in parallel in accordance with the S-cell's position. Therefore, in eq. (1), the modifiable synaptic connection $a_\ell(k_{\ell-1}, \nu, k_\ell)$ is assumed to be independent of the position n of the presynaptic cell (k_ℓ, n).

During the process of self-organization, several "representative" S-cells are selected every time when a stimulus pattern is presented. Here, we assume a certain mechanism which prohibit the selection of more than one representative from a single S-cell-plane. The detailed procedure for selecting the representatives are given later on

Suppose a representative is selected from an S-cell-plane. The afferent synapses to the representative S-cells are reinforced with the same manner as in the case of r.m.s.-type cognitron [4], and the reinforcement of the synapses of all the other S-cells in that cell-plane follows those of their representative. These relations can be quantitatively expressed as follows.

Let cell $u_{S\ell}(\hat{k}_\ell, \hat{n})$ be selected as a representative. The modifiable synapses $a_\ell(k_{\ell-1}, \nu, \hat{k}_\ell)$ and $b_\ell(\hat{k}_\ell)$, which converge to the cells of the k_ℓ-th cell-plane, are reinforced by the amount shown below:

$$\Delta a_\ell(k_{\ell-1}, \nu, \hat{k}_\ell) = q_\ell \cdot c_{\ell-1}(\nu)\cdot u_{C\ell-1}(k_{\ell-1}, \hat{n}+\nu) \quad (7)$$

$$\Delta b_\ell(\hat{k}_\ell) = (q_\ell/2)\cdot v_{C\ell-1}(\hat{n}) \quad (8)$$

where q_ℓ is a positive constant specifying the speed of reinforcement.

The cells in the S-cell-plane from which no representative is selected, however, do not have their input synapses reinforced at all.

The representatives are chosen by the following procedures. At first, we watch a group of

S-cells in a layer whose receptive fields are situated within a certain small area in the input layer. Each group contains cells from all the S-cell-planes. We call the group as an "S-column". There are a lot of such S-columns in a single layer. From each S-column, the cell which has yielded the largest output is chosen as a candidate for the representatives. In case only one candidate appears from an S-cell-plane, the candidate is unconditionally selected as the representative from that S-cell-plane. If more than two candidates appear from an S-cell-plane, however, only the one which has yielded the largest output among them is selected as the representative from that S-cell-plane. If no candidate appears from an S-cell-plane, no representative is selected from that S-cell-plane.

As it is seen from these discussions, if we consider that a single excitatory cell in the conventional cognitron [1] coresponds to a single S-cell-plane in the neocognitron, the procedures of reinforcement in the both systems are analogous to each other.

4. COMPUTER SIMULATION

The neural network proposed here has been simulated on a digital computer. In the computer simulation, we consider a seven layered network: $U_0 \rightarrow U_{S1} \rightarrow U_{C1} \rightarrow U_{S2} \rightarrow U_{C2} \rightarrow U_{S3} \rightarrow U_{C3}$. That is, the network has three stages of modular structures preceded by an input layer. The number of cell-planes in each layer is 24 for all the layers except U_0. The numbers of cells in these seven layers are 16*16, 16*16*24, 10*10*24, 8*8*24, 6*6*24, 2*2*24, and 24 from the front. In the last layer U_{c3}, each cell-plane contains only one cell, and that cell's receptive field covers the whole area of the input layer U_0.

We have presented five stimulus patterns, which are shown in the leftmost column in Fig. 1, repeatedly to the input layer U_0 of the network. The positions of presentation of these stimulus patterns have been randomely shifted at every presentation.

After a certain number of pattern presentations, each stimulus pattern has become to elicit an output only from one U_{c3}-cell, and conversely, this U_{c3}-cell has become selectively responsive only to that stimulus pattern. That is, none of the U_{c3}-cells responds to more than one stimulus pattern. It has been confirmed that the response of layer *Ucs* is not affected by the position of a stimulus pattern at all. Neither is it affected by the slight change of the shape or the size of the stimulus pattern.

Fig. 1 shows some examples of the stimulus patterns which the neocognitron has correctly recognized: All the stimulus patterns in each row of Fig. 1 have elicited the same response to U_{c3}-cells. That is, the response of layer U_{c3} is affected neither by shift of position like (a)-(c), nor by distortion of shape or size like (d)-(f), nor by some insufficiency of the pattern or some noise like (g).

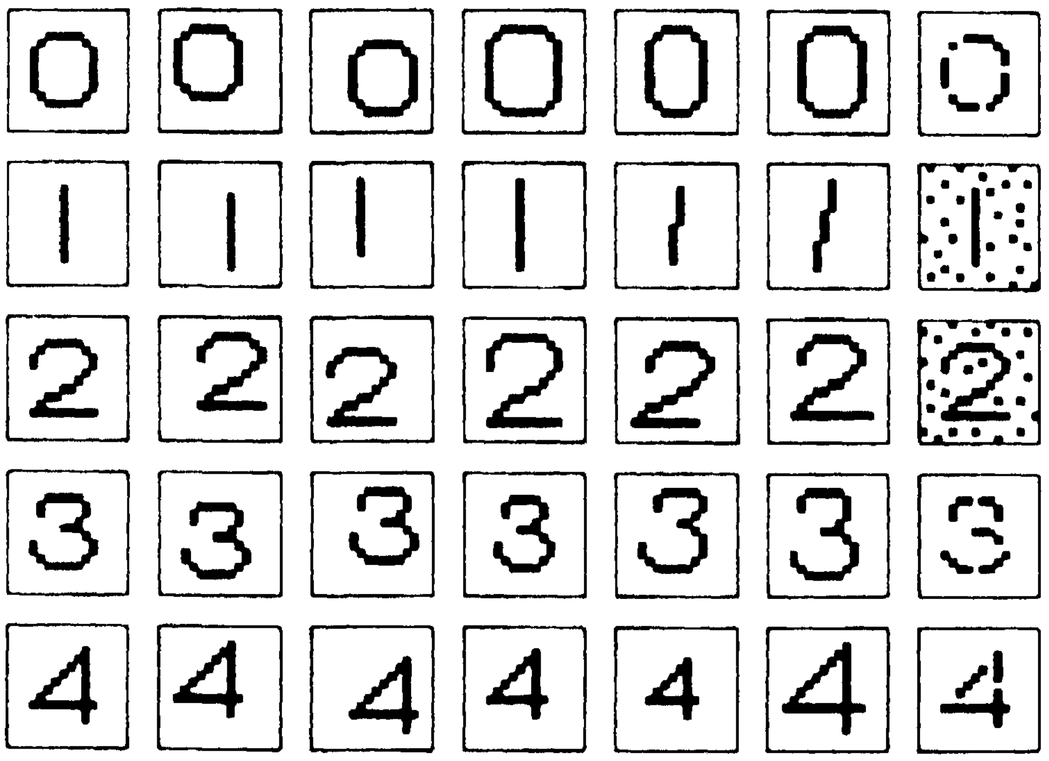


Fig. 1 Some examples of distorted stimulus patterns which the neocognitron has correctly recognized.

REFERENCES

[1] Fukushima, K. "Cognitron: A Self-organizing Multilayered Neural Network". Biol. Cybernetics 20:3/4 (1975) 121-136.

[2] Hubel, D.H., Wiesel, T.N. "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex". J. Physiol. (Lond.) 160:1 (1962) 106-154.

[3] Hubel, D.H., Wiesel, T.N. "Receptive Fields and Functional Architecture in Two Nonstriate Visual Area (18 and 19) of the Cat". J. Neurophysiol. 28 (1965) 229-289.

[4] Fukushima, K. "Improvement in Pattern-Selectivity of a Cognitron" (in Japanese). Paper of Tech. Grqup, MBE78-27, Inst. Electronics Commun. Engrs. Japan, July 1978.

RELATIONAL STRATEGIES FOR PROCESSING UNIVERSALLY QUANTIFIED QUERIES TO LARGE DATA BASES

Koichi Furukawa
Information Systems Section
Computer Science Division
Electrotechnical Laboratory
Tokyo, Japan

Relational algebraic strategies are described for processing universally quantified queries. First, a new algorithm is developed to evaluate universally quantified conditional queries of the form $[\vec{x} : (\forall \vec{y}) W1(\vec{x},\vec{y}) \rightarrow W2(\vec{x},\vec{y})]$. It decomposes the problems into two subproblems using set intersection, summary and Join operations within the frame work of relational algebra. Second, the partial multi-valued-dependency decomposition of a relation is introduced as an efficient representation schema to express a set of uniform intensional data of the form $(\forall \vec{x}/\vec{c})P(\vec{x})$. The original relation is defined in terms of the decomposed relations as a relational algebraic equation. A relational algebraic expression corresponding to a query to the original relation is transformed to the one in terms of the decomposed relations using the above equation. Furthermore, the transformed algebraic expression is converted to a simple associative search expression to one of the decomposed relations by applying the rule-based formula manipulation or symbolic evaluation method.

1. INTRODUCTION

It has been shown in Chang [1978], Furukawa [1977], Minker [1978], Kellog et al. [1978] and Reiter [1978b] that deductive logic offers considerable potential for improving on-line access to large, complex data base domains. The common feature of the current researches to that direction is the attempt to combine a deductive component with relational data bases.

Chang [1978] divided the approaches of these researches into two groups: One is the evaluational approach where intensions are used to transform queries and extensions are used to evaluate queries. The other is the non-evaluational one where both intensions and extensions are used to prove a question represented by a formula in the same manner. It has been shown in Reiter [1978b] that the evaluational approach is more feasible for data bases with very large extensions and comparatively small intensions.

Codd [1972] has defined an algorithm to convert queries expressed in relational calculus to a sequence of relational algebraic operations. Reiter [1978b] has developed a general framework in the restricted first order logic which enables one to get indefinite answers to any queries, and reformulated Codd's algorithm in his framework. Reiter [1978a] has also developed an efficient query conversion algorithm under the closed world assumption (CWA).

In this paper, a new algorithm to evaluate universally quantified conditional queries under the CWA is presented. Summary operation on a relation is added to the set of primitive operations of the relational algebra. The algorithm reduces the problem to the application of the operations of set intersection, summary and join to the answers to the subqueries.

In deductive relational data bases developed so far, only extensions have been considered to compose relations. Intensions have been stored in knowledge bases outside the relational data bases. However, it sometimes happens that we have a set of uniform intensional data of the form $(\forall \vec{x}/\vec{c})P(\vec{x})$. In this paper, a new representation schema based on the partial multi-valued dependency (PMVD) decomposition of a relation is introduced as a suitable way to express such data.

Queries to the decomposed relations are evaluated using a set of equations which give users the image of the original relations as a logical view. Furthermore, it will be shown that

universally quantified queries on those relations can be evaluated very efficiently by applying the rule-based symbolic evaluation method.

2. REVIEW OF QUERY EVALUATION UNDER THE CWA

In this section, query evaluation under the CWA developed by Reiter [1978a] will be reviewed quickly as well as some extentions.

All queries have the form $[x_1/\tau_1,\ldots,x_n/\tau_n : (q_1y_1/\theta_1)\ldots(q_my_m/\theta_m)W(x_1,\ldots,x_n,y_1,\ldots,y_m)]$ where $W(x_1,\ldots,x_n,y_1,\ldots,y_m)$ is a quantifier free formula with free variables $x_1,\ldots,x_n$, $y_1,\ldots,y_m$ and q_i is either $\forall$ or $\exists$, $i = 1,\ldots,m$. We shall use the abbreviated notation $[\vec{x}/\vec{\tau} : (q\vec{y}/\vec{\theta})W(\vec{x},\vec{y})]$ to express a typical query. The τ's and θ's, which are called types, are sets of constant signs which the variables associated with them range over. A sequence of types $\vec{\tau} = (\tau_1,\ldots,\tau_n)$ is the set $\tau_1\times\ldots\times\tau_n$.

A data base (DB) is a set of clauses containing no functional signs.

Under the CWA, if no proof of a positive ground literal exists, then the negation of that literal is assumed true. This can be viewed as equivalent to implicitly augmenting the given data base with all such negated literals.

Since worlds are completely specified under the CWA, the set of CWA answers $!Q!_{CWA}$ to $Q = [\vec{x}/\vec{\tau} : (q\vec{y}/\vec{\theta})W(\vec{x},\vec{y})]$ is defined as follows:

$$!Q!_{CWA} = \{\vec{c} : \vec{c} \in \vec{\tau} \text{ and } DB \cup \overline{EDB} \vdash (q\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\}$$

where $\overline{EDB} = \{\bar{P}c : P$ is a predicate sign, $\vec{c}$ a tuple of constant signs and $DB \nvdash P\vec{c}\}$.

The evaluation of any queries are reduced to the applications of the relational algebraic operations to the answers to atomic queries. Before listing query conversion rules, we define some of the operations of relational algebra.

Let $Q = [\vec{x}/\vec{\tau},z/\psi : W]$, and $\vec{x}$ is the n-tuple $x_1,\ldots,x_n$. Then $!Q!$ is a set of (n+1)-tuples, and the projection of $!Q!$ with respect to z, $\pi_z!Q!$ is the set of n-tuples obtained from $!Q!$ by deleting the (n+1)st component from each (n+1)-tuple of $!Q!$.

Let $Q = [\vec{x}/\vec{\tau},z/\psi : W]$. Then the quotient of $!Q!$ by z, $\Delta_z!Q!$, is a set of tuples and is defined as follows:

$$\vec{c} \in \Delta_z!Q! \text{ iff } (\vec{c},a) \in !Q! \text{ for all } a \in \psi.$$

The operater Δ_z is called the division with respect to z.

Let $Q1 = [\vec{x}/\vec{\tau},z/\psi : W1]$ and $Q2 = [\vec{y}/\vec{\theta},z/\psi : W2]$. The join of $!Q1!$ and $!Q2!$ with respect to z, $*_z(!Q1!,!Q2!)$, is a set of tuples and defined as follows:

$$(\vec{c},d,\vec{e}) \in *_z(!Q1!,!Q2!)$$

$$\text{iff } (\vec{c},d) \in !Q1! \text{ and } (\vec{e},d) \in !Q2!.$$

We sometimes denote $*_z(!Q1!,!Q2!)$ as $(!Q1! *_z !Q2!)$.

The set of query conversion rules are listed below:

Rule 1. (Decomposition of AND/OR queries)
1. $![\vec{x}/\vec{\tau} : W1 \wedge W2]! = ![\vec{x}/\vec{\tau} : W1]! \cap ![\vec{x}/\vec{\tau} : W2]!$.
2. $![\vec{x}/\vec{\tau} : W1 \vee W2]! = ![\vec{x}/\vec{\tau} : W1]! \cup ![\vec{x}/\vec{\tau} : W2]!$.

Rule 2. (Elimination of negation)
1. $![\vec{x}/\vec{\tau} : \overline{W}]! = \vec{\tau} - ![\vec{x}/\vec{\tau} : W]!$.
2. $![\vec{x}/\vec{\tau} : W1 \wedge \overline{W2}]! = ![\vec{x}/\vec{\tau} : W1]! - ![\vec{x}/\vec{\tau} : W2]!$.

Rule 3. (Elimination of existential quantifiers)
$$![\vec{x}/\vec{\tau} : (\exists\vec{y}/\vec{\theta})W]! = \pi_{\vec{y}}![\vec{x}/\vec{\tau},\vec{y}/\vec{\theta} : W]!$$
where $\pi_{\vec{y}}$ denotes $\pi_{y_1}\ldots\pi_{y_m}$ and π_{y_i} is the projection with respect to y_i.

Rule 4. (Elimination of universal quantifiers)
$$![\vec{x}/\vec{\tau} : (\forall\vec{y}/\vec{\theta})W]! = \Delta_{\vec{y}}![\vec{x}/\vec{\tau},\vec{y}/\vec{\theta} : W]!$$
where $\Delta_{\vec{y}}$ denotes $\Delta_{y_1}\ldots\Delta_{y_m}$ and Δ_{y_i} is the division with respect to y_i.

Rule 5. (Decomposition of AND queries)
$$![\vec{x}/\vec{\tau} : W1 \wedge W2]! = (![\vec{x}1/\vec{\tau}1 : W1]! * ![\vec{x}2/\vec{\tau}2 : W2]!)$$
where $\vec{x}i$ is a sub-sequence of $\vec{x}$ which appears in Wi, and $\vec{\tau}i$ is the corresponding sub-sequence of $\vec{\tau}$, for i = 1, 2. The join operation * is performed with respect to the variables included in both $\vec{x}1$ and $\vec{x}2$.

Most of the above rules are derived from Reiter [1978a]. The extended points are as follows:

1. In rule 2, the restriction that W, W1 ane W2 must be quantifier free is removed.

2. Rule 4 is new. Reiter has developed a general framework which enables one to get indefinite answers. The division operator defined by Codd [1972] has been properly generalized in the framework. However, the handling of universal quantifiers in the CWA has been ommitted in Reiter [1978a]. But there is no reason to exclude the handling of universal quantifiers in the CWA and moreover the set of conversion rules becomes more well-structured by adding these rules.

3. Rule 5 is new, too. It is easily shown that

this rule reduces the amount of computation. Let $Qi = [\vec{x}/\vec{c}:Wi]$ and $\vec{c}_i^c = \pi_{\vec{x}i}(\vec{c}_i^c)$, i.e., the compliment subsequence of $\vec{c}$, $i = 1, 2$. We would have to evaluate $Q = [\vec{x}/\vec{c}:W1 \wedge W2]$ according to $!Q! = !Q1! \wedge !Q2!$, unless we have Rule 5. But it is easily shown that

$$!Qi! = \vec{c}_i^c \times ![\vec{x}i/\vec{c}i:Wi]!$$

for $i = 1, 2$. By applying Rule 5, we can avoid the execution of the above costly Cartesian product operations.

3. THE HANDLING OF UNIVERSAL QUANTIFIERS

Let us consider how to compute the division $\Delta_z!Q!$ where $Q = [\vec{x}/\vec{c},z/\psi : (q\vec{y}/\vec{\theta})W]$. The following algorithm generates the answer $\Delta_z!Q!$.

Algorithm D.
1. Group the answer set $!Q!$ to the query Q by $\vec{x}$ and make a list of z's for each $\vec{x}$ (we denote the list by $Z\vec{x}$).
2. For each $Z\vec{x}$, test whether it includes the set ψ. If the test succeeds, then put the $\vec{x}$ in the answer set $\Delta_z!Q!$.

Rule 4 says that $\Delta_z!Q!$ is the answer set for the universally quantified query $Q0 = [\vec{x}/\vec{c} : (\forall z/\psi)(q\vec{y}/\vec{\theta})W]$. Notice that Q0 is the abbreviation of the conditional query $Q0' = [\vec{x}/\vec{c} : (\forall z)(z \in \psi \rightarrow (q\vec{y}/\vec{\theta})W)]$. The inclusion test $\psi \subset Z\vec{x}$ in Algorithm D reflects the implication in Q0', where the consequent $z \in \psi$ in Q0' corresponds to the lefthand set ψ and the antecedent $(q\vec{y}/\vec{\theta})W$ to the righthand set $Z\vec{x}$.

We can use the division operation to efficiently evaluate a class of conditional queries $Qc = [\vec{x}/\vec{c} : (\forall z/\psi)(W1 \rightarrow W2)]$ such that W1 does not contain any free variables in $\vec{x}$. The following theorem states the fact more precisely.

Theorem 1. Let $Qc = [\vec{x}/\vec{c} : (\forall z/\psi)(W1 \rightarrow W2)]$ be a universally quantified conditional query and assume that W1 does not contain any free varibles in $\vec{x}$, $Q = [\vec{x}/\vec{c},z/\psi :W2]$, and $Qz = [z/\psi :W1]$. If DB $\cup$ $\overline{EDB}$ is consistent, then

$$!Qc! = \Delta_{z' \in !Qz!}!Q! \text{ if } !Qz! \neq \phi,$$

$$= \vec{c} \quad \text{otherwise.}$$

We sometimes denote $\Delta_{z' \in !Qz!}!Q!$ as $!Q! \,/\, !Qz!$.

We cannot use Theorem 1 if W1 contains any variables in $\vec{x}$. We can deal with the general case by introducing the following cut operation:

Let $R = ![\vec{x}/\vec{c},z/\psi :W(\vec{x}, z)]!$. Then the cut of R with respect to $\vec{x}$, denoted by $\rho_{\vec{x}}(R)$, is a set of pairs $(\vec{c}_i, D_i)$ where $\vec{c}_i \in \vec{c}$ and D_i is a set of $d_k \in \psi$ such that

$$D_i = \{d_k : (\vec{c}_i, d_k) \in R\}.$$

We regard $\rho_{\vec{x}}(R)$ to be a function from $\pi_{\vec{z}}R$ to $2^{\pi_{\vec{z}}R}$ and denote D_i as $\rho_{\vec{x}}(R)(\vec{c}_i)$. $\rho_{\vec{x}}(R)(\vec{c}_i)$ is called the cross-section of $\rho_{\vec{x}}(R)$ at $\vec{c}_i$.

The set of D is exactly the same as $Z\vec{c}_i$ in Algorithm D.

Theorem 2. Let $Q = [\vec{x}/\vec{c} : (\forall z/\psi)(W1 \rightarrow W2)]$. If DB $\cup$ EDB is consistent, then

$$!Q! = (\vec{c} - ![\vec{x}/\vec{c} : (\exists z/\psi)W1]!)$$

$$\cup \{\vec{c} : \phi \neq \rho_{\vec{x}1}(R1)(\vec{c}_1) \subset \rho_{\vec{x}2}(R2)(\vec{c}_2)\},$$

where $Ri = ![\vec{x}_i/\vec{c}_i,z/\psi :Wi]!$ for $i = 1, 2$.

Since $A \subset B$ iff $|A \cap B| = |A|$ in case A and B are finite sets, we can replace the set inclusion test in Theorem 2 by a simple comparison of the cardinal numbers of two sets. This strategy has been implemented in GM-RDMS (Whitney [1974]), where an operation called summary has been used to construct a set of pairs $(\vec{c}_i, |D_i|)$ which summarize the pairs $(\vec{c}_i, D_i)$.

Example
The following example is taken from Palermo [1974]. There are four relations: SUPPLY, SUPPLIERS, PROJECT and PART, as shown in Figure 1 (The relation names are underlined to distinguish them from the corresponding predicates. They are considered to refer sets, rather than predicates).

SUPPLY	SID	PID	JID
	211	31	971
	325	32	971
	211	33	970
	211	31	972
	237	31	972
	237	31	970
	237	32	970
	237	33	970
	237	32	971
	237	31	971

SUPPLIER	SID	SLOC	SNAME
	211	NY	AA
	325	SF	XX
	237	LA	YY

PROJECT	JID	JLOC	JNAME
	970	POK	A
	971	SJ	X
	972	SJ	Y

PART	PID	PTYPE
	31	A
	32	A
	33	B

Figure 1. An example data base.

The relation SUPPLY (SID, PID, JID) is a set of tuples (x, y, z) such that a supplier x supplies a part y to a project z. SUPPLIER (SID, SLOC, SNAME) has information on suppliers' locations and their names. PROJECT (JID, JLOC, JNAME) has information on projects' locations and their names. PART (PID, PTYPE) defines the type of each part.

Let us consider the following query to the above data base:

Query. Find the name and location of suppliers each of whom has supplied at least one project located in San Jose with at least one of every part of type A.

This query is expressed as follows:

$$Q = [x/SNAME,\ y/SLOC :\ (\exists s/SID)(SUPPLIER(s,y,x) \wedge (\exists j/PID)(PROJECT(SJ,j) \wedge (\forall p/PID)(PART(p,A) \rightarrow SUPPLY(s,p,j))))]$$

The evaluation of Q proceeds as follows:

1. Let
   Q1 = [x/SNAME, y/SLOC, s/SID : SUPPLIER(s,y,x)]
   and
   Q2 = [s/SID : (∃j/JID)(PROJECT(SJ,j) ∧ (∀p/PID)(PART(p,A) → SUPPLIER(s,p,j))))].

   By applying Rule 3 and 5 to Q, we get

   $$|Q| = \pi_s(|Q1| *_s |Q2|).$$

2. Let Q21 = [j/JID: PROJECT(SJ,j)]

   and Q22 = [s/SID, j/JID : (∀p/PID)(PART(p,A) → SUPPLY(s,p,j))]

   From Rule 3 and 5, we obtain

   $$|Q21| = \pi_j(|Q21| *_j |Q221|).$$

3. Let
   R1 = |[p/PID : PART(p,A)]| and

   R2 = |[s/SID, j/JID, p/PID : SUPPLY(s,p,j)]|

   Since R1 is not an empty set,

   $$|Q22| = R2 \ / \ R1$$

   from Theorem 1.

The snapshots of the above evaluation process is shown in Figure 2, where the summary operation is used to evaluate Q22.

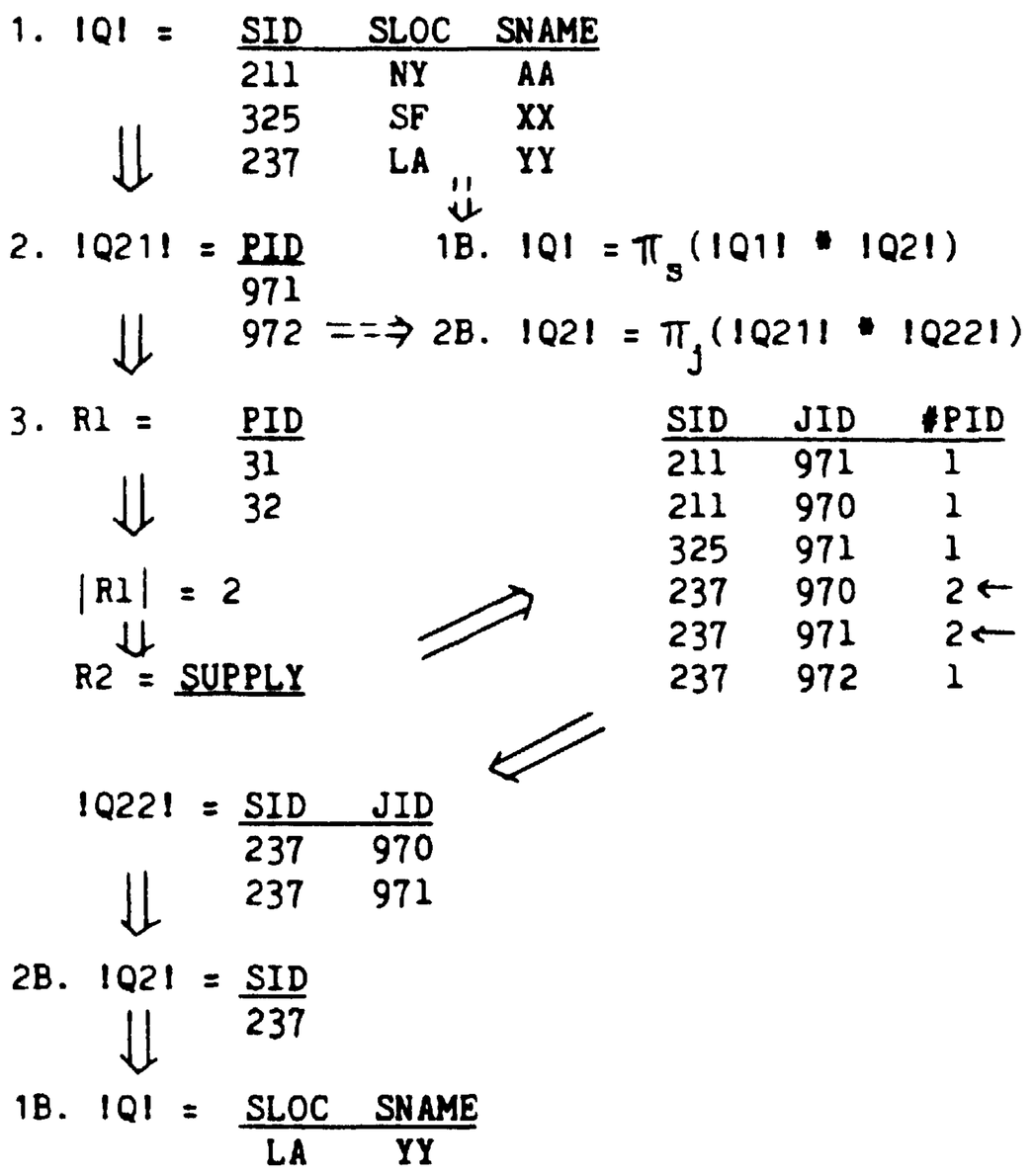


Figure 2. The process of the query evaluation.

4. THE PMVD DECOMPOSITION OQF A RELATION

Suppose that the answer of the universally quantified query Q22 in the previous example is saved. Then, we can answer to the same query immediately by simply restoring the saved answer. We need to save answers for other types, e.g. a set of supplier-project pairs (x, y) such that x supplies all parts of type B to y. These answers can be put together if we associate each type to each answers. Let SIIPPI.YALI. (SID, JID, PTYPE) be the relation built in such a way. From SUPPLYALL and PART relations, we can infer the "SUPPLY" fact, namely;

$$(\forall s/SID)(\forall j/JID)(\forall p/PID)((\exists t/PTYPE)SUPPLYALL(s,t,j) \wedge PART(p,t) \rightarrow SUPPLY(s,p,j)) \qquad (4.1)$$

Furthermore, we can remove all tuples which are infered in (4.1) from the original SUPPLY relation. We rename the reduced SUPPLY relation as SUPPLYSOME. The original SUPPLY relation is now decomposed into two relations: SUPPLYALL and SUPPLYSOME, as shown in Figure 3.

The extension of SUPPLY(s,p,j) can be obtained from not only the SUPPLYSOME relation, but also

SUPPLYALL	SID	PTYPE	JID
	237	A	971
	211	B	970
	237	B	971
	237	A	970

SUPPLYSOME	SID	PID	JID
	211	31	971
	325	32	971
	211	31	972
	237	31	972

Figure 3. The decomposed relations of SUPPLY.

the SUPPLYALL relation. Assume that there are no other relations which contain the information about SUPPLY(s,p,j). Then, the following equation holds:

$$\underline{SUPPLY}(s,p,j) = \underline{SUPPLYALL}(s,y,j) *_y \underline{PART}(p,y)) \cup \underline{SUPPLYSOME}(s,p,j) \quad (4.2)$$

This equation is used to transform queries in terms of simple user's view of relations to those in terms of system's view (Furukawa [1977]). The relational algebraic expression of the query Q22, that is,

$$!Q22! = \underline{SUPPLY}(s,p,j) \,/\, \varphi_{t=A} \cdot \underline{PART}(p,t) \quad (4.3)$$

is transformed to:

$$!Q22! = ((\underline{SUPPLYALL}(s,y,j) *_y \underline{PART}(p,y)) \cup \underline{SUPPLYSOME}(s,p,j))/\varphi_{t=A}\underline{PART}(p,t) \quad (4.4)$$

where the operator $\varphi_{f(\vec{t})}$ is defined by

$$\varphi_{f(\vec{t})}R(\vec{x},\vec{t}) = ![\vec{x} : R(\vec{x},\vec{t}) \text{ and } f(\vec{t}) = True]!$$

and is called the selection with respect to $f(\vec{t})$.

Considering the meaning of the predicate "SUPPLYALL", it is expected that !Q221! = ![s/SID,j/JID : SUPPLYALL(s,A,j)]! should be included in the answer set !Q22!. In fact, we can deduce !Q221! as the answer.

To deduce the result, we need several rules and assertions as follows.

Rules.

RU1. If $\underline{R1} = \underline{R2} *_t \underline{R3}'(t)$ and

$\underline{R3} = \varphi_{t=a_i}\underline{R3}'(t)$, then $\underline{R3} \mid \underline{R1}$.

RU2. If $\underline{R3} \mid \underline{R1}$, then

$(\underline{R1} \cup \underline{R2})/\underline{R3} = \underline{R1}/\underline{R3} \cup \underline{R2}/\underline{R3}$.

RU3. If x uniquely determines y, then

$$(\underline{R1}(y,z) *_y \underline{R2}(x,y)) \,/\, \varphi_{t=a_i}\underline{R2}(x,t) = \varphi_{y=a_i}\underline{R1}(y,z).$$

Assertions.

AS1. $\underline{SUPPLYSOME}(s,p,j) \,/\, \varphi_{t=a_i}\underline{PART}(p,t) = \phi$.

AS2. In PART(p,t), p uniquely determines t.

The deduction process is as follows:

1. From RU1, and (4.4), it is shown that

$$\varphi_{t=A}\underline{PART}(p,t) \mid \underline{SUPPLYALL}(s,y,j) *_y \underline{PART}(p,y) \quad (4.5)$$

2. From (4.5), RU2 and (4.4), !Q22! is transformed to

$$(\underline{SUPPLYALL}(s,y,j) *_y \underline{PART}(p,y))/\varphi_{t=A}\underline{PART}(p,t) \cup \underline{SUPPLYSOME}(s,p,j)/\varphi_{t=A}\underline{PART}(p,t). \quad (4.6)$$

3. From (4.6), AS2, RU3 and AS1, !Q22! is further transformed to:

$$!Q22! = \varphi_{y=A}\,\underline{SUPPLYALL}(s,y,j).$$

This is the final result and is equal to !Q221!.

If SUPLLYSOME(s,p,j) = , then all suppliers supply only all parts of some types to some projects. In this case, it is said that multi-valued dependency (MVD) PTYPE $\twoheadrightarrow$ (SID,JID) holds and it is guaranteed that the SUPPLYPART relation, that is, the hypothetical single relation which includes all information about SUPPLY and PART, can be decomposed into

$\underline{SUPPLYALL}(s,t,j) = \pi_p(\underline{SUPPLYPART}(s,j,p,t))$

and $\underline{PART}(p,t) = \pi_S\pi_j(\underline{SUPPLYPART}(s,j,p,t))$

without losing any information.

If SUPPLYSOME(s,j,p) is not empty, the above condition holds only for the subset (SUPPLY - SUPPLYSOME) of the SUPPLY relation. We say that partial MVD (PMVD) PTYPE $\overset{p}{\twoheadrightarrow}$ (SID,JID) holds in SUPPLYPART relation with SUPPLYSOME relation as exception.

The relation SUPPLYALL, unlike ordinary relations, is a set of intensional data. A tuple (s0,t0,j0) in SUPPLYALL means that the supplier s0 supplies all parts of type t0 to the project j0. On the other hand, "SUPPLYALL(s0,t0,j0)" may sometimes be interpreted as an extensional fact by itself. (s0,t0,j0) becomes an extensional

fact if it is not unusual that some suppliers supply to some projects all parts of some types. In that case, the query will be expressed directly in terms of "SUPPLYALL", instead of through "SUPPLY".

It has been suggested in Ohsuga [1979] that the information clustering by the division is an important mechanism to build a structure in the data base. The PMMD decomposition schema, together with the associated query transformation equation, enables one to realize the above mechanism in the framework of deductive relational data bases.

We finally remark the evaluation of queries other than universally quantified ones to the decomposed relations. For example, consider the query Q = [p/PID,j/JID : SUPPLY(237,p,j)]. By applying the query transformation equation (4.2) t |Q| = $\varphi_{s=237}$ SUPPLY(s,p,j), s transformed to

$$|Q| = \varphi_{s=237}(\underline{SUPPLYALL}(s,y,j) *t \underline{PART}(p,y) \cup \underline{SUPPLYSOME}(s,p,j)). \quad (4.7)$$

We need to perform the costly Join of the SUPPLYALL relation and the PART relation to evaluate (4.7). However, it may sometimes be adequate to answer in terms of SUPPLYALL and SUPPLYSOME instead of SUPPLY. In that case, we can avoid the join. It is related to the notion of approximate responses discussed in Joshi et al. [1977].

5. CONCLUSION

Practical algorithms to evaluate universally quantified queries are presented.

Furthermore, a new representation schema based on the PMMD decomposition of relations are introduced. A set of uniform intensional data of the form $(\forall \vec{x}/\vec{c})P(\vec{x})$ is expressed as a relation by using the schema. Universally quantified queries to the decomposed relations are converted to quantifier-free queries by the rule-based symbolic evaluation method.

The DBAP (Furukawa [1977]) are being expanded to realize the algorithms presented in this paper..

ACKNOWLEDGEMENT

The auther thank to Dr. Osamu Ishii and Dr. Akio Tojo for providing a chance of the present study, and to Professor Setsuo Ohsuga, Mr. Kazuhiro Fuchi and the staffs of the information systems section for their helpful discussions.

REFERENCES

1. Chang, C. L. [1978] "DEDUCE 2: Further Investigations of Deduction in Relational Data Bases," in Logic and Data Bases, (ed. H. Gallaire and J. Minker), Plenum Press, 1978, 201-236.

2. Codd, E. F. [1972] "Relational Completeness of Data Base Sublanguages," in Data Base Systems, (Ed. R. Rustin), Prentice-Hall, 1972, 65-98.

3. Furukawa, K. [1977] "A Deductive Question Answering System on Relational Data Bases," Proc. Fifth 1JCAI, 1977, 59-66.

4. Joshi, A. K. et al. [1977] "Approximate Responses from a Data Base Query System: An Application of Inferencing in Natural Language," Proc. Fifth IJCAI, 1977, 211-212.

5. Kellog, C. et al. [1978] "Deductive Planning and Pathfinding for Relational Data Bases," in Logic and Data Bases, 1978, 179-200.

6. Minker, J. [1978] "An Experimental Relational Data Base Systems on Logic," in Logic and Data Bases, 1978, 107-148.

7. Ohsuga, S. [1979] "Toward Intelligent Interactive Systems," Proc. The IFIP W.G. 5.2 Workshop Seillac II on Methodology of Interaction, (Ed. P. Ten Hagen), North-Holland, 1979-

8. Palermo, F. P. [1974] "A Data Base Search Problem," in Information Systems, (Ed. J. T. Tou), Plenum Press, 1974.

9. Reiter, R. [1977] "An Approach to Deductive Question-Answering," BBN Report No. 3649, Bolt, Beranek and Newman, Inc., 1977.

10. Reiter, R. [1978a] "Deductive Question Answering on Relational Data Bases," in Logic and Data Bases, 1978, 149-178.

11. Reiter, R. [1978b] "On Closed World Data Bases," in Logic and Data Bases, 1978, 55-76.

12. Whitney, V. K. [1974] "A Relational Data Management System," in Information Systems, (Ed. J. T. Tou), Plenum Press, 1974.

CONTROLLING KNOWLEDGE DEDUCTION IN A DECLARATIVE APPROACH

Herve Gallaire
ENSAE-CERT
2, Avenue Edouard Belin
31055 Toulouse cedex
France

Claudine Lasserre
ENSAE
10, Avenue Edouard Belin
31055 Toulouse Cedex
France

Due to its late receipt, this paper appears in full in the Supplement section, back of Volume II.

A PROBLEM SIMILARITY APPROACH TO DEVISING HEURISTICS: FIRST RESULTS

John Gaschnig
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

Here we describe an approach, based upon a notion of problem similarity, that can be used when attempting to devise a heuristic for a given search problem (of a sort represented by graphs). The proposed approach relies on a change in perspective: instead of seeking a heuristic directly for a given problem PI, one seeks Instead a problem P2 easier to solve than PI and related to PI in a certain way. The next step is to find *an* algorithm for finding paths in P2, then apply this algorithm in a certain way as a heuristic for PI. In general, the approach is to consider as candidates problems P2 that are "edge subgraphs" or "edge supergraphs" of the given problem PI. As a non-trivial application, we show that a certain restricted form of sorting problem (serving as P2) is an edge supergraph of the 8-puzzle graph (PI). A simple algorithm for solving this sorting problem is evident, and the number of swaps executed in solving an instance thereof is taken as a heuristic estimate of distance between corresponding points in the 8-puzzle graph. Using the At algorithm, we experimentally compare the performance of this "maxsort" heuristic for the 8-puzzle with others in the literature. Hence we present evidence of a role for exploiting certain similarities among problems to transfer a heuristic from one problem to another, from an "easier" problem to a "harder" one.

1. Introduction *, **

Many combinatcrially large problems cannot be solved feasibly by exhaustive case analysis or brute force search, but can be solved efficiently if a heuristic can be devised to guide the search. Finding such a heuristic for a given problem, however, usually requires an exercise of creativity on the part of the researcher.

Research to date on devising heuristics has spanned several problem-solving domains and several approaches. !n come efforts, the objective has been to optimize, using some adaptation scheme over a number of trials, the values of coefficients determining the relative weighting of several preselected terms in an evaluation function, so as to maximize the overall performance (e.g., [Samuel 1959], [Samuel 1967], [Rendell 1977]). Other approaches to automatic generation of heuristics include [Ernst, et al., 1974] and [Rendell, 1976]. Related efforts have focused on what might be called "disciplined creativity", enunciating general principles or rules of thumb that a person may apply to the problem at hand (e.g., [Polya 1945], [Wickelgren 1974]).

The approach to devising heuristics proposed here differs in perspective from these previous efforts: instead of seeking a heuristic directly, one seeks instead a problem P2 that is easier to solve than the given problem P2 and is related to P2 in a certain way. * The next step is to find an algorithm for finding paths in P2, then apply this algorithm in a certain way as a heuristic for PI. As an elementary example, the rectilinear distance function is an efficient heuristic for finding paths in a "Manhattan street pattern" graph even when some (but not too many) of the streets have been blockaded (i.e., some edges are removed from the graph). Generalizing, the approach is to consider as candidates problems P2 that are "edge subgraphs" or "edge supergraphs" of the given problem PI. As a non-trivial application, we show that a certain restricted form of sorting problem (serving as P2) is an edge supergraph of the 8-puzzle graph (PI). A simple algorithm for solving this sorting problem is evident, and the number of swaps executed in solving an instance thereof is taken as a heuristic estimate of distance between corresponding points in the 8-puzzle graph. Using the A* algorithm, we experimentally compare the performance of this "maxsort" heuristic for the 8-puzzle with others in the literature.

The general class of problems to which the present approach can be applied are those state space problems that can be represented as graphs, in which the objective is to find a path from a given initial node in the graph to

This research was eponeored by the Defense Advanced Research Project* Afency (00D), ARPA Order No 3597, monitored by the Air Force Avionica Laboratory Under Contract F33615-78-C-1551.

The views and conclusion contained in this document ara thosa of the author and should not be interpratad as representing tha official policies, eithar expressed or implied, of the Defense Advanctd Research Projects Afency or the US Government

** Author's present address Artificial Intelligence Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025

Thia perspective ia somewhat akin to that on which are based the results of backward error analysis (aa defined in tha numerical analyaia literature), in which one asks of a matrix inversion algorithm, for example, not how accurate ara the answers that it computes, but rather how different from the fiven problem ia the problem for which the computed answers are the exact anawers

a given goal node. * Such problems can be solved using the A* best-first search algorithm, which iteratively grows a tree or graph of partial paths from the initial node, at each step expanding the node (along the edge of this tree or graph) that appears to be "best". ** The definition of "best" is specified by assigning a number to each node generated, whose computation typically involves the value of a heuristic function K(s, t) used to estimate the distance in the graph from an arbitrary node s in the graph to another arbitrary node t (where t is taken to be the goal node in a particular instance of search). Devising heuristic functions that estimate distance between points in a given graph is the present practical objective.

Additional theoretical objectives motivated this work: to investigate how "similar" or "dissimilar" problems differ in structural properties, and how such structural differences relate to difference in difficulty to solve the problem. Investigations of relatively simple problems serve this purpose well. Note that we are not interested In the 8-puzzle (our principal example) *per* se, but as a concrete instance with which to investigate general principles.

Section 2 defines the notions of "edge subgraph" and "edge supergraph", and illustrates their relation to heuristic transfer using a simple example — a "Manhattan street pattern" graph and variants thereof. Section 3 applies the basic approach to a larger problem — the 8-puzzle. Sectien 4 discusses the generality of the results and poses future tasks.

2. Problem Similarity: Edge Subgraphs and Supergraphs

The basic idea considered in this paper can be illustrated using a simple example. Consider the graphs depicted in Figures Ia, Ib, and Ic, which we shall refer to as MSUB44, M44, MSUP44, respectively. We note that these three graphs have identical numbers of nodes, and that the edges of MSUB44 comprise a subset of those of M44; similarly the edges of M44 comprise a subset of those of MSUP44. We shaft say therefore that MSUB44 is an edge subgraph of M44 and that similarly M44 is an edge subgraph of MSUP44. Likewise we also say that M44 is an edge supergraph of MSUB44 and that MSUP44 is an edge supergraph of M44. These relations of edge subgraph and edge supergraph generalize formally in the obvious way for the class of problem graphs, which we define to be any finite, strongly connected graph G - (V, E) having no self-loops and no multiple edges. This definition includes the familiar problems cited in the preceding section.

To illustrate how edge subgraph or edge supergraph similarity between problem graphs is related to heuristic transfer, we now consider three cases in turn: (1) that M44 is the given problem to be solved; (2) that MSUB44 is the given problem; and (3) that MSUP44 is the given problem. In general we wish to solve an arbitrary instance of a given problem graph P, that is to find a path from an arbitrary initial node s_r in P to an arbitrary goal node s_g. (We denote such a problem instance I of a problem graph P thus: I - (s_r, s_g).) This trivial example serves as a vehicle for introducing several general concepts.

The task of finding a path between arbitrary initial and goal nodes in the M44 graph (or in some larger version of this "Manhattan street pattern" graph) is trivial. A simple algorithm for solving instances of this problem graph is readily evident: comparing the coordinates of the current node (starting with the initial node s_r), move iteratively up (or down as the case may be) until the vertical coordinate of the goal node is reached, then move right (or left) iteratively until the goal node is reached. Call this algorithm L (for "L-shaped solution path").

If MSUB44 is taken instead of M44 as the problem to be solved, the A* approach seems more attractive than attempting to devise an algorithm like algorithm L, because of the additional cases an algorithm of the latter sort must account for: besides comparing the coordinates of the current node with those of the goal node, it must also be prepared to make detours when necessary.

To use A* to solve MSUB44, one must supply a heuristic function K(s, t) that estimates distance from node s to node t. Let hp(s, t) denote the actual distance in P from s to t, assuming edges have unit weight. In the case K(s, t) - h(s, t) the distance estimate is exact; it is well known [Hart et al. 1968] that this case minimizes the number of nodes expanded -- the number is exactlv $h(s_r, S_g)$, the distance from initial node to goal node.**

For* M44, it is evident that h_{M44},(s, t) - $l^x s \sim x_t' * 'Y_s " y_t'$ where x_5 and y_s are the x and y coordinates, respectively, of node s. For MSUB44 a symbolic formula for h(s, t) is not so compact, since it must distinguish more cases. An alternative to enunciating h(s, t) for MSUB44 is to use K(s, t) - h_{M44}(s, t), i.e., use the distance function of M44 to approximate the distance function of MSUB44.

It is easy to show, as follows, that solution paths found for an arbitrary problem instance of MSUB44 using

---

* Elementary examples of such problems include the 8-puzzle, The Tower of Hanoi, the water jug problem, missionaries and cannibals problem and other "toy" problems familiar in the literature [Nilsson 1971, pp. 39-41, 77-78], [Jackson 1974, pp. 81-84, 110-115], [Raphael 1976, pp. 79-86], [Wickelgren 1974, pp 49-57, 78-80 cf.]. Somewhat less frivolous examples include certain algebraic manipulation problems [Doran & Michie 1966, pp. 254-255], [Doran 1967, pp. 114-115], and a version of the traveling salesperson problem [Doran 1968], [Harris 1974].

** The Graph Traverser algorithm [Doran & Michie 1966] and the HPA algorithm [Pohl 1970a] are essentially the same as A*.

* Note that the 8-puzzle graph consists of two disjoint components. For present purposes, we consider search within one such component. Also, we assume that a problem graph is specified in practice by a successors function: SUC(s) denotes, for any node s in the graph, the set of nodes v_i for which there exists an edge from s to v_i. Typically SUC(s) is implemented by a set of operators, each of which transforms a given state s into another state v_i, provided the operator's precondition is satisfied. In the 8-puzzle context, one such operator might have the effect of moving the hole upward if it is not in the top row of the board.

** To achieve this bound, one must be careful in selecting a strategy for resolving ties among nodes for which equal values were assigned.

$K(s, t) = h_{M44}(s, t)$ will be of minimal length. If P1 and P2 are any problem graphs such that P1 is an edge subgraph of P2 (denoted $P1 \subseteq_e P2$), then the distance between arbitrary nodes s and t in P2 is never more than the distance between the corresponding points in P1, i.e., $h_{P2}(s, t) \leq h_{P1}(s, t)$. Hence using the heuristic function $K(s, t) = h_{P2}(s, t)$ for solving P1 it follows that $K(s, t) \leq h_{P1}(s, t)$. Hence by the A* admissibility theorem ([Hart, et al. 1968], [Gelperin 1977]) the solution path found is always of minimum length when using $K(s, t) = h_{P2}(s, t)$ in solving P1.*

As a third case, take MSUP44 as the problem to be solved. Now $h_{MSUP44}(s, t) \leq h_{M44}(s, t)$, hence taking $K(s, t) = h_{M44}(s, t)$ overestimates the distance from s to t in MSUP44. Instead of using A*, however, note that algorithm L can be used to find paths in MSUP44, although they will not necessarily be of minimum length. (Whether this is important depends on the application.) In general, if $P2 \subseteq_e P1$ then an algorithm that finds paths in P2 can also be used to find paths in P1 (assuming the same encoding scheme is used for identifying corresponding nodes in P1 and P2).

These simple examples illustrate a general approach: given a problem graph P1 to be solved, first identify a problem graph P2 that is an edge subgraph or edge supergraph of P1. We call P1 the given problem and P2 the transfer problem. Then find a way of computing $h_{P2}(s, t)$. (In the preceding examples a simple formula for h_{M44} was readily evident. When a formula for h_{P2} is not evident, we instead attempt to devise an algorithm for computing the values of $h_{P2}(s, t)$ by finding a path in P2 from s to t and counting the number of steps in the path. See section 3.) Then use $h_{P2}(s, t)$ as a heuristic function for solving instances of P1: given a goal node s_g, whenever A* calls for a value to be assigned to a node s in P1, compute $h_{P2}(s,s_g)$.

3. Example: A Sorting Algorithm Used as Heuristic for the 8-Puzzle

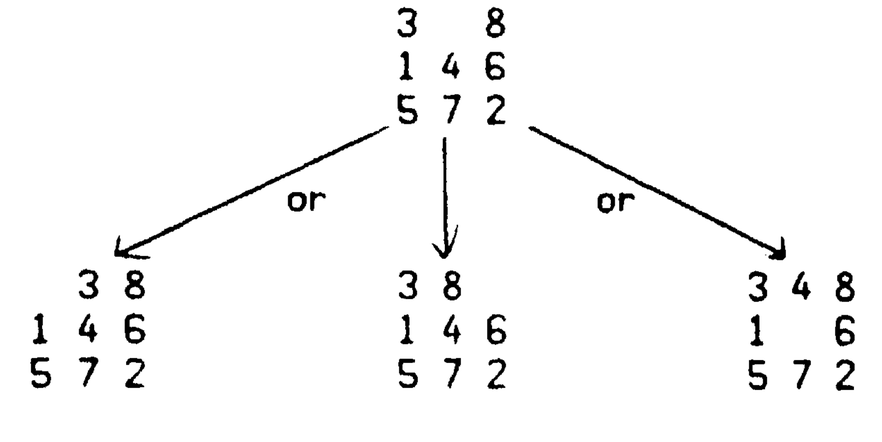
Section 3.1 defines the "9MAXSWAP" graph and shows it to be an edge supergraph of the 8-puzzle graph. Section 3.2 defines an algorithm, called MAXSORT, for finding paths in the 9MAXSWAP problem graph. Section 3.3 reports experimental results measuring the performance of MAXSORT when used as the basis of a heuristic distance-estimating function for the 8-puzzle, and compares this observed performance with those of other known heuristics for the 8-puzzle. This section also proposes and tests another related heuristic for the 8-puzzle. Section 3.4 reports experimental measurements of how closely the heuristic function based on MAXSORT approximates the distance function of the 8-puzzle. This larger example serves as a non-trivial application and as a vehicle for introducing additional concepts.

---

* A* is analyzed mathematically in [Hart, et al. 1978], [Pohl 1970a], [Pohl 1970b], [Nilsson 1971], [Harris 1974], [Vanderbrug 1976], [Munyer 1976], [Munyer & Pohl 1976], [Ibaraki 1977], [Pohl 1977], [Gelperin 1977], [Martelli 1977], [Gaschnig 1979], and elsewhere. Attempting to apply such analytic results is not a major concern here; in Section 3.3 we measure heuristic performance experimentally.

3.1. The MAXSWAP problem

The 8-puzzle is a one-person game the objective of which is to rearrange a given configuration of eight tiles on a 3x3 board into another given configuration by iteratively sliding a tile into the orthogonally adjacent empty location, like so:



To apply the present approach (which might be called the "edge subgraph/supergraph transfer" approach) to the 8-puzzle, first we must identify a suitable edge subgraph or edge supergraph of the 8-puzzle. Toward this end we note that an 8-puzzle tile configuration can be considered a permutation of the sequence 1,2,...,9, letting 9 correspond to the hole in the 8-puzzle, and numbering the nine locations of the 8-puzzle board in left to right, top to bottom manner, say. Hence the topmost of the four configurations depicted above corresponds to the sequence 398146572.

Now we define the "N-swap" graph to have N! nodes, corresponding to the N! permutations of the sequence 1,2,...,N. An edge connects two vertices of this graph iff the corresponding permutations differ by a single swap. In an obvious way, the behavior of any sorting algorithm which operates by sequentially interchanging pairs of array elements can be mapped into sets of paths in this graph. Taking N = 9, there is evident a one-to-one correspondence between the nodes of 9-Swap (but not the edges) and those of the 8-puzzle graph.

By noting that the effect of moving a tile in the 8-puzzle graph is to exchange two elements of a 9 element vector, it is easy to see that the 8-puzzle graph is an edge subgraph of 9SWAP. Hence the 8-puzzle can be viewed as a restricted version of permutation sorting, such that every swap must include the largest element (i.e., 9), and that certain swaps are forbidden, depending on the location of the largest element.

If we impose the restriction on sorting N-element permutations that every swap must exchange the largest element, N, with some other element, we obtain what might be called the "N-MAXSWAP" graph. For illustrative purposes, Figure 2 depicts the 4MAXSWAP graph. Note that $8\text{-puzzle} \subseteq_e 9MAXSWAP \subseteq_e 9SWAP$, hence for arbitrary nodes s and t, $h_{9SWAP}(s, t) \leq h_{9MAXSWAP}(s, t) \leq h_{8\text{-puzzle}}(s, t)$. Hence the distance functions of both 9SWAP and 9MAXSWAP underestimate the distance function of the 8-puzzle, but that of 9MAXSWAP is a closer approximation of the

8-puzzle's distance function than that of 9SWAP. * We will use 9MAXSWAP as the transfer problem for the 8-puzzle in this example; next, we must demonstrate the transfer.

3.2. The MAXSORT algorithm

We describe now what seems to be a most peculiar way of solving the 8-puzzle. Given a problem instance, A* iteratively expands nodes in the 8-puzzle graph and requires that a number be assigned to every new node generated, as the basis for deciding which node to expand next. To obtain each such number we propose to solve an entire instance of 9MAXSWAP using an algorithm defined below, and return a number measuring its performance (in fact, the number of swaps it executes). If solving instances of 9MAXSWAP itself required search, this approach might consume more time than if the 8-puzzle were solved using breadth-first search. However, the point of this example is that the 9-Maxswap problem is simpler than the 8-puzzle — one can readily devise an efficient algorithm for it, as follows.

The basis for an algorithm which we call "MAXSORT" (defined by a SAIL procedure in the test) for finding paths in an N-MAXSWAP graph is the observation that the largest element, N, in the permutation can always be swapped with the element whose proper place in the permutation it occupies, except when N is in the N'th position of the permutation. We illustrate using N = 4. To sort the permutation P = 2341 (into 1234), the algorithm first swaps 3 and 4 producing 2431, in which the element 3 now occupies its proper place. Next swapping 2 and 4 puts 2 in its proper place and then swapping 4 and 1 puts both those elements in their proper places and the algorithm terminates. To implement this policy of simply swapping the largest element 4 with the element whose proper place 4 is occupying, MAXSORT uses an internal auxiliary array B, and begins by assigning to B[i] the location of element i in the input permutation, i.e., B[I:4] = 4123 for the above example. Then MAXSORT simply swaps iteratively P[B[4]] with P[B[B[4]]], updating both P and B at each iteration, until the sort is complete, with the followng exception to this rule.

Sorting 4321 using MAXSORT, the first swap produces 1324, but now 4 is in its proper place, and so we must do something other than swap 4 with itself indefinitely. Instead, the algorithm swaps 4 with the leftmost element of P that is not in its proper place, i.e., 3 in this case, and we proceed as before. (Determining this leftmost element is accomplished efficiently — using the variable called avail for bookkeeping, the block named "available-spot-found" is executed at most N/3 times.) So each swap executed by MAXSORT moves zero or one new element, other than N, into its proper place and never moves an element, other than N, from its proper place. Table 1 shows the current permutation, the contents of the elements of B, and the value of the variable avail for each step in the above example.

(Trace this sequence as a path in the graph in Figure 2.) *

iteration	permutation	b array	avail
	4 3 2 1	4 3 2 1	1
1			
	1 3 2 4	1 3 2 4	2
2			
	1 4 2 3	1 3 4 2	2
3			
	1 2 4 3	1 2 4 3	3
4			
	1 2 3 4	1 2 3 4	4

Table 1. MAXSORT trace for permutation 4321

To apply MAXSORT as a heuristic for the 8-puzzle, we take N = 9. Let $K_{MAXSORT}(p)$ denote the number of swaps executed by MAXSORT given permutation p. Then given an 8-puzzle instance with arbitrary initial node p and goal node q = 123456789, the function $K_{MAXSORT}$ can serve as a heuristic for the 8-puzzle: we take $K(s) = K_{MAXSORT}(s)$, obtained for any state s by executing MAXSORT on s and counting the number of swaps it executes. For example if p = 296134758 and q = 123456789, then MAXSORT executes seven swaps when sorting p, hence K(p) = 7. (It happens that $h_{8\text{-puzzle}}(p,q) = 9$ in this case.) For other goal nodes q, we compute $K_{MAXSORT}(p, q)$ by simply using MAXSORT to sort p into q by first transforming the instance (p, q) into the canonical form (p', 123456789) and then sorting p' by MAXSORT. The permutation p' is the image of p under the transformation that maps q into 123456789, e.g., if p = 258491367 and q = 485297361, then 4 in p becomes 1 in p', 8 in p becomes 2 in p', and so on, so that p' = 432159786. Note that every swap must include as one of its elements not 9, but rather the image of 9 under this transformation, i.e., 5 in this example. (The trivial generalization of the MAXSORT code to accomplish this transformation is omitted here for brevity.)

Since 8-puzzle $\subseteq_e$ 9MAXSWAP it follows that $h_{9MAXSWAP}(s, t) < h_{8puzzle}(s, t)$. Hence if $K_{MAXSORT}(s, t) = h_{9MAXSWAP}(s, t)$ it follows from the A* admissibility theorem that solution paths found using $K_{MAXSORT}(s)$ as heuristic function for the 8-puzzle are

---

* If tha number of edges in 9MAXSWAP were close to the number in the) 8-puzzle we could infer that the distance function of the former ia a close approximation of (hat of the latter, but such ia not the case Tha 8-puzzle, 9MAXSWAP and 9SWAP each has 9' • 362,880 mades Tha 9MAXSWAP graph haa 4.9' . 1,451,520 adfaa (Tha number of edges in the N-MAXSWAP graph is N' (N-D/2) Nodaa in tha 8-puzzle graph are incident to 2, 3, or 4 adfaa, in tha proportion 4.4.I, respectively Hence the number of edges •• 9'2.4/9 + 3-4/9 ♦ 4.1/9) / 2 - 9>=4/3 • 483,840, a factor of 3 fawar than in 9MAXSWAP By companion, tha 9SWAP graph haa 18.9' • 6,531,840 adfaa, a factor of 3 375 mora than in 9MAXSWAP (Tha *number* of edges in N-SWAP ,e N! N(N-I)/4)

* The general behavior of MAXSORT for 4MAXSWAP can be depicted graphically as follows Since the execution of MAXSORT for an arbitrary 4 element permutation corresponds to a path in the 4MAXSWAP graph (see Figure 2), the union over all 4-element permutation of the execution paths of MAXSORT (i.e., to sort into ascending order) is a subgraph of 4MAXSWAP. In fact this subgraph is a spanning tree of 4MAXSWAP (see Figure 3.)

of minimal length. * In words, to prove the premise is to show that for any 9-element permutations p and q, the number of swaps executed by MAXSORT in sorting p into q is minimal, i.e., it equals the minimum distance from p to q in the 9-MAXSWAP graph. Instead of attempting to prove this (its relevance beyond the example at hand being minimal), we note that it has been observed that heuristic functions that overestimate distance in a graph often are more efficient than similar functions that always underestimate the true distance ([Doran & Michie 1966], [Nilsson 1971], [Gaschnig 1977], [Gaschnig 1979]). Hence it may advantageous for efficiency in solving the 8-puzzle if $K_{MAXSORT}$ does not find minimal length paths in 9MAXSWAP, provided that a guarantee of minimal length solution paths in the 8-puzzle is not mandatory.

3.3. Experimental Performance Measurement Results

How efficient is the function $K_{MAXSORT}$ as a heuristic function for the 8-puzzle? To find out, we selected randomly a set of over 800 problem instances of the 8-puzzle, and solved each of these with A* using $K_{MAXSORT}$ as the heuristic function. As a measure of search efficiency, for each such problem instance (s_r, s_g) we measured the value of $X(s_r, s_g)$, the number of nodes expanded before a solution path is found to problem instance (s_r, s_g) when using $K_{MAXSORT}$ as a heuristic function with A*. This sample of problem instances is identical to the one used in [Gaschnig 1977] and [Gaschnig 1979], as is the experimental procedure followed, hence those results and the present results are comparable. The problem instances (s_r, s_g) in this sample are grouped together according to the value of $h(s_r, s_g)$, the length of a minimum length path in the graph from s_r to s_g.

Figure 4 plots the observed values of XMEAN(N), the mean number of nodes expanded as a function of depth of the goal node (i.e., the mean over all instances (s_r, s_g) in the sample such that $h(s_r, s_g) = N$). Superimposed in Figure 4 for comparison purposes are the analogous experimental data reported in Figure 1 of [Gaschnig 1977] and [Gaschnig 1979] for three particular 8-puzzle heuristic functions there called K_1, K_2, and K_3. Figure 4 shows that $K_{MAXSORT}$ is slightly better than $K_1(s)$, which computes the number of tiles out of place in tile configuration s with respect to the goal node. Figure 4 shows also that $K_{MAXSORT}$ usually expands many more nodes than do K_2 or K_3.** So it turns out that $K_{MAXSORT}$ is a relatively inefficient heuristic for the 8-puzzle, compared with these other known heuristic functions, but inspection of Figure 4 reveals that it expands far fewer nodes in solving the 8-puzzle than does breadth-first search.

---

* As a matter of convenience, we sometimes write K(s) instead of K(s, t) when t is understood implicitly. For example, in any given A* search node t is understood to be the goal node.

** K_2 computes the number of tiles out of place, each weighted by the orthogonal distance the tile must move to its desired spot, assuming no other tiles block the path; K_3 equals K_2 plus another term measuring the degree to which the outer tiles in s agree in rotational order to those in the goal node. See [Doran & Michie 1966], [Nilsson 1971], [Gaschnig 1977], or [Gaschnig 1979] for additional details.

4. Analysis of Distance Function Approximation

The experimental results in Figure 4 pose a challenge: why is $K_{MAXSORT}$ a relatively inefficient heuristic for the 8-puzzle, and is it coincidental that its performance is comparable to that of K_1? One approach to answering such questions is to determine how closely $K_{MAXSORT}(s, t)$ approximates $h_{8\text{-puzzle}}(s, t)$. Figure 5 plots experimental measurements of the range of $K_{MAXSORT}$'s estimates of distance to the goal node in the 8-puzzle vs. the actual distance to the goal. KMIN(i) is defined with respect to $K_{MAXSORT}$ to be the minimum value of $K_{MAXSORT}(s, t)$ over all node pairs (s, t) such that $h(s, t) = i$, and similarly KMAX(i) is the maximum of these values. Hence KMIN(i) and KMAX(i) bound the values computed by $K_{MAXSORT}$ for nodes that happen to be distance i from the goal. The data plotted in Figure 5 were recorded during the same experiment represented in Figure 4, in the following way. The value of $K_{MAXSORT}(s_r, s_g)$ was recorded for each problem instance (s_r, s_g) in the sample set. The value of $h(s_r, s_g)$ equals the length of the solution path found, which is of minimum length since $K_{MAXSORT}$ underestimates distance in the 8-puzzle graph. In addition we measured the value of $K_{MAXSORT}(s, s_g)$ for each node s along the solution path found for each problem instance in the sample set. Since the solution path found is of minimum length, the value of $h(s, s_g)$ is known for each such node s. In all, Figure 5 represents over 10,000 distinct observations of $K_{MAXSORT}(s, t)$ vs. $h_{8\text{-puzzle}}(s, t)$. Superimposed for comparison in Figure 5 are analogous experimental measurements of KMIN(i) and KMAX(i) for K_1, taken from Figure 2 in [Gaschnig 1977] or [Gaschnig 1979].

Figure 5 shows both that $K_{MAXSORT}$ is a poor approximation of $h_{8\text{-puzzle}}$, and that the approximation is only slightly better than that of K_1.

5. Discussion

We have described a general principle of problem similarity for path-finding state-space problems -- the edge subgraph and edge supergraph relations. We have demonstrated, using the 8-puzzle as case study, how this principle of "edge subgraph/supergraph transfer" can be exploited to aid in devising a heuristic function for a given problem, by transferring a heuristic from an "easier" problem to a related "harder" one. We have measured experimentally the efficiency of the resulting heuristic function devised for the 8-puzzle, demonstrating at least a limited practical utility.

These first results leave many questions unanswered. Our intention here has been merely to introduce the idea for further consideration. Hence the remainder of this discussion focusses on extensions to the present efforts.

Additional insight might be obtained by applying the transfer approach to additional problem graphs, including some in which the transfer problem is an edge subgraph of the given problem instead of an edge supergraph, as in the 8-puzzle case study. In particular, we note of the 8-puzzle case study that a transfer problem (9MAXSWAP) was readily apparent, that an efficient algorithm for 9MAXSWAP was readily devised, and that this algorithm proved to be a relatively inefficient heuristic function for the 8-puzzle. How formidable these hurdles are for

other problems remains to be determined.

Another objective is to attempt to devise for the 8-puzz!e another transfer problem which is a closer approximation to it than is the 9MAXSWAP graph, that is to identify a problem graph that is an edge supergraph of the 8-puzzle and an edge subgraph of 9MAXSWAP.

It may be interesting to apply this transfer concept in reverse fashion: given a particular heuristic function for a given problem, identify the graph to which it is equivalent. For example, one might attempt to determine whether a graph corresponding to the K_1 function is an edge supergraph of the 9MAXSWAP graph, or to identify a graph corresponding to the function K_2 A theory about the equivalence of problem graphs and heuristic functions along these lines is conceivable.

Acknowledgement. Richard Korf offered helpful comments on an earlier draft of this paper.

6. References

Doran, J., "An Approach to Automatic Problem Solving," in Machine Intelligence 1, N. Collins and D. Michie (eds.), American Elsevier Publ. Co., New York 1967.

Doran, J., "New Developments of the Graph Traverser," in Machine Intelligence 2, E. Dale and D. Michie (eds.), American Elsevier Publ. Co., New York 1968.

Doran, J., and D. Michie, "Experiments with the Graph Traverser Program," Proc. Royal Society of London, Series A, Vol. 294, 1966, pp. 235-259.

Ernst, G.W., Banerji, R.B., Hookway, R.J., Oyen, R.A., and Shaffer, D.E., "Mechanical Discovery of Certain Heuristics", Report 1136-A, Jennings Computing Center, Case Western Reserve Univ., Cleveland, Ohio, January 1974.

Gaschnig, J., "Exactly How Good are Heuristics?: Toward a Realistic Predictive Theory of Best First Search", Proc. Intl. Joint Conf. on Artificial Intelligence, Cambridge, Mass., August 1977, pp. 434-441.

Gaschnig, J., Performance Measurement and Analysis of Certain Search Algorithms, Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., May 1979.

Gelperin, D., "On the Optimality of A*," Artificial Intelligence, Vol. 8, 1977, *pp.* 69-76.

Harris, L, "Heuristic SEarch under Conditions of Error," Artificial Intelligence, Vol. 5, No. 3, North Holland Publ. Co., Amsterdam, 1974, pp. 217-234.

Hart, P., N. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Trans. Sys. Sci. Cybernetics, Vol. 4, No. 2, 1968.

Ibaraki, T., "Theoretical Comparisons of Search Strategies in Branch-and-Bound Algorithms," International Journal of Computer and Information Sciences, Vol. 5, No. 4, December 1976, pp. 315-344.

Jackson, P. Introduction to Artificial Intelligence, Petrocelli Books, New York 1974.

Martelli, A., "On the Complexity of Admissible Search Algorithms," Artificial Intelligence, Vol. 8, 1977, pp. 1-13.

Munyer, J., "Some Results on the Complexity of Heuristic Search in Graphs," Technical report HP-76-2, Information Sciences Dept., U. Cal. Santa Cruz, September 1976.

Munyer, J., and I. Pohl, "Adversary Arguments for the Analysis of Heuristic Search in General Graphs," Technical report HP-76-1, Information Sciences Dept., U. Cal. Santa Cruz, July 1976.

Nilsson, N., Problem Solving Methods in Artificial Intelligence, 1971.

Pohl, I., "First Results on the Effect of Error in Heuristic Search," Machine Intelligence 5, B. Meltzer and D. Michie (eds.), Edinburgh University Press, Edinburgh 1970 (1970a).

Pohl, I., "Heuristic Search Viewed as Path-Finding in a Graph", Artificial Intelligence, Vol. 1, 1970 (1970b).

Pohl, I., "Practical and Theoretical Consideration in Heuristic Search Algorithms," in Machine Intelligence 8, E. Elcock and D. Michie (eds.), Ellis Howard Ltd., Chichester, England 1977.

Polya, G., How to Solve It, Princeton University Press, Princeton, N. J. 1945.

Raphael, B., The Thinking Computer: Mind Inside Matter, W. H. *Freemsn* & Co., San Francisco 1976.

Rendell, L, "A Method for Automatic Generation of Heuristics for State-Space Problems", Report CS-76-10, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, February 1976.

Rendell, L, "A Locally Optimal Solution of the Fifteen Puzzle Produced by an Automatic Evaluation Function Generator," Report CS-77-36, Dept. of Computer Science, University of Waterloo, December 1977.

Samuel, A., "Some Studies in Machine Learning Using the Game of Checkers," in E. Feigenbaum and J. Feldman (eds.), Computers and Thought, McGraw-Hill 1963, pp. 71-105.

Samuel, A., "Some Studies in Machine Learning Using the Game of Checkers II. Recent Progress", IBM J. Research and Development, Vol. 11, No. 6, (1967), pp. 601-617.

Schofield, P. "Complete Solution of the Eight Puzzle," in Machine Intelligence 1, N. Collins and D. Michie (eds.), American Elsevier Publ. Co., New York 1967.

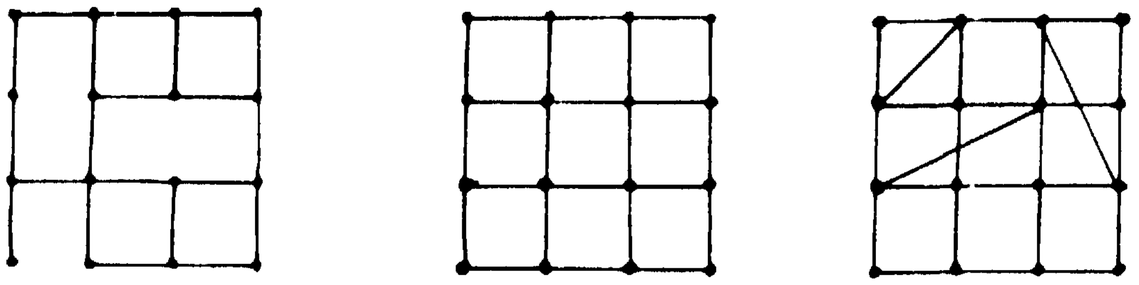
Slagle, J., and Farrell, C, "Experiments in Automatic Learning for a Multipurpose Heuristic Program," Communications A.C.M., Vol. 14, No. 2, *pp.* 91-99.

Swinehart, D., and B. Sproull, SAIL, Stanford Artificial Intelligence Laboratory Operating Note 57.2, January 1971.

Vanderbrug, G., "Problem Representations and Formal Properties of Heuristic Search/' Information Science, Vol. 11, No. 4, 1976.

Wickelgren, W., How to Solve Problems, W. H. Freeman and Co., San Francisco 1974.

Givtn t touristic function K for a given problem graph P1, or* can define another graph P2 thustht nodtt in P2 corrttpond to tht nodtt in PI; an edgentcta two nodtt a and t in P2 iff K (s , t) = 1 wheather or not P2 satifies the conditions of probltm graph dtptndt on tht vtlut computed by K.



(a) MSUB44 (b) M44 (c) MSUP44

Figure 1. A "Manhattan street pattern" graph and two variants thereof

Note that MSUB44 $\leq_t$ M44 $\leq_t$ MSUP44

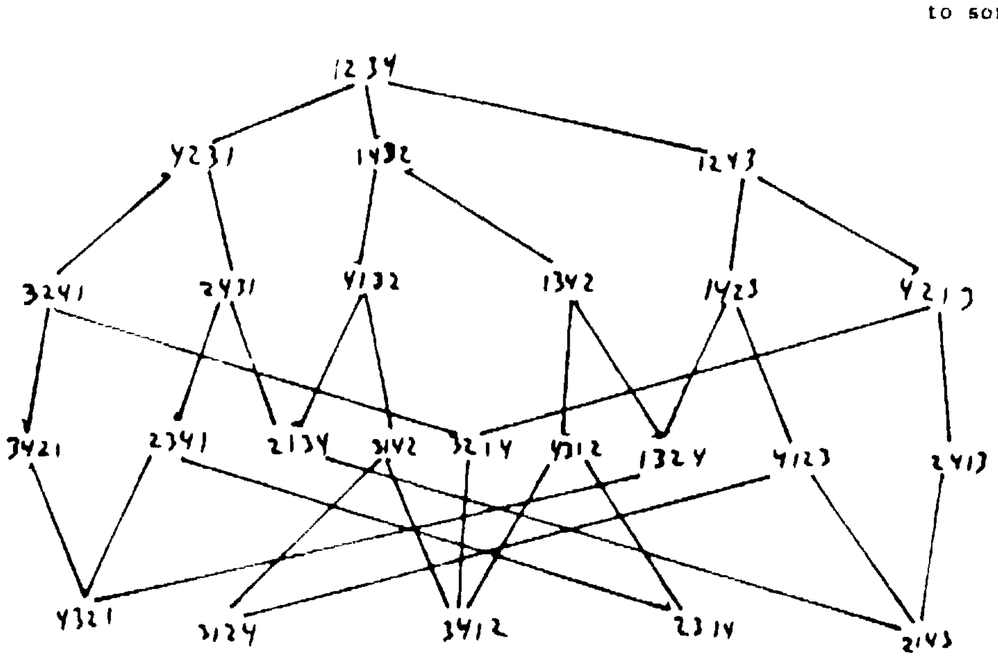


Figure 2 4MAXSWAP graph

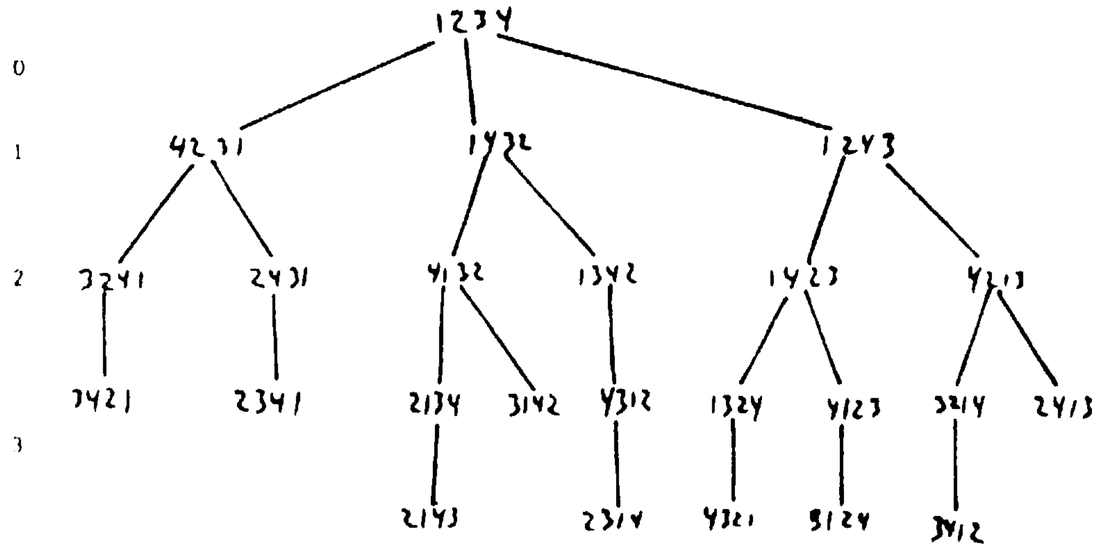


Figure 3. The union of the paths taken by MAXSORT in sorting any 4-element permutation is a spanning tree of 4MAXSWAP (and hence an edge subgraph of 4MAXSWAP; compare with Figure 2)

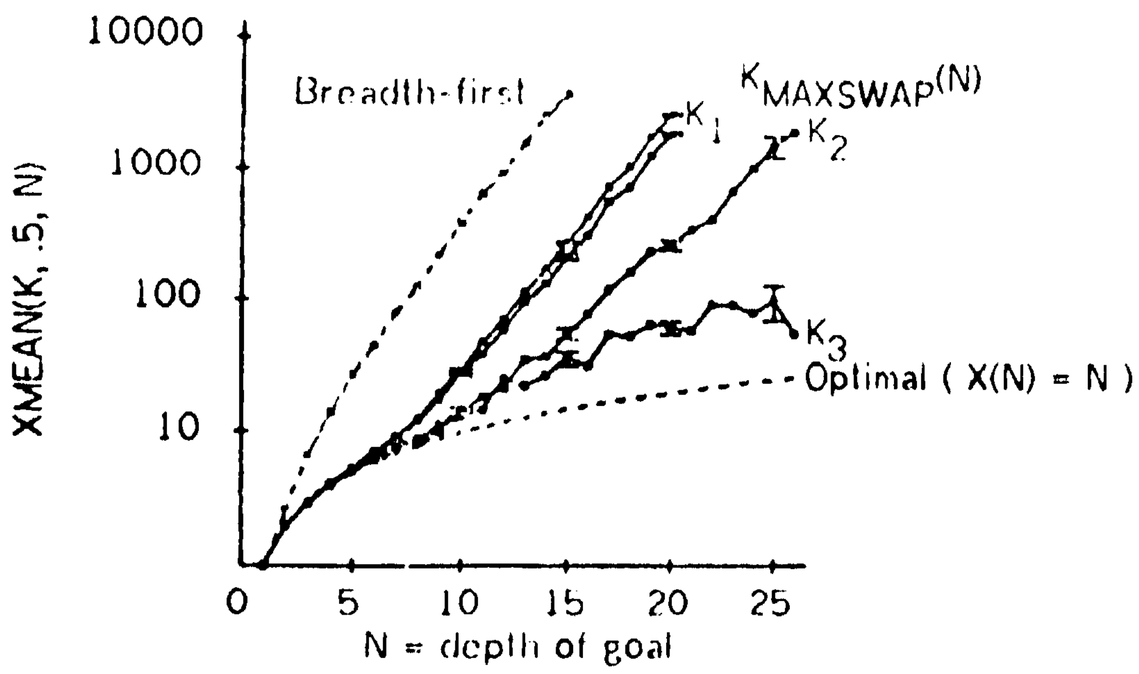


Figure 4 Mean number of nodes expanded vs. depth of goal
A* search of the 8-puzzle, comparing heuristic K_{MAXSAP} and $K_{WMAXSWAP}$ with heuristics K_1, K_2, and K_3
40 problem instances for most values of N
760 to 895 algorithm executions per heuristic function

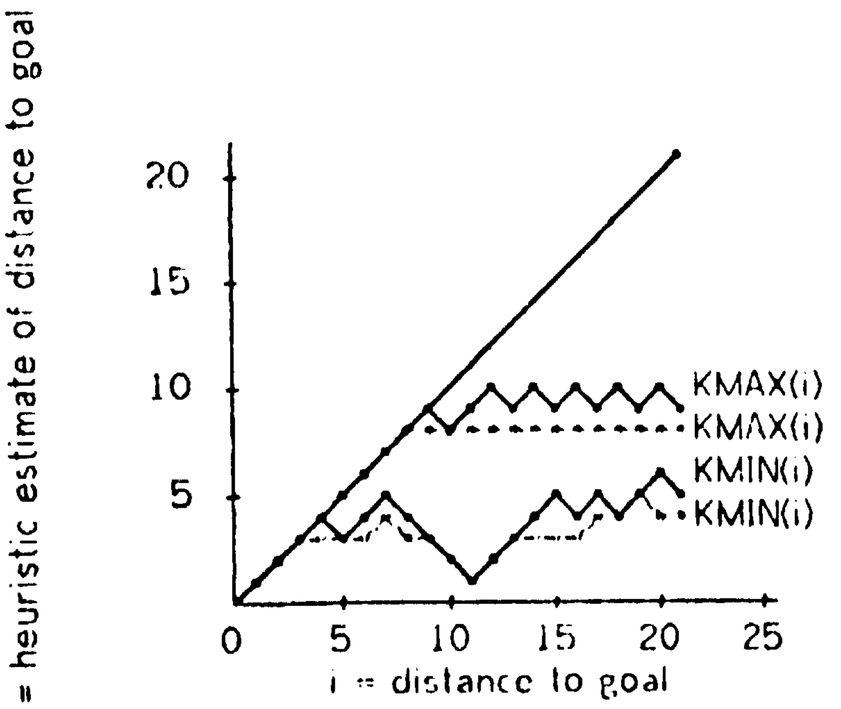


Figure 5
Heuristic estimate of distance to goal vs. actual distance
8-puzzle heuristic $K_{MAXSWAP}$ (solid) compared with heuristic K_1 (dash)

PRELIMINARY PERFORMANCE ANALYSIS OF THE PROSPECTOR CONSULTANT SYSTEM FOR MINERAL EXPLORATION

John Gaschnig
Artificial Intelligence Center
SRI International
Menlo Park, CA 94025

Abstract

To demonstrate that the performance of an expert knowledge-based system is comparable to that of the experts it emulates, it is useful to subject the system to an appropriate objective evaluation. The Prospector consultant system is intended to aid a geologist in evaluating the mineral potential of an exploration site. Here we report the results of a preliminary performance analysis of three Prospector ore deposit models. Using data from known deposits as test cases, we compare the system's performance in detail with analogous target values supplied by the model designer based on the same input data. These calibration results measure how well a model embodies the model designer's intentions, and identify particular sections of a model that would benefit from revision. We discuss limitations of the present experiments and future work. Put briefly, we report how work-a-day performance analysis instruments can accelerate the model design and refinement process in expert systems.

1 Testing Prospector Models

Performance analysis of exert systems can serve several useful purposes: (1) to demonstrate objectively and convincingly that the competence of a system is (or is not) comparable to that of the experts it emulates; (2) by exposing the inner workings of a complex system and knowledge base, to aid the ongoing efforts of the system implementors to achieve a level of performance that merits a formal evaluation by outside experts. Performance analysis techniques can be applied to systems of diverse domains of expertise, such as medical diagnosis (e.g., [10]), speech understanding (e.g., [6]), and game playing (e.g., [4]). In this paper we report another instance of the performance analysis of an expert system — the Prospector consultant system for mineral exploration. The present work emphasizes the work-a-day use of performance analysis as an aid to system implementors in refining a knowledge base. The performance data provide an objective, detailed, quantitative measure of the current performance of a model, and pinpoint those sections of the model that are not performing exactly as intended, thereby establishing priorities for future revisions. Hence besides reporting specific experimental results for the Prospector system, the present work provides an example of an experimental methodology that may be adapted to (or improved upon in adapting it to) other expert systems.

The Prospector system is intended to emulate the reasoning process of an experienced exploration geologist in assessing a given prospeot site for its likelihood of containing an ore body of the type represented by the model he or she designed [2] [3]- Like other rule-based expert consultation systems (e.g., [8], [5], [7]), Prospector engages the user in a dialogue, accepting volunteered observations and prompting the user with questions about relevant factors not volunteered. Here we report the results of preliminary performance measurement experiments for a Prospector ore deposit model concerning porphyry copper deposits (denoted PCDA).

The basis for a Prospector computation is a set of ore deposit models formulated by expert economic geologists. We perform a separate evaluation for each model. In this paper "Prospector" always means the Prospector system executing some particular model.

2 Experimental Methodology

We wish to compare Prospector to human geologists in terms of the accuracy with which the presence or absence of a particular type deposit at a particular location can be predicted, but there exist no standard quantitative measures of human performance. Lacking an absolute scale of comparison, we employ instead a relative scale; our principal objective here is to measure how closely Prospector's conclusions agree with those of the model designer.

To compare Prospector's conclusions with those of its model designer, we ask the model designer to express his or her conclusions on the same -5 to 5 certainty scale used by Prospector. This scale measures degree of subjective certainty: the value 5 indicates absolute certainty that a

geological characteristic is present or that a final or intermediate model conclusion is true; -5 indicates absolute certainty that a characteristic is not present or that a conclusion is false; zero indicates no information; values between 0 and 5 and between 0 and -5 denote degrees of uncertainty. Values on this subjective certainty scale map (in a piece-wise linear, one-to-one onto fashion) to probability values, which are used exclusively for computation purposes. (Prospectors reasoning about uncertainties is based on a modified Bayesian probability updating scheme [1]; see [9] for a description of the subjective certainty scale used in MYCIN.) Since Prospector models have a hierarchical structure, we wish to compare both Prospector's overall conclusion and its major subconclusions with those of the model designer.

The test sites chosen here are all exemplars of the PCDA model. The experimental procedure is as follows: (1) for each test site, the model designer completes a questionnaire listing all the questions (about geological characteristics of the test site) asked by the model; (2) for each such input data set, the model designer assigns to each of several major sections of the model a target certainty value, expressing his Judgment as to the extent to which the conclusion represented by that section is established for that test site; (3) Prospector is run once for each input data set, and the final certainty values assigned to each node in the model are recorded; (4) these raw performance and target values are analyzed in ways described subsequently.

Additional details about the experiment methodology are given in [2]. The fact that all the present test cases are exemplars of their respective models, and the fact that the model designer supplied the input data concerning the test cases, are limitations; the present tests are more necessary than sufficient conditions for good performance. Despite these limitations, the preliminary results reported in subsequent sections have proved useful in the ongoing model refinement process.

3 Performance Analysis of the PCDA Model

We now evaluate one version of a model for a class of porphyry copper deposits (PCDA) designed by Prof. Marco Einaudi of Stanford University. Input data was available for three test cases, namely the known deposits called Yerington (Nevada), Bingham (Utah), and Kalamazoo (Arizona). Prospector scored 4.769, 4.721, and 4.756, respectively for these three sites, indicating acceptable performance of the PCDA model (since these three sites are considered examples of the PCDA model).

The target certainty values assigned by Prof, Einaudi for calibration purposes are given either in the form of a single number (on a -5 to 5 scale), or as two numbers establishing an upper and lower bound on a certainty interval. The estimates are listed in Table 1 on the left, with the scores as determined by execution of Prospector recorded on the right. For brevity, Table 1 lists data only for the Yerington deposit. The symbols PCDA, FPTS, FRE, etc., in Table 1 denote nodes in the PCDA model hierarchy corresponding to major sections and subsections of the model (indicated by indentation).

Table 1. Prospector Scores for Several Levels ofthe PCDA Model — Yerington Deposit

Name of Model Node	Einaudi's Estimate	Prospector Score
PCDA	4.5 to 5.0	4.769
FPTS	4.5 to 5.0	4.528
FRE	4.5	4.540
FPCDAIS	4.5 to 5.0	4.787
FCDS	5	4.524
FIS	5	4.744
FAMR	4.5 to 5.0	4.225

In most cases shown in Table 1 Prospector agrees very closely with Prof. Einaudi's estimate. Conclusions about these data can be expressed quantitatively by first identifying the values in Table 1 with a concise notation, then defining a simple formula for the relative error of Prospector in predicting Prof. Einaudi's estimates. We let C(X, Y, Z) denote the certainty score given to model node Z by agent X for site Y, where X denotes Prospector or Einaudi. For example, C(Prospector, Yerington, FPCDAIS) - 4.787. When Einaudi are an interval of certainty values instead of a single value, we use the midpoint of the interval as the value of C. Then an error measure is given by

$$E(Y, Z) = \frac{C(Einaudi, Y, Z) - C(Prospector, Y, Z)}{C(Einaudi, Y, Z)}$$

For example, E(Yerington, FPCDAIS) = (4.75 - 4.787) / 4.75 =-.008, meaning that Prospector's prediction is accurate to within 0.8% in this case. Supplementing Table 1 with analogous data for the Bingham and Kalamazoo deposits, we oan compute the value of E for 3 * 7 = 21 different instances. For 5 of the 21 data points Prospector predicted Einaudi's estimate to within 1%, while 15 of the 21 data points show agreement to within 10%. The grand average over the 21 data points is 10.3%. For convenience, we list these 21 values of E in Table 2, expressed as percentages. In Table 2, "Average" denotes the average of absolute vales.

Table 2. Relative Error (E) of Prospector Scores as predictors of Einaudi's Estimates (derived from data in Table 1 and analogous data)

	Yer.	Bingham	Kal.	Average
PCDA	-.3%	-4.9%	-11.9%	5.7%
FPTS	4.7	-18.6	-4.7	9.3
FRE	-.9	-13.6	49.0	21.2
FPCDAIS	-.8	.4	-.9	.7
FCDS	9.5	51.9	5.6	22.3
FIS	5.1	5.1	5.1	5.1
FAMR	11.1	5.6	5.6	7.6
Average:	4.1	14.3	11.8	10.3

Note that the average data in the rightmost column suggest the FRE and FCDS sections as candidates for model revisions. More extensive data and interpretation are found in [2].

4 Conclusions and Future Work

We have applied a simple experimental methodology for measuring quantitatively the performance of Prospector ore deposit models in some detail. The results indicate that the PCDA model reflects fairly faithfully the Judgments of its designer. Although covering a limited number of cases that include only exemplars of the model, these performance analysis results prove useful in identifying particular sections of a model that would most benefit from "fine-tuning", hence establishing precise priorities for model revisions. Directions for future work include analogous experiments on additional cases, sensitivity analysis of input data and rule strength values, and incorporation of performance analysis into Prospector's explanation system.

ACKNOWLEDGEMENTS

This work was supported in part by the Office of Resource Analysis of the U.S. Geological Survey under Contract 14-08-0001-15985, and in part by the RANN Division of the National Science Foundation under Grant AER77-04499. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author, and do not necessarily refleot the views of either the U.S. Geological Survey or the National Science Foundation.

REFERENCES

1. Duda, R. 0., P. E. Hart, and N. J. Nilsson, "Subjective Bayeslan Methods for Rule-Based Inference Systems," National Computer Conference 1976 (AFIPS Conference Proceedings, Vol. 45), pp. 1075-1082, 1976.
2. Duda, R. 0., P. E. Hart, P. Barrett, J. Gaschnlg, K. Konolige, R. Heboh, and J. Slocum, "Development of the Prospector Consultation System for Mineral Exploration," Final Report, SRI Projects 5821 and 6415, Artificial Intelligence Center, SRI International, Menlo Park, California (October 1978) (1978a).
3. Duda, R. 0., P. E. Hart, N. J. NIlsson, and G. L. Sutherland, "Semantic Network Representations in Rule-Based Inference Systems," Technical Note 136, Artificial Intelligence Center, SRI International, Menlo Park, California (March 1977). Appears also in Pattern-Directed Inference Systems, D. A. Waterman and F. Hayes-Roth (eds.) (Academic Press, New York, 1978) (1978b).
4. Gillogly, J. J., "Performance Analysis of the Technology Chess Program," Report CMU-CS-78-109, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania (March 1978).
5. Martin, N., P. Friedland, J. King, and M. Stefik, "Knowledge Base Management for Experiment Planning in Molecular Genetics", Proc. Fifth IJCAI, Cambridge, Mass., August 1977, pp. 882-887.
6. Paxton, W. H., "A Framework for Speech Understanding,", Technical Note 142, Artificial Intelligence Center, SRI International, Menlo Park, California (June 1977).
7. Pople, H. E., "The Formation of Composite Hypothese in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning," Proc. Fifth IJCAI, Cambridge, Mass., August 1977, pp. 1030-1037.
8. Shortliffe, E. H., Computer-Based Medical Consultations: MYCIN, Elsevier, New York (1976).
9. Shortliffe, E. H., and B. G. Buchanan, "A Model of Inexact Reasoning in Medicine," Math. Biosci., Vol. 23 (1975), pp. 351-379.
10. Yu, V. L., et al., "Evaluating the Performance of a Computer-Based Consultant," Heuristic Programming Project Memo HPP-78-17, Dept. of Computer Scienoe, Stanford Univ., September 1978.

The Role of Plans in Automated Consultation

Michael R. Gcnesereth
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Mass. 02139

Abstract: Consultation is a method widely used in computer centers for helping people to use unfamiliar computer systems or languages. Unfortunately, consultants are scarce, expensive, and often unavailable when needed. Thii paper describes the implementation of an automated consultant for MACSYMA, called the Advisor.

The Advisor's implementation is based on the assumption that the MACSYMA novice behaves in accordance with a standard heuristic problem solving algorithm, called MUSER. This assumption is supported by a body of data on the problem solving behavior of MACSYMA users. In solving his problem, the user implicitly generates a goal-subgoal graph, called a "plan". The plan is a direct proof that the commands used actually achieve the goal (in terms of the user's beliefs), and the problem solving algorithm constitutes a grammar for such plans.

The key to the consultation process is the analysis of the user's plan. First, the plan is reconstructed by a combined process of dialogue and automatic plan recognition, and then the underlying beliefs are checked for errors. Because of the MUSER model, the Advisor is able to diagnose not only standard errors but also more general misconceptions.

1. *Introduction*

Consider a person trying to solve a problem with a cornput er system he docs not fully understand, and assume that he has encountered a difficulty due to his lack of knowledge of that, system. For example, he might be unaware of the capabilities available or not know the names of its commands, or he might get an unexpected result he cant explain. The simplest way for him to acquire just the information he needs is to ask a consultant for help. Then, armed with the consultant's advice, he may surmount the difficulty and solve, the problem. Consultation is a method widely used in computer centers as well as in domains like business, law, and medicine; and, as computer technology becomes more pervasive and computer systems more complex, the need for consultation will grow. Unfortunately, human consultants are scarce, expensive, and often unavailable when needed. This paper describes the implementation of an automated consultant to fill this need.

The type of difficulty described above can be called a *resource usage difficulty* to distinguish it from a difficulty inhering in the problem or a difficulty in learning about the resource without a specific problem in mind. A resource usage difficulty is one which is encountered in trying to solve a problem using a resource and due to the problem solver's lack of knowledge of that resource. The consultant's task is to remedy the deficiency in the user's knowledge underlying a resource usage difficulty.

The hardest resource usage difficulty to handle is a violated expectation which the user can't explain. In other cases, the deficiency in the user's knowledge is implicit in his question, and the answers can simply be looked up in a data base (e.g. "how do 1 invert a matrix?" or "what are the arguments to MAP?"). In the case of an unexplainable violated expectation, even the user is unaware of the nature of his "misconception". The consultant's first step, therefore, is to identify this misconception, after which he can generate his advice.

The method described here for the identification of the user's misconception is based on the assumption that the user's actions are "rational", i.e. that he has some "plan" for achieving his goal. This plan explains why he chose the commands he did in terms of his beliefs about the input-output specifications of those commands. The key to the identification of the user's misconception is the reconstruction and debugging of this plan.

The plan reconstruction procedure is based on the assumption that the user of a complex resource behaves (up to ordering of strategics) in accordance with a stand-

and heuristic problem solving procedure, called ,MUSER. MUSER is a largely domain independent problem solver with the ability to invoke domain dependent methods when necessary. In solving a problem the user implicitly generates a graph of goals and subgoals, in accordance with this algorithm. MUSER is guaranteed to produce only correct solutions (given a correct data base), and so this goal-subgoal graph constitutes a direct proof that the user's commands achieve his goal (in terms of his beliefs). In the consultant described here, the notion of the user's plan (as proof) is identified with the goal-subgoal graph of his solution.

With this view, plan reconstruction is a complex "parsing" problem, in which the user's actions are the "sentences" and in which the MUSER algorithm is the "grammar". The plan reconstruction process consists of heuristic suggestion of partial parsings together with forward and backward symbolic evaluation of the problem solving procedure to filter these suggestions. Some of the partial parsings are suggested by the patterns in a library of freqiir.iit.ly recurring errors. Others arc suggested by correct facts about MACSYMA and the structure of the MUSER model. Although the recognition of standard errors plays a large part in effective consultation, it is possible to reconstruct plans containing misconceptions using just the MUSER model and correct facts about MACSYMA.

The method of consultation is exemplified by an automated consultant for MACSYMA novices, called the Advisor. It is a program distinct from MACSYMA, with its own separate data base and expertise. The Advisor accepts a description of a resource usage difficulty from its user and generates advice tailored to his need. Eventually, the Advisor should be able to converse in English, however, at present, all communication is conducted in the Advisor's LISP-likc data base language.

This does not mean that the consultation technology described here is in any way restricted to MACSYMA. Consultation is useful in any situation where the user of a resource must do some problem solving with partial know ledge of the resource. The Advisor could equally well be applied to any complex, interactive computer system, such as a text editor, a document production program, or a data base query system.

2. *An Example of* a *Resource* Usage *Difficulty*

As a concrete example of a resource usage difficulty, consider the MACSYMA interaction and consultation session shown in figure 1. The command lines (e.g. C6) are typed by the user; the display lines (e.g. D6) are MACSYMA's response. A command line may be terminated by a ";" or a "$"; if the latter is used, the display line is suppressed (e.g. C7). Expressions are entered in linear form (e.g. (3-2*Z)/Z); ":" is the assignment operator; and other commands are applied using functional notation (e.g. RATSIMP(D4)). Parentheses delimit compound statements.

(C6) RATSIMP(D4);

(D6) $(-3X-3)Z+X^2+3X$

(C7) (A:COEFF(D6,X,2),B:COEFF(D6,X,1),
C:COEFF(D6,X,0))$

(C8) (-B+SQRT(B↑2-4*A*C))/(2*A);

(D8) 0

(C9) HELP()$

User: I was trying to solve *D6* for *X* and I got 0.

Advisor: Did you expect *COEFF* to return the coefficient of *D6*?

User: Yes, doesn't it?

Advisor: *COEFF*(*EXP*, *VAR*, *POW*) returns the correct coefficient of VAR^{POW} in *EXP* only if *EXP* is expanded with respect to *VAR*. Either expand first and then use *COEFF* or use *RATCOEF*.

User: Ok, thanks. Bye.

Fig. 1 - An example of MACSYMA consultation

In (his example, the user's difficulty was due to his "misconception" that COEFF always returns the coefficient of its argument. This misconception gave rise to the erroneous use of COEFF in line C7 without a preliminary expansion of DG. However, the user didn't observe (his "bug" until the zero appeared in line D8. Being unable to explain the violation of his expectations, the user then appealed to the on-line consultant for help.

One common suggestion for eliminating such difficulties is to switch the names COEFF and RATCOEFF. One reason (his hasn't been done is that COEFF is much more efficient than RATCOEFF; by giving COEFF the more common name, the hope is that users will try the more efficient command first. Even if they were switched, not all problems would be eliminated. For example, a user who wants to solve a polynomial over a polynomial domain would need a version of COEFF with the present

semantics; seeing a command called COEFF he might believe that it does what he wants and would get the wrong answer. So long as there are commands with different interpretations depending on the the type of problem the user is trying to solve, there will be violated expectations and a need for consultants.

3. The *MUSER Model* and P/ans

In order to discover the misconception underlying the user's violated expectation, it is first necessary to know why he expected it. The user must have had in his mind some argumcnt explaining how his particular sequence of commands was structured to achieve his expectation. For single commands, an adequate explanation is simply a belief about the command involved. For a sequence of several commands, the explanation requires an enumeration of the intrinsic properties of the individual commands together with an argument showing how they co-operate to achieve the expectation A *plan* is a formalization of this notion of co-operation, essentially a formal proof that the command sequence results in the expectation.

This view of plans raises two questions. How does one charact crize plans, i.e. is there any syntax for or constraint on plan structure? Second, how does the Advisor acquire knowledge of the user's plan and how is it debugged? The an swer to the first question is given as a formal model for the problem solving behavior of the MACSYMA novice. The Advisor's techniques for plan reconstruction and debugging are described in the sections thereafter.

3.1 The MUSER Model

In solving problems MACSYMA novices do not randomly explore all possible combinations of commands. In fact, the analysis of more than a hundred problem solving sessions supports the conjecture that novices act in accordance with a standard heuristic problem solving procedure called MUSER. The differences between users stem from differences in their knowledge of MACSYMA and their ordering of strategies. The strategics themselves seem to be common.

The MACSYMA user typically has a mathematical problem he is trying to solve and approaches MACSYMA for its powerful abilities at algebraic manipulation. Thus, in any MACSYMA problem, there are two distinct domains involved, viz. mathematics and MACSYMA. In the MUSER model, knowledge about these domains is grouped into three categories (see figure 2). The "Task Environment" is a body of mathematical facts and procedures; the "Static Model of MACSYMA" contains the input-output specifications of MACSYMA's commands; and the "State Description" is a data base of facts about the current MACSYMA (such as variable bindings and function definitions). The MUSER problem solver is a domain independent procedure which accepts a problem statement in a domain independent formalism similar to predicate calculus and produces a sequence of MACSYMA commands that solves it. In doing so, it utilizes the facts stored in the data bases and implicitly generates a goal-6ubgoal graph, or "Plan".

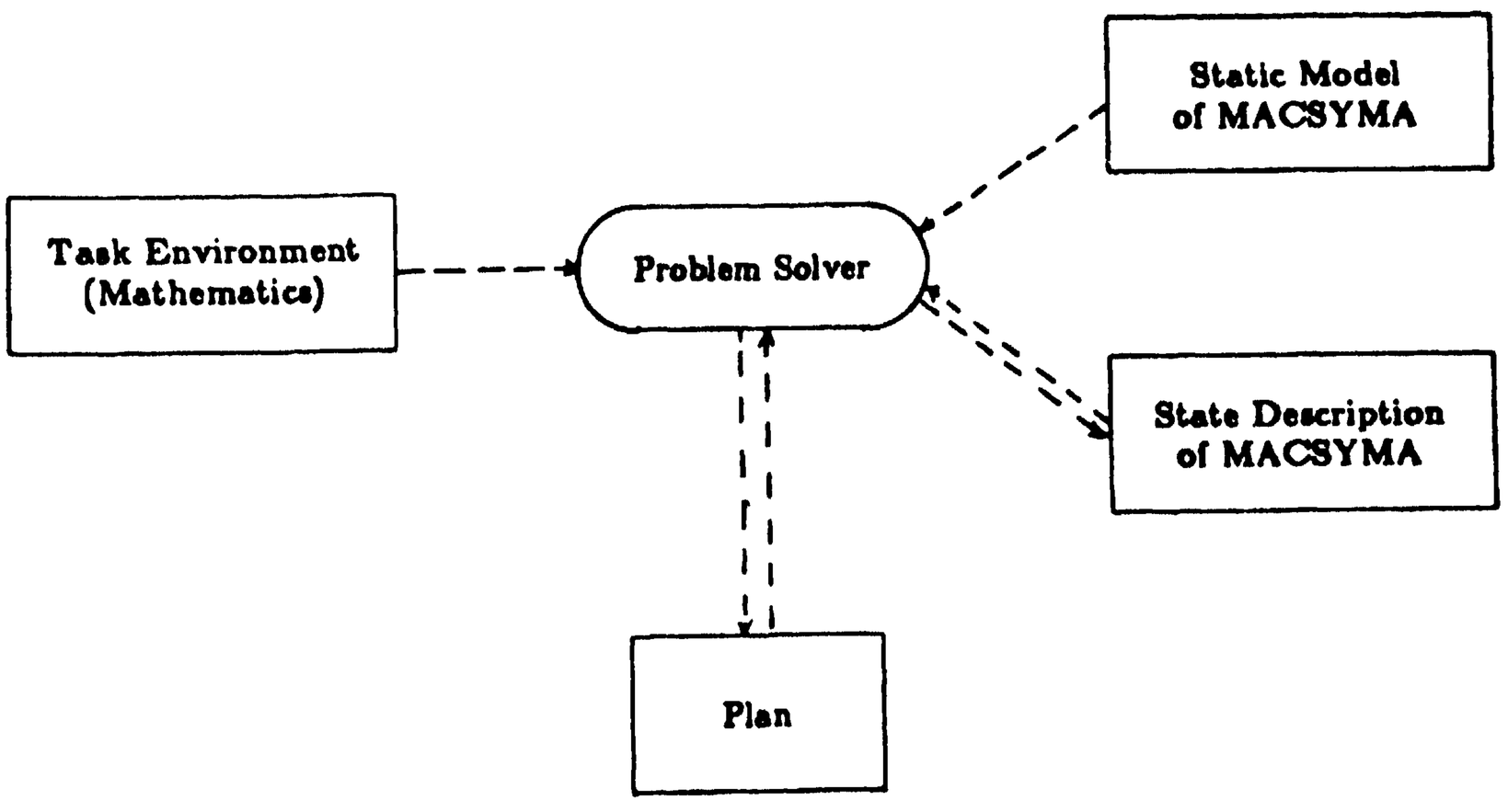


Fig. 2 - MUSER's data structures

The MUSER problem solver is guaranteed to produce only correct Rotations (given a correct data base), and so this goal-subgoal graph constitutes a direct proof that the commands achieve the goal (in terms of the facts in the data bases). Any errors in a solution must derive from incorrect entries in one of the data bases. A resource usage difficulty often arises due to an error in the user's static model of MACSYMA, and the purpose of consultation is to correct such errors.

The MUSER problem solver is represented as a "parameterized procedural net", or PPN. A PPN is a hierarchical network of actions at varying levels of detail in which each action node has associated with it sets of input and output objects and sets of 'prerequisites" and "postrcquisitcs" (conditions which must be true for the action to succeed and those which become true after its execution). In the procedural nets described by Sarerdoti [Sacerdoti), each action node is described in terms of specific objects in the world being modeled. In a parameterized procedural net, the flow of data into and out of an action is described in terms of the inputs and outputs of other actions without naming any specific real world objects Thus, a PPN is a kind of partial program. Except for its hierarchical character and the explicit rep* resentation of prerequisites and postrequisitcs, this formalism is similar to the dataflow graphs described by Dennis [Dennis).

Figure 3 shows a very small fragment of the MUSER PPN The square boxes represent problem solving methods at varying levels of detail Methods at different levels are connected by vertical "supergoal-subgoal" links. Links with crossbars denote conjunctive subgoals; those without, denote alternativesugbgoals. The rounded boxes represent data base assertions In order to simplify the presentation and emphasize their hierarchical character, PPNS are drawn here as trees without control or dataflow links and with various problem solving methods duplicated (e.g. "Obtain an object"). In looking at such abbreviated diagrams, the reader should bear in mind that the action nodes are. actually interconnected through their inputs and outputs and through their prerequisites and post requisites

Each problem solving method in MUSER satisfies a particular type of goal by a particular method or a choice of methods. For example, the goal of "Obtain an object" is to "obtain" the object specified as its argument (either to print it out or pass it as argument to a function). In MACSYMA one can do this cither by finding an already computed expression (the "Find a variable" method) or by constructing one anew (the "Construct an object" method).

In the "Find a variable" method, MUSER searches its data base to find a variable with the goal as value and then evaluates it. In the terms used in figure 3, it looks a variable *v* such that *Val(v)* = *g,* where *g* is the specified goal.

The "Fetch" method is MUSER's data base access function It searches the data base for a fact of the form specified in the assertion attached to it. (This includes more than exact matches. "Fetch" employs a fast but limited inference algorithm [Gcnescrcth 1977) to do property inheritances and set intersections as well.) In this case, the goal *g* would be supplied as an input to "Fetch" and the variable *v* would be returned as output.

The "Action" method is MUSER's way of executing a command in MACSYMA. Given a command and a set of objects as inputs, "Action" will cause MACSYMA to call the command with the objects as arguments and will return an internal representation of the answer as its output. In the case of "Find a variable", the command is MACSYMA's evaluator MEVAL.

The alternative to obtaining a pre-computed object is to construct a new one. The goal of the "Construct an object" method is to produce an object defined as /(a). The strategy is to search the data base for a command c which computes /, given preconditions *p* on the argument and general prerequisite *q,* as shown in figure 3. The outputs of "Fetch" are the command c, the preconditions p, and the prerequisite q. Having found such a command, the method then obtains the argument *a* in a form that satisfies *p,* achieves *q,* and executes the command. The extension to multiple argument functions is straightforward.

The arguments of a function are obtained by the "Obtain arguments" method If a particular argument must satisfy a precondition, MUSER uses the "Obtain and Convert" method If not, it simply calls "Obtain" recursively.

The strategy of "Obtain and Convert" is to obtain the argument and then transform it to satisfy the precondition. In doing so, it searches for a suitable transformation, achieves its prerequisites/and applies the transformation, as shown.

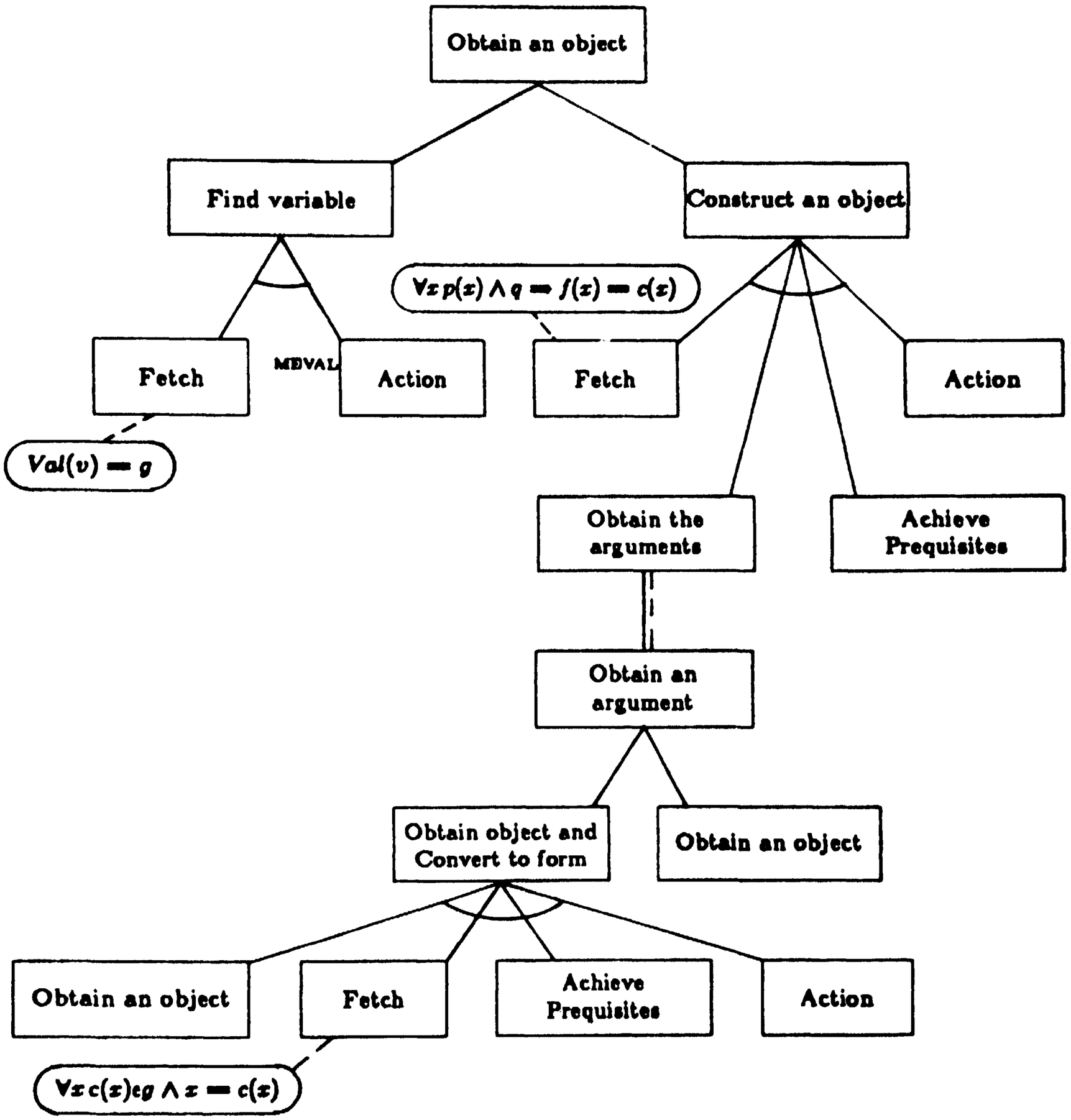


Fig. 3 - A fragment of the MUSER Model

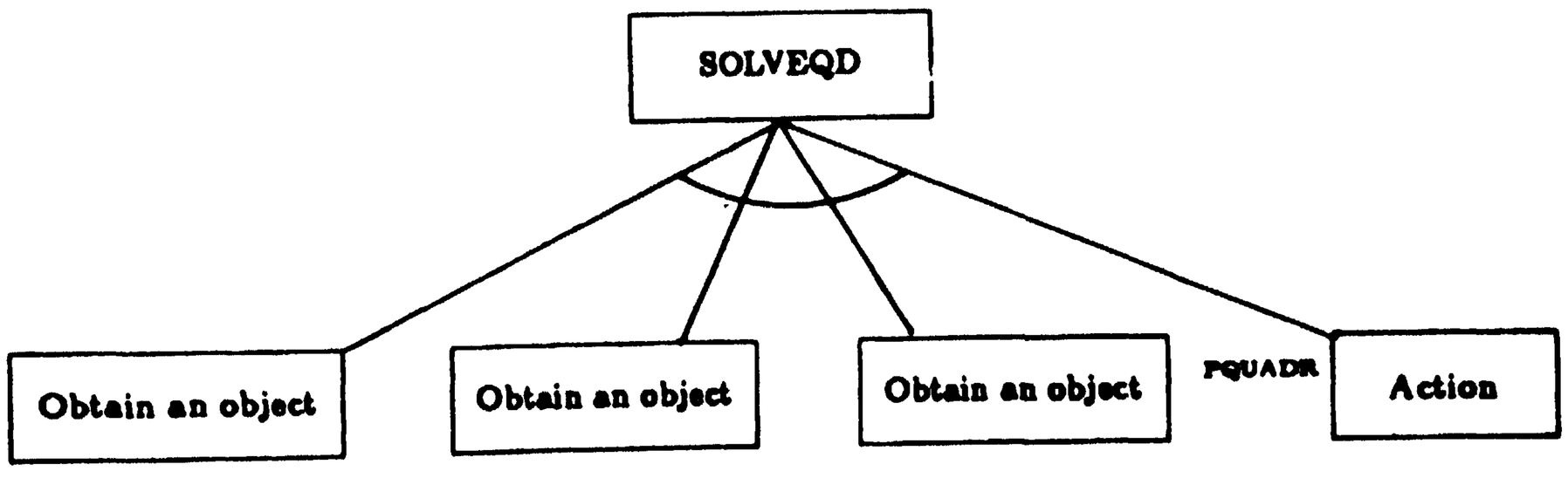


Fig. 4 - The domain dependent method SOLVEQD

The full MUSER model is considerably larger than the small fragment shown here (see [Gcncsercth 1978]). In addition, MUSER has the ability to invoke domain dependent procedures when appropriate. As MUSER generates each subgoal in solving a problem, it searches its data base for any domain dependent methods with a matching goal. If a procedure is found, it is used just as if it were a method in the MUSER model. As an example, consider the procedure SOLVEQD shown in figure 4. SOLVEQD computes the coefficients of the quadratic and then plugs the results into the quadratic formula. If MUSER needed to obtain the root of a quadratic, it would find SOLVEQD and invoke it with the corresponding "Obtain an object" method as supergoal.

3.2 Plans

A plan is the result of executing a parameterized procedural net. with particular bindings for the parameters. More formally, a plan is a PPN in which all network objects are bound to real world objects, all parent-child cycles have been eliminated by copying, and all disjunctions have been resolved Additionally, a plan includes pointers to all data base assertions accessed in constructing it.

Figure 5 shows a plan for computing the coefficient C1G6 of X in an expression G6. The data base documents that the COEFF command computes the coefficient of an expression so long as it is expanded. In trying to obtain the argument in expanded form, MUSER finds that the EXPAND command returns an expanded version mathematically equivalent to its argument. Hence, EXPAND is called on the value of DO, and the result (GE) is passed as argument to COEFF. Note that this plan is completely correct, unlike the user's plan in figure 1.

4. Overview of the *Consultant*

In handling a violated expectation, the Advisor'6 first step is to reconstruct the user's plan. When the Advisor is called, it has a data the relevant sequence of MACSYMA commands; and, if the user has not already supplied it, the Advisor obtains from him a statement of his overall goal. The commands are assumed to represent the fringe of a plan in whirh the root is the user's goal. The purpose of plan reconstruction is to fill in the intervening structure.

The Advisor's plan reconstruction procedure has three parts The first step is to convert the user's actions into a single-level dataflow graph. The second step is mechanical plan recognition. In principle, the plan could be obtained by asking the user to describe in detail how he intended to achieve his goal However, human consultants are able to bypass most and sometimes all such questioning, and it would be preferable if the Advisor could do as well. Furthermore, in practice the user often won't know what terms to use in describing his intentions, or the description might be too tedious for his patience to bear Therefore, to be effective, a consultant must reconstruct as much of the user's plan as possible with as little questioning as possible. The third phase of plan acquisition is user interrogation When the Advisor's plan recognition procedure fails to produce a complete plan, the Advisor tells the user what it has figured out and asks for help.

The plan recognition procedure is a heuristic "parsing" procedure that searches the. partially reconstructed plan for plan fragments (from its "Plan Library") or error fragments (from its "Error Library"). Since the user's plan is assumed to be an instantiation of the MUSER model, the Advisor also uses the MUSER model as data in inferring further struct ure. All possible parsings are simultaneously explored until a single plan is found that ties the user's goal to his dataflow graph. The first such "parsing" of the graph to be found is tentatively adopted by the Advisor as the correct plan If at some later 6tagc the Advisor finds that its < entative plan does not correspond with the user's, the Advisor eliminates it from consideration and searches for another parsing. In some cases, the recognizer can rule out potential plans by checking the component data base assertions against its model of the user's beliefs. If the tentative plan includes a misconception the U6cr is known not to possess, the plan is rejected and another sought. In the episode shown in figure 1, the Advisor was able to reconstruct the plan shown in figure 6 In comparing this plan to the one shown as figure 5, note the different data base assertion about the command COEFF and the resulting use of the "Obtain" method instead of "Obtain and Convert".

Once the Advisor has a version of the user's plan, it tries to identify the misconception. The first step in doing this is to acquire a suspicion of what might be wrong. Suspicion can be aroused either by recognition of some standard error in the user's plan *or* by a general model debugging process Once a suspicion is aroused, the Advisor confirms that it is a misconception by asking the user whether or not he believes it In the example, the Advisor found that

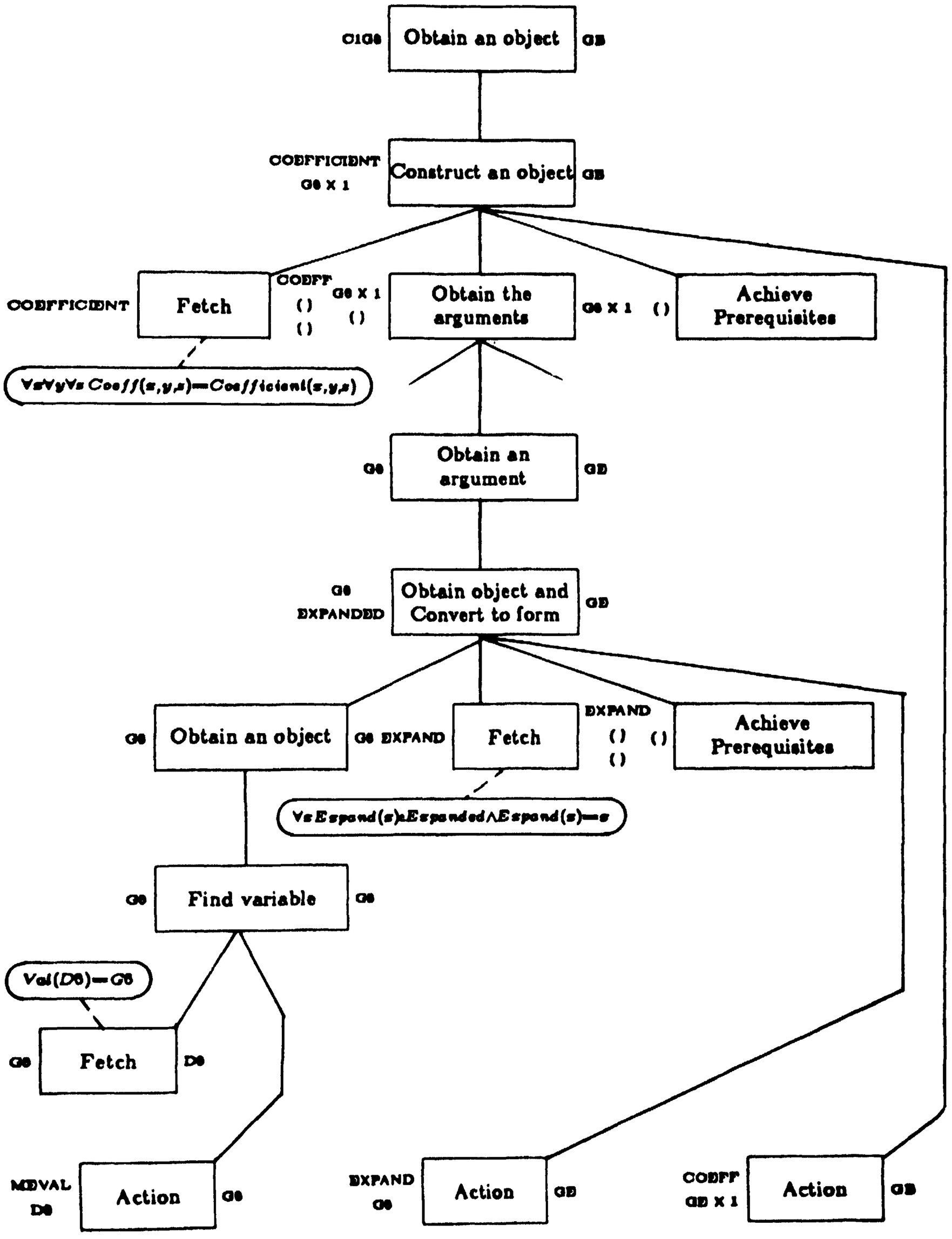


Fig. 5 - Plan to compute the coefficient of an expression

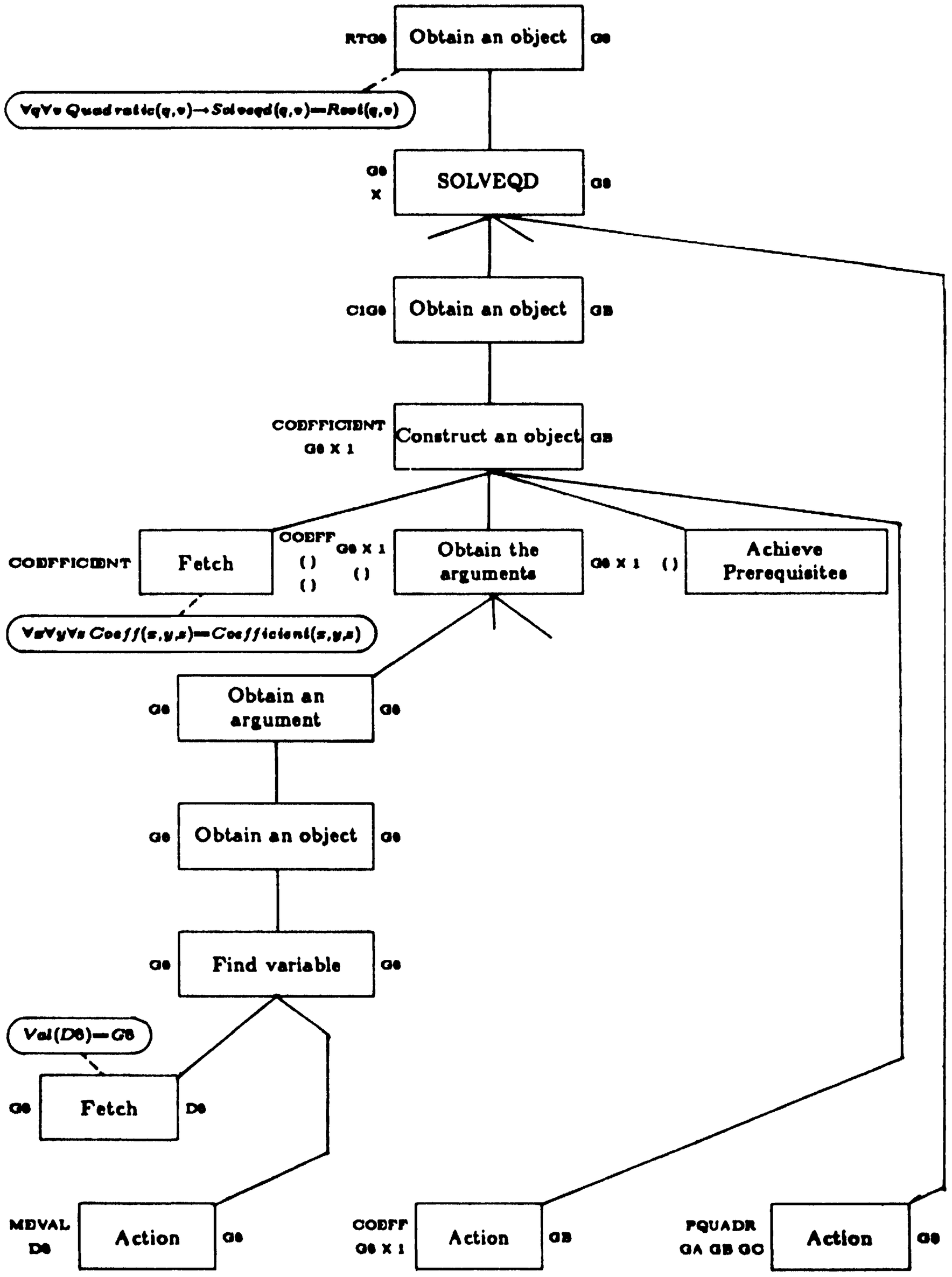


Fig. 6 - The plan for the MACSYMA session shown in Fig. 1

the assertion about COEFF in the reconstructed plan was incorrect and then asked the user whether he believed it.

Once a misconception has been identified, the Advisor correct s it and generates advice In the current implementation all standard errors have scripts associated with them. When such an error is detected and confirmed, the script is recited to the user. These scripts arc prepared by (he Advisor's programmer and contain all the necessary directions. In other cases, the Advisor follows a more general procedure.

In this more general procedure, correcting the misconception is easy, the user is simply told that his misconception is wrong. If there is a corresponding correct fact about the command or variable involved, this information is then supplied. In this example, the user was first told the correct information about COEFF.

Unfortunately, simply relating the correct information about the wrong command doesn't always help the user find the right command or option. Misconception correction is intended to improve the user's knowledge about the command or variable used. Helping him to useful information to achieve the goal involved is the purpose of the "misconception alternative" step of advice generation. In order to find an alternative, the Advisor executes (on its own data base) the same "Fetch" call with which the user retrieved his misconception (as indicated in the plan). If it succeeds, the information is conveyed to the user, as in the example If it fails, no further action is taken In either case the Advisor considers its work done and the episode ends.

5. *Summary*

The Advisor is implemented in MACLISP and consists of about T>0 pages of very compact code, including the data base manipulation routines and the MUSER problem solver but not counting the data base about MACSYMA. (At the moment the data base includes information about a few dozen commands only.) As mentioned earlier, there is no natural language interface. Preliminary testing of the system by hand translating consultation sessions into the Advisor's internal language shows that the major shortcoming arc its lack of knowledgcof the task environment (mathematics) and the weakness of the MUSER model. A full description of the implementation and the theory on which it is based can be found in [Genesereth 1978]'

The important contributions of this research are (1) its emphasis on the need for consultation in any sufficiently complex domain and (2) its consultation technique. In general, a consultant is necessary whenever one is faced with a problem solving situation in a domain one docs not fully understand. The lack of knowledge may be incidental, as it is when the domain or device is fairly simple but time constraints make it impossible for the U6cr to learn all that is necessary (e.g. using a calculator or oscilloscope). Or it may be essential, as when the domain is very complex and the user can't possibly learn everything (eg. MACSYMA).

The implementation described here is based on the belief that a good consultant must possess not only substantial knowledge of its material but also a good model of its user's knowledge. The key point of the paper is that plan reconstruction and analysis is a good way to gain this user model. The consultant described here utilizes a formal domain-independent problem solving model as a grammar for plan reconstruction. Although the Advispr also uses an error library, the plan grammar allows it to diagnose bugs it has not previously encountered. This not ion of a person's "plan" for solving a problem is central to consultation and should prove useful for man-machine interaction in general.

References

Denni*, J.. "First Version of a Data Flow Procedure Language", Memo 0:, M I T . Lab. for Comp. Science, 1975.

Genesereth, M.R..: "Automated Consultation for Complex Computer Systems", Ph.D. thesis, Harvard University, Sept. 1978.

Genesereth, MR.: "A Fast Inference Algorithm Based on the Constrained Propagation of Marks in a Semantic Network", Mathlab memo 5I>, M.I.T. Lab. for Computer Science, Dec. 1976.

Mathlab Group. "MACSYMA Reference Manual", Lab. Computer Sci., M.I.T., Dec. 1977.

Sacerdoti, E D.: "The Nonlinear Nature of Plans", TN 101, AI Group, Stanford Research Institute, 1975.

DETECTION AND MEASUREMENT

Donald B. Gennery
Artificial Intelligence Laboratory
Computer Science Department
Stanford University
Stanford, California 94305

A method of detecting and measuring objects for the purpose of representing three-dimensional outdoor scenes, using data such as that obtained from stereo vision or from a scanning laser rangefinder, is described. Objects are approximated by ellipsoids. Segmentation of the objects from the background and from each other is done by finding the ground surface, forming a preliminary segmentation by clustering the points above the ground by more than a threshold, fitting ellipsoids to match these clusters and to avoid obscuring the other points, and adjusting the clusters according to the fits. The method is designed to produce results useful for obstacle avoidance and navigation in an exploring vehicle, such as a Mars rover. An example is given showing results obtained from a stereo pair of pictures of Mars from the Viking Lander.

1. Introduction

In a previous paper [11 a stereo vision system and its possible use in an exploring vehicle was described. This paper describes further work towards a complete system for an exploring vehicle, specifically, the use of three-dimensional data for the detection of objects and the measurement of their position, size, and approximate shape. Although the three-dimensional data could be obtained from a laser rangefinder, the object detector is designed to be tolerant of errors in this data, such as mistakes produced by incorrect matches in stereo vision data and poor accuracy of distances from stereo.

Many approaches are possible In describing the shapes of objects. For example, generalized cones [2] can be used to describe complicated objects. However, in some cases the resolution of the data is insufficient to produce detailed information about the shape. In other cases, the objects are so irregular as to make such detailed descriptions very difficult. However, in such cases information about the position and size and some crude information about the shape may still be quite useful. For example, for obstacle avoidance in a roving vehicle, this sort of information is adequate. Furthermore, this sort of information for each object in a large scene containing many objects amounts to quite detailed information concerning the whole scene, and thus it would be useful for navigation.

* This work was performed under NASA contract NASW-2916 and the Defense Advanced Research Projects Agency contract MDA905-76-C-0206.

For these reasons, and because of its convenient mathematical properties, here each object will be approximated by an ellipsoid. By "object" we do not necessarily mean an actual physical object, but merely a portion of the scene that can be reasonably approximated by an ellipsoid. Thus, if we use as an example a vehicle exploring Mars, an object may be a single rock on the Martian surface, two or more adjacent rocks, or merely a bump in the ground. Also, an L-shaped physical object might be represented as two objects.

This ellipsoidal representation should be quite appropriate for representing rocks on Mars, because rocks probably tend to resemble more nearly ellipsoids than any other simple shape. However, it could also be used to represent cars in a parking lot or trees in a field, for example, especially in aerial photographs where the resolution may be poor compared to the size of the objects, and in other cases where precise object description or recognition is not necessary but rather an overall description of the scene is desired.

The stereo vision processing or laser rangefinder results in data representing the three-dimensional position of a large number of points distributed over the scene. The first step in the processing of this three-dimensional data is to find the ground surface. A method of doing this was previously described [l l In general, an entire scene would be partitioned into small areas, in each of which the ground would be approximated by a plane or paraboloid. Then points which are above the ground by a sufficient amount (depending on the computed accuracy of the points, the roughness of the ground, and the minimum size of object that is of interest) are candidates for points on objects.

These above-ground points are clustered to produce preliminary groupings of points which correspond roughly to objects. An ellipsoid is fit to each cluster by first computing an initial approximation based upon the moments of the points in the cluster and then iterating a weighted nonlinear least-squares adjustment to fit the ellipsoids to these points and to avoid obscuring other points. Then, according to the relative positions of the ellipsoids and points, clusters can be broken or merged, and the process repeats until the apparently best segmentation is found. Each of these steps will be described in the following sections.

The object detection and measurement process as described here uses only the three-dimensional position information. The brightness information is discarded after the stereo processing. However, a more complete system would use both types of information. Perhaps an edge detector could be applied to the brightness data in the regions near the outlines of the ellipsoids in order to refine the boundaries of the objects, for example.

Matrix notation will be used throughout this paper. Matrices will be denoted by capital letters. The transpose of a matrix A will be denoted by A^T, and the inverse of A will be denoted by A^{-1}. The trace of a square matrix A (sum of the diagonal elements, which is equal to the sum of the eigenvalues) will be denoted by tr(A). A vector in three-dimensional space will be represented by a 3-by-l matrix containing the Cartesian coordinates, usually denoted by X with an appropriate subscript. (Hohn [3] provides a good text on matrix algebra.)

2. Preliminary Clustering

Once the ground surface has been determined, all points that are above this surface by more than a threshold are clustered to form an initial approximation to the segmentation of the scene into objects.

Various clustering techniques could be used here. One possibility is a relaxation method, such as Zucker's [4] However, at present, the clustering is done by using the minimal spanning tree of the points. (The minimal spanning tree is the tree connecting all of the points such that the sum of the edges is minimum.) This is computed by using the nearest neighbor algorithm [5} (The length of the edges of the tree is defined here as the three-dimensional Euclidean distance between the points.) Then the tree is broken at every edge whose length is greater than twice the average length of the adjacent edges [51 However, a minimum length for an edge to be broken (related to the resolution of the data) is specified, so that the method will not be overly sensitive to local fluctations in the data. Also, a maximum can be specified, beyond which all edges are broken.

S. Initial Approximations to Ellipsoids

Since each ellipsoid will be fit to a cluster of points by an iterative process, an initial approximation is needed. A good approximation Increases the likelihood of convergence, decreases the number of iterations required, and can be used as the result in case the iterations do not converge. This initial approximation is obtained from the three-dimensional moments, through the second order, of the points in the cluster.

An ellipsoid can be represented by the following matrix equation:

$$(X-X_c)^T W(X-X_c) = 1$$

where X is a vector of the three-dimensional coordinates of any point on the surface of the ellipsoid, X_c similarly is the position of the center of the ellipsoid, and W is a positive-definite symmetrical 3-by-3 matrix. (See, for example, Hohn [3].) Let M denote the inverse of W. (The square roots of the eigenvalues of M are the lengths of the semi-axes of the ellipsoid.) The relationship between the computed moments and the matrices X_c and M depends on the distribution of points over the ellipsoid. If the object has been viewed from a single point, we will have points distributed nonuniformly over half of the surface. (Actually slightly less than half will be seen because of perspective. Also, in stereo vision, both cameras must see each point, so that with a single pair of cameras only the common area seen from both camera positions will appear. These two effects will be neglected below, however.)

We assume here that the object is seen from a single viewpoint by a raster scanning device which produces points distributed uniformly in the image plane. Such a device might be a scanning laser rangeflnder or an area-based stereo system. Actually, because of missing points, the distribution will not be uniform. It would be possible to estimate the actual distribution by computing higher-order moments, but this might be overly sensitive to randomness in the distribution or an inadequate density of points, so it is not attempted here. As an approximation, we assume an orthogonal projection instead of a central projection. Let X_s denote :he vector of normalized first moments (centroid) and M_5 denote the matrix of second moments about X_s obtained with this distribution, and let X_0 denote the position of the camera.

The relationship connecting X_s and M_s to X_c and M can be derived by first considering the case of a sphere of radius r. A little integration shows that in this case the eigenvalue of M_s corresponding to the eigenvector X_0-X_c is $r^2/18$, the other two eigenvalues are both $r^2/4$, and X_s is X_c plus 2r/3 times the unit vector in the X_0-X_c direction. All three eigenvalues of M should be r^2 in this case. An ellipsoid can be considered to be a distorted sphere (using stretching and skew distortions). Thus the ellipsoid can be considered to be stretched in the various directions by the amount given by the square roots of the ratios of the above eigenvalues, but in computing the displacement of the center, instead of r the distance from X_c towards X_0 to the ellipsoid surface must be used. Thus the displacement of the center is the vector

$2(X_o-X_c)/3$ divided by the scalar $\sqrt{(X_o-X_c)^T W(X_o-X_c)}$. Since X_c, X_s, and X_o are colinear, X_c can be replaced by X_s without changing the value of this ratio. Also, W ($=M^{-1}$) can be replaced by $M_s^{-1}/18$ in this expression, because of the stretching discussed above. Thus X_c can be computed from X_s by translating by this amount. To compute M, we can take 4 times M_s to account for the factor of 4 in two dimensions, but this leaves 14 out of the factor of 18 by which we need to stretch the moments in the direction toward the camera. This extra amount can be added by adding 14 times the moment produced by a fictitous point at the intersection of the X_s-to-X_o line and the surface of the ellipsoid corresponding to M_s. In order to keep the ellipsoid to a reasonble shape when there are not enough points to determine it well, M as obtained above is averaged with a scalar matrix whose diagonal elements are s (which represents a sphere of radius $\sqrt{s}$), with the average weighted so that the sphere represents four additional points in the moment computation. The value of s is determined so that it is the average of the two components of the second moments at right angles to X_o-X_s, but limited by the average of all three components (the three eigenvalues), including the effect of 14 times the effect of the fictitious point, as above, as an upper limit, and excluding this effect, as a lower limit. This avoids putting undue weight on the X_o-X_s dimension when the ellipsoid is long in this direction, since this dimension is less reliable because of the factor of 18 compression.

By combining the above information, the computation of the initial approximation can be expressed as follows:

$$X_s = \frac{1}{n}\sum X_p$$

$$M_s = \frac{1}{n}\sum (X_p - X_s)(X_p - X_s)^T$$

$$M_t = 4M_s + \frac{14(X_o-X_s)(X_o-X_s)^T}{(X_o-X_s)^T M_s^{-1}(X_o-X_s)}$$

$$X_c = X_s - \frac{2\sqrt{2}(X_o-X_s)}{\sqrt{(X_o-X_s)^T M_s^{-1}(X_o-X_s)}}$$

$$s = \min\left[\max\left(\frac{1}{3}\mathrm{tr}(M_s),\ 2\,\mathrm{tr}(M_s) - 2\frac{(X_o-X_s)^T M_s (X_o-X_s)}{(X_o-X_s)^T(X_o-X_s)}\right),\ \frac{1}{3}\mathrm{tr}(M_t)\right]$$

$$M = \frac{n}{n+4}M_t + \frac{4}{n+4}sI$$

$$W = M^{-1}$$

where X_p represents any point in the cluster, n is the number of points in the cluster, the summations are over these points, and I is the 3-by-3 identity matrix.

4. Iterative Solution for Ellipsoids

The adjustment of the ellipsoids is done by a modified least-squares approach. Each ellipsoid is adjusted so as to minimize the weighted sum of the squares of two kinds of discrepancies: the amounts by which the points (usually points in the cluster being fit) miss lying in surface of the ellipsoid, and the amounts by which the ellipsoid hides any points as seen from the camera position. (In the latter case, the discrepancies actually should be considered separately for each camera that sees the point in question. However, for narrow-angle stereo we use as a reasonable approximation the assumption that the "camera" is at the midpoint of the stereo baseline.) Including the second kind of discrepancy is useful in helping to determine the size and shape of the object when the points on the object itself do not contain sufficient information. Also included in the weighted sum of squares to be minimized are *a priori* terms which tend to force the ellipsoid by default to become a sphere near the ground when the points do not constrain it well.

The first kind of discrepancy above optimally should be defined as the length of the normal from the point in question to the surface of the ellipsoid. However, computing this requires solving a sixth-degree equation. Therefore, as an expedient the distance between the point and the surface along a straight line from the center of the ellipsoid to the point is used instead. In order to be consistent with this definition, the second kind of discrepancy is defined as follows. The midpoint of the two intersections of the surface of the ellipsoid with a line from the camera to the point is first found. Then the discrepancy of the first kind is computed for this midpoint. Both kinds of discrepancies are illustrated in Figures 1 and 2.

Now we must consider exactly for which points which kind of discrepancy is computed. There are five regions of space to consider, according to whether the point is to the side of the ellipsoid as seen from the camera (that is, the line through the camera postion and the point does not intersect the ellipsoid), is in front of the ellipsoid as seen from the camera, is inside the front portion of the ellipsoid (in front of the surface of midpoints as defined above), *is* inside the back portion of the ellipsoid, or is behind the ellipsoid. Also, there are two kinds of points to consider, according to whether or not the point is in the cluster which is assumed to correspond to this object. This produces ten combinations in all, which are illustrated in Figures 1 and 2. They divide into four categories.

First, if the point is not in the cluster and is either in front of the ellipsoid or is to the side, there is no discrepancy and this point is not included in the computations.

Second, if the point is in the cluster and is either in front, inside the front half, or to the side, or if the point is not in the cluster and *ii* inside the front half, the first kind of discrepancy is used.

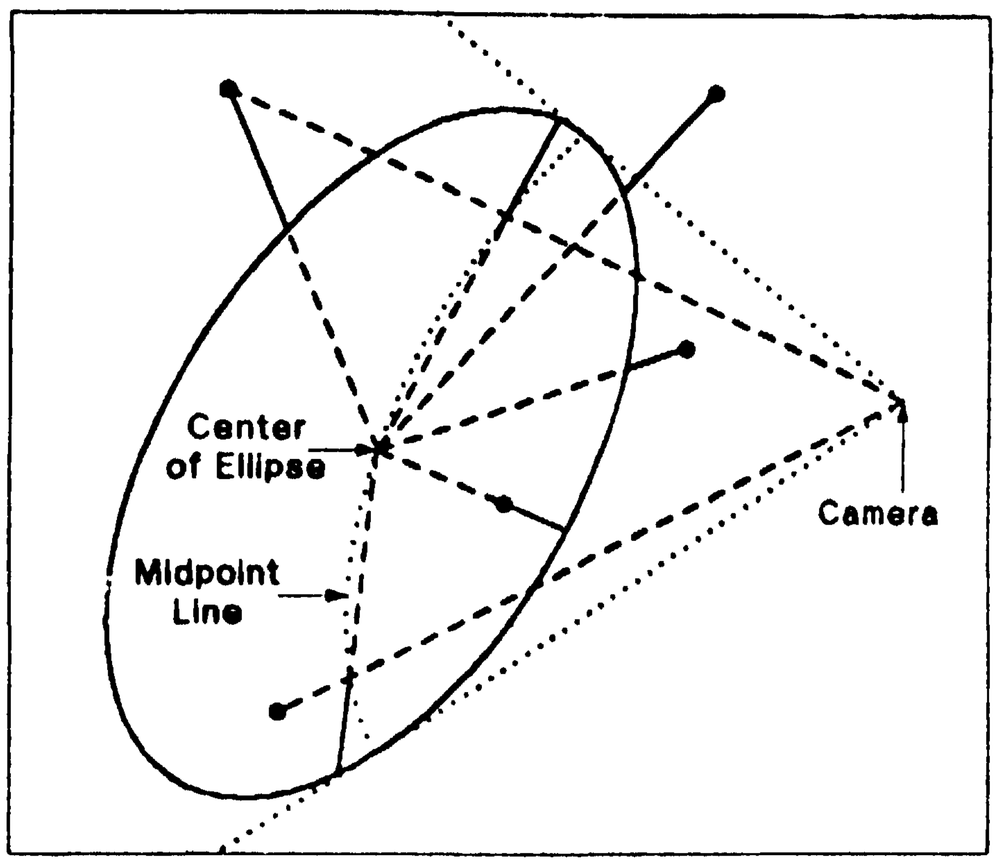


Figure 1. Two-dimensional version of adjustment, showing points belonging to this cluster. Solid dark straight lines are discrepancies to be minimized.

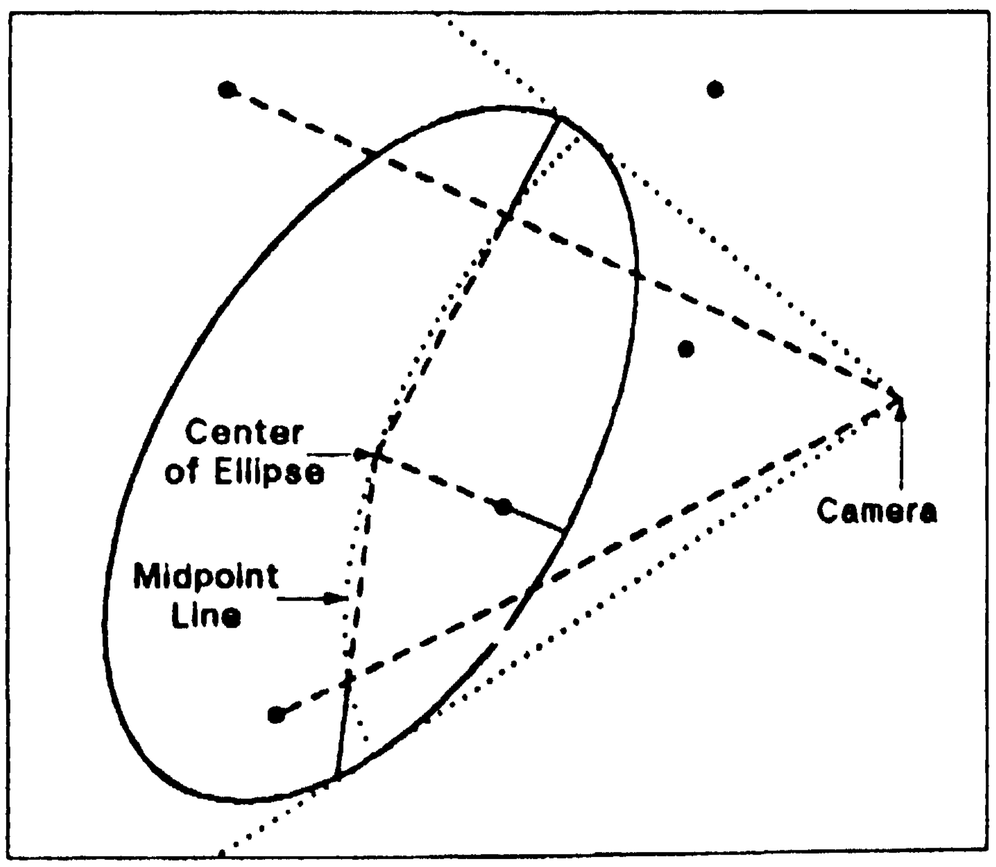


Figure 2. Two-dimensional version of adjustment, showing points not in this cluster. Solid dark straight lines are discrepancies to be minimized.

Third, if the point is not in the cluster and is behind the ellipsoid, or if either kind of point is inside the back half, the second kind of discrepancy is used.

Fourth, if the point is in the cluster and is behind the the ellipsoid, both kinds of discrepancies are used, and the point acts as two points in the computations. This is because there are two separate components of error in this case: the object does not extend enough on the side to hide the point, but it apparently bulges out in back (relative to the ellipsoid) to include the point.

In order to derive the mathematics for dividing space into the above five regions, consider the equation for the ellipsoid, as previously stated,

$$(X-X_c)^T W(X-X_c) = 1$$

and the equation of a straight line through the camera position X_o and the point in question X_p, in parametric form,

$$(X-X_o) = u(X_p-X_o)$$

These can be combined to produce

$$[u(X_p-X_o)+X_o-X_c]^T W[(X_p-X_o)+X_o-X_c] = 1$$

the roots of which determine the intersections of the line and ellipsoid. This equation is equivalent to

$$au^2 + bu + c = 0$$

where

$$a = (X_p-X_o)^T W(X_p-X_o)$$

$$b = 2(X_p-X_o)^T W(X_o-X_c)$$

$$c = (X_o-X_c)^T W(X_o-X_c) - 1$$

The roots of the above equation in u determine the region of space in which X_p lies. If the roots are imaginary ($b^2-4ac < 0$), the point is to the side of the ellipsoid. If the average of the two roots ($-b/2a$) is positive, the point is in front of the midpoint surface, if negative, it is behind the surface. If the roots are real, the point is in front of, behind, or inside the ellipsoid according to whether both roots are greater than unity, both roots are less than unity, or unity lies between the roots, respectively. Alternatively, we can use the fact that the point is outside of the ellipsoid if and only if $(X_p-X_c)^T W(X_p-X_c) > 1$.

The discrepancy of the first kind is

$$\epsilon = \sqrt{(X_p-X_c)^T(X_p-X_c)}\left(1 - \frac{1}{\sqrt{(X_p-X_c)^T W(X_p-X_c)}}\right)$$

In order to compute the discrepancy of the second kind, the midpoint of the intersections of the camera-point line with the ellipsoid is first obtained as follows:

$$X_u = -\frac{b}{2a}(X_p-X_o) + X_o$$

Then the discrepancy of the second kind is

$$\epsilon = \sqrt{(X_u-X_c)^T(X_u-X_c)}\left(1 - \frac{1}{\sqrt{(X_u-X_c)^T W(X_u-X_c)}}\right)$$

Because there may be erroneous points in the data, points

which have large discrepancies relative to the size of the ellipsoid are given less weight in the solution. The weighting function used is

$$\omega = \frac{1}{1 + 2\left(\frac{\sqrt{(X-X_c)^T W(X-X_c)} - 1}{\mathrm{tr}(W)}\right)^2 \sigma^2}$$

where X represents X_p or X_u for discrepancies of the first or second kinds, respectively, and σ is the component of standard deviation of measurement errors in X_p propagated into the discrepancy. (If these are unknown, σ can be zero.) Thus the dimensionless quantity to be minimized (by adjusting X_c and W) is $\Sigma\omega\epsilon^2$, plus some additional terms for *a priori* values yet to be discussed. However, this quantity is minimized only with respect to the effects of X_c and W acting through ϵ and not their effects through ω. In order to solve the above nonlinear problem, the Gauss method [6] is used. This method is equivalent to using the partial derivatives of the discrepancies to approximate the nonlinear problem by a linear statistical model [7], solving the linear problem, and iterating this process until it converges.

On any one iteration the following is done. The current values of X_c and W are used to compute for each point the value of ϵ as define above and the 1-by-9 matrix P, which consists of the partial derivatives of ϵ with respect to the three elements of X_c and the six unique elements of W. (W is symmetrical.) The following summations over all of the points are computed, in which each point in the first category above is not used, each point in the second or third categories appears once, and each point in the fourth category appears twice:

$$H = H_0 + \sum P^T \omega P$$

$$C = C_0 + \sum P^T \omega e$$

(H_0 and C_0 are used for the *a priori* values yet to be discussed.) Then the 9-by-1 matrix of corrections is

$$D = vH^{-1}C$$

where v is a factor used to improve convergence because of the very nonlinear nature of the problem. (Currently v = 0.5 on early iterations, but v = 1 after a test indicates that this will produce more rapid convergence.) The elements of D are subtracted from the corresponding elements of X_c and W to obtain the improved approximations for the next iteration.

Now the *a priori* values will be discussed. In some cases the points affecting the ellipsoid will be insufficient in number or insufficiently distributed to determine all parameters of the ellipsoid very well. It is therefore desirable to have *a priori* values for some of the parameters with appropriate weight in the solution to constrain them to reasonable default values when the points do not contain sufficient information. When there is ample information in the points, the *a priori* values will have very little effect because of their small weight. The *a priori* values currently used are the ground surface height directly under X_c for the vertical component of X_c, with weight 0.1/tr(M), equality for the diagonal elements of W, with weight $\mathrm{tr}(M)^2/10$, and zero for the off-diagonal elements of W, with weight $\mathrm{tr}(M)^2/10$, where $M = W^{-1}$. (Including tr(M) as shown scales things correctly so that the solution is invariant under a scale factor change.) The effect of the W terms is to try to force the ellipsoid into a spherical shape. These *a priori* terms are put into the solution in the following way. The diagonal element of H_0 corresponding to the vertical component of X_c is 0.1/tr(M), the three diagonal elements corresponding to the off-diagonal elements of W are each $\mathrm{tr}(M)^2/10$, and the 3-by-3 submatrix on the diagonal of H_0 in the position corresponding to the diagonal elements of W consists of 2/3 on its main diagonal and -1/3 elsewhere multiplied by $\mathrm{tr}(M)^2/10$. All other elements of the 9-by-9 matrix H_0 are zero. Then

$$C_0 = H_0 G$$

where G is a column matrix of the current values of X_c and W, arranged as in D, with the height of the ground directly under the center of the ellipsoid subtracted from the element of G corresponding to the vertical component of X_c. H_0 and C_0 are used in the summations for H and C as previously shown.

5. Breaking and Merging Clusters

Because the preliminary clustering is dependent on local information, it may not produce the best segmentation based on more global information. Therefore, after ellipsoids have been fit to all of the preliminary clusters, these clusters may be tentatively broken into smaller clusters and merged into larger clusters, new ellipsoids are fit to these clusters by the same process previously described, and a decision on whether to keep or reject each of these actions is made based on the goodness of fit of the ellipsoids to the points.

In order to decide where to break a cluster, for each edge in the portion of the original minimal spanning tree which connects this cluster the quantity $\lambda(1-\rho)$ is computed, where λ is the length of the edge and ρ is the minimum of $\sqrt{(X_p-X_c)^T W(X_p-X_c)}$ for the two points connected by the edge. Then the cluster is tentatively broken at the edge for which this quantity is maximum, of all such edges such that each new cluster formed has at least four points at least one of which has $(X_p-X_c)^T W(X_p-X_c) > 1$ (that is, it is outside the old ellipsoid). This process tends to break the cluster at places furthest inside the ellipsoid, but connecting points that are outside the ellipsoid. If this new clustering is accepted by the criteria described below, the process repeats on the new clusters.

After the above breaking process is finished, any two clusters are tentatively merged if $(X_a-X_c)^T W(X_a-X_c) < 4$ for either cluster, where X_a is X_c for the other cluster, provided that these two clusters were not previously one cluster before breaking. If there is competition for the merging, the cluster

pair with the minimum value for this quantity is merged first. If a merger is accepted, further mergers can take place on these clusters by this same process.

The criteria for accepting two clusters or one that resulted from a tentative break or merger are as follows. If $(X_a-X_c)^T W(X_a-X_c) < 1$ for either small cluster, where X_a is X_c for the other small cluster (that is, the center of one ellipsoid is inside the other ellipsoid), the single cluster is chosen. Otherwise, the following quantity is computed for each of the three ellipsoids:

$$q = \frac{\Sigma \omega \epsilon^2}{n-3}$$

where E and w are the discrepancies and weights from the last iteration, as defined in the previous section, and n *is* the number of points in the cluster corresponding to this ellipsoid. (If the initial approximation is used as the result, e and w are obtained from the first iteration, and the denominator is n instead of n-3.) Then the two small clusters are chosen if the sum of their two values of q is less than the value of q for the single cluster. Otherwise, the single cluster Is chosen.

6. Results

Fig. 3 shows a stereo pair of pictures taken from the Viking Lander I on the surface of Mars. Each picture is 256 pixels by 256 pixels. The pixel spacing *is* 0.04 degrees in azimuth and elevation. (Thus the field of view is about 10 degrees.) The azimuth and elevation from the left camera to the center of the picture are about 18 degrees and -20 degrees, respectively, relative to the camera positions. The two cameras are 0.8187 meters apart. The distances to the points in the scene range from about 3 meters to about 4.5 meters.

Fig. 4 shows the left picture enlarged, with arrows indicating the points found by the stereo processing. The point of each arrow is at the center of an eight-pixel-square window which was matched to a corresponding area in the other picture. (Thus the resolution of the reduced stereo data is 0.32 degrees in azimuth and elevation.) The arrows are dropped perpendicularly from the point in three-dimensional space computed for this point to a nominally horizontal reference plane 1.5 meters below the cameras, with the base of the arrows on this plane, and arc then projected into the picture. Fig. 5 shows the same data as Fig. 4 except that the bases of the arrows rest on a ground plane computed from this data by the ground surface finder. Fig. 6 is the same as Fig, 5 except that it shows only points computed to be at least 5 centimeters above the ground surface.

The 5-centimeter height threshold was used for selecting the points to cluster in the object finder. The minimum distance for breaking the minimal spanning tree to form the initial clusters was also 5 centimeters, and the maximum distance for connecting points was 20 centimeters. (Using zero and infinity for this minimum and maximum produced a sightly different Initial clustering but identical final results.)

Fig. 7 shows the points in Fig. 6 in a nominally vertical orthogonal projection (perpendicular to the reference plane). The figure covers an area one meter by one meter. The symbol for each point represents height in centimenters above the ground plane, with the letters A, B, C, etc. representing the values 10, 11, 12. etc. The points are connected to show the minimal spanning trees that were computed. Solid lines connect points within each initial cluster.

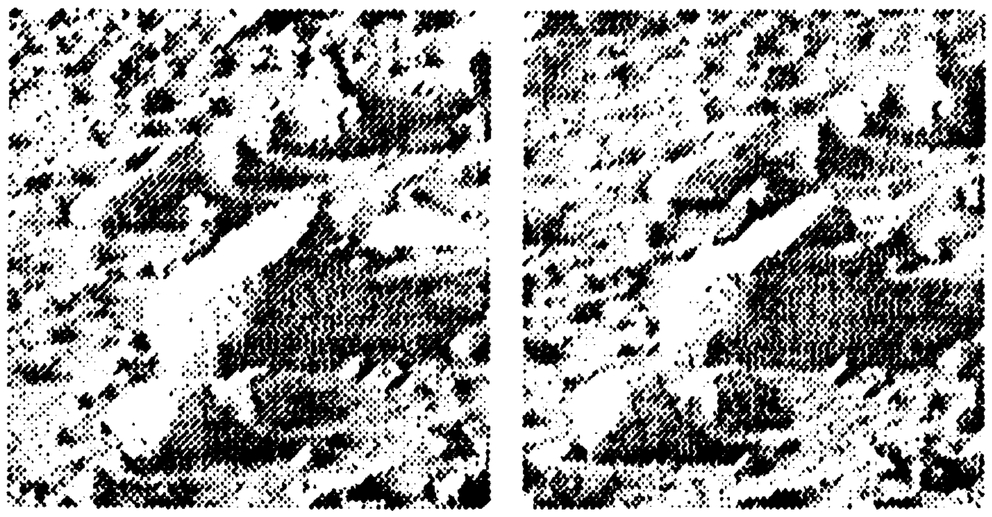


Figure 3. Stereo pair of Martian surface.

Figure 4. Points found by stereo processing, showing heights above reference plane.

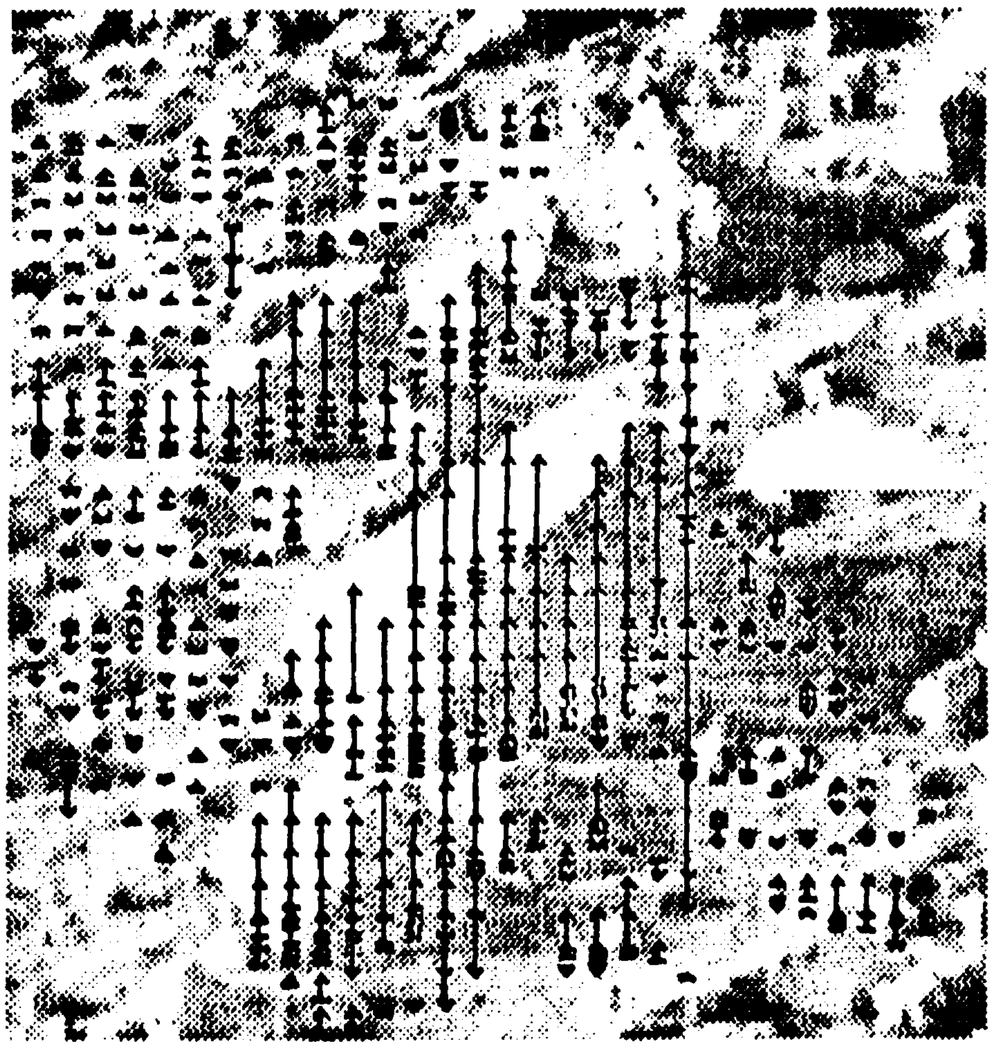


Figure 5. Heights above computed ground plane.

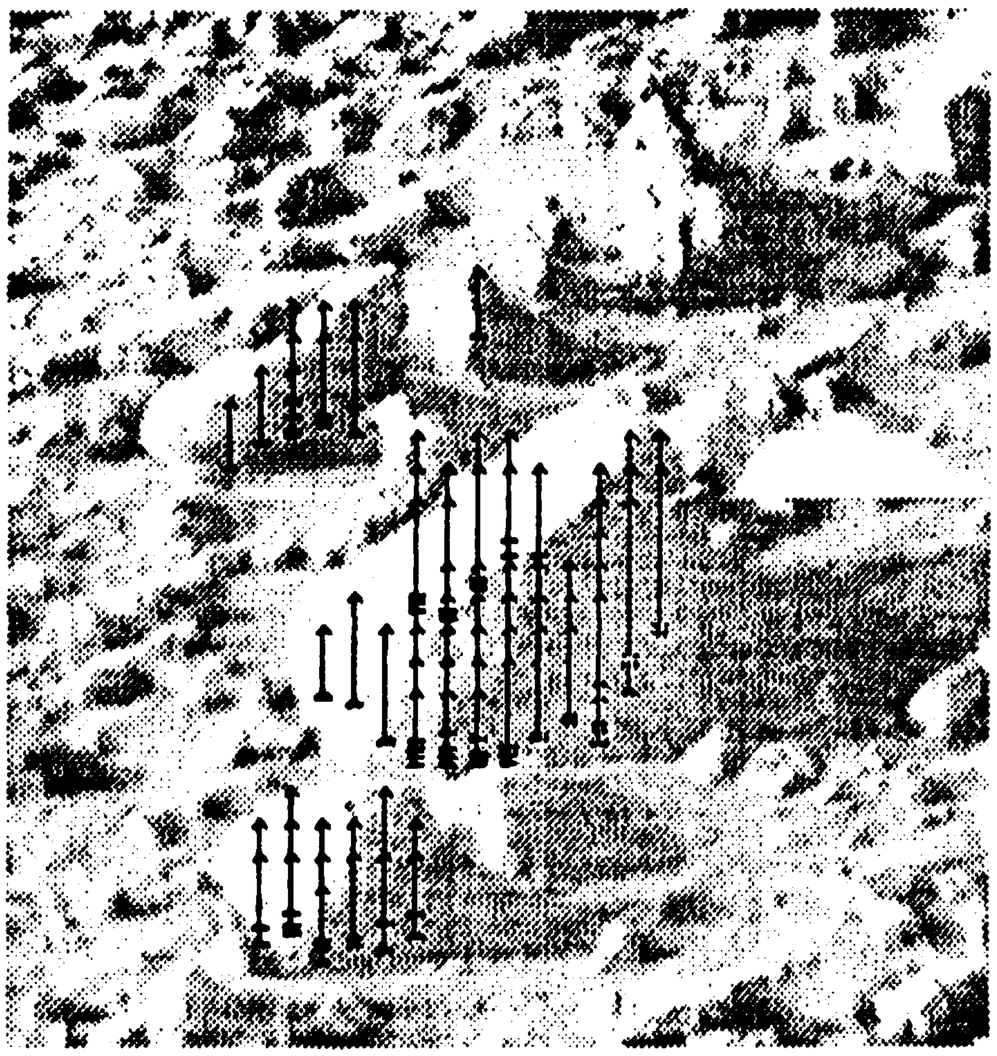


Figure 6. Heights above computed ground plane, for points above 5 cm.

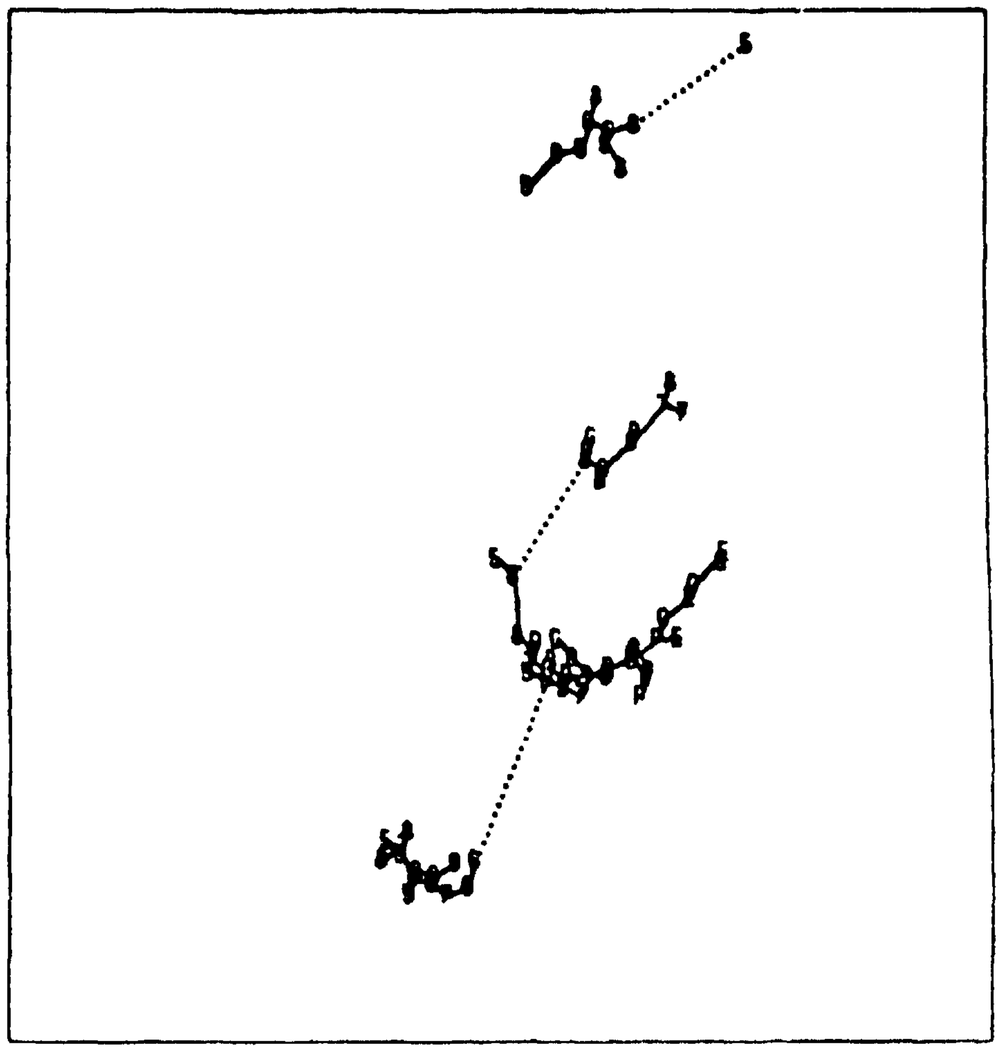


Figure 7. Mimimal spanning tree connecting points above 5 cm, vertical view. Solid lines show initial clusters.

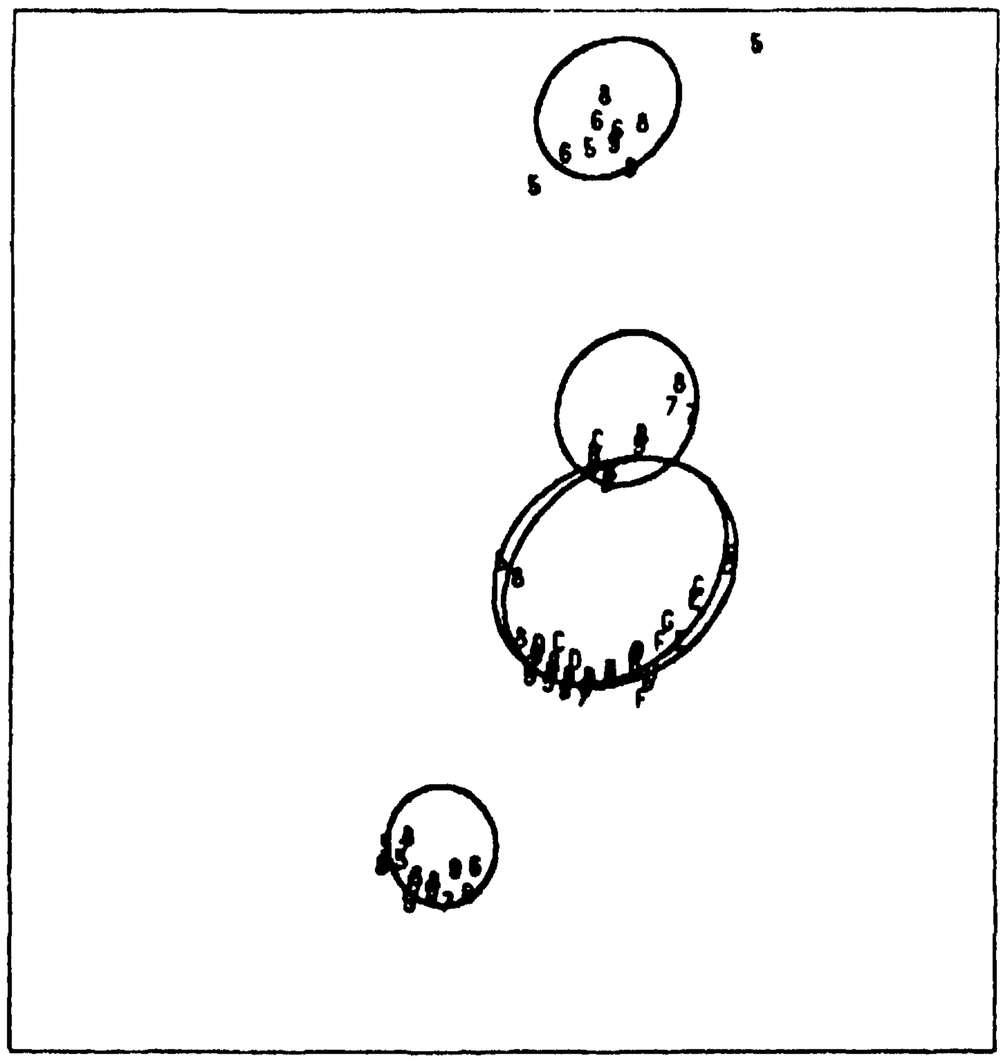


Figure 8. Ellipsoids fit to initial clusters.

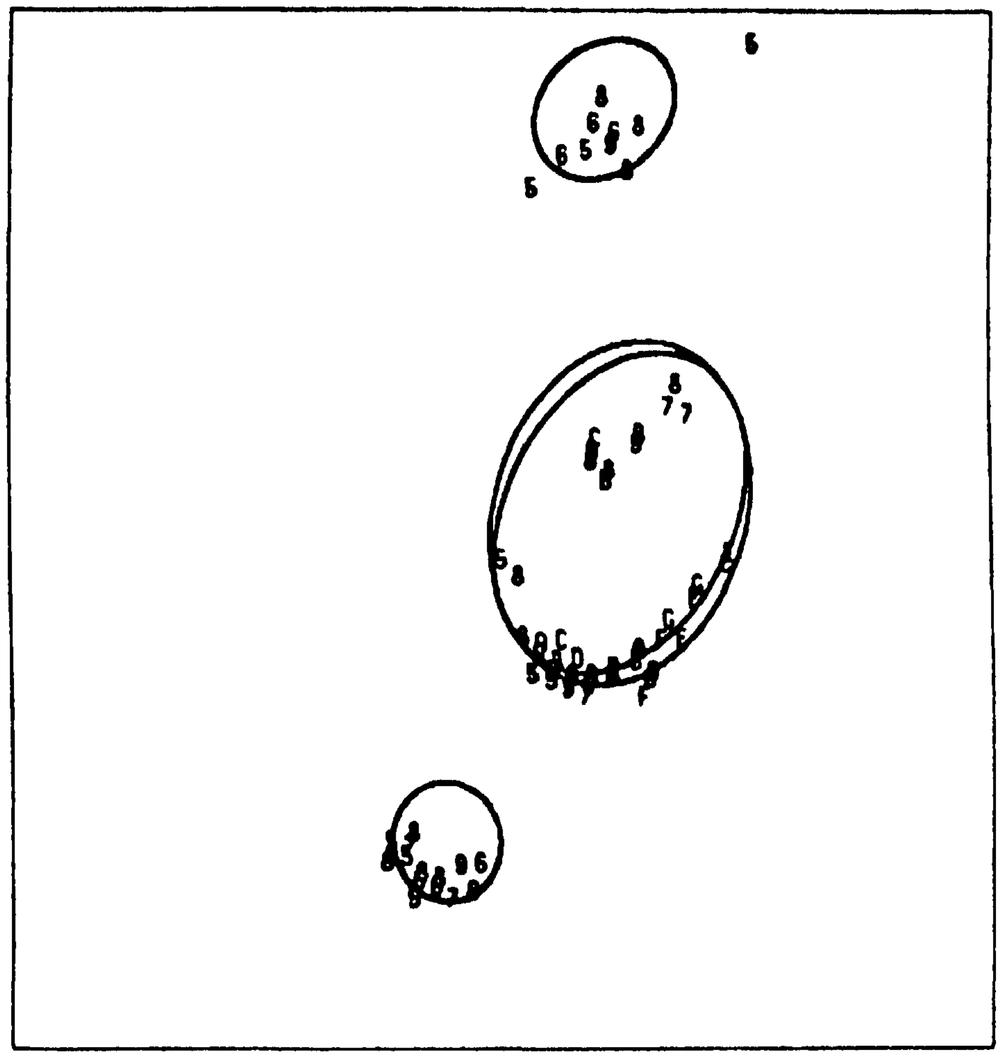


Figure 9. Ellipsoids fit to final clusters.

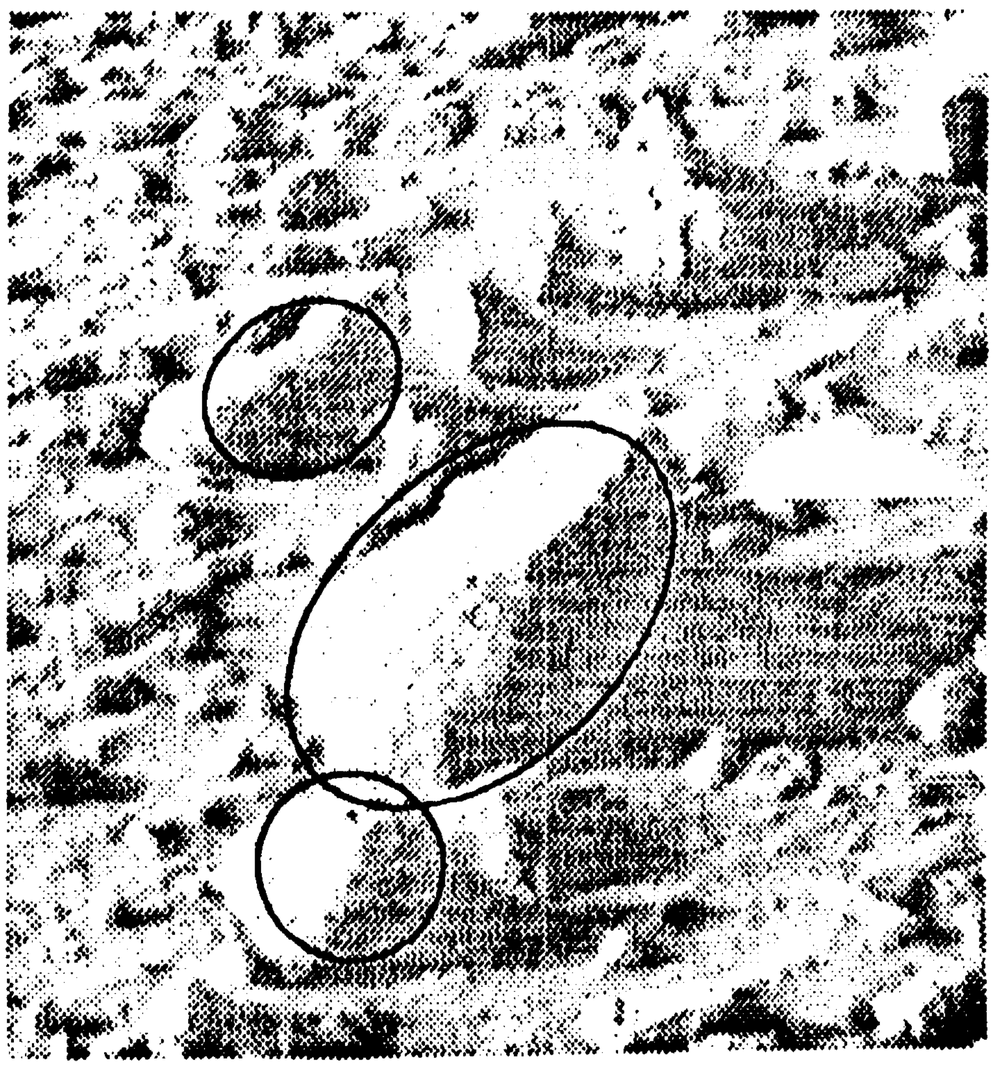


Figure 10. Ellipsoids fit to final clusters, projected into left picture.

Fig. 8 shows the ellipsoids that were fit to the initial clusters. Each ellipsoid is represented by two ellipses. One ellipse is the orthogonal projection of the ellipsoid onto the reference plane. The other ellipse is the intersection of the ellipsoid with a plane through the center of the ellipsoid and parallel to the reference plane. Only the clustered points are shown here, as in Fig. 7. However, as previously described, any of the points shown in Fig. 4 may have been involved in the adjustment of the ellipsoids. Remember that the fit is done in three dimensions, whereas Fig. 8 shows a two-dimensional projection.

Fig. 9 shows in the same way the results of the breaking and merging operations. The two clusters in the center (corresponding to the large rock in the center of the pictures) were merged into one, and a new ellipsoid is shown for this cluster. The other clusters were not changed.

These results were projected into the left picture to produce Fig. 10. The outline of the ellipsoids as they would be seen from the left camera are superimposed on the picture. The lengths of the principal axes of the large ellipsoid in the center are 36.5, 30.5, and 19.8 centimeters.

Acknowledgment

The Mars pictures were supplied by Elliott Levinthal and Sidney Liebes of the Viking Lander Imaging Team.

References

[1] Cennery, D.B. "A Stereo Vision System for an Autonomous Vehicle." In *Proc. IJCAIJ7.* MIT, Cambridge, Mass., August 1977, pp. 576-582.

[2) Nevatia, R. & Binford, T.O. "Description and Recognition of Curved Objects," *Artif. hull.* Vol. 8, February 1977, pp. 77-98.

[3] Hohn, F.E. *Elementary Matrix Algebra* (Third Edition). The MacMillan Company, 1973.

[4] Zucker, S.W. "Relaxation Labelling and the Reduction of Local Ambiguities," In *Proc. Third International Joint Conf. on Pattern Recognition.* San Diego, Calif., November 1976, pp. 852-861.

[5] Dud a, R. & Hart, P. *Pattern Recognition and Scene Analysis.* Wiley, 1973.

[6] Bard, Y. *Nonlinear Parameter Estimation.* Academic Press, 1974.

[7] Craybill, F.A. *An Introduction to Linear Statistical Models,* Volume I. McGraw-Hill Book Company, 1961.

A FRAMEWORK FOR CONTROL IN PRODUCTION SYSTEMS

M.P. Georgeff
School of Mathematical Sciences
Flinders University
Bedford Park, S.A., 5042.
Australia

A formal scheme for representing control in production systems is defined. The scheme allows control to be directly specified independently of conflict resolution and thus allows the issues of control and nondeterminism to be treated separately. Unlike previous approaches, it allows control to be examined within a uniform and consistent framework. It is shown that the scheme provides a basis for implementing control constructs which, unlike existing schemes, retain all the properties desired of a knowledge based system——modularity, flexibility, extensibility and adaptive capacity. Within the formalism it is also possible to provide a meaningful notion of the power of control constructs. This enables the types of control required in production systems to be examined and the capacity of various schemes to meet these requirements to be determined.

1. INTRODUCTION

Over the years a range of different mechanisms have been proposed for representing and using knowledge about general and possibly ill-defined problem domains. Of these, production systems [12] have been among the most promising, and have been applied to a diverse collection of problems, including mass spectroscopy [A] , medical diagnosis [15], electronic circuit design [10] and automated theory formation in mathematics [8]. Most theorem provers can also be viewed as production systems (e.g. PROLOG [16]).

Informally, a production system consists of a set of modules or procedures called productions and a data base on which these productions operate. Now one of the most fundamental and significant characteristics of production systems is the lack of explicit control information ——that is, production invocation can only be achieved indirectly through the data base. The primary effect of this indirect means of production invocation is to produce a system which is strongly modular, flexible and adaptive, and thus well-suited as a knowledge based system. However, it is also perhaps the most significant factor in complicating the programming of production systems, in making the behaviour flow more difficult to analyse, and in increasing the difficulty of an adequate formalization [1].

Now there are two reasons why we would like to have control. The first, and to which we alluded above, is that the solutions to many problems are most naturally represented by sequences of actions rather than by sets of actions in which the order of application is unimportant. We tend to use plans or strategies, even when we are manipulating declarative knowledge or facts about the world (e.g. consider the wide use made of procedural attachment in most knowledge based systems). Secondly, any large production system that does not somehow constrain the number of productions that are active at any one time becomes so inefficient as to be unworkable. The problem is: how do we achieve such control without sacrificing the *raison d'etre* of production systems.

Many production systems use some form of control information. Usually, this remains hidden in the productions or in the data base in the form of special flags and other hand-crafted markers [11]. Other systems use more explicit means of control (e.g. NASL [10], annotated systems [6]), but the control structures are limited and are often difficult to access or modify. But most importantly, no system provides a uniform framework in which control issues can be addressed, nor a direct means of implementing control constraits——in every case the control schemes are essentially ad hoc.

In this paper we approach the problem from the other direction. That is, we formally characterize the notion of control applied to production systems, and then consider how best to implement such a scheme. The scheme we propose, which we will call a *controlled production system,* simply consists of a production system together with a control device called a

control language. This provides us with a uniform framework for representing control in production systems, and allows us to examine in a very general way how the various approaches to realizing (implementing) control affect the properties and behaviour of the system as a whole.

In fact, it first appears that the notion of control in production systems is quite obvious, and hardly needs formalizing. However, if we examine the control schemes of current production systems——apart from their ad hoc nature——we see control being used in quite different ways. For example, in most schemes (e.g. PROLOG) control is *intended* to simply enhance the efficiency of the system——given enough time, the system without the control component would find the same solutions as the system with control. In other schemes (e.g. [13]), and *in fact* in most of the a fore mentioned schemes (e.g. PROLOG), control can be used in the same manner as in procedural languages——that is, the solutions obtained depend critically on the order in which the productions are invoked. Such confusion leads toad hoc systems the behaviour of which it is very difficult to predict and the solutions to which it is very difficult to validate. That is, it leads to a programming methodology quite the opposite of that favoured for procedural languages. Further, domain specific control knowledge is difficult to realize as it becomes confused with efficiency issues |3]. Formalizing the notion of control avoids these difficulties, and allows system efficiency to be treated as a separate issue.

2. CONTROLLED PRODUCTION SYSTEMS

2.1 An Informal Description

Informally, a production system (PS) consists of a set of modules or procedures called *production rules* and a *data base* or *working memory* to which these rules are applied. Each production rule is an expression or string of symbols which consists of two parts called the *lefthandside* (LHS), or *antecedent,* and the *righthandside* (RHS), or *consequent* . These respectively denote a *condi tion* which is to be satisfied before the production can be applied or invoked, and an *action* which specifies the result of application of the production to the data base.

In the simplest execution scheme, the conditions in each production are evaulated for the current state of the data base, one of the satisfied productions is selected, and the action specifled by that production then executed. The procedure is then repeated for this new state of the data base. Execution terminates either when there are no satisfied productions or when some desired state of the data base is achieved. In general execution is nondeterministic, as at any stage during execution more than one production may be applicable. The set of productions applicable at each stage of execution is known as the *conflict set*, and the selection procedure is usually called *conflict resolution*.

For example, consider the following production system where states of the data base are words over the alphabet {S,A,B,C,a,b,c} and where the productions (which are to be interpreted in the normal rewriting sense) include

p_1: S → ABC
p_2: A → aA
p_3: B → bB
p_4: C → cC
p_5: A → a
p_6: B → b
p_7: C → c

For an initial state S of the data base, the set of possible final states is the set of strings $\{a^{\ell}b^{m}c^{n}: \ell \geq 1, m \geq 1, \text{ and } n \geq 1\}$. Depending on how the system is implemented, it can be used either as a generator of these strings or as a recognizer for these strings.

Suppose that we now wish to introduce explicit control constraints into such a system. One way to achieve this is to allow productions to throw special symbols into the data base to constrain invocation of other productions (e.g. [11]). However, this scheme has numerous drawbacks. Firstly, in order that the desired constraint is effected, it is necessary to invest the productions that respond to these special symbols with a higher priority of invocation than all other productions that might also be satisfied by the current state of the data base. We thus end up with two classes of symbols in the data base, or equivalent!y, two classes of productions. Nothing is wrong with this, of course, except that we have changed the nature of the production system. Secondly, it is clear that the above scheme will not work unless the special symbols are known to be unique to the invoking and the invoked productions, and such uniqueness can only be established by reference to the condition part of all other productions. Other problems are also present in the above scheme. The system loses its potential extensibility as these special symbols essentially invoke productions by name rather than by content. Further, augmentation and modification of control information is extremely difficult.

Other contol schemes have been proposed which avoid some of these problems (e.g. [13]), but control is still achieved indirectly through the conflict resolution scheme. One consequence of this indirection is that code is also indirect and cumbersome, as, for example, that

needed for keeping loop productions dominant during iteration [13]. What is worse, control information can only be expressed having a full knowledge of the conflict resolution scheme, and this is not always possible nor desirable - consider, for example, systems that involve a dynamic conflict resolution scheme e.g. TEIRES1AS [2], Furthermore, in order that control constraints are not eventually overridden, production sequencing determined during conflict resolution must be irrevocable, at least for control elements, and this brings into question the function of the conflict resolution scheme as a means of handling non-determinism.

We thus adopt an alternative approach whereby we specify control information explicitly and independently of the conflict resolution scheme. To do this we will simply require that any constraints on production invocation be specified by means of a language over the production set. We will call such a language a *control language,* A production sequence and the relation it defines on the data base is then only allowed if this production sequence is included in the control, language. Thus at each stage of execution, the control language restricts the set of productions that may be considered for invocation and only a subset of the total production set is *active.* The only productions that can enter the conflict set are those that both have their condition satisfied by the current state of the data base *and* are contained in the active production set. We will call a production system together with a control language a *controlled prodution system* (CPS).

For example, consider the above production system together with a control language defined by the regular expression

$p_1(p_2p_3p_4)^*p_5p_6p_7$

Then the set of final states of the data base given the initial state S is the set of strings $\{a^nb^nc^n : n\geq 1\}$. Note that this controlled production system is nondeterministic and that after executing production *p4* the conflict set will contain two productions (namely, p2 and p5).

Thus it is seen that a controlled production system differs from the more usual production systems only in that it has an explicit and independently specified control structure. This control structure acts as a constraint on production invocation——it effectively reduces the possible interactions between productions. In one sense, the control structure provides private channels of communication between productions. This allows the power of a production system to be increased without increasing the complexity of the productions. For example, if we allow productions that check for a symbol not being in the current state of the data base, then it can be shown that context-free rewriting systems with regular control languages generate all the recursively enumerable languages [14].

2.2 Formal Definitions

A *production system* (PS) is a triple

$$P = (\Sigma, D, h) \qquad (2.1)$$

where Σ is an alphabet called the set of *production names*, D is a set called the *data base* and h, called the *interpretation* of Σ, assigns to each element of Σ a pair (q,r), where q is a total predicate on D and r is a relation on D. For any production name p, we will say that the first co-ordinate of h(p) is the *condition* denoted by p and that the second co-ordinate of h(p) is the *action* denoted by p. Elements of D will usually be referred to as *states* of the data base. For each production p, we will also require that all states of the data base which satisfy the condition of p be in the domain of the action of p.

We now introduce the notion of a controlled production system. We take as our control device a language over Σ. Formally a *control language* over Σ is any subset of Σ^*, where Σ^* is the free monoid over Σ with identity λ.

We define a *controlled production system* (CPS) to be a quadruple

$$C = (\Sigma, D, h, C) \qquad (2.2)$$

where C is a control language over Σ and (Σ, D, h) is a production system. A CPS (Σ, D, h, C) with $C = \Sigma^*$ is equivalent to the PS (Σ, D, h). We can therefore consider a PS to be a special case of a CPS.

We are now in a position to define the execution of a CPS. Let C be a CPS as defined in (2.2). We first define a *state of execution* (or simply *state*) of C to be a pair $s = (u,x)$ where u is a prefix of some word in C and x is an element of D. Now let u and up be prefixes for some word in C, where $u \in \Sigma^*$, $p \in \Sigma$ and $h(p) = (q,r)$. Then we say a state (up,x_2) is *directly computed* from a state (u,x_1), denoted $(u,x_1) \Rightarrow_C (up,x_2)$, if and only if we have $q(x_1)$ = true and $(x_1,x_2) \in r$. If the above holds, we will also say that the state (up,x_2) results from *execution* of production p in state (u,x_1).

Let $\Rightarrow_C^*$ denote the reflexive transitive closure of the relation $\Rightarrow_C$. Then the *relation computed by* C is defined to be the set

$R(C) = \{(x,y) : (\lambda,x) \Rightarrow_C^* (w,y)$ for some w in C$\}$

Informally, a CPS C may be (nondeterministically) executed for some initial state x of the data

base D in the following manner. At each moment of time, execution is at some state. Initially this state is $\langle \lambda, x \rangle$. Suppose that execution has arrived at some state (u,y>. Now a production p can be considered for evaluation if up is a prefix of some word in C. Let us call the set of all such productions the *active production set.* Suppose this set is empty. Then execution terminates successfully if u is an element of C and terminates unsuccessfully if u is not in C. Otherwise, execution may either terminate successfully (if u is in C) or continue by evaluating the condition of each active production with respect to the current state of the data base. All of those productions which are satisfied form what is known as the *conflict Bet.* If the conflict set is empty, , then execution terminates unsuccessfully. Otherwise , a production p is (nondeterministically) selected from this set, and execution continued from a (not necessarily unique) state <up,z>, where (y,z> is an element of the action denoted by p. The final state of the data base is obtained on successful termination of execution.

The most obvious way to implement such a scheme is simply to augment the recognize-act cycle of classical production systems with a stage that determines, on the basis of the control language, the set of active productions (or equivalently masks off the non-active productions). However, it is not clear how the control language should be specified.

3. SYSTEM BEHAVIOUR

it is first necessary to consider how the control language and its means of specification will affect the behaviour of the system as a whole. In particular, we need to consider how the imposition of a control language will affect the additivity and adaptive behaviour of the system.

3.1 Additivity and Adaptive Behaviour

Extensibility and the capacity for self modification are important characteristics of intelligent systems, and any control scheme should allow of such behaviour. Consider for example the problem of learning to add two positive integers given only the successor function. We will need the following two actions:
DEP(d) which places d in the data base and
PUT(p,q,r,c) which (i) adds the production with name p, LHS q and RHS r to the production set and (ii) adds the symbol string c to the control language. GENSYM is a parameterless procedure which generates a new symbol (production name), and s is the successor function.

The initial production set for solving this problem is as follows

p_1: (ready) →READ; DEP(count 0)
p_2: (count ?x_1)(m ≠?x_1)(n ?x_2)→DEP(count (s ?x_1)); DEP(n (s ?x_2))
p_3: (count ?x_1)(m ?x_1)(n ?x_2) →WRITE(?x_2)
p_4: (n ?x_1) →DEP(n_0 ?x_1)
p_5: (count ?x_1)(n ?x_2)(n_0 ?x_3)→?x_4←GENSYM; PUT(?x_4,((m ?x_1)(n_0 ?x_3)),(WRITE ?x_2),$p_1 p_4$?x_4)

(The notation obeys typical AI language conventions i.e. variables prefixed by ? have values bound to them and when bound must match that value, and variables prefixed by ≠? are taken to match anything other than the value bound).
With the control language $p_1 p_2{}^* p_3$ the system performs addition without learning anything, with the control language $p_1 p_4 p_2{}^* p_3 p_5$ we have essentially the example in [17], and with the control language $p_1 p_4 (p_2 p_5)^* p_3$ the system learns intermediate results as well.
The important point to note is that adaptive behaviour and additivity - in the sense of being able to modify the production set - is not precluded by the introduction of a control component. Further, a control language need not, and rarely will, specify all and only those production sequences that represent solution paths, but will simply place (greater or lesser) constraints on production invocation. Thus the synergic properties of the system will also be retained.

3.2 Semantic Specifications

In the above example we achieved additivity at the expense of having to explicitly augment the control language. Now in many applications such explicit modification of the control language will be difficult.

For example, consider the problem of cascading two amplifiers [10]. We could represent this problem using productions of the form

t_0 (make cascade-amp)--->(make collector)(make emitter)(make couple)
t_1 : (make collector) -►...
t_2: (make emitter) -->...
t_3 : (make couple) --->...

together with the control constraint given by the regular expression $(t_1 t_2 + t_2 t_1) t3$. In English, this simply says that to cascade two amplifiers we first construct a common collector and a common emitter (in any order) and then couple them. Now if we add to the production set a new method t_4 for making collectors this new method will never be considered unless we explicitly substitute ($t_1 + t_4$ for each occurence of t_1 in the above control sequences. Furthermore, we should do the same thing for every control sequence in which t_1 appears, not just those appearing above.

The problem clearly lies in the means used for specifying the control language. Let us briefly consider what's going on. Most languages are specified syntactically because of a desire to describe the language solely in terms of its alphabet without regard to any semantic interpretation of the language. But in the present case there is no need to be so restrictive. As the interpretation is a component of the formal model (see eqn 2.2), it can also be used in specifying the control language, that is, the control language can be specified by its *semantic content* rather than its syntactic form. Thus in the above example, we could have specified the control constraint semantically:

if p_1 is a production that makes a collector (or whose LHS mentions 'collector')
and p2 is a production that makes an emitter
and p_3 is a production that couples them,
then p1P2P3 and p2p1p3 are in the control language.

p1,p2, and P3 are variables ranging over the set of productions, and the specification says, in effect, that a production that makes a common collector and a production that makes a common emitter should be invoked before a production that couples the two components.

Semantic specification thus avoids the problem of explicit modification of the control language, and additivity is better realized. Most other properties desired of knowledge based systems are also retained - the system remains highly modular and flexible, and retains its explanatory capacity. Of course, the dependence of these properties on the means of production reference is well known [2] -the only difference is that here we are using semantic reference to effect control constraints rather than to improve system performance.

Semantic control information is also the type of information most likely to be possessed by an expert interacting with such a knowledge based system. For example, consider a world consisting of some blocks and a (large) table. Now we know that in an optimum solution we will never put one block on top of another block that has to be subsequently cleared, which we may express as follows:

if the action part of a production puts a block x on a block y then it cannot be followed (no matter how much later) by a production whose condition part requires y to be clear.

We can then interpret this directly as a semantic specification of the control language. (Interestingly, such a simple control constraint considerably reduces the search space as, for example, production sequences of the form

... (PUTON x y) ... (PUTON y z) ...

are excluded from consideration.)

One problem with implementing such a scheme is that in the general case it is very difficult to extract the required semantic information from a body of code. The problem is not so severe in (controlled) production systems because the code usually consists of a number of relatively small, independent chunks of knowledge, and these often represent computationally primitive operations. More important, however, is that in a CPS the control information is specified in such a way that semantic criteria can be used as a means of invocation——how the semantic information is obtained can be treated as a separate issue [3].

The other problem is one of system efficiency. Even in cases where the object level system is made more efficient, the time spent in evaluating semantic criteria can easily offset these gains. However, if the production set is fixed then it is not necessary that the semantic criteria be evaluated at execution time——in many cases it will be more efficient to transform the semantic specification into a syntactic specification at compile time. In these cases, semantically based invocation can be expected to provide both a powerful and relatively efficient means of control.

4. CONTROL LANGUAGE DEVICES

Any desired control constraint can be specified by a control language. Tn turn, however, we require some device for specifying control languages. Now we are in a better position to construct such a device if we know what class of languages the device is required to generate. Thus in this section we examine some typical AI control constructs and see how they are placed within the standard hierarchy of languages [14].

4.1 Control Language Types

One of the simplest classes of languages is the type 3 or regular languages. As control languages they are surprisingly powerful——thus, as mentioned in Section 2.1, they can increase the power of context-free rewriting systems to that of a Turing machine. As the solutions to many problems are often conceptualized as a sequence of actions, regular control languages also improve construetibility.

Both direct sequencing and iterative control can be specified by means of regular control languages (e.g. see the example of Section 3.1). This class of languages is also powerful enough to realize partitioned production systems (procedures [11], packets or multiple production

memories [9]). If we consider the control language to be generated by a transition network, then each state in that network defines a set of active productions, these being the productions that label the outgoing arcs. Each transition either loops and thus leaves control in the same state (i.e. with the same set of active productions) or transfers control to another state (i.e. to another set of active productions). Thus as long as we continue to loop on a given state we effectively operate on a subset of the entire production memory.

Context-free control languages can be used to specify recursive procedures or goal-subgoal control constructs where the subgoals are to be achieved in a specified order. This is readily seen as follows. Let the control language be generated by a context-free grammar. We can interpret each non-terminal symbol appearing in the grammar as the name of a procedure in the CPS, and each terminal symbol simply as the name of a production in the CPS. Now (top-down) left-right generation of control language sentences produces possible execution sequences of the CPS, where each expansion of a non-terminal symbol is interpreted as a call to the procedure having that name. In AI terms, each expansion of a non-terminal is equivalent to the reduction of a goal to a sequence of subgoals. Such control constraints correspond to the SUCCESSOR and SUBTASK relations in NASL [10] and to the fall-back and held-result-useage control constructs described in [13].

Let us consider again the problem of constructing a cascade of amplifiers. The solution proposed in Section 3.2 was not really sufficient, as firstly it did not restrict the scope of the control constraint to the case of cascading amplifiers, and secondly because it required the subtasks (i.e. the making of the common collector, etc.) to be primitive. However, we can overcome both these difficulties by letting the production names stand as non-terminals in a context-free grammar which contains the rules

$$t_0 \rightarrow t_1 t_2 t_3 \qquad t_0 \rightarrow t_2 t_1 t_3$$

The result is that in trying to achieve t_0, we have to achieve t_1 and t_2 before t_3, where this time t_1, t_2 and t_3 may themselves be expanded into further (sequences of) subtasks. (In fact, we have really changed the entire nature of the object level production system, as it now need contain only primitive productions (i.e. productions with no subtasks). For our purposes this is not important, but it does indicate that much domain specific knowledge, in our terms at least, occurs at the control level).

One particularly interesting aspect of context-free CPSs is that they encompass augmented transition networks (ATNs) [18]. An ATN is simply a controlled production system where the context-free control language is specified not by a phrase structure grammar but by a recursive transition net. The conditions and actions on each arc of the transition net are simply the conditions and actions denoted by the LHS and RHS of the production corresponding to that arc. Thus an input word to an ATN simply acts as a control word (strictly a homomorphism of a control word) over the productions attached to each arc of the network.

There are also a number of control constructs which cannot be specified by context-free control languages. For example, the control language of the blocks world example (Section 3.2) is not a context-free language. Similarly, direct-result-usage control constructs [13] need language generators of the form merge (t_1, t_2), where t_1 and t_2 act as non-terminals in a phrase structure grammar and merge (t_1, t_2) is to be interpreted as providing all merges of the words generated by t_1 with those generated by t_2. Again, such languages are not context-free.

Of course, we are not suggesting that we should use phrase structure grammars for specifying control languages - what we are saying is that we need to make sure our control device, whatever it is, has the power to describe the types of control that we need. For example, if we are to allow context-free control languages - and the above considerations suggest that we should - then the type of productions used in TEIRESIAS [2] for specifying production orderings at the meta-level are not going to be powerful enough at the control level as they can only describe regular languages.

4.2 Control Device Structure

We have so far said nothing about the structure of the device to be used in specifying the control language. If the specification is to be syntactic, then any of the standard methods——explicit enumeration, property specification, finite state automata, phrase structure grammars, augmented transition networks, etc.——could be used. These devices could also be used where the control language is to be specified semantically, but would need to be augmented with some inference device. Alternatively, the system could be provided with a number of control primitives, such as those used in NASL [10] or described in [13]. Of course, for a CPS these primitives would be implemented through a separate control device, rather than indirectly through the conflict resolution scheme.

However, all the above approaches limit, to a greater or lesser degree, the additivity and

flexibility of the system to the object level. A potentially more powerful approach is thus to specify the control language using a second level or *control level* CPS. This reflects very closely what McDermott had in mind when designing NASL - that the order of steps within and between subplans be itself rule governed. One advantage of this approach is that the programmer has available a language and framework in which he can easily define his own invocation criteria. Furthermore, the representation of knowledge at both the object level and control level is uniform, and additivity and flexibility is preserved at both levels.

The control language of the control level CPS would need to be specified, and one could envisage a hierarchy of CPSs, each determining the control language of the one below it. There is probably not much advantage in such a wealth of control language CPSs. Of course, this is not to suggest that there should be no further levels of knowledge - for example, it might be desirable to handle non-determinism using a meta-level CPS [1,3,5].

5. CONCLUSIONS

The importance of control information should not be underemphasized. In placing constraints on production invocation, control reduces the interaction between knowledge units. The more control constraints we impose, the fewer patterns of interaction have to be explored, and the smaller and less complex the search space. To paraphrase Hayes [7] it is precisely the resulting restrictions on interactions that makes control so useful.

In this paper we have presented a uniform framework in which control issues can be addressed, and which provides a basis for implementing such control. We have attempted to show that the introduction of a explicit control device need not result in the loss of any of the properties desired of a knowledge based system, and in particular that additivity and adaptive behaviour can be retained. Given that the power of production systems is equal to that of controlled production systems, the question remains as to whether it is advantageous to introduce control at all. At an intuitive level it would appear that the answer is yes - it is not difficult to find problem areas where expert knowledge is expressed in terms of sequences of actions. Further, if we look at the formal language domain, regulated (controlled) rewriting systems are usually simpler to construct than equivalent (uncontrolled) rewriting systems. Of course, until a system is actually implemented and applied to typical AI problems, the question must still remain open.

REFERENCES

[I] Davis, R. G King, J.J. "An Overview of Production Systems", in Mach. Int. 3: Machine Representations of Knowledge, ed. Elcock & Michie, Wiley, N.Y., 1977.

[2] Davis, R. "Generalized Procedure Calling and Content-Directed Invocation" SIGPLAN'/SIGART Newletter, August, 1977, pp. 45-54.

[3] Davis, R. & Buchanan, B.C. "Meta-Lcvcl Knowledge: Overview and Applications", Proc. IJCAI-77, 1977, pp. 920-927.

[4] Feigenbaum, F..A., Buchanan, B.C. & Lederberg, J. "On Generality and Problem Solving—a case study involving the DENDRAL program", STAN-AIM-131. Stanford Uni., Ca., 1971.

[5] Georgeff, M.P. "A Framework for Control in Production Systems" Stanford AI-Memo, Stanford Uni., Ca., 1979.

[6] Goldstein, I.P. & Grimson, E. "Annotated Production Systems: A Model for Skill Acquisition" Proc. IJCAI-77, 1977, pp. 311-317.

[7] Hayes, P.J. "In Defense of Logic", Proc. IJCAI-77, 1977, pp. 559-565.

[8] Lenat, D.B. "Automated Theory Formation in Mathematics", Proc. IJCAI-77, 1977, pp. 833-842.

[9] Lenat, D.B., & McDermott, J. "Less Than General Production System Architectures". Proc. IJCAI-77, 1977, pp. 928-932.

[10] McDermott, D. "Flexibility and Efficiency in a Computer Program for Designing Circuits", AI-TR-402, MIT AI Lab, 1977.

[II] Moran, T.P. "The Symbolic Imagery Hypothesis: A Production System Model", Comp. Sci. Dept., Carnegie-Mellon University, December, 1973.

[12] Post, E. "Formal Reductions of the General Combinatorial Problem", Am Jnl Math, 65, 1943, pp. 197-268.

[13] Rychener, M.D. "Control Requirements for the Design of Production System Architectures" SIGPLAN/SIGART Newsletter, 1977, pp. 37-44.

[14] Salomaa, A. Formal Languages, Academic Press, N.Y., 1977.

[15] Shortliffe, E.H. MYCIN: Computer-Based Consultations in Medical Consultations, American Elsevier, 1976.

[16] Warren, D.H.D. "Implementing Prolog—Compiling Predicate Logic Programs" AI-Report, Edinburgh Uni., 1977.

[17] Waterman, D.A. "Adaptive Production Systems" Proc. IJCA1-75, 1975, pp. 296-303.

[18] Woods, A.T. "Transition Network Grammars for Natural Language Analysis", Comm ACM, 13:10, 1970, pp. 591-606.

A MULTI-LEVEL PLANNING AND NAVIGATION SYSTEM FOR A MOBILE ROBOT; A FIRST APPROACH TO HILARE

Georges Giralt, Ralph Sobek*, and Raja Chatila

Laboratoire d'Automatique et d'Analyse des Systemes du CNRS
7, avenue du Colonel-Roche
31400 Toulouse - France

This paper describes the current state of HILARE: a modular progressively-built mobile robot aimed at general robotics research. The computer organization comprises of local mini and micro processors coupled with a remote time-shared system acting as a consulting facility. A multi-level decision-making system is presented with world models, inference rules, and algorithms particular to each level. The navigation planner uses a geometric model wherein 2-space is partitioned into polygonal areas based on perceptual and/or initial information. A cost function is proposed which provides support for optimal or E-optimal path finding.

1. GENERAL DESCRIPTION AND PURPOSE OF HILARE

The HILARE** project started in Sept. 1977 as an attempt to provide the LAAS robotics group, as well as several other teams interested in that field in Toulouse, with a flexible and powerful experimental support for advanced research work on Robotics: a particular blend of Artificial Intelligence, Computer Science, Control Theory, and Instrumentation.

The choice of an autonomous mobile robot was made mainly for two reasons: 1) capacity to provide a variety of problems at different levels of generality and difficulty in a large domain - including perception, learning, decision-making, communication, etc., which all have to be considered within the scope of the specific constraints of robotics: on-line computing, cost considerations, operating ability and reliability; and 2) possibility to design a modular system which could be incrementally built and yet providing at every step for interesting and useful research in many directions.

Research goals were programmed in these domains in accordance with the development of the system configuration. We emphasize that HILARE has no practical purpose in itself yet we are deeply concerned in every stage of our work with methods and techniques, software and instrumentation for advanced applied robotics.

* Presently at LAAS; visiting researcher from the University of California, Berkeley, Calif. Visit made possible in part by the National Science Foundation Grant INT78-09263.

** Heuristiques Integrees au Logiciel et aux Automatismes dans un Robot Evolutif.

To date the system architecture (see Fig. 1) consists of (cf. /1/): 1) Perception: video camera and laser range-finder mounted on a two-axes scanning system and integrated to provide fast rough scene analysis consistent with navigation needs; 2) Locomotion: one free front wheel and two motorized rear drive wheels; the microprocessor controlled step motors allow for a variety of speed and trajectory commands: straight lines, arcs with specified angular shift and rotation center; and 3) Computer System.

One of the more relevant characteristics which will aid advanced robotics with real-world constraints, such as cost-effectiveness, is an appropriate computer system structure. Our organization (being investigated in several projects at LAAS) consists of 1) local distributed computing facilities coordinated by a 16-bit MITRA 15 minicomputer, and 2) a remote time-shared IBM 370/168 system. Local facilities include on-

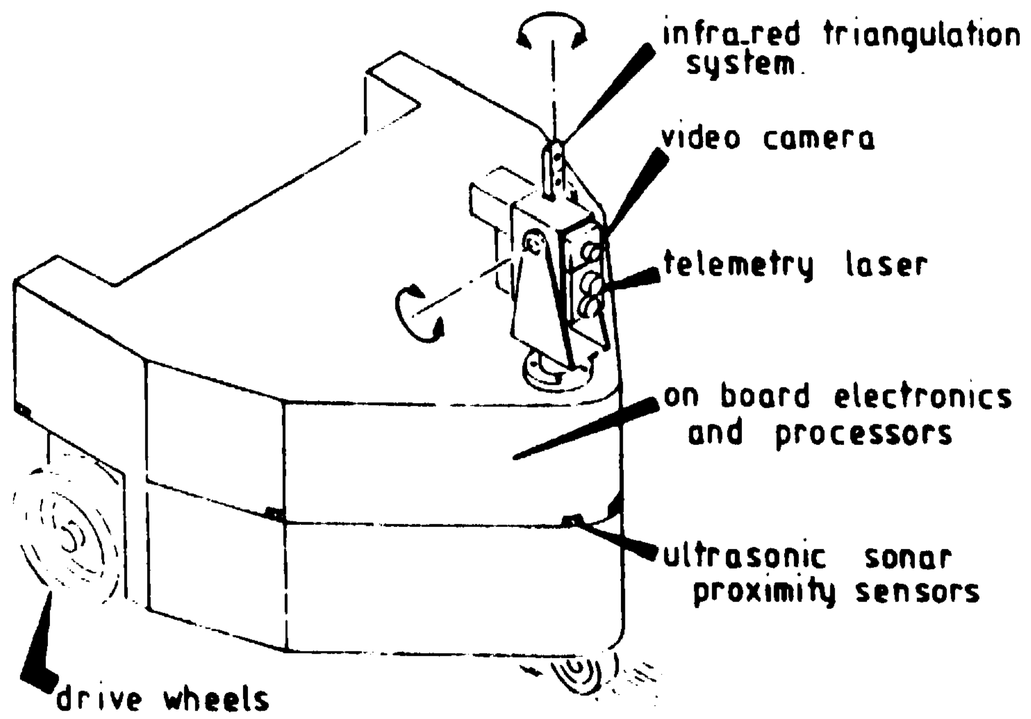


Fig. 1 - Present Configuration of HILARE

board speciallized processing units (e.g., image preprocessing) together with 3 Intel 8085 microcomputers. The MITRA 15 supports executive, decision, and communication functions. The remote computer is programmed as a consultant system (see §2); it provides powerful resources in tasks such as learning, reprogramming, procedure optimization, large data-base consulting, etc.

2. DISTRIBUTED DECISION-MAKING

In the system design of HILARE we have decided on a distributed decision-making capacity which allows robustness, efficiency, economy of operation and timely system response. Our design can be viewed as decisions through multiple cooperating expert modules/5/ together with a high-level coordinator in a hierarchical means-ends structure.

The expert modules have their expertise in a variety of overlapping domains (e.g., object identification, navigation, exploration, itinerary planning). The modules consist of 1) specialized and redundant knowledge bases, 2) algorithms and heuristics, 3) local error-processing capabilities, and 4) communication procedures. Modules may access one another as primitive action-units. The coordinator activates modules based on a means-ends analysis/2/ of the current situation.

2.1 High-Level Decision-Making and Planning

We are designing and programming a relational-level planner at an advisor level. A plan at this level is a flexible dynamic structure which coordinates the achievement of desired goals/8/. Goal understanding includes 1) acknowledging or refusing an user order, 2) handling temporal and logical constraints, and 3) finding action-units which can achieve goals/7/. The planner must also include coordination of error processing, broadcast, and recovery procedures/9/.

Action-units may be primitive or macro actions (viz., entry-points into expert modules) /3/. The planner's required knowledge about actions includes limiting constraints and activation, continuation, and termination conditions.

Due to the incremental and open-ended nature of HILARE's design we consider Production Systems (PSs) a viable research tool for relational-level problem-solving. Thus, we are writing the planner in terms of a PS architecture in the spirit of /6/.

2.2 Relational-Level World Model

The world model is contained in the PS data base: embedded non-recursive list structures which are associatively accessed by the pattern matcher during rule matching. The model is a hierarchy of object-centered concepts of the form

(name feature pattern-body)

which describe space and objects. Space is defined by places (rooms, work-areas), frontiers (doors), and locations. Places have the property of surface connexity and are connected to other places by means of frontiers. Locations are elements within places which can be identified by point coordinates, relational descriptions, or feature descriptions.

Objects are defined by features (e.g. , shape, color, and dimension) based on sensory perception. We define macro-objects by recursive assembling and aggregation of objects.

3. NAVIGATION

In this section we consider the basic problem of moving a robot from an initial state (R) to the target (G) within a given place. This involves obstacle avoidance, path-finding, and search trajectory minimization/4,10/.

3.1 Navigation World Model

Currently, obstacles are defined as polyhedra, whose floor projections fully determine the navigation problem, and they can be located either by initial information or by robot perception (see Fig. 2). Each obstacle projection is represented as an ordered list of segments in counter-clockwise sequence: e.g.,

$$Obs_1 S_1\}, S_2\}, \ S_3, \ S_4)$$

A segment is coded using the Cartesian coordinates of its leftmost point(X,Y), its angle with a reference axis(), and its length(r):

$$S. \ \{<X,Y>, \ \Phi, \ r\}$$

Empty areas are defined as convex polygonal cells which include obstacle segments. A trajectory within such cells can be considered as a straight line between entry and exit segments. In such cells ordering of segments S. is clock-

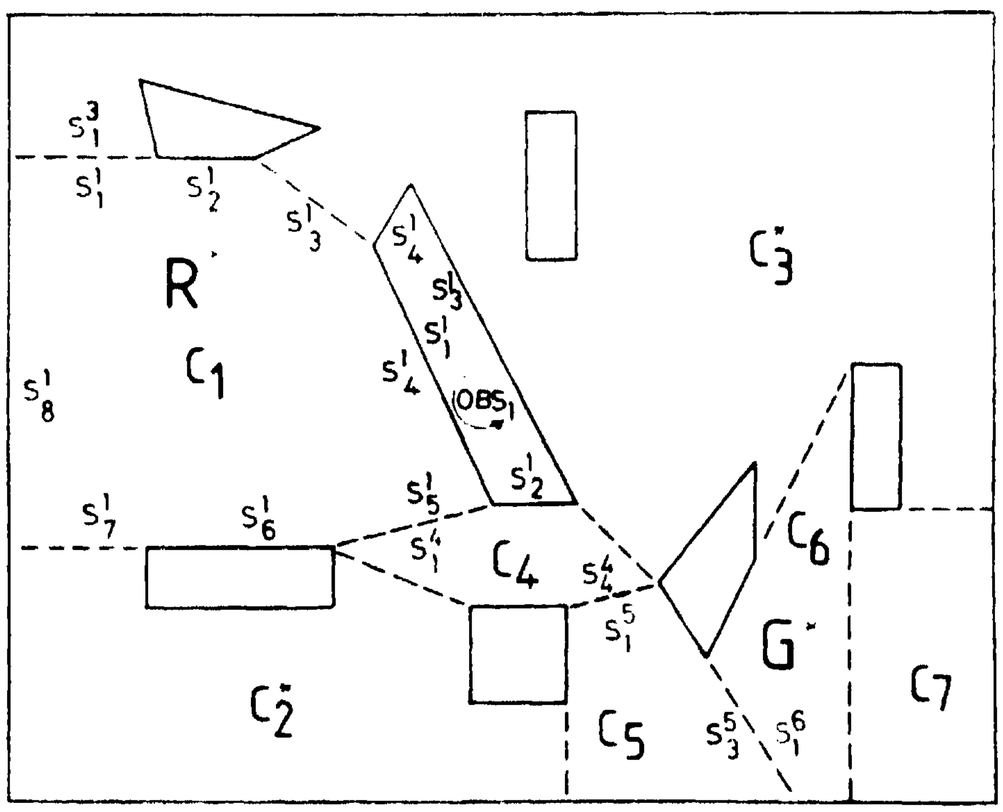


Fig. 2 - Cellular Decomposition of a World

wise, e.g., C_1 $\{S_1^1, S_2^1, \ldots, S_8^1\}$.

Our partition algorithm which divides a known world into C-cells and obstacles is as follows. Starting at any vertex a C-cell is defined by following segments in a clockwise order as long as the angle between two successive segments is acute. When not, the algorithm jumps to the next visible segment, such that an acute angle is formed and a convex cell is possible, and continues until a convex polygon is formed.

Any area including possibly unknown obstacles is defined as a polygonal *-cell, which is allowedly non-convex, e.g., C_3^* $\{S_1^3, S_2^3, \ldots, S_{15}^3\}$.

The topological properties of a place - divided into C-cells, *-cells, and obstacles - are graph represented. Two adjacent cells have connectivity through common segments which are traversable by the robot. Thus, a connectivity graph (see Fig. 3) provides the structure necessary for path finding.

3.2 Decision Procedure

The example in Fig. 2 produces among possible paths a single path between R and G which is completely determined since it does not include *-cells: $C_1(R) \xrightarrow{S_5^1} C_4 \xrightarrow{S_4^4} C_5 \xrightarrow{S_3^5} C_6(G)$. In most cases several paths would exist and a choice would be made through ε-optimality concepts.

We define our cost function as: $F = \Sigma_i (F_i^1 + F_i^2)$; i = cell index. F_i^1 measures every cost associated with determined trajectory travel. To minimize distance and energy requirements we have: $F_i = \omega_1 d_i + \omega_2 |\phi_i| + \omega_3 n_i$, where for cell i 1) d_i is path length, 2) $|\phi_i|$ is the angle of planned direction change, 3) n_i is the number of predicted robot stops, and 4) ω_i are weights.

F_i^2 measures uncertainty and is evaluated over *-cells: $F_i^2 = \omega_4 d_i (I*V - 1)$, where I is a decreasing positive function of the cell information which can be obtained by the robot at the entry segment: $I = S_T/S$, and V measures path viability due to estimated obstacle clustering: $V = 1/(1-r)$, S is the maximal perceived area at the considered entry and S_T is the total *-cell surface. T is the estimated ratio of aggregated obstacle surface over the total *-cell surface (S_T).

The minimization of F provides a method for path-finding within a world which can be stepwise perceptually identified by the robot.

In the case of partially known worlds cell constuction is defined by the perceptual system as HILARE discovers its world. It may occur that non-traversable narrow cells are created; whereas another partitioning would allow traversal.

3.3 Object Close-Distance Navigation

Consistent with our viewpoint on multi-level decision-making the problem of navigation can be resolved more efficiently with the introduction of a decision level which takes into consideration a variety of navigation conditions which call for on-board closed-loop control, such as narrow corridor navigation, etc. This control is performed by HILARE using 10 close-range ultrasonic sensors regularly distributed over its surface. The higher decision levels including navigation set the objectives and the constraints for close-distance navigation with take-over and restitution of control locally defined.

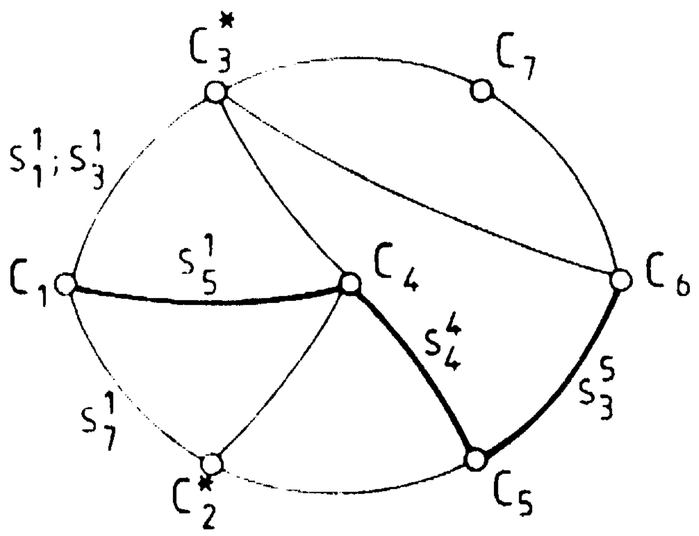


Fig. 3 - Connectivity Graph

REFERENCES

1. Briot, M., G. Bauzil, and R. Chatila. "Presentation d'HILARE, un Robot Mobile." Journees d'Automatique de 1'IRISA, Rennes (Jan. 1979).
2. Fikes, R. and N. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving." Art. Int. 2:3/4 (1971) 189-208.
3. Fikes, R., et_ al. "Learning and Executing Generalized Robot Plans." Artificial Intelligence 3:4 (1972).
4. Hart, P., et_ al. "Artificial Intelligence — Research and Applications." Annual Technical Report, Contract DAHC04-72-C-0008. SRI (Dec. 1972).
5. Lenat, D. "Beings: Knowledge as Interacting Experts." In Proc. of IJCAI-75. Tblisi, USSR (Sept. 1975) pp. 126-133.
6. Rychener, M. "Production Systems as a Programming Language for Artificial Intelligence Applications." Ph.D. Thesis, Computer Science Dept. Carnegie-Mellon University (Dec. 1976).
7. Sacerdoti, E. A Structure for Plans and Behavior. New York: Elsevier Publishing Co., 1976.
8. Sobek, R. "Automatic Generation and Execution of Complex Robot Plans." Master's Report. EECS Dept., Univ. of Calif., Berkeley(Sept.1975).
9. Srinivas, S. "Error Recovery in Robot Systems." Ph.D. Thesis. Computer Science Dept., California Institute of Technology (1977).
10. Thompson, A. "The Navigation System of the JPL Robot." In Proc. of IJCAI-77. Cambridge, Mass. (Aug. 1977) pp. 749-757.

META - INTERPRETATION

OF RECURSIVE LIST-PROCESSING PROGRAMS

Daniel GOOSSENS
Univsite ParisVIII
Route de la Tourelle
75012 Paris, France

Due to late receipt, this paper appears in full in the Supplement section, back of Volume II.

PROGRAM SYNTHESIS FROM NATURAL DEDUCTION PROOFS

Shigeki Goto
Musashino Electrical Communication Laboratory
Nippon Telegraph and Telephone Public Corporation
Musashino-shi Tokyo, 180 JAPAN

There have already been many program synthesizers which are based on classical logic. There is, however, some evidence that intuitionistic logic is more suitable for computer programs. This paper presents a program synthesis method based on intuitionistic logic. The synthesizing method is essentially an application of Godel's interpretation. An experimental program synthesizer NJL, which performs Godel's interpretation, is implemented in LISP. NJL takes natural deduction proofs as input and produces LISP programs.

1. Introduction

In this paper, programs on natural numbers will be treated mainly and the number theory based on intuitionistic logic is considered.
Program synthesis by theorem proving approach treats the logical formula which represents the specification of the desired program. Typical specification forms: (Manna and Waldinger [1])

$$\forall \bar{x}[\varphi(\bar{x}) \supset \exists \bar{z}\psi(\bar{x},\bar{z})] \quad \text{------(1)}$$

Here, x is an input vector x=(x1,x2,... ,xm) and I is an output vector z=(z1,z2,... ,zn), Φ(x) is an input predicate and $\psi(\bar{x},\bar{z})$ is an output predicate. The aim of program synthesis is to construct a function F such that z=F(x).
The present method is based upon Godel's interpretation, which transforms (1) into (2) below:

$$\exists \bar{z}\forall \bar{x}[\varphi(\bar{x}) \supset \psi(\bar{x},\bar{z}(\bar{x}))] \quad \text{------(2)}$$

Godel's interpretation assures that there exists a term t such that $\forall \bar{x}[\varphi(\bar{x}) \supset \psi(\bar{x},\bar{t})]$ holds. The term t is determined through the proof of (1) and the synthesized program can be defined as $F=\lambda \bar{x}.\bar{t}$.

2. Logical framework

To handle logical formulas in an intuitionistic framework, a good choice is to adopt the natural deduction system, which was devised by G.Gentzen. It is mentioned in many logic textbooks (e.g. Troelstra[4]).
Before giving Godel's theorem, several definitions are necessarily introduced, since Godel's interpretation utilizes primitive recursive functionals of finite type.

Def.I (Type)

1. Symbol 0 is a type.
2. If σ and τ are types, then $\sigma(\tau)$ is also a type.

Although the same symbol "0" is used, type symbol 0 should be distinguished from natural number 0.

Def.2 (Finite type functionals)

1. Every natural number is a type 0 functional.
2. If type σ functional and type τ functional are already defined, then a type $\sigma(\tau)$ functional is a certain map from the set of type T functionals into the set of type σ functionals.

The set of finite type functionals is an extension of the natural number. It is necessary to extend intuitionistic number theory to treat finite type functionals.

Def.3 (System FT)

1) FT terms and primitive recursive functionals:
   i) 0 (natural number) is a type 0 term.
   ii) Each free variable of type σ is a type σ term.
   iii) If f is a type 0 term and S is the successor function, then S(f) is also a type 0 term.
   iv) If f is a type σ term and x is a bound variable of type $\underline{\tau}$, then λxf is a type $\sigma(\tau)$ term.
   v) If f is a type $\sigma(\tau)$ term and x is a type t term, then f(x) is a type σ term.
   vi) If g is a type τ term and h is a type $\tau(0)(\tau)$ term, then p[g,h] is a type $\tau(0)$ term. $\rho[\ ]$ means the primitive recursion,
   vii) A term is called a primitive recursive functional, if and only if it contains no free variables.
2) FT axioms and rules:
   i) All axioms and rules of intuitionistic number theory, except what concern the quantifiers, are taken as FT axioms and rules.

ii) Additional axioms and rules are:

Axioms for λ (lambda)

$(\lambda x f(x))(t)=f(t)$,

$\lambda x f(x)=f$,

If f and g are the same, except bound variables, then f=g.

Axioms for $\rho[\ ]$

$\rho[g,h](0)=g$,

$\rho[g,h](S(n))=h(n,\rho[g,h](n))$.

Axiom for ( )

$f=g \wedge x=y \supset f(x)=g(y)$

Substitution rule

Any term can be substituted for a free variable of the same type.

FT has no quantifier symbols. To treat quantifiers, the system QFT is formulated.

Def.4 (QFT)

QFT is a quantificational extension of FT.

1) Each formula in FT is also a formula in QFT.

2) If A(a) is a QFT formula and "a" is a free variable, then $\forall x A(x)$ and $\exists x A(x)$ are also QFT formulas, where x is a bound variable of the same type as "a" and x never occurs in A(a).

QFT axioms and rules are not explicitly defined here, since they are not used in this paper.

To each formula of intuitonistic number theory, Gödel's interpretation associates a formula of the form $\exists x \forall y A(x,y)$, which is called $\exists\forall$-formula in QFT.

Def.5 ($\exists\forall$-formula)

To each formula A in intuitionistic number theory, a $\exists\forall$-formula A* in QFT is associated. The definition is by induction on the logical complexity of A.

0) If A is a prime formula, then A* is A.

From (1) to (4), it is assumed that B and C are subformulas of A and that B* and C* are already defined. B* is $\exists x \forall y B'(x,y)$ and C* is $\exists u \forall v C'(u,v)$.

1) If A is $\neg B$, then A* is $\exists z \forall x \neg B'(x,z(x))$.

2) If A is $B \wedge C$, then A* is $\exists x \exists u \forall y \forall v (B'(x,y) \wedge C'(u,v))$.

3) If A is $B \vee C$, then A* is $\exists d \exists x \exists u \forall y \forall v [(d=0 \wedge B'(x,y)) \vee (d \neq 0 \wedge C'(u,v))]$.

4) If A is $B \supset C$, then A* is $\exists w \exists z \forall x \forall v [B'(x,z(x,v)) \supset C'(w(x),v)]$.

For (5) and (6), it is assumed that F*(a) is $\exists x \forall y F'(a,x,y)$.

5) If A is $\forall u F(u)$, then A* is $\exists z \forall u \forall y F'(u,z(u),y)$.

6) If A is $\exists u F(u)$, then A* is $\exists u \exists x \forall y F'(u,x,y)$.

On the interpretation, the important theorem holds.

Theorem (Gödel)

If a formula $A(\bar{z})$, containing at most $\bar{z}$ free, is provable in intuitionistic number theory and $\exists\forall$-formula $A^*(\bar{z})$ is $\exists \bar{x} \forall \bar{y} A'(\bar{x},\bar{y},\bar{z})$, then there exists a vector $\bar{t}$ of primitive recursive functionals in FT, such that $A'(\bar{t}(\bar{z},\bar{y}),\bar{y},\bar{z})$ is provable in FT.

The proof is so lengthy that it cannot be shown here (Troelstra[4]). Roughly speaking, the theorem is proved by interpreting axioms and rules of intuitionistic number theory into FT axioms and rules. For example, the induction rule (IND) in intuitionistic number theory can be interpreted in FT as follows:

$$\text{IND}\quad \frac{\begin{matrix}[A(a)] \\ A(0) \quad A(Sa)\end{matrix}}{A(t)}$$

a is an eigen variable.
S is the successor function.
t is any term.

It is not expedient to apply Gödel's interpretation directly to a natural deduction, since it is not easy to handle the assumption dependency. The present system converts the original natural deduction proof into the LJ-like sequential form. The conversion technique is described in Gentzen[5]. For simplicity, it is assumed that neither A(0) nor A(Sa) depends on other assumptions.

$$\frac{\rightarrow A(0) \quad A(a) \rightarrow A(Sa)}{\rightarrow A(t)}$$

If A(a) is $\exists z \forall w B(z,w,a)$, then the corresponding induction rule in FT is

$$\frac{B(t0,w,0) \quad B(z1,t1(a,z1,w2),a) \rightarrow B(t2(a,z1),w2,Sa)}{B(\rho[t0,t2](t),w,t)}$$

Other axioms and rules are interpreted similarly and Gödel's theorem can be proved by induction on the complexity of $A(\bar{z})$ proof.

3. Synthesizer NJL

This section describes synthesizer NJL, which is a PDP-11 program written in LIPQ[3]. NJL is named after NJ (intuitionistic natural deduction) and Lisp. The rest of this paper illustrates an NJL synthesis example. Gödel's theorem is applied to program synthesis as follows:

(1) Make a logical formula expressing the specification of the desired program. Here, the "quotient remainder theorem" is taken for an example:
$\forall x1 \forall x2 [x2 \neq 0 \supset \exists z1 \exists z2 (x1=x2.z1+z2 \wedge z2<x2)]$, where x1 is a dividend, x2 is a divider, z1 is the quotient and z2 is the remainder.

(2) Prove the formula of (1) in the number theory. A brief sketch of the proof is

$$\text{IND}\quad \frac{\begin{matrix}[QR(a,b)] \\ QR(0,b) \quad QR(Sa,b)\end{matrix}}{\dfrac{QR(t,b)}{\forall x1 \forall x2.QR(x1,x2)}}$$

t is any term.

where Q(a,b) stands for ∃z1∃z2(a=b.z1+z2∧z2<b). NJL is equiped with proof checking capability.

(3) Convert the proof of (2) into the sequential form. This step is automatically performed in NJL system.

(4) Interpret the proof of (2) into FT. First, NJL makes the corresponding ∃∀-formulas to the sequential forms of (3). For instance, formula of (1) is associated with a ∃∀-formula

```
((E (Z1 X1 X2) (Z2 X1 X2)) ((X1 X2)
 (IMP NIL (IMP (NE X2 0)
               (AND (EQ X1 (+ (* X2 (Z1 X1 X2))
                              (Z2 X1 X2) ))
                    (LT (Z2 X1 X2) X2) )))))
```

In NJL, formulas are expressed in prefix form, i.e. ((E X)(A X)) for ∃xA(x), ((X)(A X)) for ∀xA(x) and (IMP A B) for A⊃B. The outer (IMP NIL A) means a sequent →A.

Next, the desired terms are determined by applying Gödel's theorem. The determination process is really a proof of Gödel's theorem for the formula of (1). NJL represents primitive recursive functionals by LISP programs.

Def.6 (Representation in LISP)

1) The natural number 0 is represented by the integer 0 in LISP.

2) Each free variable of type σ is represented by a variable in LISP program.

3) If f is a type 0 term and represented by f' in LISP, then S(f) is represented by (ADD1 f') in LISP1.6 or (+ f') in LIPQ.

4) If f is a type σ term, represented by f' in LISP, and x is a type τ term, represented by x' in LISP, then λxf is represented by (LAMBDA (x') f') in LISP.

5) If f is a type σ(τ) term represented by f' and x is a type τ term represented by x', then f(x) is represented by (f' x') in LISP.

6) If g is a type τ term represented by g' and h is a type τ(0)(τ) term represented by h', then ρ[g,h] is represented by

```
(LABEL R001
  (LAMBDA (X)
    (COND ((ZEROP X) g')
          (T (h' (SUB1 X) (R001 (SUB1 X)))))))
```

or

```
(DE R001 (X)
   (COND ((ZEROP X) g')
         (T (h' (SUB1 X) (R001 (SUB1 X))))))
```

7) Every term which contains no free variables, i.e. primitive recursive functional, is represented by a LISP program.

In this example, Z1 and Z2 below are the synthesized programs.

```
(DE Z1 (X1 X2)
 (COND ((ZEROP X1) 0)
       (T (COND ((LT (S (Z2 (- X1) X2)) X2)
                 (Z1 (- X1) X2) )
                (T (S (Z1 (- X1) X2))) ))))

(DE Z2 (X1 X2)
 (COND ((ZEROP X1) 0)
       (T (COND ((LT (S (Z2 (- X1) X2)) X2)
                 (S (Z2 (- X1) X2)) )
                (T 0) ))))
```

This ends the example.

It is well-known that induction rule corresponds to recursive program (Manna and Waldinger [1]).

4. Concluding remarks

Since Godel's interpretation utilizes the primitive recursive functional, which is an extension of the primitive recursive function on natural numbers, the synthesized program becomes necessarily total (Sato[6]).

NJL can be applied to LISP linear lists. In that case, an induction schema LS2 (McCarthy[7]) is used instead of IND.

ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks to Professor M.Sato at the University of Tokyo for his kind directions and advice. He also gratefully thanks Dr. N.Ikeno and LIPQ group of Musashino Electrical Communication Laboratory for their guidance and encouragement.

REFERENCES

[1] Z.Manna and R.J.Waldinger, Toward automatic program synthesis, Comm. ACM, vol.14, no.3, pp.151-165, 1971.

[2] S.Goto, Program synthesis through Godel's interpretation, Proceedings of The International Conference on Mathematical Studies of Information Processing, 1978.

[3] I.Takeuchi and H.Okuno, A list processor LIPQ, 2nd USA-JAPAN Computer Conference Proceedings, pp.416-421, August, 1975.

[4] A.S Troelstra, Metamathematical investigation of intuitionistic arithmetic and analysis, Lecture Notes in Math. 344, Springer, 1973.

[5] M.E.Szabo (ed), The collected papers of Gerhard Gentzen, North-Ho11and, 1969.

[6] M.Sato, Towards a Mathematical Theory of Program Synthesis, IJCAI-79.

[7] J.McCarthy, Representation of Recursive programs in First order logic, The International Conference on Mathematical Studies of Information Processing, 1978.

RESULTS IN
KNOWLEDGE BASED PROGRAM SYNTHESIS

Cordell Green, Richard P. Gabriel, Elaine Kant
Beverly I. Kedzierski, Brian P. McCune, Jorge V. Phillips
Steve T. Tappel, Stephen J. Westfold

Systems Control, Inc.
1801 Page Mill Road
Palo Alto, California 9-4304

Abstract. This paper reviews the entire PSI program synthesis system, summarizing progress made during the past two years. PSI synthesizes efficient programs from several types of abstract specifications. The paper presents a brief summary of PSI, an example dialogue demonstrating its performance, and a discussion of Its present capabilities. Explanation of the detailed operation of the system is omitted in this short paper. For an overview of prior work, see [Green-76]; for more details see [Ginsparg-78], [Steinberg-79], {Barstow-79], and [Kant-79] [Biermann-76] surveys related work in the automatic programming field.

Keywords and phrases. Automatic programining, knowledge based programming, program synthesis. program specification, program acquisition.

1. Summary of the PSI Program Synthesis System

The PSI program synthesis system is a computer program that acquires high level descriptions of programs and produces efficient implementations of them. PSI's operation *is* factored into two parts: the *acquisition phase,* which acquires the high level description, and the *synthesis phase,* which produces a program from it. Simple symbolic computation programs are specified through dialogues between the user and PSI. The specification techniques available include natural language, input-output pairs, partial traces, and a high level specification language currently under development. These specifications are integrated In the program model. It is also possible to specify a problem directly in the modelling language. LISP programs are produced from the model, but experiments have shown that the system can be extended to produce code in a block structured language such as PASCAL.

The design and implementation of PSI have involved many people. The system's modules are currently the responsibility of the following individuals: parser/interpreter, Beverly Kedzierski; explainer, Richard Gabriel; trace and example inference expert and domain expert, Jorge Phillips; program model builder, Brian McCune; and efficiency expert, Elaine Kant. Additional personnel have worked on various projects within these experts. *Steve* Tappel wrote the rule expander for the program model builder; Stephen Westfold enhanced the examples component of the Inference expert. A description of the interactions of the phases and modules of PSI can be found In [Green-77].

2. Recent Progress on and Present Capabilities of PSI

The entire PSI system can be run by a knowledgeable user. A number of tests of the entire system have been made PSI has successfully produced LISP code that implemented the specifications given *in* English dialogues for four variants of CLASS, a symbolic pattern classification program.

CLASS is a simple program that includes many programming constructs necessary for more complex programs. CLASS first inputs a sample set of items (e.g., a list of job qualifications). Then the program repeatedly inputs a trial set (e.g.. the qualifications of an applicant), tests whether the sample set (required qualifications) is a subset of the trial set (applicant qualifications), and prints "fit" or "didn't fit" accordingly. The program is exited if the word "quit" is typed instead of a trial set.

2 .1 Acquisition Phase

The following dialogue was held with the PSI acquisition phase to specify CLASS.

PSI: What is the name of the program you want to write?
USER: CLASS.
PSI: Does CLASS have a special program type?
USER: Classification.
PSI: Please describe CLASS.
USER: CLASS first inputs a concept. Then it repeatedly accepts an Input scene from the user, classifies it based on whether or not it fits the concept, and outputs this classification to the user.
PSI: What is the structure of the concept?
USER: A concept is a collection of relations. Each relation is just a string.
PSI: What Is the exit test of the loop?
USER: Stop when the user types "quit" instead of a scene.
PSI: What is the structure of the scene?
USER: The same as the concept.
PSI: Please define what it means for a scene to fit a concept.
USER: A scene fits a concept if all the relations in the concept are in the scene.
PSI: What Is the format of the classification output?
USER: Output "fit" if the scene fit, "didn't fit" if it didn't.
PSI: Are we finished?
USER: Yes

From the above dialogue the parser/interpreter [Ginsparg-78] produces a program net, a common representational base used by all modules of the acquisition phase. The parser/interpreter understands over seventy programming concepts and has a vocabulary of more than 175 words. Its programming concepts include data and control structures, primitive operations, and more complicated algorithmic ideas (e.g., user-program interchanges, set construction, quantification). The system can sometimes determine the meaning of unknown words (e.g., what concept they represent) from the context in which they appear. The dialogues that the system has understood include those specifying many variants of CLASS, several variants of NEWS (a news story retrieval program), TF (a learning program that uses CLASS as a subroutine), and graph reachability.

The dialogue moderator [Steinberg-79] chooses which of the questions posed by the parser/interpreter to ask next. A number of simulated dialogues have been gathered, with a member of the PSI group playing the role of PSI and people not part of the group as users. The question choosing algorithm of the dialogue moderator has been improved by comparing its behavior with the data from these dialogues. The moderator also has mechanisms (not yet interfaced to the rest of PSI) to answer the question, "Where are we?", and most of the mechanism needed to handle a request to change topic. The moderator has handled dialogues for NEWS and variants of CLASS

The questions that are asked of the user are quite readable and coherent. Questions use the same terms as the user did in previous sentences of the dialogue. For example, rather than asking for the definition of "A00I8'', PSI asks what it means for "a scene to fit a concept". The question generation system has been used in the dialogues for CLASS, NEWS, RECIPE (a recipe retrieval program), and TF. It produces about twenty substantially different sentence types. The question generator is being expanded into a more general explainer that will explain *PSV's* understanding of the program specification given by the user.

PSI allows programs to be specified by the use of traces and examples. The trace component of the inference expert [Phillips-77] handles simple loop and data structure inference such as that needed for the CLASS and TF dialogues. The examples component determines from an example input-output pair for a certain data object a suitable program transformation that could have carried the object from its initial to Its final state.

An initial version of a domain expert for information retrieval has been Implemented, using the program net as an interface with the rest of the system. The domain expert has been used in the generation of a variant of NEWS.

Preliminary designs are complete for an additional program specification technique, a formal system with the flavor of a very high level programming language. The language allows manipulation of abstract algebraic structures such as mappings and sets. The semantic support available through the domain expert will allow the use of domain specific jargon in this language. The language will allow the user to specify quickly and precisely program descriptions that have already been well thought out.

The program model builder [McCune-77] *uses* the *program* net produced by the other acquisition modules to construct a complete and consistent model of the program. There are about 350 rules in the model builder's knowledge base. Rules incorporate knowledge of mappings and primitive operations for accessing them, of procedures and procedure invocations, and of type coercion. The model builder also resolves type-token ambiguities and transforms expressions to canonical forms It has built a number of program models that are variations of CLASS as part of the PSI *system.* Separately the model builder has successfully constructed a model for RECIPE.

The rule expander for model building rules makes writing such rules easier. Rule preconditions are written in a concise declarative language. Then the rule expander translates the declarative form into the required fetch and test operations, taking Into account any ordering constraints that the preconditions may have and avoiding retesting preconditions unnecessarily.

A program has been written that prints concise, readable versions of program models. The Internal representation of the model is designed for programming efficiency and is hard for people to understand. Listings in the concise notation are thus valuable for debugging. The model is printed in a very high level language, using a syntax similar to PASCAL. Any or ail of the parts of a model may be printed, and cross-reference tables are available to index the concise listing and the original model.

The program model interpreter executes models interpretively as an alternative to coding them and running the target program. It correctly interprets all program models available. The interpreter parses arbitrary input data, including those which are of any type occurring in a tree of legal alternatives.

2 2 Synthesis Phase

The program model is refined into target language code during the synthesis phase. Dividing PSI into two separate phases allows program optimization to take different runtime environments into account. The program can be specified once and a program model built. Then different target language programs can be produced for different size estimates, probabilities, or cost functions. The programs will have the same input-output behavior, but the code will be optimized differently based on the data structure sizes or other such parameters.

For example, recall that CLASS reads a sample set of items, then repeatedly inputs a trial set and tests whether the sample set *is* a subset of the trial set. Since the universe of the sets is not known, a fast subset test using a bit map is not possible. So the subset test is implemented as an enumeration through the elements of the sample set, testing each element for membership in the trial set. When the trial set is small, a simple list (the same as the Input format) is a good choice of representation for the sets. When the trial set *is* large, however, representation as a hash table may prove more efficient because the membership test Is much faster. The efficiency expert checks whether such savings outweigh the cost of the representation conversion.

The knowledge base of the coder [Barstow-79] has about 450 rules. These rules have been used to code a variety of programs, including graph reachability and prime number finding. The sets and mappings used in these programs can be represented as lists, arrays, Boolean mappings, or property lists. Several versions of CLASS, RECIPE, NEWS, TF, and insertion and selection sorts have been coded. Rules about *reusing* the space in arrays have been written and used to synthesize (n-place selection and insertion sorts.

The efficiency expert [Kant-79) has been used with the coder to write RECIPE, NEWS, insertion and selection sorts, and several variants of CLASS. In all cases different implementations are selected when different data structure sizes (for example) are assumed. More than one representation for the same data structure can be used in a program. Efficiency rules suggest the circumstances under which various representations are plausible or implausible, which greatly reduces the search space from the original space of all legal programs. Space-time cost estimates are used to compare different implementations and to identify the decisions that may have the greatest Impact on the global program cost; the decision making resources are allocated accordingly.

3 Acknowledgment

This paper describes research being done at Systems Control, Inc. The research is supported in part by the Dtfense Advanced Research Projects Agency under DARPA Order 3681, Contract NOOOI4-79.C-OI27, which is monitored by the Office of Naval Research. The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of SCI, DARPA, ONR, or the US Government.

4. References

[Barstow-79] David R. Barstow, *Knowledge Based Program Construction,* Elsevier North-Holland, Inc. 1979.

[Biermann-76] Alan W. Biermann, "Approaches to Automatic Programming", in M Rubinoff and M. Yovits, editors, *Advances In Computers,* Volume 15, Academic Press, Inc., 1976, pages 1-63.

[Ginsparg-78] Jerrold M. Cinsparg, *Natural Language Processing in an Automatic Programming Domain,* PhD thesis, STAN-CS-78-67I, Computer Science Department, Stanford University, June 1978.

[Green-76] Cordell Creen, The Design of the PSI Program Synthesis System", *Proceedings Second International Conference on Software Engineering,* October 1976, pages 4-18

[Green-77] Cordell Creen, "A Summary of the PSI Program Synthesis System", *Proceedings of the Fifth International Joint Conference on Artificial Intelligence-1977,* August 1977, pages 380-381.

[Kant-79] Elaine Kant, *Efficiency Considerations in Program Synthesis A Knowledge Based Approach,* Ph.D. thesis, Computer Science Department, Stanford University. 1979 (In progress).

[McCune-77] Brian P. McCune, "The PSI Program Model Builder. Synthesis of Very High Level Programs", Proceedings of the Symposium on Artificial Intelligence and Programming Languages, *SIGART Newsletter,* Number 64 August 1977, pages 130-139.

[Phillips-77] Jorge V. Phillips, "Program Inference from Traces Using Multiple Knowledge Sources", *Proceedings of (he Fifth International Joint Conference on Artificial Intelligence-1977,* August 1977, page 812.

[Steinberg-79] Louis I. Steinberg, *A Dialogue Moderator for Program Specification Dialogues in the PSI System,* Ph.D. thesis. Computer Science Department, Stanford University, 1979 (in progress).

A TOO LEVEL MODULAR SYSTEM FOR NATURAL LANGUAGE UNDERSTANDING

Giovanni Guida and Marco Somalvico
Milan Polytechnic Artificial Intelligence Project
Istituto di Elettrotecnica ed Elettronica
Politecnico di Milano
Piazza Leonardo da Vinci n. 32
1-20133 Milano, Italy

This paper is intended to propose a new methodological approach to the conception of natural lan guaqe understanding systems. This contribution is supported by the design and implementation of DONAU. The system is based on a two level modular architecture. The horizontal level allows an independent and parallel development of the single segments of the system (syntactic analyser, semantic analyser, information extractor, legality controller). The vertical level ensures the possibility of changing the definition of the semantic domain on which particular versions of the system can be oriented and specialized in a simple, incremental, and user-oriented way. DONAU is written in LISP and is available from the authors.

1. INTRODUCTION

The conception of modular and flexible archite£ lures has been proposed recently in the develop merit of natural language understanding systems [2]. Basic exigencies of unification, flexibi- lity, and intercommunication have motivated the development of the DONAU (Domain Oriented NAtu- ral language Understanding) system, which has been successfully experimented on an UNIVAC 1110 at the Milan Polytechnic Artificial Intel- ligence Project [3] .

The original contribution of DONAU consists in the adoption of a two level modular architec- ture for a system able of understanding natural language.

In the development of the DONAU two level archi tecture, the morphologic and syntactic analysis has been designed and implemented on the base of the PIAF system which has been conceived, de_ vcloped, and successfully tested in the past at the IMAG Laboratory of the University of Gre_ noble (France) [4] .

Two versions of DONAU have been already comple- tely implemented and experimented: a first one, oriented on robotics, for the programming and control of the SUPERSIGMA robot developed and operating at the Milan Polytechnic Artificial Intelligence Project; a second one, oriented on data bases, for the inquiry of a relational data base about the oil resources in the world proposed by the International Institute for Ap- plied System Analysis, Laxenburg, Austria. A third version, centered on program synthesis,is presently being designed *[1]*.

THE TWO LEVEL MODULAR STRUCTURE

This section illustrates the modular structure of DONAU and the adopted design criteria. The main criterion which has guided the conception of DONAU has been that one of two level modu- larity.

The first level, called horizontal level,re- fleets the distinction between the different conceptual activities concerning the natural language understanding process. Eachone of the se activities (i.e., morphologic and syntactic analysis, semantic analysis, information ex- traction, and legality control) is associated with a bimodular segment which is precisely defined by its input-output characteristics. The second level of the modular architecture, called vertical level, is related with the di- stinction of the different nature of the two modules which compose the bimodular segment. The first module, called top module, embeds the domain-independent, general-purpose algo- rithm associated with the conceptual activity proper of the segment. The second module, cal- led bottom module, consists of a framework con- taining the domain-dependent, special-purpose information which directly relates with the particular semantic domain in which a version of the DONAU system is conceived to be operat- i ng.

An advantage of this two level architecture lies in an efficient balance between the chara£ teristic of generality, required by a natural language understanding system, and the exigency of efficiency, implied by an actually useful system.

Another feature is the simplicity in designing different versions of the same system, each-one devoted to a particular instance of a semantic domain- Each new version can be obtained, in fact, by simply changing, in each segment, only the bottom module, i.e., the domain oriented information. A further quality is the possi_bility of having incrementslity in the definition and set up of the system, i.e., the capability of modifying and increasing the content of the specification information on the base of the requirements and suggestions which the experimentation of the system might propose.

The fundamental assumption on which the DONAU project is based is that one of artificial understanding of natural language. We claim that the understanding of natural language, having as ultimate goal the communication between the man and an artificial system, does not require the capability of following all the peculiarities and the nuances of the possibly different ways of expressing an operative information in natural language. The goal of the understanding process is to obtain just that kernel of information which is necessary to activate in the desired and correct way the artificial system which is the target of the man-machine interaction and which can perform only a bounded, usually quite small, number of activities. This approach to natural language understanding can be useful, in our opinion, for avoiding some of the linguistic difficulties encountered in early research works. Moreover, this standpoint allows to largely extend the capabilities of the system in accepting a great variety of free natural language sentences, still avoiding the explosion of the complexity of the parsing algorithm consequent to an increased complexity of the input sentences which might actually be encountered in real applications.

3. UNDERSTANDING INPUT SENTENCES

The first step of the parsing consists of the morphological and syntactic analysis. The classical methods of finite state recognizers and dependence grammars [4] are adopted for this activity. The result of the syntactic analysis, constituted by a (small) set of dependence trees [3], is then supplied as input of the semantic analyser.

The role of the semantic analysis is to discriminate, among the different dependence trees, that one which is consistent with the semantic description of the natural language supplied to the system.The specification information of this segment is constituted by a complex framework called model-list network.This network is lists.The model-lists are lists of symbols representing morphological types and semantic types, which denote the categories of words and elementary constructs considered as relevant ones for defining the semantics of the natural language in a particular application domain. Each model-list is associated to a semantic type and constitute therefore a transformation rule. A dependence, tree is discriminated by the semantic analyser if to its nonterminal nodes model-lists can be associated in such a way to satisfy an appropriate set of matching conditions [3] The output of the semantic analyser is represented by a semantic tree embedding all the relevant information so far obtained in the parsing.

The third segment of DONAU is the information extractor. Its role is to individuate within the semantic tree the operative significant content which is sufficient and not redundant for expressing the meaning of the original natural language sentence. The elimination of the superfluous information is performed by a covering algorithm on the base of the content diagram [3] defined on the lexicon. The result of the whole parsing activity is an executable formal statement which can directly be accepted and executed by the interacting artificial system.

Finally, the legality control consists in the evaluation of the proposed operative content within a legality base which represents the non-linguistic context of the artificial system connected with DONAU.

4. EXPERIMENTAL RESULTS AND CONCLUSIONS

Two versions of DONAU have been developed.

RB-DONAU, centered on the semantic domain of robotics, is running, at present time, on an UNIVAC 1110 computer and is programmed in 1108-LISP. The software architecture of the system reflects quite closely the conceptual schema described in section 2. The program occupies about 25 Kwords memory and the specification information, relative to a lexicon of about 200 words, 5 kwords. The comprehension of simple sentences of 5 to 10 words requires about 1 second (with compiled LISP programs).

DB-DONAU, centered on the domain of data base inquiry [1], has been developed with the precise aim of verifying the actual possibility of changing the semantic domain along the lines suggested by the vertical modularity of the system. The experimental activity done has shown a good flexibility of the model proposed for achieving the modularity and incrementality goals initially stated for the project.

Several projects are presently developing around the DONAU system. In particular, the following goals are considered as relevant ones :

- to improve the diagnostic capabilities of DONAU and to develop an high-level testing and tracing subsystem;
- to define and implement an experience subsystem, which can guide the user's tuning and extension activities on the base of the past use of the system;
- to integrate the two steps of syntactic and semantic analysis as parallel coroutines for increasing the efficiency of the parser.

APPENDIX

In this appendix we present an example of natural language understanding on a simple sentence :

ELENCA I DEPOSIT! DEL KUWAIT AVENTI RISERVA DI PETROLIO MAGG10RE DI 1500 TON AL 1.9.1978
(list the deposits of Kuwait having an oil reserve greater than 1500 ton on 1.9.1978).
Two different dependence trees are obtained:

((ELENCA verb)*((DEPOSITI sub) (I art)*
((KUWAIT sub) (del prep)*) ((AVENTI verb)*
((RISERVA sub)*((PETROLIO sub) (DT prep)*)
(MAGGIORE adj) ((1500 num) (DI prep)"*
(TON sub)) ((AL prep)* (1.9.1978 date))))))

((ELENCA verb)*((DEPOSITI sub) (I art)*
((KUWAIT sub) (DEL prep)*))
((RISERVA sub) (AVENTI Verb)*((PETROLIO sub)
(Dt prep)*)(MAGGIORE adj)((1500 num)(DI prep)*
(TON sub)) ((AL prep)*(1.9. 1978 date))))

The notation (x y*z) is used,where x is the root of the tree, y are the left subtrees and z are the right subtrees.
Only the first tree is discriminated by the semantic analyzer; the corresponding semantic tree is :

((sentence: (ELENCA verb))*((object: (DEPOSITI sub)) (T art)*((spec: (KUWAIT sub)) (DEL prep)*)
((compl: (AVENTI verb))*((compl-spec: (RISERVA sub))*((spec: (PETROLIO sub)) (DI prep)*)
(MAGGIORE adj) ((spec: (1500 num)) (DI prep)*-
(TON sub)) ((date: (AL prep))*(1.9. 1978 date))))))

The resulting information tree is then:

((sentence: (ELENCA verb))*((object: (DEPOSITI sub))*(spec: (KUWAIT sub)) ((compl:)"*
((compl-spec: (RISERVA sub))*(MAGGI0RE adj)
((spec: (1500 'num) * (TON sub)) ((date: *
(1.9.197P Hate))))))

ACKNOWLEDGMENTS

We are grateful to the researchers of the IMAG Laboratory (University of Grenoble, France) which have cooperated to this project and to the students of the Politecnico di Milano for their contributions in the experimental work.

REFERENCES

[1] Bernorio, M., Bertoni, M., Dabbene, A., Somalvico, M. "A Domain Oriented Natural Language Understanding System for Man-machine Interaction with Dynamic DATA Bases". In Proc. IIASA Workshop Natural Languages for Interaction with Data Bases,Laxenburg, Austria, October,1978, pp.239-264.

[2] Hendrix, G.G. "A Natural Language Interface Facility and its Application to a Data Base" In Proc. IIASA Workshop Natural Language for Interaction with Data Bases, Laxenburg, Austria, October, 1978, pp.87-94.

[3] Guida, G., Somalvico, M. "DONAU: A General Purpose Domain Oriented Natural Language Understanding System". Memo MP-AIM-99, Politecnico di Milano, Milan, Italy, October, 1978.

[4] veillon, G. "Modeles et algorithmes pour la Traduction Automatique". These, University Scientifique et Medicale de Grenoble, Grenoble, France, 1970.

[5] Woods, W.A "Transition Network Grammars for Natural Language Analysis", Comm. ACM 13:10 (1970) 591-606.

A NATURAL PROGRAMMING CALCULUS

Ake Hansson Sten-Ake Tarnlund

Department of Computer Science
University of Stockholm and
The Royal Institute of Technology
106 91 Stockholm
Sweden

We shall lay down a programming calculus in which we develop a methodology for reasoning about data and programs. Our language, L, is ordinary first order predicate logic. A subset of this language, Lp, is our programming language that can be run efficiently in PROLOG. The calculus consists of a natural deduction system that is used for deducing programs and a system for efficient computation of programs. The axioms of the calculus characterize the data structures. Definitions are used for program specifications and mappings between data structures. The programs are deduced from the axioms and the definitions. Examples of deduced programs are a LISP-program and a program that Burstall and Darlington could not obtain in their transformation system.

1. INTRODUCTION

Our main purpose is to develop a methodology for reasoning about data and programs, and especially to enable us to arrive at a programming calculus. In this connection we shall take up a system of natural deduction invented by Jaskowski and Gentzen, and from this we shall deduce programs.

Natural deduction systems are natural in several ways. One important point is their characterization of informal reasoning. In particular, the structure of unformalized reasoning preserves its structure when it is formalized. In fact, the suggestion by Lukasiewics to formalize reasoning from assumptions seems to be a first step towards natural deduction systems.

We can think of a natural deduction system as a formalization of means~ends analysis in the GPS-system (see Newell & Simon [14]). Taking this view, a derivation of a program follows a general structure. An induction hypothesis is an initial state. A program specification together with axioms of data structures or data structure mappings are operators that transform an initial state to a final state that is the program. The general structure of a deduction is of great interest not only for our understanding of program deductions but also for developing methods for automated reasoning. Unfortunately, we have to leave out the deductions for reasons of space. A detailed description of the deductions is found in Hansson & Tarnlund [7].

We shall make use of Quine's *[16]* natural deduction system for classical logic. Natural deduction systems are not restricted to classical logic but apply also to many other systems e.g., second order logic, modal logic and set theory. A comprehensive description of natural deduction systems as well as of their history is given by Prawitz [15].

Our language is first order predicate logic (L) defined in its standard way with bound and free variables, primitive signs for quantification (V and 3), conjunction (&), disjunction (v), conditional (-->), negation (~) and bi-conditional (<--►). In section *2* this language is used for writing down axioms e.g., to characterize data structures, and in sections *3~5* for definitions of programs and. data structure mappings. We use the inference rules of the natural deduction system to derive programs from the axioms, definitions and mappings.

The language L should be distinguished from the restricted language of Horn clauses Lp, which is our programming language (see Colmerauer et al. [6], Hayes [8] and Kowalski ho]). Thus a deduction of a program starts from axioms and definitions in L and arrives at a Horn clause program in Lp. Horn clause programs do not restrict the set of computable functions (see Tarnlund [19]) and since the advent *of* PROLOG (see Roussel [18] and Warren [21]) they can also be run efficiently by a resolution-like theorem prover (see Robinson [17]).

The programming language evidently becomes an integrated part of a logic calculus, in the sence of Church *[4] ,* where the deductions are determined by inference rules and axioms. It should be noted that our system consists of a natural deduction system for reasoning and a resolution system for the efficient computation of programs.

*This work was supported by the Swedish Natural Science Research Council.

2. DATA STRUCTURES

We now turn to the notion of data structures, which are premises in our deductions. We shall have to write down these structures explicitly in order to make formal inferences from them. These formalizations are axioms in our calculus.

We shall take up four data structures: simple linear lists, difference lists, binary trees and ordered-binary trees. Our characterization of these data structures has already been introduced in Tarnlund [20].

A simple linear list is empty (0) or constructed (x.y) from an element x and a list y. We formalize this idea in our first axiom.

Axiom 1.

$$\forall w\{list(w) \leftrightarrow w=\emptyset \vee \exists x\exists y[w=x.y \;\&\; element(x) \;\&\; list(y)]\}$$

We are using a dot, . , for list constructions which like the extensions of the elements are supposed to be defined elsewhere.

Generally, there are useful lemmas that could be derived from the axioms which make the deductions of programs shorter. Following Clark & Tamlund [5] these lemmas consist of an induction step and a base. For reasons of space we have to leave them out.

Our next axiom characterizes a more complicated linear list called a difference list or a d-list for short. It is useful for direct access to the rear of a linear list and for concatenating two lists efficiently. We may think of a d-list as a pair <u,w> of linear lists u and w, where each element on w occurs in the same order at the rear of u e.g., u=a.b.c.x and w=c.x giving the pair <a.b.c.x,c.x>, which denotes the simple list a.b.0. Our second axiom formalizes the idea of a d-list.

Axiom 2.

$$\forall w\{d\text{-}list(w) \leftrightarrow \forall z\exists u\exists v[w=<u,z> \;\&\; (u=z \vee u=v.z \;\&\; element(v) \vee d\text{-}list(<u,v>) \;\&\; d\text{-}list(<v,z>))]\}$$

We assume that the pair constructor, < > , is already defined.

Binary trees are interesting and useful data structures not only for their own sake but also because any tree can be represented by a binary tree. An object is a binary tree if and only if it is empty or constructed t(x,y,z) where y is an element, moreover, the subtrees x and z are binary trees themselves. This idea is formalized in our third axiom.

Axiom 3.

$$\forall w\{binary\text{-}tree(w) \leftrightarrow w=\emptyset \vee \exists x\exists y\exists z[w=t(x,y,z) \;\&\; element(y) \;\&\; binary\text{-}tree(x) \;\&\; binary\text{-}tree(z)]\}$$

When unambiguous a binary tree is called a tree.

The concept of ordering is fundamental in programming. We shall need a "belongs-to" relationship in order to specify the idea of ordering.

An element u belongs to a list w if and only if w is identical with a simple list x.y and u is identical with x or u belongs to the list y. This is written more precisely in our fourth axiom.

Axiom 4.

$$\forall u\forall z\{u \in z \leftrightarrow list(z) \;\&\; \exists x\exists y[z=x.y \;\&\; (u=x \vee u \in y)]\}$$

This inductive definition is naturally viewed from the front of the list towards the rear. However, it is equally natural to view a list from the rear towards the front. But in order to achieve the latter idea we have to make use of a d-list since a linear list does not work in the latter direction. The following picture might be useful for understanding our next axiom.

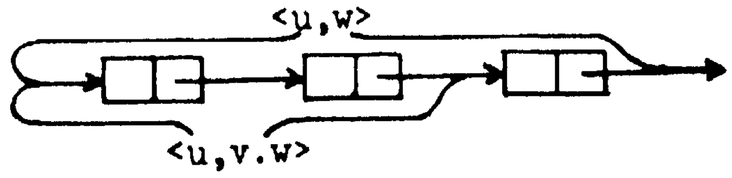


Fig.1 Two d-lists.

An element x belongs to a d-list <u,w> if and only if there exists a one-element shorter d-list <u,v.w> such that x is identical with v or belongs to a d-list <u,v.w>. We make this notion concise in an axiom.

Axiom 5.

$$\forall x\forall z\{x \in z \leftrightarrow \exists z'[d\text{-}list(z) \;\&\; d\text{-}list(z') \;\&\; \forall w\exists u\exists v[z=<u,w> \;\&\; z'=<u,v.w> \;\&\; (x=v \vee x \in z')]]\}$$

An element u belongs to a binary tree t(x,y,z) if and only if u is identical with y or u belongs to x or z. This idea is written precisely in an axiom.

Axiom 6.

$$\forall u\forall w\{u \in w \leftrightarrow binary\text{-}tree(w) \;\&\; \exists x\exists y\exists z[w=t(x,y,z) \;\&\; (u \in x \vee u=y \vee u \in z)]\}$$

We finish this section with an axiomatization of an ordered binary tree; ob-tree for short. An empty tree is an ob-tree and a tree t(x,y,z) is ordered if and only if each element on the left subtree x io less than the element y and each element on the right subtree z is greater than y, moreover, the trees x and z are ob-trees.

Axiom 7.

$$\forall w\{ob\text{-}tree(w) \leftrightarrow binary\text{-}tree(w) \;\&\; [w=\emptyset \vee \exists x\exists y\exists z[w=t(x,y,z) \;\&\; \forall u[(u \in x \rightarrow u<y) \;\&\; (u \in z \rightarrow y<u)] \;\&\; ob\text{-}tree(x) \;\&\; ob\text{-}tree(z)]]\}$$

We shall also need to distinguish between an empty list and a constructed list i.e., $\emptyset \neq x.y$, as

well as between an empty tree and a constructed tree i.e., $\emptyset \neq t(x,y,z)$. We can now, for example, derive that there is no element on an empty list or on an empty tree i.e., $\sim(u \in \emptyset)$. We shall also assume some standard axiomatization of arithmetic.

3. DEDUCTION FROM INDUCTIVE ASSUMPTIONS

The intention of our programming calculus is to arrive at systematic methods for deducing programs. Generally speaking, a deduction of a program follows from a program specification and from the axioms of data structures given in section 2 or mappings between data structures. These axioms, specifications and mappings are written in our first order language L. By contrast, a program is written in the language Lp, which usually is Horn clauses and a subset of L.

A deduction is built up on the inference rules in Quine's natural deduction system [16]. In addition we shall also make use of rules for identity.

It is a point of methodological interest that the programs are written inductively on a data structure. This fact is used for deducing programs from their specification by an inductive argument i.e., a program is deduced from an induction hypothesis. Moreover, this method of deducing programs gives a correctness proof of the programs for free. A simple example will shed light on these ideas.

Suppose we wish to deduce a program which checks whether or not an element x is less than all elements of a list l. We owe this "lessall"-program to Manna & Waldinger [12]. First, we write down a definition in the language L which is a precise specification of the program.

Definition 1.

$$\forall x \forall l\{\text{lessall}(x,l) \leftrightarrow \text{element}(x) \;\&\; \text{list}(l) \;\&\; \forall z[z \in l \rightarrow x<z]\}$$

This definition is explicit in contrast to an implicit one written inductively on a data structure. As we shall see explicit definitions contain more alternative programs than implicit definitions and so we consider them more abstract.

We observe that definition 1 contains a simple list as data structure. In fact, a data structure provides a general rule for generating the structure of a program deduction. First, we assume that there is a program on the data structure e.g., lessall(x,l), where x is an element and l a constructed list. From this assumption a consequent is deduced e.g., lessall(x,u.l), where u.l contains one element more than l. In this way, by the deduction theorem, we can deduce a part of a program, which is an induction step. Secondly, we can readily complete the program by an induction base e.g., lessall(x,$\emptyset$).

We can sum up our deductions in Hansson & Tärnlund [7] in the following theorem, which is a program that checks whether or not an element and a list satisfy the "lessall"-relationship.

Theorem 1.

lessall(x,$\emptyset$) ← element(x)
lessall(x,u.l) ← x<u & element(u) & lessall(x,l)

We follow a convenient convention for logic programs and omit universal quantifiers whose scope is the entire sentence. We can omit the "type"-checks if we assume that the program is used with respect to its intended interpretation. In this way we arrive at the following simple program which we can run efficiently in PROLOG.

lessall(x,$\emptyset$)
lessall(x,u.l) ← x<u & lessall(x,l)

3.1 Program correctness

It is to be noticed that the deduction of our program ensures that the program is correct. For example, a correctness statement of the "lessall"-program is:

$$\forall x \forall l\{\text{element}(x) \;\&\; \text{list}(l) \;\&\; \text{lessall}(x,l) \rightarrow \forall z\{z \in l \rightarrow x<z\}\}$$

A meta-argument tells us that this statement is already proved during the deduction of the program. A central idea about the deduction is that it is an induction argument on lists consisting of a base case and an induction step i.e.,

$$P(\emptyset) \;\&\; \forall x \forall y[P(y) \rightarrow P(x.y)] \rightarrow \forall z P(z)$$

where the antecedent consists of the base case and the induction step of the "lessall" program respectively. So, by induction we prove that the program is true of exactly all elements and lists in the "lessall"-relationship, i.e., we have a model of

lessall(x,l)

which by definition 1 is true for

$\forall z\{z \in l \rightarrow x<z\}$

It remains to be proved that the program terminates. We can, in fact, do this by induction on the data structure following methods developed in Clark & Tärnlund [5].

4. DEDUCTION FROM AN ABSTRACT SPECIFICATION

Deductions of programs from specifications usually contain several alternatives of data structures as well as a choice between operations on data structures. We shall say that a specification is less abstract when it provides fewer deductions. In this way an abstract specification gives more power and a natural idea would be to write down just abstract definitions. Up to a point such a

development has great advantages, as we shall see in section 4.1, where different programs are deduced from alternative assumptions about data structures and in section 4.3, where new programs are deduced from a choice between operations on data structures.

Unfortunately, all specifications are not abstract enough to provide alternative deductions of programs. This is certainly true of specifications that are very close to being programs themselves. In order to arrive at alternative programs in this situation we can of course try to write down a more powerful specification, but then we are faced with the problem of showing that the original specification satisfies the more abstract one. However, we can also deduce an alternative program from the original specification by introducing a mapping between data structures. We take up this idea in section 4.2.

4.1 Hypothesis About Alternative Data Structures

Evidently we can make use of our natural deduction system for hypothetical reasoning introduced in the preceding section and thus systematically deduce programs from alternative hypothesis.

Let us get a firmer idea by an example. Suppose that we have an abstract definition of programs for computing a list of factorials. Let us further assume that we first of all want to deduce a program with the factorials in descending order in the list and, secondly, to deduce a program with the factorials in ascending order. We owe the first of these programs to Burstall & Darlington [2]. The idea of putting each factorial at the front of the list is a stack operation in character and leads to a list of factorials in descending order. By contrast, the idea of putting each new factorial at the rear of the list is a queue operation in nature and gives the factorials in ascending order. These broad outlines of deducing programs from abstract specifications with alternative hypothesis about data structures are depicted in figure 2.

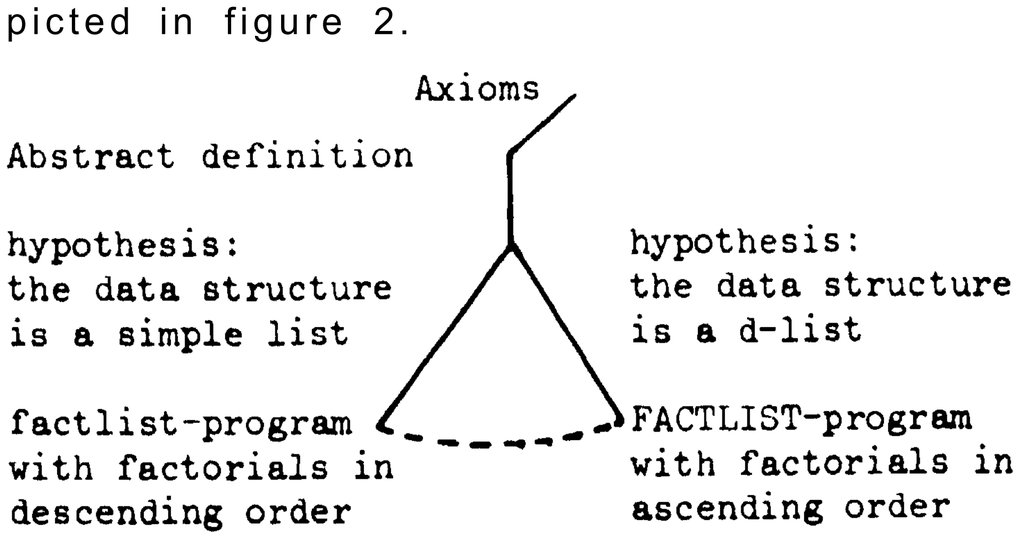


Fig.2 Deductions of alternative programs from the same specification

We shall use bounded quantifiers. We say that

$\forall_{1\leq y\leq n} F(y)$ is equivalent to $F(1) \& F(2) \ldots \& F(n)$ and

$\exists_{1\leq y\leq n} F(y)$ is equivalent to $F(1) \vee F(2) \ldots \vee F(n)$.

Let us now write down an abstract specification of factlist.

Definition 2.

$$\forall n \forall l \{ factlist(n)=l \leftrightarrow n=0 \,\&\, l=\emptyset \vee n\geq 1 \,\&\, \forall_{1\leq y\leq n} \exists n' [fact(y)=n' \rightarrow n' \in l] \,\&\, \forall n' \exists_{1\leq y\leq n} [n' \in l \rightarrow fact(y)=n'] \}$$

Obviously, the elements in the list could have any property. In fact, the only information we use about the factorials is that "fact" is a function from element to element. We have

fact: element → element

We can now deduce the two programs factlist and FACTLIST respectively. The deductions are identical for the first 7 steps but branch as illustrated in figure 2, when we make assumptions about data structures. The left branch follows from the assumption that the data structure is a simple list and the right branch from the assumption that it is a d-list (see Hansson & Tärnlund [7]). We obtain the following program.

Theorem 2.

$$factlist(0)=\emptyset$$
$$factlist(n+1)=u.l \leftarrow n\geq 0 \,\&\, list(l) \,\&\, fact(n+1)=u \,\&\, factlist(n)=l$$

In Hansson & Tärnlund [7] we also have a deduction of a FACTLIST-program.

Theorem 3.

$$FACTLIST(0)=\langle u,u\rangle$$
$$FACTLIST(n+1)=\langle u,w\rangle \leftarrow n\geq 0 \,\&\, d\text{-}list(\langle u,v.w\rangle) \,\&\, fact(n+1)=v \,\&\, FACTLIST(n)=\langle u,v.w\rangle$$

We recall from section 3 that the deductions of theorems 2 and 3 also give us correctness of these programs.

4.2 Data Structure Mappings

According to the dotted line in fig.2 a deduction requires a correspondence between the data structure simple list and d-list respectively. This one-to-one correspondence between the element in a stack and a queue is depicted in our next figure

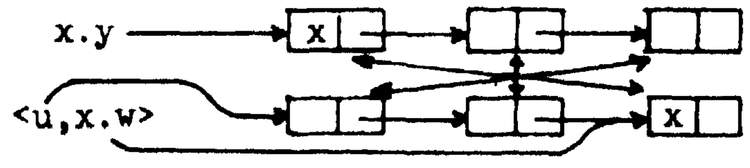


Fig.3 A one-to-one correspondence between a simple list x.y and a d-list <u,w>.

Let us give these correspondences names, domains and ranges in a usual way.

map1: simple list → d-list
map2: d-list → simple list.

We now have to define these correspondences more precisely in order to carry out a deduction from the factlist program with factorials in descending order to a program with factorials in ascending order.

Definition 3.

$$\forall l \forall q\{map1(l)=q \leftrightarrow l=\emptyset \;\&\; \forall u(q=<u,u>) \vee \exists x \exists y \forall w \exists u[list(l) \;\&\; l=x.y \;\&\; d\text{-}list(q) \;\&\; q=<u,w> \;\&\; map1(y)=<u,x.w>]\}$$

A definition of map2 is straightforward when we have map1.

Definition 4.

$$\forall q \forall l\{map2(q)=l \leftrightarrow map1(l)=q\}$$

We have two functions: factlist and FACTLIST.

factlist: element → simple list
FACTLIST: element → d-list.

This brings us to the following definitions of these functions.

Definition 5.

$$\forall n \forall q\{FACTLIST(n)=q \leftrightarrow \exists l[factlist(n)=l \;\&\; map1(l)=q]\}$$

Definition 6.

$$\forall n \forall l\{factlist(n)=l \leftrightarrow \exists q[FACTLIST(n)=q \;\&\; map2(q)=l]\}$$

These definitions and the following factlist program:

$$factlist(0)=\emptyset$$
$$factlist(n+1)=m.l \leftarrow n \geq 0 \;\&\; fact(n+1)=v \;\&\; list(l) \;\&\; factlist(n)=l$$

give a FACTLIST-program again. Hence, we have obtained theorem 3 twice, but this time by a mapping between data structures (see Hansson & Tärnlund [7]).

We can, of course, deduce the factlist program in theorem 2 in an analogous way from FACTLIST. Consequently, the programs factlist and FACTLIST are equivalent.

4.3 Hypotheses About Alternative Operations on a Data Structure

We shall now treat the idea of keeping the same data structure but making alternative hypotheses about operations on the data structure. In this way, a program is first deduced which inserts a new element into an empty subtree of an ordered binary tree such that the new tree is also ordered. Secondly, a program is deduced which inserts a new element into the root of an ordered binary tree and keeps the new tree ordered.

The important point about these programs is their operation on their data structure. The first program inserts the new element in one of its subtrees until an empty tree is reached. By contrast, the second program inserts its new element into the root of the binary tree and reorganizes the subtrees so that they stay ordered. We owe this program to Keith Clark.

Let us now write down an abstract specification of these programs. We are in fact, a bit strict and make the definition true only for ordered binary trees. We think of an insert relation between individuals u, v and w if and only if u is an ob-tree, v is an element and w is an ob-tree, moreover, each element on w is on u or is identical to v.

Definition 7.

$$\forall u \forall v \forall w\{insert(u,v,w) \leftrightarrow ob\text{-}tree(u) \;\&\; element(v) \;\&\; ob\text{-}tree(w) \;\&\; \forall z[z \in w \leftrightarrow z=v \vee z \in u]\}$$

In Hansson & Tärnlund [7] we first deduce one case of a leaf insert program and secondly one case of a root insert program. The first 14 steps in the deductions are identical until we make a choice between operations on the data structure. We have, thus, the following case of a leaf insert program.

Theorem 4.

$$insert(t(x,y,z),v,t(x',y,z')) \leftarrow v<y \;\&\; z=z' \;\&\; ob\text{-}tree(t(x,y,z)) \;\&\; insert(x,v,x')$$

Let us now write down a corresponding case for a root insert program.

Theorem 5.

$$insert(t(x,y,z),v,t(x',v,t(x'',y,z))) \leftarrow v<y \;\&\; x_2=t(x',v,x'') \;\&\; x=x_1 \;\&\; ob\text{-}tree(t(x,y,z)) \;\&\; insert(x_1,v,x_2)$$

We depict a relationship between input and output in our root insert program in the next figure. Notice that subtree x is partitioned in two sub-

trees so that x' contains all elements less than v and x" all elements greater than v.

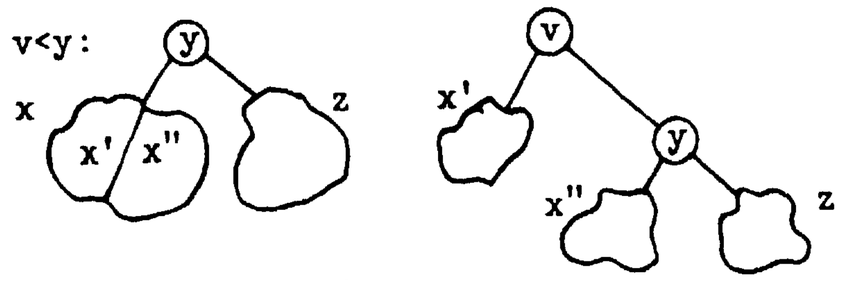


Fig.4 Root insert

In this way we can deduce the entire programs for leaf insert and root insert, respectively.

Theorem 6.

```
insert(t(x,y,z),v,t(x',y,z')) ← v<y & z=z' &
                ob-tree(t(x,y,z)) & insert(x,v,x')
insert(t(x,y,z),v,t(x',y,z')) ← y<v & x=x' &
                ob-tree(t(x,y,z)) & insert(z,v,z')
insert(t(x,y,z),v,t(x',y,z')) ← v=y & x=x' &
                z=z' & ob-tree(t(x,y,z))
insert(∅,v,t(∅,v,∅))
```

Theorem 7.

```
insert(t(x,y,z),v,t(x',v,t(x",y,z))) ← v<y &
      x2=t(x',v,x") & x=x1 & ob-tree(t(x,y,z)) &
                                  insert(x1,v,x2)
insert(t(x,y,z),v,t(t(x,y,z'),v,z")) ← y<v &
      z2=t(z',v,z") & z=z1 & ob-tree(t(x,y,z)) &
                                  insert(z1,v,z2)
insert(t(x,y,z),v,t(x',v,z')) ← v=y & x=x' &
                z=z' & ob-tree(t(x,y,z))
insert(∅,v,t(∅,v,∅))
```

We recall that we obtain a correctness proof of the programs.

5 DEDUCTION FROM A NON-ABSTRACT SPECIFICATION

The abstract character of the specification of factlist in the preceding section gives the advantage of two methods of deducing a program, namely, by choice of data structure or by data structure mapping. Unfortunately, there are quite natural specifications that are less abstract e.g. already bound to a data structure. This situation obviously eliminates the method that provides choices between data structures for deduction of alternative programs. In such a case we can either redefine our specification more abstractly or use the method of data structure mapping, shown in section 4.2. We shall take up the latter method again on a specification of a twist relation. We owe this program to Burstall & Darlington [2].

5.1 Data Structure Mappings for Deducing Alternative Programs

The twist program is a function.

twist: binary tree → binary tree

An example of this program is displayed below.

twist:

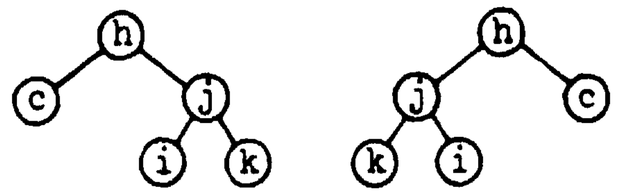


Fig.5 Twist of a binary tree

Let us now consider the problem of substituting this data structure for a "LISP"-tree structure and in this way arrive at a TWIST-program, which "TWISTs" a LISP-tree. Thus we want to deduce a TWIST-program, which is a function from LISP-trees to LISP-trees i.e.,

TWIST: LISP-tree → LISP-tree

An example of the TWIST-function is displayed in the next figure.

TWIST:

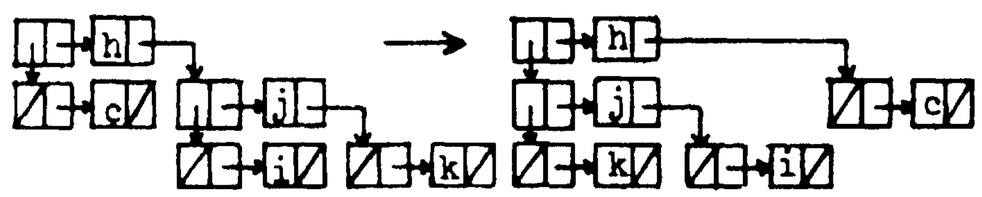


Fig.6 TWIST of a LISP-tree

This TWIST-program, although written in logic and potentially run in PROLOG, is a LISP-program in character, and thus it is natural to ask for a LISP-program. At this point three sub-problems demand solution, a definition of a mapping between a binary tree and a LISP-tree, a definition of TWIST in twist and an axiomatization of relevant LISP-functions. Let us sum up this situation in a figure.

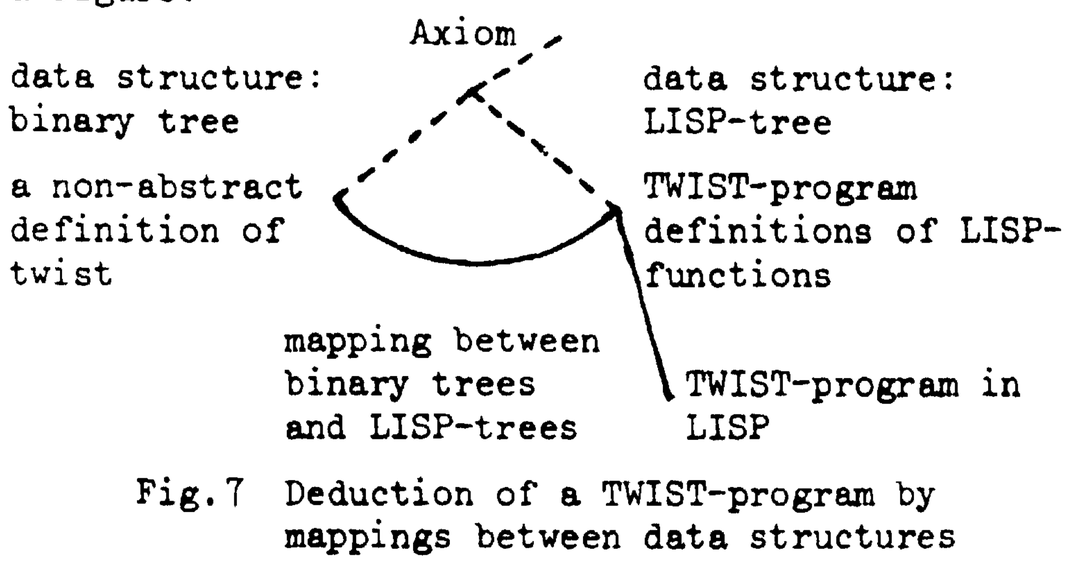


Fig.7 Deduction of a TWIST-program by mappings between data structures

Informally, we need a one-to-one correspondence between the data structures binary tree and LISP-tree.

map1: LISP-tree → binary tree
map2: binary tree → LISP-tree

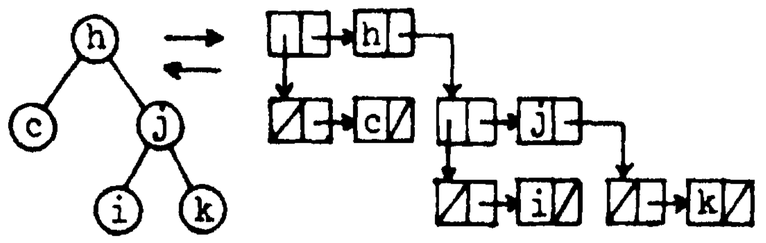


Fig.8 Functions between data structures

Before we can write down this correspondence more precisely we shall have to axiomatize a data structure LISP-tree built up on a LISP-list cons(x,y), and so we have to introduce the latter data structure.

Axiom 8.

$$\forall x\forall y\exists z\{cons(x,y)=z \leftrightarrow (element(x) \vee \exists u\exists w\ cons(u,w)=x) \ \&\ (y=nil \vee \exists u\exists w\ cons(u,w)=y)\}$$

A LISP-tree is a subset of a LISP-list. In fact, it is either nil or constructed, cons(x,cons(u,y)), where x is a LISP-tree, u an element and y a LISP-tree. We have:

Axiom 9.

$$\forall l\{LISP\text{-}tree(l) \leftrightarrow l=nil \vee \exists x\exists y\exists u[l=cons(x,cons(u,y)) \ \&\ element(u) \ \&\ LISP\text{-}tree(x) \ \&\ LISP\text{-}tree(y)]\}$$

As usual we distinguish between an empty list and a constructed list.

Axiom 10.

$$\forall x\forall y\sim\{cons(x,y)=nil\}$$

We can now write down the functions map1 and map2.

Definition 6.

$$\forall l\forall t\{map1(l)=t \leftrightarrow l=nil \ \&\ t=\emptyset \vee \exists u\exists l_1\exists l_2\exists t_1\exists t_2[l=cons(l_1,cons(u,l_2)) \ \&\ LISP\text{-}tree(l) \ \&\ binary\text{-}tree(t) \ \&\ t=t(t_1,u,t_2) \ \&\ map1(l_1)=t_1 \ \&\ map1(l_2)=t_2]\}$$

We get map2 immediately from map1.

Definition 7.

$$\forall l\forall t\{map2(t)=l \leftrightarrow map1(l)=t\}$$

Let us now specify twist on binary trees, in fact, the definition is quite natural although it is not abstract enough to provide a deduction of TWIST, without the mappings map1 and map2.

Definition 8.

$$\forall t_1\forall t_2\{twist(t_1)=t_2 \leftrightarrow t_1=\emptyset \ \&\ t_2=\emptyset \vee \exists u\exists t_{11}\exists t_{12}\exists t_{21}\exists t_{22}[t_1=t(t_{11},u,t_{12}) \ \&\ t_2=t(t_{22},u,t_{21}) \ \&\ binary\text{-}tree(t_1) \ \&\ binary\text{-}tree(t_2) \ \&\ twist(t_{11})=t_{21} \ \&\ twist(t_{12})=t_{22}]\}$$

We can now define TWIST by three mappings. First, from a LISP-tree l_1 to a binary tree t_1.

map1: $l_1 \rightarrow t_1$

Secondly, from a binary tree t_1 to a (twisted) binary tree t_2.

twist: $t_1 \rightarrow t_2$

Finally, from a binary tree t_2 to a LISP-tree l_2.

map2: $t_2 \rightarrow l_2$

We are now ready to define TWIST precisely.

Definition 9.

$$\forall l_1\forall l_2\{TWIST(l_1)=l_2 \leftrightarrow \exists t_1\exists t_2[map1(l_1)=t_1 \ \&\ twist(t_1)=t_2 \ \&\ map2(t_2)=l_2]\}$$

We still have to deduce a TWIST-program which is independent of twist and the functions map1 and map2. We can deduce the following program that "TWISTs" a LISP-tree, without any mapping function (see Hansson & Tärnlund [7]).

Theorem 8.

$$TWIST(nil)=nil$$
$$TWIST(cons(x,cons(y,z)))=cons(z',cons(y,x')) \leftarrow TWIST(x)=x' \ \&\ TWIST(z)=z'$$

Moreover, our inductive proof shows that the TWIST-program is total, i.e.,

Lemma 1.

$$\forall w\exists u[LISP\text{-}tree(w) \rightarrow TWIST(w)=u]$$

Before proceeding to a derivation of a LISP-program let us pause a moment for a comment on the deductions. Although they are made by hand several of them have also been deduced in Weyrauch's FOL system [22]. For example see appendix 1 in Hansson & Tärnlund [7] for a deduction in FOL of the TWIST-program in theorem 8. It is of course a natural development to get these deductions automatically. However, FOL is useful for systematic developments of programs, and in addition it checks the inference and ensures that we arrive at correct programs.

5.2 Deduction of a LISP-program

It is a natural development within our calculus to go a little further and deduce a LISP-program that twists a tree. In order to carry out this program we need an axiomatization of the LISP functions car and cdr.

Axiom 11.

$$\forall x\forall y\{car(x)=y \leftrightarrow \exists w\ cons(y,w)=x\}$$

Axiom 12.

$$\forall x\forall y\{cdr(x)=y \leftrightarrow \exists w\ cons(w,y)=x\}$$

We shall make use of a LISP-syntax containing "if ... then ... else ..." as proposed by McCarthy [13]. So we have to define this form.

Definition 10.

$$\forall x\forall y\{B(\text{if } A \text{ then } x \text{ else } y) \leftrightarrow (A \rightarrow B(x)) \ \&\ (\sim A \rightarrow B(y))\}$$

In this way we do not extend our natural deduction system which we might have done and thus introduce an inference rule for "if ... then ... else ..." introduction and elimination respectively (see Manna [11]). We get the following LISP-program for TWIST (see Hansson & Tärnlund [7]).

Theorem 9.

```
TWIST(w)= if w=nil then nil else
        cons(TWIST(cdr(cdr(w))),cons(car(cdr(w)),
                                     TWIST(car(w))))
```

It is worthwhile noting that our method laid down in this paper also applies to arbitrary programming languages provided their programming constructs are axiomatized like axioms 11 and 12. This axiomatic method seems to be a new approach to program verification.

Unfortunately, no other programming language than pure LISP, except logic itself, of course, has been given a treatment that is rigorous and precise enough for a programming calculus (see Boyer & Moore [1], Cartwright [3] and McCarthy [13]). In particular, the specification of PASCAL (see Hoare & Wirth [9]) does not suffice. It is obviously undesirable to restrict the generality of programming languages so they do not fit into a programming calculus. Therefore, developments of new programming languages should demand solution of two problems; design of the language and axiomatization of the language constructs, for example, as outlined above.

6. CONCLUSION

A methodology for reasoning about programs and for systematic development of programs is of great interest. In this connection we have employed a deductive methodology and obtained a calculus for programming without a detour for a formalization of the programming language. Thus, it is an important point that our language for data structures and program specifications is first order predicate logic, a subset of which is our programming language. This is of great advantage when we deduce programs. Unfortunately, we have not, for reasons of space, displayed any deductions. In fact, they are all examples of hypothetical reasoning formalized in a natural deduction system. By contrast, a program is executed by a PROLOG-like system. Hence, the calculus consists of two distinct logical systems. We have focused on a methodology for systematic development of programs, but it should be noted that the deductive methodology ensures that a derived program is correct. Consequently, we also obtain a methodology for proving that programs are correct.

REFERENCES

[1] Boyer R. & Moore J.S., Proving Theorems about LISP Functions, JACM vol 22 no 1, 1975

[2] Burstall R.M & Darlington J., A Transformation System for Developing Recursive Programs, JACM vol 24 no 1, 1977

[3] Cartwright R.S., A Practical Formal Semantic Definition for Typed LISP, Computer Science Dept. Stanford University, 1977

[4] Church A., Introduction to Mathematical Logic, Princeton, 1956

[5] Clark K. & Tärnlund S-Å., A First Order Theory of Data and Programs, IFIP-77, Toronto

[6] Colmerauer A. et al., Un Système de Communication Homme-Machine en Français, Groupe d'intelligence Artificielle U.E.R de Luminy Universite d'Aix-Marseille, Luminy, 1972

[7] Hansson Å. & Tärnlund S-Å., A Natural Programming calculus, Dept. of Computer Science University of Stockholm and The Royal Institute of Technology, 1979

[8] Hayes P., Computation and Deduction, Proc. MFCS Conference Czechoslovakian Academy of Sciences, 1973

[9] Hoare C.A.R. & Wirth N., An Axiomatic Definition of the Programming Language PASCAL, Acta Informatica 2, 1973

[10] Kowalski R., Predicate Logic as Programming Language, Proc. IFIP-74, North-Holland, 1974

[11] Manna Z., Mathematical Theory of Computation, McGraw-Hill Inc., 1974

[12] Manna Z. & Waldinger R., Synthesis: Dreams=Programs, Memo AIM-302, AI-lab., Computer Science Dept. Stanford University, 1977

[13] McCarthy J., Representation of Recursive Programs in First Order Logic, Draft, AI-lab., Computer Science Dept. Stanford University, 1977

[14] Newell A. & Simon H.A., Human Problem Solving, Englewood Cliffs, N.J. Prentice-Hall, 1972

[15] Prawitz D., Natural Deduction, A Proof-Theoretical Study, Almqvist & Wiksell, Stockholm, 1965

[16] Quine W.V., Methods of Logic, Second Edition Routledge & Kegan Paul, London, 1959

[17] Robinson J.A., A Machine-Oriented Logic Based on the Resolution Principle, JACM vol 12 no 1, 1965

[18] Roussel P., PROLOG: Manuel de Référence et d'utilisation, Groupe d'intelligence Artificielle, U.E.R de Luminy, Marseille, 1975

[19] Tärnlund S-Å., Horn Clause Computability, BIT 17, 2, 1977

[20] Tärnlund S-Å., An Axiomatic Data Base Theory, in Logic and Data Bases (eds.) Gallaire H. and Minker J., Plenum Press N.Y. 1979

[21] Warren D., Implementing PROLOG - Compiling Predicate Logic Programs, Dept of Artificial Intelligence, Edinburgh University, No 39, 1977

[22] Weyhrauch R.W. & Filman R.E., An FOL Primer, Stanford AI-lab, Memo AIM-288, Computer Science Dept. Stanford University, 1976

INCREASING TREE SEARCH EFFICIENCY FOR CONSTRAINT SATISFACTION PROBLEMS

Robert M. Haralick*
Electrical Engineering Dept.
Computer Science Dept.
Virginia Polytechnic Institute
and State University
Blacksburg, Virginia 24061

Gordon L. Elliott
Electrical Engineering Dept.
Virginia Polytechnic Institute
and State University
Blacksburg, Virginia 24061

in this paper we explore the number of consistency checks made by a tree search in order to solve binary constraint satisfaction problems. We show analytically and experimentally that the two principles of first trying the places most likely to fail and remembering what has been done to avoid repeating the same mistake twice improve the standard backtracking search. We experimentally show that a lookahead procedure called forward checking (to remember the future) which employs the most likely to fail principle performs better than standard backtracking, Ullman's, Waltz's, Mackworth's, and Haralick's discrete relaxation in all cases tested, and better than Gaschnigs backmarking in the larger problems.

1. INTRODUCTION

Associated with search procedures are heuristics. In this paper we provide a theory which explains why two heuristics used in constraint satisfaction searches work. The heuristics we discuss can be given a variety of one line descriptions such as:

> Lookahead and remember your future in order to succeed in the present.
> To succeed, try first where you are most likely to fail.
> Remember what you have done to avoid repeating the same mistake.
> Lookahead to the future in order not to worry about the past.

We will attempt to show that for a suitably defined random constraint satisfaction problem, the average number of consistency checks performed by a search procedure which employs these principles will be smaller than that required by the standard backtracking tree search.

To begin our discussion, we need a precise description of the constraint satisfaction problem we are attempting to solve by a search procedure. We assume that there are M units (some authors call these variables instead of units). Each unit has a set of L possible values or labels for each pair of units. The constraint satisfaction problem we consider is to determine all possible assignments f of labels to units such that for every pair of units, the corresponding label assignments satisfy the constraints. More formally, if U is the set of units and L is the set of labels, then the binary constraint R can be represented as a binary relation on $U \times L$: $R \subseteq (U \times L) \times (U \times L)$. If a pair of unit labels $(u_1,\ell_1,u_2,\ell_2) \in R$, then labels ℓ_1 and ℓ_2 are said to be consistent or compatible for units u_1 and u_2. A labeling f of all the units satisfies the constraints if for every pair u_1,u_2 of units $(u_1 f(u_1), u_2, f(u_2)) \in R$. Haralick et. al. (1978) call such a labeling a consistent labeling.

The problem of determining consistent labelings is a general form of many problems related to artificial intelligence. For example, scene labeling and matching (Barrow and Tenenbaum, 1976, and Rosenfeld et. al., 1976), line interpretation (Waltz, 1972), edge labeling (Haralick, 1978), graph homomorphisms and isomorphisms (Ullman, 1969), graph coloring (Harary, 1969) boolean satisfiability (Haralick et. al., 1978), and proposition theorem proving (Kowalski, 1975) are all special cases of the general consistent labeling problem. Ullman (1966), Waltz (1972), Rosenfeld et. al. (1978 and 1979), and Gaschnig (1977 and 1978) attempt

to find efficient methods to solve the consistent labeling problem. Knuth (1975) also analyzes the backtracking tree search, which is the basis of most methods used to solve the consistent labeling problem.

For the purpose of illustrating the search required to solve this problem, we choose the N-Queens problem. Here, the unit set corresponds to the row coordinates on a checkerboard and we denote them by positive integers. The label set corresponds to the column coordinates on a checkerboard and we denote them by alphabetic characters. Hence, the unit-label pair (1,A,2,D) satisfies the constraint R, [(1,A,2,D)rR], since a queen on row 1 column A cannot take a queen on row 2 column D. But, the unit label pair (1,A,3,C) does not satisfy the constraint R because queens can take each other-diagonally.

Using the number letter convention for unit-label pairs, Figure 2 illustrates a portion of a backtracking tree trace for the 6 Queens problem. Notice how the unit 5 labels A, C, E, and F occur twice in the trace, each time being tested and failing for the same reason: incompatibility with units 1 or 2. These redundant tests can be eliminated if the fact they failed can be remembered or if units 1 or 2 could lookahead and prevent 5 from taking the labels A, C, F, or F. The remembering clone by Gaschnig's backmarking (1977) and the forward checking approach described in this paper help eliminate these problems. Notice that once unit 3 takes label E (Figure la) the only labels left for units 4 and 6 are incompatible. The forward checking algorithm will not discover this future incompatibility. However, the first time label B is associated with unit 4, there is absolutely no label possible for unit 6. Hence, the search through the labels for 5 and 6 are entirely superflous and forward checking will discover this (Figure lb). The lookahead procedures (discrete relaxation) of Ullman (1966), Waltz (1972), Rosenfeld (1976), Mackworth (1977), and Montanari (1974) help alleviate the problem Illustrated in Figure la as well as in Figure lb.

Section 2 gives a brief description of the full and partial looking ahead, forward checking, backchecking, and backmarking procedures and concludes with a comparison of these problems created randomly. These results show that standard backtracking is least efficient in most cases, and backmarking and forward checking are the most efficient for the cases tried.

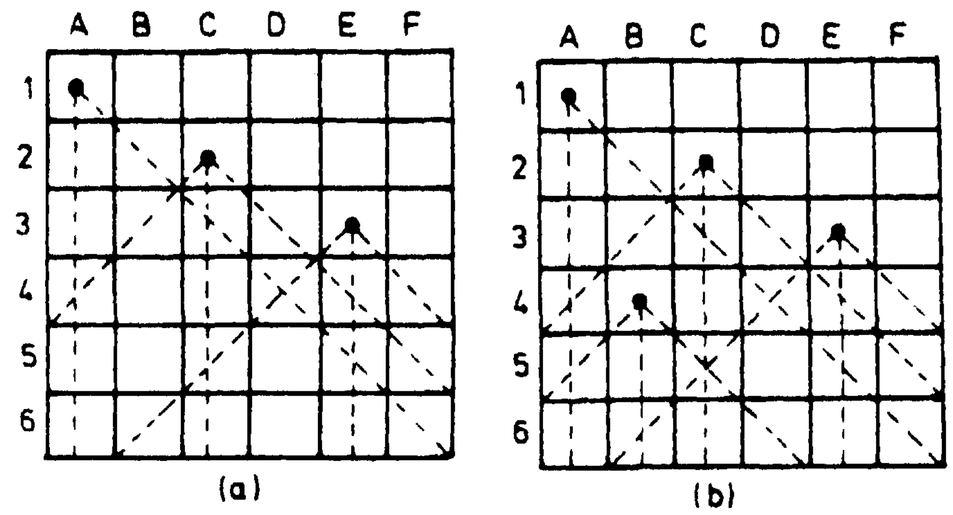


Figure la illustrates how the labeling A,C,E for units 1,2,3 implies that the onlv labels for units 4 and 6 are incompatible in the 6 Queens problem.

Figure lb illustrates how the labeling A,C,E,B for units 1,2,3,4 implies that there is no label for unit 6 in the 6 Queens problem.

```
1  A
   2  A,B
   2  C
      3  A,B,C,D
      3  E
         4  A
         4  B
            5  A,B,C
            5  D
               6  A,B,C,D,E,F
            5  E,F
         4  C,D,E,F
      3  F
         4  A
         4  B
            5  A,B,C,D,E,F
         4  C,D,E,F
```

Figure 2 illustrates a segment of a tree trace that the standard backtracking algorithm produces for a 6 Queens problem. No solutions are found in this segment. The entry 2 A,B, for example, indicates that labels A and B were unsuccessful at level 2, but 2 C succeeds when checked with past units, and the tree search continues with the next level.

In section 3, we give a statistical analysis of constraint satisfaction searches and demonstrate the statistical reason why forward checking requires fewer expected consistency

checks than standard backtracking. In section 4 we explore other applications of the fail first or prune early tree search strategies and show that such particular strategies as choosing the next unit to be that unit having fewest labels left and testing first against units whose labels are least likely to succeed reduce the expected number of consistency tests required to do the tree search. Finally, by changing the unit search order dynamically in every tree branch so that the next unit is always the one with fewest labels left, we show experimentally that not only does performance improve for each procedure, but that forward checking now does better than backmarking in the larger problems tested.

2. SOME PROCEDURES FOR TREE SEARCH REDUCING

In this section we give brief descriptions of five procedures which can be used within the standard backtracking framework to reduce tree search operations. They are called full and partial looking ahead, forward checking, back-checking, and backmarking. Each of these procedures invests resources in additional consistency tests at each point in the tree search in order to save (hopefully) more consistency tests at some point later in the tree search.

For ease in explaining these procedures, we call those units already having labels assigned to them the past units. We call the unit currently being assigned a label the current unit and we call units not yet assigned labels the future units. We assume the existence of a unit label table which at each level in the tree search indicates which labels are still possible for which units. Past units will of course have only one label associated with each of them. Future units will have more than one. The tree search reducing procedures invest early to gain later. Hence, the result of applying any of them in the tree search will be to decrease the number of possible labels for any future unit or reduce the number of tests against past units.

The section concludes with a comparison among the tree search reducing procedures which indicates that backtracking is least efficient in most cases, and that backmarking and forward checking are the most efficient for the cases tested. The lookahead procedures can be ordered in increasing order of efficiency as full looking ahead, partial looking ahead, and forward checking.

2.1 Looking Ahead

Waltz filtering (Waltz, 1972), a procedure by Ullman (1966), discrete relaxation (Rosenfeld, Hummel, Zucker, 1976), and the Y operator of Haralick et. al. (1978) are all examples of algorithms that look ahead to make sure that (1) each future unit has at least one label which is compatible with the labels currently held by the past and present units and (2) each future unit has at least one label which is compatible with one of the possible labels for each other future unit. Looking ahead prevent the tree search from repeatedly going forward and then backtracking between units u and v, v < u, only to ultimately discover that the labels held by units 1 through v cause incompatibility of all labels between some unit w, w > u, and some past, current, or future unit.

Because looking ahead in this manner cannot remember and save most of the results of tests performed in the lookahead *of* future units with future units for use in future lookaheads, the full savings of looking ahead are not realized for many problems. A partial look ahead that does not do all the checks of full look ahead will perform better than backtracking and one that checks only future with past and present units (neglects future with futures) will do much better because all tests it performs can be usefully remembered.

The procedure LA TREE SEARCH and its associated subroutines CHECK-FORWARD and LOOK FUTURE (Figure 3 a,b, and c) is a formal description of the full looking ahead algorithm, which can easily be translated into any structured recursive language. U is an integer representing the unit, and will increment at each level of the tree search. It takes on the value 1 at the initial call. F is a one dimension array indexed by unit, where entry f(u) for unit u is the label assigned to u. T and NEW T are tables, which can be thought of as an array of lists, T(u) is a list of labels which have not yet been determined to be not possible for unit U. (We implemented T as a 2 dimension array, with the number of entries in each list (or row) stored in the first position of the row. This implementation uses approximately (NUMBER_OF_UNITS)^2 x (NUMBER OF_LABELS) words of memory for table storage). The tree search is initially called with T containing all labels for each unit. All other variables can be integers. EMPTY_TABLE and NUMBER_OF_UNITS have obvious meanings.

The function RELATI0N(U1,L1,U2,L2) returns *TRUE*

if (U_1 ,L_1 ,U_2 ,L_2) *t* R, otherwise it returns *FALSE.* CHECK_FORWARD checks that each future unit label pair is consistent with the present label F(u) for unit u as it copies the table T into the next level table NEWJT, LOOK_FUTURE then checks that each future unit label pair in NEWJT is consistent with at least one label for every other unit, and deletes those that are not.

In this implementation CHECK_FORWARD and LOOK FUTURE return a flag, EMPTY ROW FLAG, if a unit is found with no possible consistent labels- Thus the next level of the tree search will not be called, otherwise each entry in NEW T is consistent with U,F,(u), and therefore, all the past unit-label pairs.

```
RECURSIVE PROCEDURE L_A_TREE_SEARCH(U,F,T);
FOR F(U) = each element of T(U) BEGIN
   IF U < NUMBER_OF_UNITS THEN BEGIN
      NEW_T = CHECK_FORWARD(U,F(U),T);
      CALL LOOK_FUTURE(U,NEW_T);
      IF NEW_T is not EMPTY_ROW_FLAG
         CALL L_A_TREE_SEARCH(U+1,F,NEW_T)
         END;
   ELSE
      Output the labeling F;
   END;
END L_A_TREE_SEARCH;
```

Figure (3a)

```
PROCEDURE CHECK_FORWARD(U,L,T);
NEW_T = empty table;
FOR U2 = U+1 TO NUMBER_OF_UNITS BEGIN
   FOR L2 = each element of T(U2)
      IF RELATION(U,L,U2,L2) THEN
         ENTER L2 in the list NEW_T(U2);
   IF NEW_T(U2) is empty THEN
      RETURN (EMPTY_ROW_FLAG); /* NO consistent
       Labels */
   END;
RETURN (NEW_T);
END CHECK_FORWARD;
```

Figure (3b)

```
PROCEDURE LOOK_FUTURE(U,NEW_T);
IF U+1 ≥ NUMBER_OF_UNITS THEN RETURN;
FOR U1 = U+1 TO NUMBER_OF_UNITS BEGIN
   L1_CONSISTENT = FALSE;
   FOR L1 = each element of NEW_T(U1)
      FOR U2 = U+1 TO NUMBER_OF_UNITS except
       skipping U1 BEGIN
         L2_CONSISTENT = FALSE;
         FOR L2 = each element of NEW_T(U2);
            IF RELATION(U1,L1,U2,L2) THEN BEGIN
               L2_CONSISTENT = TRUE;
               BREAK for L2 loop;
               END;
         IF NOT L2_CONSISTENT THEN
               BREAK for U2 loop;
         END for U2 loop;
      IF L2_CONSISTENT THEN
         L1_CONSISTENT = TRUE;
      ELSE
         Delete L1 from list NEW_T(U1);
         END for L1 loop;
      IF NOT L1_CONSISTENT THEN BEGIN
         NEW_T = EMPTY_ROW_FLAG;
         RETURN;
         END;
   END for L1 loop;
RETURN;
END LOOK_FUTURE;
```

Figure (3c)

2.2 Partial Looking Ahead

Partial looking ahead is a variation of looking ahead which does approximately half of the consistency that full looking ahead does while checking future with future units. Each future unit label pair is checked only with units in its own future, rather than all other future units. Thus partial looking ahead is less powerful than full looking ahead in the sense that it will not delete as many unit label pairs from the lists of potential future labels. We will, however, see that partial looking ahead does fewer total consistency checks than full looking ahead and standard backtracking in all cases tested.

The checks of future with future units do not discover inconsistencies often enough to justify the large number of tests required, and these results cannot be usefully remembered. Since partial looking ahead does fewer of these less useful tests, it is more efficient. A look ahead that checks only future with current or past units can have better performance since these more powerful tests can also be usefully remembered.

The formal algorithm for partial looking ahead requires only a minor modification of the L00K_ FUTURE procedure (Figure 3b). To change LOOK FUTURE into PARTIAL_LOOK_FUTURE, the second and fifth lines must be changed. The *"FOR* UI..." loop becomes *"FOR* UI = U+I *TO* NUMBER OF UNITS "1 *BEGIN"* and *"FOR* U2..." becomes *"FOR* U2 = UI+1 *TO* NUMBER_OF_UNITS

2.3 _Forward Checking

Forward checking is a partial lookahead of future units with past and present units, in which all consistency checks can be remembered for a while. This method is similar to looking ahead, except that future units are not checked with future units, and the checks of future units with past units are remembered from checks done at past levels in the tree search. Forward checking begins with a state of affairs in which there is no future unit having any of its labels inconsistent with any past unit-label pairs. This is certainly true at the base of the tree search, since there are no past units with which to be inconsistent. Because of this state of affairs, to get the next label for the current unit, forward checking just selects the next label from the unit label table for the current unit. That label is guaranteed to be consistent with all past unit label-pairs. Forward checking tries to make a failure occur as soon as possible in the tree search by determining if there is any future unit having no label which Is consistent with the current unit-label pair. If each future unit has consistent labels, it remembers by copying all consistent future unit-label pairs to the next level's unit label table. If every future unit has some label in the unit label table which is consistent with the current unit-label pair, then the tree search can move forward to the next unit with a state of affairs similar to how it started. If there is some future unit having no label in the unit label table which is consistent with the current unit-label pair, then tree search remains at the current level with the current unit and continues by selecting the next label from the table. If there is no label then it backtracks to the previous unit label table.

The formal algorithm for forward checking is the Procedure LA TREE_SEARCH (Figure 3a) with line 5, the call to LOOKFUTURE, removed. Forward checking is just looking ahead, omitting the future with future checks.

2.4 Backchecking

Backchecking is similar to forward checking in the way it remembers unit label pairs which are known to be inconsistent with the current or any previous unit label. However, it keeps track of them by testing the current unit label only with past unit label pairs and not future ones. So if, for instance, labels A, B, and C for unit 5 were tested and found incompatible with label B for unit 2, then the next time unit 5 must choose a label, it should never have A, B, or C as label possibilities as long as unit 2 still has the label B.

Each test that backchecking performs while looking back from the current unit u to some past unit v, forward checking will have performed at the time unit v was the current unit. Of course, at that time, forward checking will also have checked all future units beyond unit u. Hence, backchecking performs fewer consistency tests, an advantage. But backchecking pays the price of having more backtracking and at least as large a tree as forward checking. Backchecking by itself is not as good as forward checking.

2.5 Backmarking

Backmarking (defined in Gaschnig, 1977, and also discussed in Gaschnig, 1978) is backchecking with an added feature. Backchecking eliminates performing some consistency checks that were previously done, had not succeeded, and if done again would again not succeed. Backmarking also eliminates performing some consistency checks that were previously done, had succeeded, and if done again would again succeed. To understand how backmarking works, recall that the tree search by its very nature goes forward, then backtracks, and goes forward again. We focus our attention on the current unit u. We let v be the lowest ordered unit to which we have backtracked (has changed its label) since the last visit to the current unit u. Backmarking remembers v. If v = u, then backmarking proceeds as backchecking. If v < u, then since all the labels for unit u had been tested in the last visit to unit u, any label now needing testing, needs only to be tested against the labels for units v to u-1, which are the ones whose labels have changed since the last visit to unit u. That is, the tests done previously against the labels for units 1 through v-1 were successful and if done again would again be successful because labels for units 1 through v-1 have not changed and the only labels permitted for the current unit u are those which have passed the earlier tests.

2.6 Experimental Results

In this section we compare the six procedures, partial and full looking ahead, backtracking, backchecking, forward checking, and backmarking, on the N-Queens problem for $4 \le N \le 10$, and on a random constraint problem, finding all solutions. We assume that the unit order is fixed in its natural order from 1 to N and that all consistency tests of the current unit with past units or future units begin with the lowest ordered unit. The label sets will consist of all N columns; no consideration is given to the various symmetries peculiar to the N-Queens problem.

Our comparison will be in terms of three kinds of graphs and tables:

1. The number of nodes visited is largest for the middle levels of the tree search, with the looking ahead procedure having fewest nodes at each level. (Figures 4 and 6)
2. The bulk of the consistency tests are done at shallow depths of the tree search for the look ahead types of peocedures (Figure 5)
3. Forward checking and backmarking do the fewest consistency tests among the algorithms using the normal unit orders in most cases tested. (Tables 1 and 2)

3. STATISTICAL MODEL FOR CONSTRAINT SATISFACTION SEARCHES

Our statistical model for random constraint satisfaction is simple. The probability that a given consistency check succeeds is independent of the pair of units or labels involved and is independent of whatever labels may already have been assigned to past units. Hence, $P((u_{K+1}, \ell_{K+1}, u, \ell) \in R | \ell_1, \ldots, \ell_K$ are consistent labels of $u_1, \ldots, u_K) = P((u_{K+1}, \ell_{K+1}, u, \ell) \in R$ for every u, ℓ.

In our analysis, we will assume that a given pair of units with a given pair of labels is consistent with probability p, p being independent of which units, which labels, or any past processing; and there will be M units total. If each unit has the same number L of possible labels, then any K-tuple of labels for any K units has probability $p^{K(K-1)/2}$ of being consistent since each labeling must satisfy $K(K-1)/2$ consistency checks. Since there are L^K possible labelings of K units, the expected number of consistent labelings is

$$L^K p^{K(K-1)/2}$$

Using our statistical model, the expected number of consistency checks at level K in the tree search can be derived:

$$L^K p^{(K-1)(K-2)/2} \frac{1-p^{K-1}}{1-p}$$

We might note that the N-Queens problem differs some from the assumption of our statistical model. In terms of number of solutions, N-Queens has fewer solutions than the expected number of solutions of its associated random constraint satisfaction problem. In terms of consistency checks, N-Queens requires more checks than the expected number of checks of its associated random constraint satisfactior problem.

The computation of the number of labelings at any depth K for the forward checking algorithm is considerably more complicated, but the result is nearly as simple as for backtracking. The expected number of consistent labelings at level K out of M units will be

$$L^K p^{K(K-1)/2} [1 - (1-p^K) L]^{M-K}$$

Notice that for K = M this is the expected number of solutions to the problem, and the computation agrees with the one for the standard backtracking tree search. The expected number of consistency checks at level K will be

$$L^{K+1} p^{(K-1)(K+2)/2} [1-(1-p^{K-1}) L]^{M-K-1} (M-K)$$

Figure 6 shows that the expected number of consistent labelings for random constraint satisfaction problems agrees closely with the actual number in the average of several experiments.

4.1 Optimizing the Consistency Check Order in Tree Searching

Suppose we are solving a constraint satisfaction problem and suppose units $1, \ldots, K$ have already been assigned labels $\ell_1, \ldots, \ell_K$ are we are trying to find a label ℓ_{K+1} for unit K+1. The label ℓ_{K+1} must come from some set S_{K+1} of labels and it must be consistent with each of the previous labels $\ell_1, \ldots, \ell_K$, that is, we must have $(k, \ell_k, K+1, \ell_{K+1}) \in R$ for $k=1, \ldots, K$. To determine the label ℓ_{K+1}, we sequentially go through all the labels in S_{K+1} and perform the K consistency checks: $(k, \ell_k, K+1, \ell_{K+1}) \in R$. If one check fails, then we try the next label in S_{K+1}. If all checks succeed, then we can continue the

depth first search with the next unit.

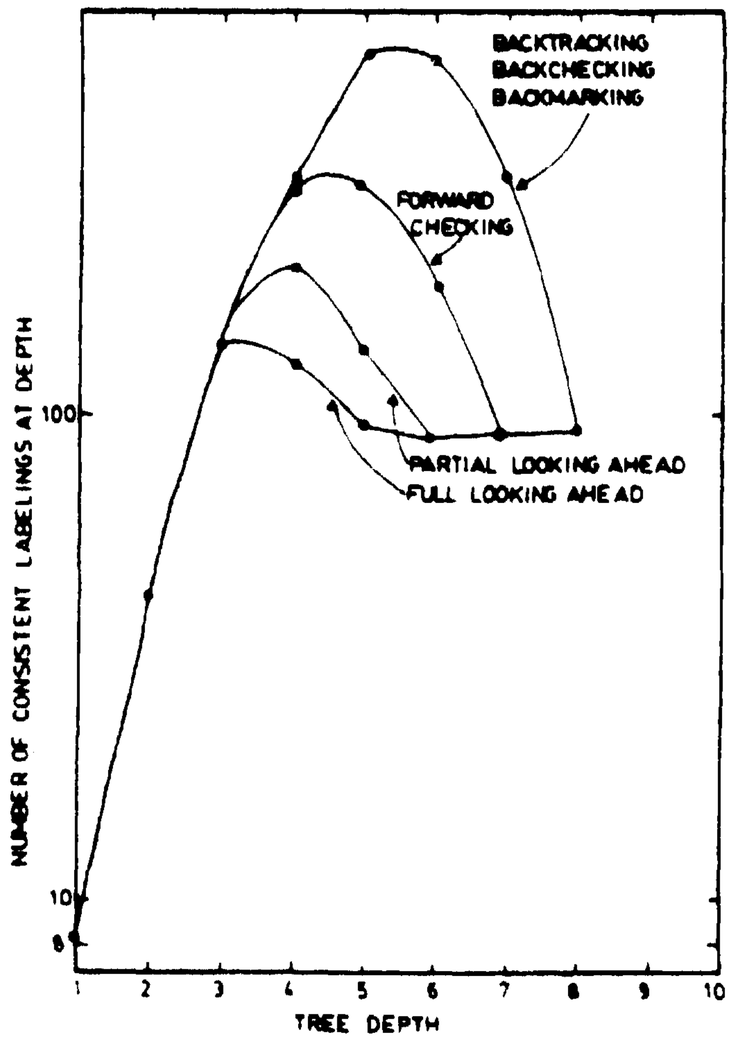


Figure 4 illustrates number of consistent label ings as a function of tree depth, for the 8 queens in the natural unit order.

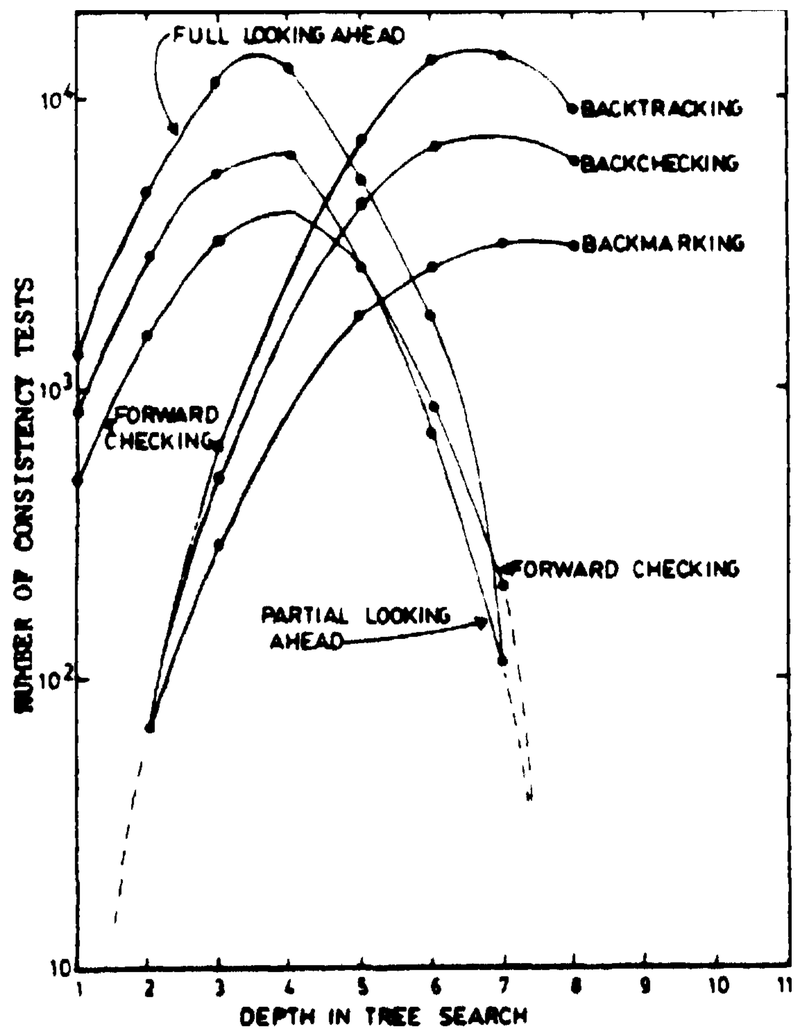


Figure 5 compares the number of consistency tests made at each level in the tree search for six different procedures, for the 8 queens problem in the natural unit order.

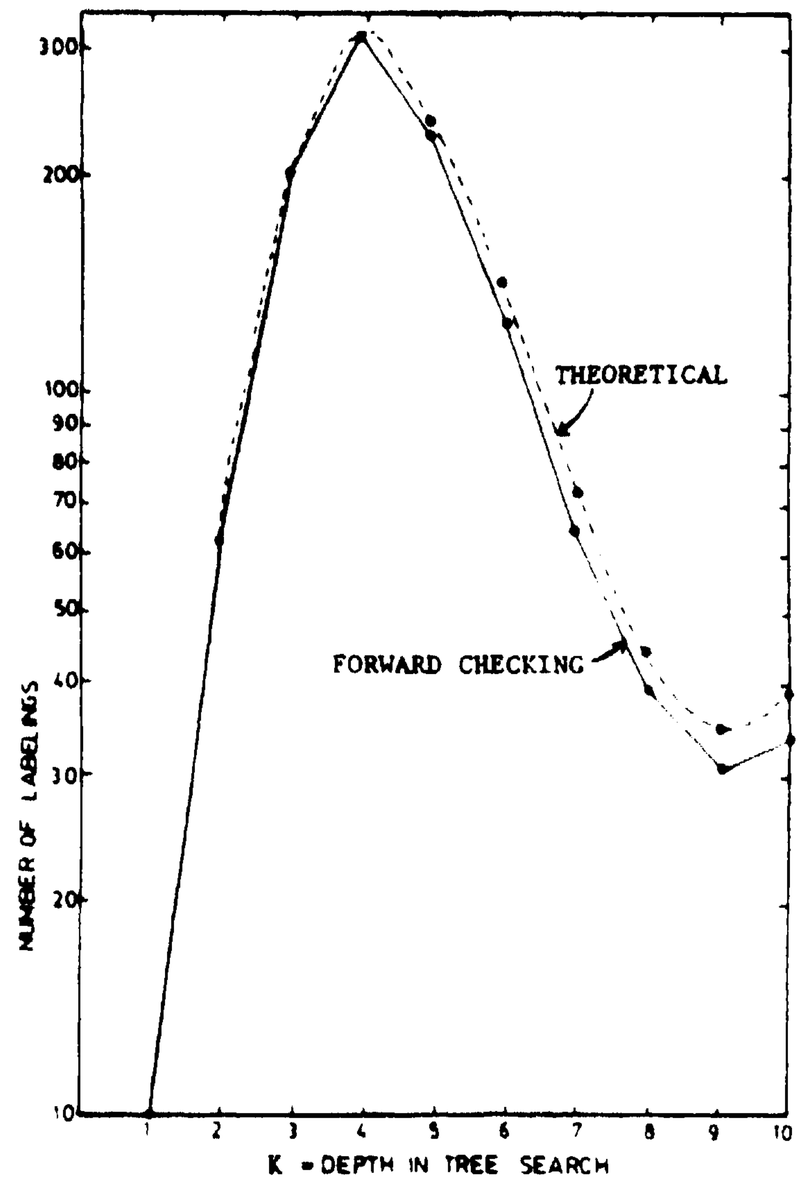


Figure 6 indicates the number of consistent labelings to depth K in the tree search for the average of 25 random constraint satisfaction problems with probability of consistency check success of .65 and number of units = number of labels = 10. The dotted curve is the theoretical expected number of labelings for such a problem.

The optimizing problem for consistency checking is to determine an order in which to perform the tests which minimizes the expected number of tests performed. To set up the optimizing problem, we must have some knowledge about the degree to which a previous unit's label constrains unit (K+1)'s label. For this purpose we let P(k) be the probability that the label I_{k_k} for unit k is consistent with some label for unit K+1. We assume that the consistency checks are independent events so that the probability of the tests succeeding on units 1 through K is $\prod_{k=1}^{K} P(k)$.

For each order of testing, these probabilities determine the expected number of tests in the following way. Let $k_1,\ldots,k_K$ be a permutation of 1,...,K designating the order in which the consistency checks will be performed. The expected number of tests performed can be shown

to be

$$1 + \sum_{i=1}^{K-1} \prod_{j=1}^{i} P(k_j)$$

We can minimize the expected number of tests by having $k_1,\ldots,k_K$ be any permutation of $1,\ldots,K$ satisfying $P(k_1)\leq P(k_2)\leq,\ldots,\leq P(k_K)$. Hence, we must choose to test against those past units whose label is most likely to fail against the current unit label pair.

To illustrate the advantage of using an optimum consistency test order, we consider the 10 Queens problem for the standard back tracking procedure which checks the current units' label against the past units' labels. In the N-Queens problem when the units are naturally ordered from 1 to N and the current unit is K, then the fail first principle states that tests with past units must be done in the order of decreasing constraints. Hence, the test order should be first unit K-1, then K-2, up to unit 1. Backtracking requires 1,297,448 tests when done in the wrong order (unit 1, 2,...,K-1) and 1,091,85b tests when done in the right order.

4.2 Optimizing Tree Search Order

Every tree search must assume some order for the units to be searched in. The order may be uniform throughout the tree or may vary from branch to branch. It is clear from experimental results that changing the search order can influence the average efficiency of the search. In this section we adopt the efficiency criterion of branch depth and we show how by always choosing the next unit having smallest number of label choices we can minimize the expected branch depth.

Let $k_1,\ldots,k_M$ be the order in which the M units are searched on the tree. Let $P_n(k_n|k_1,\ldots,k_{n-1})$ be the conditional probability that some label for unit k_n will succeed when unit k_n is the n^{th} one in the tree search order given that units $k_1,\ldots,k_{n-1}$ are the first n-1 units searched in the branch. We assume that the probability of a label for unit k_n succeeding depends only on the number of units preceding it in the tree search and not upon which particular units they are. That is,

$P_n(k_n|k_1,\ldots,k_{m-1}) = P_n(k_n|v_1,\ldots,v_{n-1})$ for all units combinations $v_1,\ldots,v_{n-1}$

This conditional independence assumption justifies the use of the notation $P_n(k_n)$ to designate the probability that some label succeeds for unit k_n when it is the n^{th} unit in the tree search, and we will call the probability that an arbitrary label for unit u will succeed when checked against another arbitrary unit's label the success probability for unit u.

Units which are searched later in the tree typically have lower probability for a label succeeding since the label must be consistent with the labels given all the earlier units. We want some way to compare the probability of success for the same unit in different tree searches. Since the success probability depends only on the unit and its level in the tree and since units later in the tree have lower success probabilities, we assume that the success probability for a unit u when it is at level i in one tree search is related to the success probability of unit u when it is at the first level of another tree search by a constant factor a where $0<a<1$:

$$P(u)\ \alpha^{i-1} = P_i(u)$$

The best search order is the one which minimizes the expected length or depth of any branch. When the units are searched in the order $k_1,\ldots,k_M$, the expected branch depth is given by

$$1 + \sum_{n=1}^{M-1} \prod_{j=1}^{i} P_j(k_j)$$

This is minimized when the unit chosen at each level is that unit whose success probability is smallest. Thus at level j we choose unit k_j, where

$$P_j(k_j) \leq P_j(u) \quad \text{for } u \neq k_1,\ldots,k_{j-1}$$

Letting the unconditioned success probability of one unit with another be q we have, $P_j(k_j) = \alpha^{j-1}[1 - (1 - q^{n(k)}j)]$. Since $0\leq q\leq 1$, this expression is minimized by choosing k_j to be that unit having the smallest number of possible labels.

To illustrate the advantage of using a locally optimal unit order for each branch in the tree search, we consider the improvement achieved on the N Queens problem and the order N random relation problem for $4\leq N\leq 10$. The number of consistency tests required is given in Tables 1 and 2.

The reason why optimal unit order usually improves forward checking more than backroarking is that forward checking has more information about future units than backmarking. Therefore, forward checking's choice of the next unit most likely to fail is more likely to produce a unit which fails than backmarking's choice.

Some improvement is shown in the larger N-Queens problems, and considerable improvement appears in the larger order N random constraint problems. This improvement increases with problem size in the random constraint problem with p = .65.

V. CONCLUSION

We have shown analytically and experimentally the efficacy of the remembering and fail first principles in constraint satisfaction tree search problems. A new search procedure called forward checking has been described and it combined with optimal unit order choice leads to a more efficient tree search than looking ahead or backmarking in the large problems tested. This suggests that the entire set of look ahead operators described by Haralick et. al. (1978), Haralick and Shapiro (1979a, 1979b), the discrete relaxation described by Waltz (1972) and Rosenfeld et. al. (1976) would be more efficiently implemented by omitting the consistentcy tests required by future units against future units. Future analytic and experimental work needs to be done to determine if this in fact is generally true. Further work also needs to be done to explore the tradeoff between data structure overhead and number of consistency tests so that the lowest CPU time algorithms can be determined.

Table 1 Number of Operations in N-Queens Problem for Normal and Optimal Unit Order

N	Backtracking	Backchecking	Full Looking Ahead		Part Looking Ahead		Backmarking		Forward Checking	
	Normal	Normal	Normal	Optimal	Normal	Optimal	Normal	Optimal	Normal	Optimal
4	84	80	99	99	97	97	76	76	76	76
5	405	356	598	578	485	431	276	276	282	282
6	2016	1496	2095	2082	1703	1708	944	921	964	946
7	9297	6042	8942	8941	6511	6318	3236	3168	3338	3229
8	46752	27450	35323	35211	25882	25062	12308	12095	13024	12108
9	243009	131538	153455	151275	112327	106247	50866	50027	55326	49856
10	1297558	643658	661017	636377	496455	449666	220052	211635	242174	205970

Table 2 Number of Operations in Average of 5 Random Constraint Satisfaction Problems with Consistency Check Success Probability .65, for Normal and Optimal Unit Order

N = number of units = number of labels

N	Backtracking	Backchecking	Full Looking Ahead		Part Looking Ahead		Backmarking		Forward Checking	
	Normal	Normal	Normal	Optimal	Normal	Optimal	Normal	Optimal	Normal	Optimal
4	243	195	230	223	184	174	132	133	133	121
5	1043	731	760	712	599	554	425	382	414	355
6	4637	2514	2543	2288	1966	1700	1288	1159	1273	979
7	12040	6722	5722	5156	4697	3875	3175	2486	3057	2069
8	25893	13490	10779	9306	9111	7233	6089	4300	5978	3425
9	118086	55318	30799	25904	26788	19174	21170	10246	18616	8673
10	163983	73260	44655	36675	41232	28872	28314	14892	26288	12022

References

(sorry, no space)

EXPERIENCE WITH ROBOT
IN 12 COMMERCIAL, NATURAL LANGUAGE DATA BASE QUERY APPLICATIONS

Larry R. Harris
Department of Mathematics
Dartmouth College
Hanover, New Hampshire 03755

Tne aoility to unaerstand Natural Language has long been a goal of Artificial Intelligence Research, and it is still far from being solved, however, in tne early 1970's tne AI research techniques reached a point whereby certain applications became feasible for tne first time. Since that time, several systems sucn as PLANES[1], LiFER[2], and ROBOT[3,4,5] have been built tnat nave demonstrated tnat tne current state of the art is sufficient for quite gooD natural language data base query.

i'ne implementation of the R0301 system has been geared for high performance and instaliaoility in accual real world environments. As sucn, it offers the AI researcn community some insignt into tne difficulties encountered when putting tne current: AI tecnnology in tne nands of people in the real world. This paper discusses cne unexpected linguistic and semantic difficulties encountered in the 12 commercial applications to wnicn ROBOT nas been applied during the last year and a nalt.

mere are many differences between nacurai language systems developed for researcn and natural language systems developed for production. Not tne least or tnese differences is tne fact tnat tne production systems are eventually used in tne aosence of tne developers of tne system. AS sucn, valuable insights into tne problem of natural language processing can be gained. In spite of tne fact tnat all of us speak English, none of us Know exactly wnat the user is going to type. Beyond this, we don't Know now tne user's questions will cnange after several months of use. This is valuable information tnat only systems tnat nave actually been installed for long periods of time in the user's environment can provide.

The ROBOT system has oeen applied to 12 different applications during tne past 18 montns. In eacn case, tne system nas oeen "turned over" to users for evaluation. Tne pnrase "turned over" is jargon of tne software industry meaning tnat tne program is now completely in tne nanus of tne users, This is particularly ditficult to do for natural language systems, since so much application-specific tuning is required. But the system must be officially turned over if we wish to observe the unencumbered user. Naive users, in particular, are greatly influenced when technicians or researchers are observing tneir use of a computer system. It is only after the system is completely in their hands that we can begin to gain accurate data on how well the system satisfies their needs.

2. THE 12 APPLICATIONS

ROBOT has now been completely turned over to users at the following companies: Bigelow-Sanford Carpet, Commercial Union Assurance, Du Pont Chemical, Planning Research Corporation, ana Braastreet and Software Ag of Nortn America.

The following is a list of the different data bases to which ROBOT has been applied in these installations
By ROBOT development staff: Customer file, Homeowner's Insurance file, Employee relations file, Car file, Employee file, Data dictionary, Financial file and budget control; by local staff: Sales file, Membership file, Accounting file and Planning file.

The variation in the different discourse domains of these applications illustrates the importance of the domain-independent parsing techniques employed by ROBOT (see [5]). The sizes the applications range from 1,000 records in the test files to over 2.4 million records.

3. ACTUAL USER REQUESTS

The following questions were selected from log files in response to a survey requesting interesting queries that have been asked by local users that ROBOT was able to answer. Although this sample is biased towards requests that ROBOT can handle, it does demonstrate the level of sophistication that users expect from the system.

SHOW SUBTOTALS OF DIRECT COMMISSION BY MONTH FOR 1ST QTR '76 IN REGION 4 WHERE NET ARISING AMOUNT IS AT LEAST $150,000.

FOR DOMESTIC MACHINERY, PRINT THE JULY 1976 SALES DOLLARS AND SALES QUANTITIES, SORTED BY SALES CLASS.

FOR COMMERCIAL DIVISION AND THOSE PRODUCTS IN EXCESS OF $500,000 YEAR-TO-DATE, SHOW ME ALL TIME PERIODS WHOSE PRE-TAX EARNINGS ARE MORE THAN $200,000 AND PRINT THEIR NAME.

FOR SUMMARIES WITH LOSS RATIO GREATER THAN 200, REPORT REGION, BRANCH AND LOSS RESERVE BY MONTH.

PRINT ALL PRODUCTS WHOSE NOVEMBER 1977 SALES QUANTITIES ARE BETWEEN 100,000 AND 300,000 AND WHOSE PRE-TAX EARNINGS EXCEED $1,000,000.

4. PROBLEMS ENCOUNTERED

Although the overall experience with ROBOT at all of these sites has been quite good, there have nonetheless been several very difficult problems of a linguistic or semantic nature that have surfaced. The remainder of this paper discusses these problems and potential solutions for them. A problem is defined as any desired response to a request that cannot be produced by appropriate dictionary definitions. Note that this is a rather strict definition since it presumes that the parser and the set of primitives that can be used to create dictionary definitions are fixed. Even the simplist change to the ATN grammar is considered a major problem because this cannot be done by local staff at the site. A solution is defined to be a domain-independent extension to ROBOT and/or the dictionary utility that allows the desired response to be produced by appropriate dictionary definition. The importance of the domain-independence of the solution should be clear from the variety of applications that ROBOT currently deals with. It is also clear that researchers who do not share our concern for domain-independent solutions may find the following problems very easy to solve in any given application. With this disclaimer we can now begin discussing the problems known as:

The "ME OR IN" Problem,
The Snowmobile Problem,
The Matrix Problem, and
The Summary Problem.

4.1 The "ME OR IN" Problem

This problem arises as a result of coded data in the data base. The dictionary must contain enough information about the coding scheme so that it can be made transparent to the user. Typically, users want to have the flexibility of referring to the data in either the long form or the coded form. For example, in a coded state field, the user may want to refer to New Hampshire either as NH or New Hampshire. The specific problem arises when a coded item intersects with other words. For example, the abbreviation for Maine is ME, and the abbreviation of Oregon is OR and the abbreviation for Indiana is IN. It is particularly hard for the parser to tell in some uses of the word IN whether its the preposition "in" or Indiana. ROBOT normally deals with lexical ambiguity by considering all interpretations and then heuristically selecting the best one. This approach is fine in most circumstances, but does not work very well when one interpretation is much more common than the other. For example, the request "PRINT FOR ME THE NAMES OF ALL SECRETARIES." would be detected as ambiguous. Does it refer to all secretaries or just those in Maine? When you are aware of the information the program has to work with, it's hard to fault it for entering into a clarification dialog with the user in cases like this. But when you are constantly badgered by the system to help weed out such obscure interpretations, it can get very tiresome.

The proper solution for this problem seems to be an improved heuristic that is perhaps a little more ruthless in weeding out interpretations using coded data items. Unfortunately the environment does not provide the normal clues, such as capitalization, that help people make this distinction. Also the typewritten form of expressions tends to exacerbate the problem since people tend to use coded references rather frequently as a way of reducing typing. The current, admittedly short term, solution of ROBOT is to restrict selected coded data items so that they cannot be used as words in a request. This allows the person that builds the dictionary to decide whether the ambiguous meaning is desirable or not. This has the effect of forcing users to refer to Indiana, Oregon and Maine in their full form, where other states can be referenced either way.

4.2 The Snowmobile Problem

This problem arose in the context of the question "How many snowmobiles are there?". This question was directed towards a homeowner insurance file that included information on snowmobile coverage. The interpretation ROBOT gave to this request was "How many policies have snowmobile coverage?". Needless to say, the answers to these questions are very different. The subtlety of this problem isn't so much linguistic as it is in the data representation. If the file being queried would have contained one record for each snowmobile, then ROBOT would have answered it properly. But the file contained one record per policy and a field containing the number of snowmobiles covered by that policy. ROBOT's dictionary defined the meaning of snowmobile to be any number greater than 0 in this field. From this it should be clear why ROBOT first found the policies with at least one snowmobile and then counted the policies.

The point is that the computation of the answer depends on how the data is stored. The syntactic structure of the request is of little help in figuring what computation is required to answer the request. Consider the following requests:

"How many snowmobiles are there?"
"How many secretaries are there?"
"How many salaries are there?"

These are all syntactically identical. One might even argue that they are semantically homomorphic. But each request requires that the answer be computed in a totally different way. We have already seen that the snowmobile request requires summing the contents of the snowmobile field. Assuming that secretary is an instance of a job title, then the number of secretaries is computed by counting the number of records with secretary occurring in the job field. The proper response to the request about salaries may well be to give the number of different salaries that occur in the file. Needless to say, the algorithm required to compute this is very different from the other "How many" questions. ROBOT is currently being fitted with enough domain-independent facilities to permit the inclusion of enough domain-specific knowledge to know which algorithm to employ in each case.

4.3 The Matrix Problem

This problem arises because there are many ways of referring to data aggregates. For example, one of the applications had historical data in several fields such as profit and total sales. Thus, users could to refer to the profit in 1976 or the 1972 total sales. In many requests the year specifier would refer to several fields. For example, "In 1972, what were the profits and total sales." In other cases several years might apply to one field. For example, "Give me the total sales for 1972 and 1973."

These questions could not be processed because they require combining the syntactic information gained from the parse with domain-specific information about the data base. The answers to these questions are generated in different ways depending on whether the years themselves are stored as data items. In the particular instance in which this problem was first encountered, ROBOT required very direct scoping of the time referent. For example: "List the 1972 profits and the 1972 total sales?", or "List the 1972 total sales and the 1973 total sales."

Because this was perceived as a rather severe linguistic limitation, ROBOT'S dictionary facility now allows the specification of how the time referents and the periodic fields are related. This information can then be

A zoom lens and a pan-tilt head of the camera help the accurate pointing and measuring.

2.2 Pointer Operation

A user manipulates a joystick to locate the spot at the place wherever he vants. Then the spot position in the work space coordinates system is calculated "based on the triangulation and is stored in the data area with a name for the later reference. The spot is also controllable by the computer.

2.3 Geometry Editor

Using obtained points from the real world, this module makes basic geometric elements. They are lines, planes, vectors, rotation matrices and so on. Generation of basic bodies and synthesis to complicated objects are also performed. Furthermore non-geometric information about objects is added to construct better model.

2.4 Graphics and Scope Accomodation

Calculating the direction and the range of view of the TV camera, the graphics generates the projected skelton image of objects from the stored data. With this generated image, the user can easily grasp three dimensional shapes and spatial relations of objects.

3. EXAMPLES OF THE SYSTEM PERFORMANCE

The hardware of this system is completed, and the software implementation is now in progress. Fig. 2 through 4 illustrate how the system will work.

A vise and its handle are placed in the work space and are monitored. (Fig. 2)
The generated model of the handle and handle-shaft are displayed on CRT. (Fig. 3)
The real scene and the generated image are superimposed on the monitoring display. This provides us convenient ways for verification of the generated model. (Fig. 4)

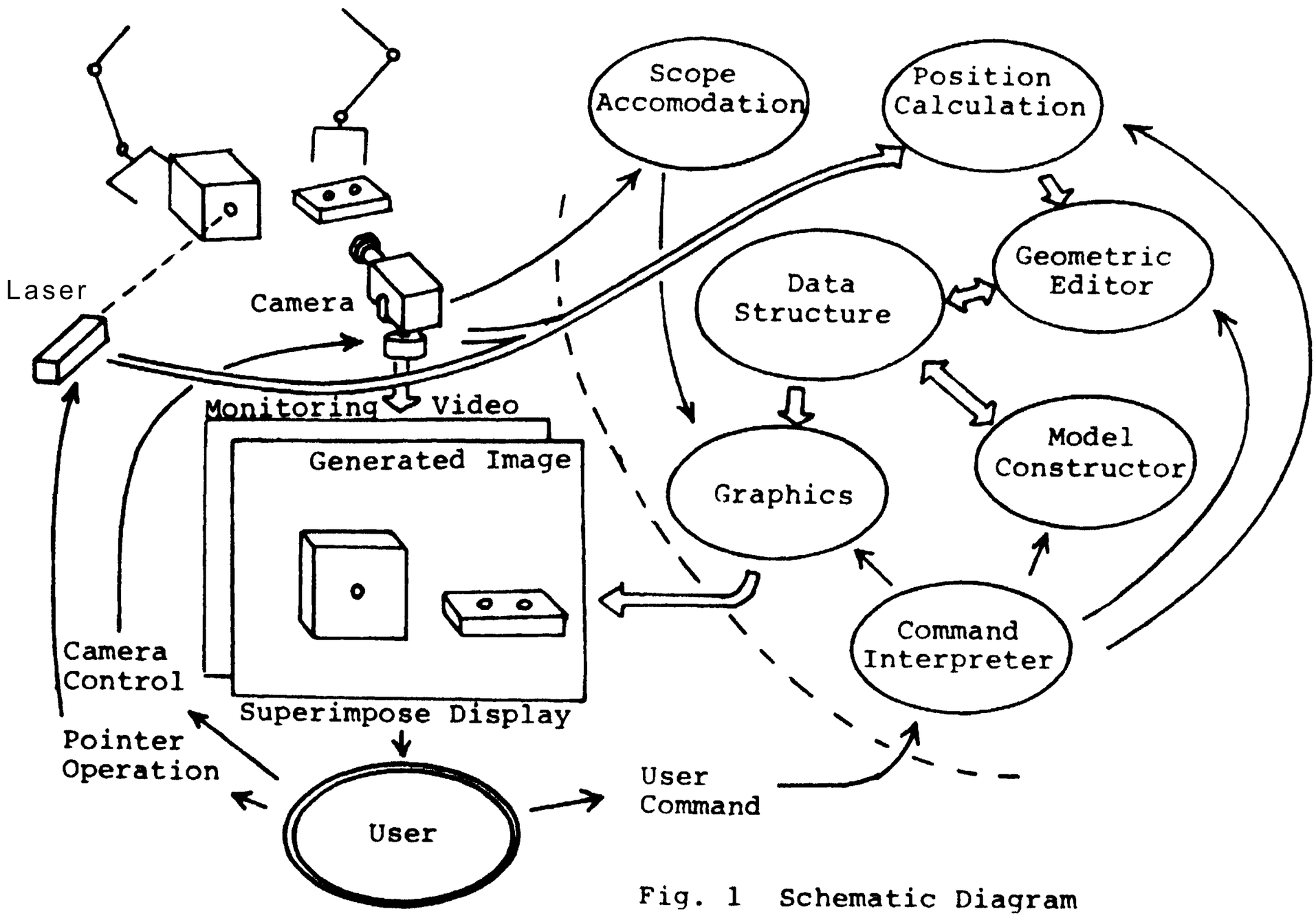


Fig. 1 Schematic Diagram

4. CONCLUSION

The superimpose display allows us to monitor the real world and the internal data simultaneously. Judging whether the generated image and the real scene are well overlapped or not, we can supervise the process and the result of model construction. Thus, combination of the laser pointer and the superimpose display promises natural man-machine interaction.

Since active scanning of the laser spot by computer is possible, the laser pointer will be developed to a semi-automated local vision.

REFERENCES

[1] Grossman, D. D. and Taylor, R. H. (1978) "Interactive Generation of Objects Models with a Manipulator." IEEE Trans. Systems, Man, and Cybernetics., Vol. SMC-8, pp667-6T9

[2] Ishii, M. and Nagata, T. (1976), "Feature Extraction of Three-Dimensional Objects and Visual Processing in a Hand-Eye System Using Laser Tracker." Pattern Recognition, Vol. 8, pp 229-237

[3] Hosaka, M. and Kimura, F. (1977), "An Intreactive Geometrical Design System with Handwriting Input." Information Processing 77, ppl.67-17P, North-Holland, Amsterdam

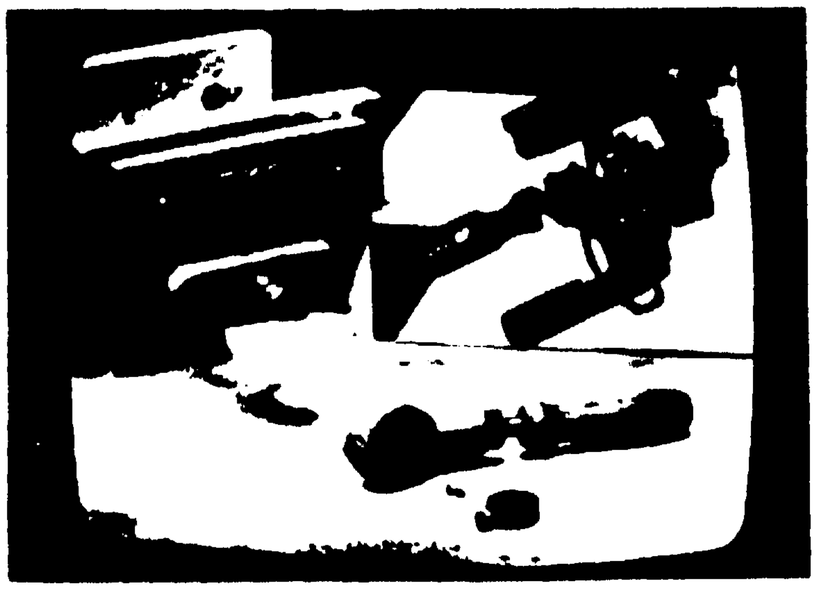


Fig. 2 Real Scene

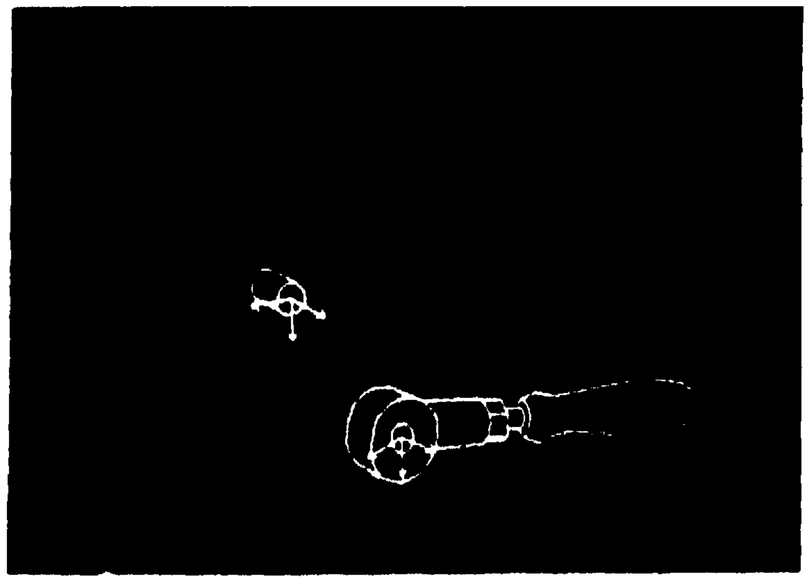


Fig. 3 Generated Model

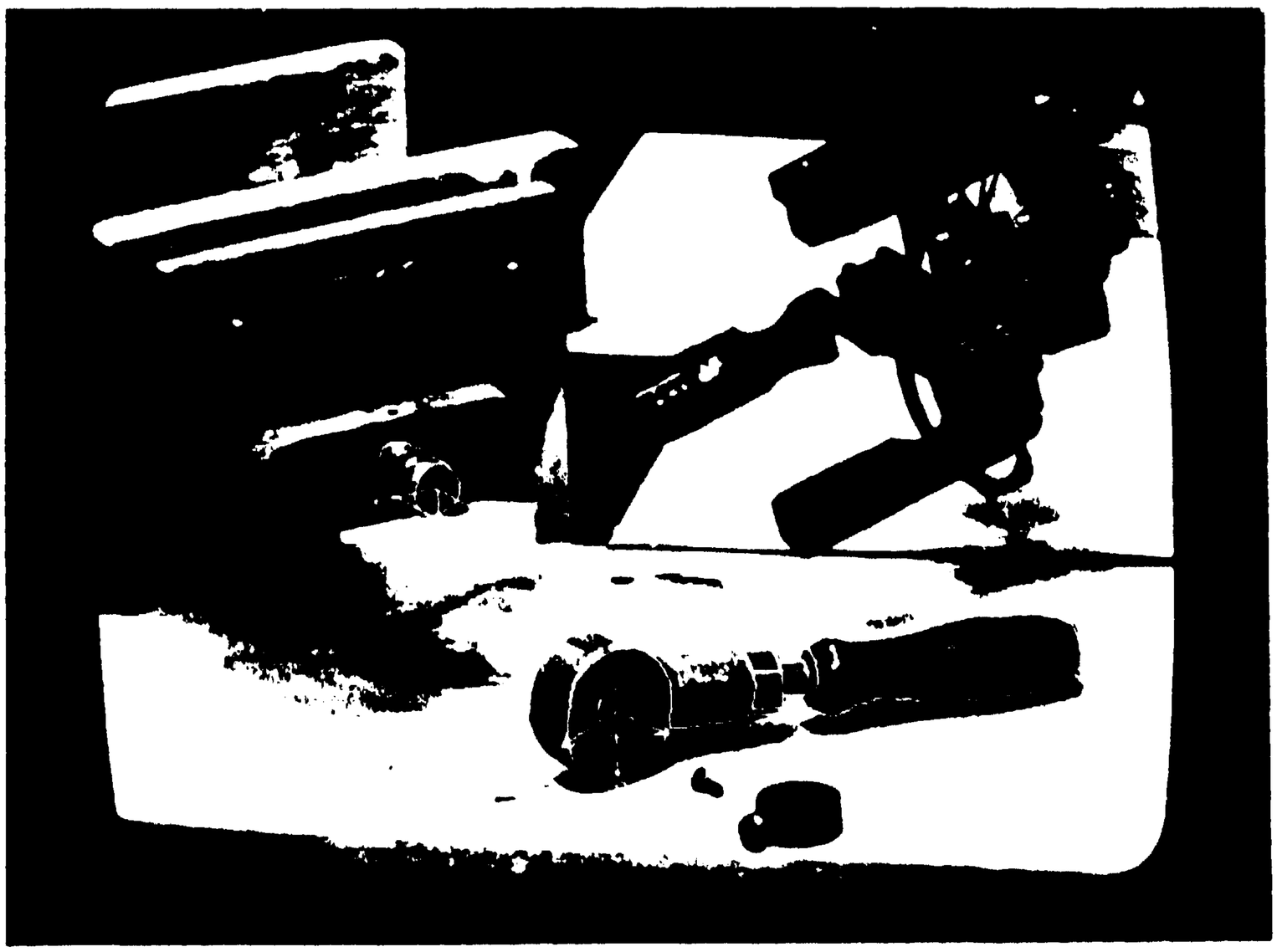


Fig. 4 Superimposed Display

GRACEFUL INTERACTION IN MAN-MACHINE COMMUNICATION

Phil Hayes and Raj Reddy,

Computer Science Department,
Carnegie-Mellon University,
Pittsburgh, PA 15213.

Compared to humans, current natural language dialogue systems often behave in a rigid and fragile manner when their conversations deviate from a narrowly conceived mainstream, e.g. when faced with ungrammatical, unclear, or unrecognizable input, ambiguous descriptions, or requests for clarification of their own output. We believe that the time is now ripe to construct systems which can interact gracefully with their users when such contingencies arise. Graceful interaction is not a single skill, but a combination of several diverse abilities. We list these components, and describe one of them - the ability to communicate robustly. Detailed descriptions of all the components appear in [4], along with details of a system architecture for their integrated Inplementation.

1. Introduction

A great deal of interest has recently been shown In computer systems capable of engaging in a dialogue with a human being in more or less natural language. This interest is embodied in numerous systems including GUS [1] LIFER [51 PAL [61 PARRY [71 and PLANES [8], and in the work of Codd [21 Grosz [31 and others. Such systems typically respond accurately and appropriately to straightforward requests and questions or otherwise uneventful dialogue within their domain of discourse. Compared to human beings, however, the performance of current dialogue systems appears quite rigid and fragile. Most current systems cannot, for instance: respond reasonably to input not conforming to a rigid grammar; ask for and understand clarification if their user's input Is unclear; offer clarification of their own output if the user asks for it; or interact to resolve ambiguous descriptions. A dialogue system cannot hope to interact naturally and gracefully with its users, unless it can cope with these and the many other contingencies, so common in ordinary human conversation.

Graceful interaction, then, involves dealing appropriately with anything a user happens to say, rather than just those inputs in the mainstream of a conversation. Even though little attention has been paid to it in work to date (PARRY [7] being the main exception), we believe that graceful interaction is of critical importance in making natural language interfaces (for both typed and spoken input) a practical reality, and thus in making computer systems more accessible to casual or naive users, and more pleasant and natural for everyone.

Graceful interaction is *$n* idea whose time has come and is ripe for implementation. We believe that graceful interaction is not a monolithic skill, but can be decomposed into a set of abilities and behaviours, none of which is significantly beyond the current state of the art. These components of graceful interaction are listed in the next section. In this short note, it is only possible to give a few details on just one of the components; we discuss alt of them fully in [4} In that paper, we also: claim that besides being necessary, our list of components is sufficient for graceful interaction, at least in a certain (large) class of application domains; propose an architecture for the integrated implementation of all the components mentioned; and give a worked example of our design in operation.

2. Components of Graceful Interaction

Graceful interaction is not a single monolithic skill. Rather, it seems to be composed of a number of diverse abilities and behaviours, including:

Flexible parsing: The ability to deal with naturally used natural language, with all the ellipses, idioms, grammatical errors, and fragmentary utterances it can contain.

Robust communication: The set of strategies needed to ensure that a listener receives a speakers utterance, and interprets it correctly.

Focus mechanisms: The ability to keep track of what the conversation is about, as the items under discussion shift; this is important for the resolution of ellipsis and anaphora, as well as for continuity in the conversation.

Explanation facility: The ability to explain what it can and cannot do, what it has done, what it is trying to do, a*nd* why, both for response to direct questions, and as a fall-back when communication breaks down.

Identification from descriptions: The ability to recognize an object from a description; including the ability to pursue a clarifying dialogue if the original description is unclear.

While none of these components of graceful interaction has been entirely neglected in the literature, no single current system comes close to having most of the abilities and behaviours we describe, and several are not possessed by any current systems.

These components are based on phenomena observable in naturally occuring human dialogues. We believe it is very important for a gracefully interacting system to conduct a dialogue in as human-like a way as possible; if the strategies the system employs for clarifying its incomprehensions of the user, resolving ambiguous

descriptions supplied by the user, etc. are not the same as those a human would use in the same situation, then the user will feel that the interaction is not natural, and hence not graceful. Furthermore, most of the components involve *cooperation* between speaker and listener; if the user tries to employ any of these techniques, an inability of the system to cooperate will appear most ungraceful.

Unfortunately, the aim of being as human-like as possible must be tempered by the limited potential for comprehension of any forseeable computer system. Until a solution is found to the problems of organizing and using the range of world knowledge possessed by a human, practical systems will only be able to comprehend a small amount of input, typically within a specific domain of expertise. Graceful interaction must, therefore, supplement its simulation of human conversational ability with strategies to deal naturally and gracefully with input that is not fully understood, and to steer a conversation back to the system's home ground.

Space restrictions make it impossible to discuss all the components of graceful interaction mentioned above (see instead [4]), so we will concentrate on just one of them - the set of techniques for robust communication.

3. Robust Communication and Implicit Confirmation

During the course of a conversation, it is not uncommon for people to misunderstand or fail to understand each other. Such failures in communication do not usually cause the conversation to break down; rather, the participants are able to resolve the difficulty, usually by a short clarifying sub-dialogue, and continue with the conversation from where they left off. Current computer systems are unable to take part in such clarifying dialogues, or resolve communication difficulties in any other way. As a result, when such difficulties occur, and they can be neither eliminated nor anticipated, a computer dialogue system is unable to keep up its end of the conversation, and a complete breakdown is likely to result; this fragility lies in stark and unfavourable contrast to the robustness of human dialogue.

Since a speaker typically has no way of telling whether his listener has received his message correctly, detection of communication difficulties must rest on some convention of acknowledgement, commonly agreed upon by speaker and listener. Explicit acknowledgement by a listener of everything a speaker says would be tedious for humans. Instead, people use a convention that we will call the principle of implicit confirmation;

> A speaker assumes his message has been received and interpreted correctly by his listener, unless the listener indicates otherwise.

While this approach is obviously very efficient when there is no difficulty with communication, it places a large burden on the listener. Unless the listener can determine that he has not received the message correctly, the error will go undetected, and the conversation may become quite confused. However, a human listener normally has enough expectations about the sorts of things the speaker might say to make this a rare occurrence.

Although we have been using the terms, "speaker" and "listener", the same communication difficulties can arise for typed dialogues. While the errors typically arise from different sources (e.g. misrecognition of words v. spelling errors), for present purposes, we can continue to blur the distinction between spoken and typed dialogues. In what follows, we consider two different methods by which a listener can indicate incomprehension, discussing how much, if at all, current systems use these techniques, and noting modifications required for systems with limited ability to comprehend. Examples will use dialogues between a hypothetical directory assistance system (S) and its user (U).

3.1. Explicit Indications of Incomprehension

The principle of implicit confirmation requires a listener to give an explicit indication whenever he fails to comprehend a message or suspects that he may have received a message incorrectly. In this section we deal with the most straightforward way a listener can do this: by asking the speaker what he said or meant. Such questions vary according to how much of the original utterance the listener thinks he understood, how informative he tries to be about the nature of the problem, and how much he wants to influence the speaker's subsequent reply.

Phrases such as "I beg your pardon" are the least informative way of indicating incomprehension. They provide the original speaker with no clue about what went wrong with the communication, and so *are* most appropriate when the failure is due to a transient problem such as a sudden noise. A speaker is much more likely to modify his communication in a way that will allow it to succeed at a second attempt if his listener gives him some information about the nature of the problem.

The clues the listener can provide depend on the degree to which he failed to understand the utterance. If he understood nothing at all, the only clue he can provide is what he expected the speaker to say:

S: This is directory assistance.
U: GARBLE
S: Excuse me? May I find you a number?

If he has understood something, he can show what he has understood while indicating incomplete comprehension:

U: What is the number for GARBLE?
S: Whose number did you want?

If he has understood enough to narrow the possibilities down to a small number, he can list them. The information given by these various strategies to the original speaker allows him to concentrate on imparting the information that did not get across.

The way in which incomprehension is indicated can influence the form of the subsequent clarification. As Codd

[2] points out, a gracefully interacting system with limited power of comprehension should therefore phrase its indications of incomprehension to maximize its chances of an understandable reply. For instance, if a system does not understand "extension" in "What is the extension for Jim Smith?", it is unwise to reply "What is an extension?", but better to ask "Do you want Jim Smith's number or his address?". On the other hand, such directiveness is *never* guaranteed success, and in particular, a user cannot always be expected to cooperate by choosing one of the alternatives in a multiple choice question. For instance, reasonable responses to the last question include "The number", "The first one", "Both", "I want to phone him", "Do you know where he is now?", "I want the number, but 1 meant Joe Smith". This raises the possibility of a user's clarification also being misunderstood, and of a sequence of misunderstandings and clarifications failing to converge. In such cases, a gracefully interacting system will have to become more and more explicit about the nature of its problems and its limitations, and rely on the user to accommodate himself.

In summary, a gracefully interacting system must be able to express its own incomprehension of or uncertainty about its user's input, and be able to deal with the user's reports of his own problems in comprehending the system. While most systems can indicate incomprehension, most do it in a more or less uninformative and undirective way. As far as we know, no current system can respond appropriately to explicit complaints of incomprehension by its user.

3.2. Echoing

Echoing is a way of confirming uncertain Interpretations without the requirement for a reply associated with an explicit request for confirmation. If a listener wants to be sure that his interpretation of the speaker's utterance (or more commonly part of it) is correct, he *need* only echo his interpretation. If the original speaker does not comment on the echo, it is implicitly confirmed. Thus:

U: What is the number for ?ter Smith?
S: Walter Smith ... His number is 5592.
U: Thank you.

Of course, the original speaker still has the option of confirming the echo explicitly; and if the echo is incorrect, he must indicate that explicitly.

A gracefully interacting system should be able to issue echoes when it is uncertain in its comprehension, and to accept any corrections its user offers. It must also be constantly on the watch for echoes by the user of what It says. As far as we know, none of these aspects of echoing has ever been implemented in a dialogue system.

4. Summary

In this short note, we have tried to show the importance of graceful interaction for man-machine communication, outline the scope of the problems involved in constructing gracefully interacting systems, and provide a small amount of detail about one of those problems. We believe that graceful interaction is an idea whose time has come, that it can be decomposed into a set of relatively independent components, none significantly beyond the current state of the art, and that these components can be made to perform together in a truly gracefully Interact'ng system. Details of the proposed components, including robust communication, flexible parsing, focus mechanisms, explanation facility, identification from descriptions, and generation of descriptions, can be found in [4], along with a description of the system architecture in which we propose to integrate all these components. A gracefully interacting system conforming to this architecture Is currently under implementation.

References

1. Bobrow, D. G.f Kaplan, R. M., Kay, M., Norman 9. A., Thompson, K, and Winograd, T. GUS: a Frame-Driven Dialogue System. *Artificial Intelligence 8* (1977), 155-173.

2. Codd, E. F. Seven Steps to RENDEZVOUS with the Casual User. In Klimbie, J. W. *and* Koffeman, K. L, Ed., *Proc. IF IP TC-2 Working Conf. on Data Base Management* Systems, North Holland, Amsterdam, 1974, pp. 179-200.

3. Grosz, B. J. The Representation and Use of Focus in a System for Understanding Dialogues. Proc. Fifth Int. Jt. Conf. on Artificial Intelligence, MIT, 1977, pp. 67-76.

4. Hayes, P. J., and Reddy, R. An Anatomy of Graceful Interaction in Man-Machine Communication. Tech. report. Computer Science Department, Carnegie-Mellon University, 1979.

5. Hendrix, G. G. Human Engineering for Applied Natural Language Processing. Proc. Fifth Int. Jt. Conf. on Artificial Intelligence, MIT, 1977, pp. 183-191.

6. Sidner, C. A Progress Report on the Discourse and Reference Components of PAL A. I. Memo. 468, MIT A. I. Lab., 1978.

7. Parkison, R. C, Colby, K. M., and Faught, W. S. Conversational Language Comprehension Using Integrated Pattern-Matching and Parsing. *Artificial Intelligence 9* (1977), 111-134.

8. Waltz, D. L. An English Language Question Answering System for a Large Relational Data Base. *Comm. ACM 2l, 7* (1978), 526-539.

MODELING PLANNING AS AN INCREMENTAL, OPPORTUNISTIC PROCESS*

Barbara Hayes-Roth, Frederick Hayes-Roth,
Stan Rosenschein, Stephanie Cammarata

The Rand Corporation
Santa Monica, California 90406

Planning is the process of formulating an intended course of action. In this paper we present a model of planning and describe the current version of an INTERLISP simulation of the model. We also review psychological results which confirm the model's basic assumptions for human planning behavior.

We have been studying planning--the process by which a person or a computer program formulates an intended course of action. Our goal is to develop a model of the planning process that is both computationally feasible and psychologically reasonable. Toward this end, we have found it useful to adopt many of the basic features of the Hearsay-II system [1,4, 7, 8, 9]. In this paper, we describe our model of the planning process, the current version of an INTERLISP implementation of the model, and some of the psychological research that supports it.

1. THE ERRAND-PLANNING TASK

We have focused our initial efforts on an errand-planning task. The planner begins with a list of desired errands and a map of a town in which she or he must perform the errands. The errands differ implicitly in importance and the amount of time required to perform them. The planner also has prescribed starting and finishing times and locations. Ordinarily, the available time does not permit performance of all of the errands. Given these requirements, the planner decides which errands to perform, how much time to allocate for each errand, in what order to perform the errands, and by what routes to travel between successive errands.

* The work reported here was supported by Contract No. N00014-78-C-0039 from the office of the Director of Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research. Perry Thorndyke and Doris McClure made valuable contributions to the research reported.

In performing this task, the planner makes many decisions. These decisions exploit different kinds of knowledge and address different aspects of the planned activity. The following examples illustrate the variability in decisions a planner might make:

1. I'll go to the drug store after the bank.

2. I'm going to do all of the errands in the northeast corner of town and then the errands in the southeast corner.

3. The dentist is more important than the hardware store.

4. The drugstore, the dentist, and the bank are all in the same general area.

5. I'm going to try to find an errand that is on my route to the northeast corner of town.

6. I'm going to see where the errands are on the map.

7. I'm going to avoid backtracking.

8. First I'd better decide which errands are the most important ones.

Planners can also vary considerably in the order in which they make these decisions. For example, a planner might begin by making very abstract decisions about the gross features of the plan (e.g., decision 2 above) and use these decisions to guide subsequent decisions about the details of the plan (e.g., decision 1 above). Alternatively, the planner might begin by making decisions about certain details of

the plan before deciding upon any particular gross organization for the plan. Similarly, the planner might decide upon intended actions in the order in which she or he plans to perform them. Alternatively, the planner might decide upon intended actions in some other order.

In order to accommodate the different kinds of decisions, the different kinds of knowledge they reflect, and differences in the order in which planners make them, we built our model around the following features of the Hearsay-II system: (a) multiple cooperating knowledge sources (referred to below as specialists); (b) incremental, opportunistic problem-solving behavior; (c) structured communication among knowledge sources via a blackboard; and (d) an intelligent scheduler to control knowledge source activity.

2. THE PLANNING MODEL

In our model, the planning process comprises the independent and asynchronous operation of many distinct specialists (knowledge sources). Each specialist makes tentative decisions for incorporation into a tentative plan. All specialists record their decisions in a common data structure, called the blackboard. They also establish linkages on the blackboard to reflect causal or logical relationships among various decisions. The blackboard enables the specialists to interact and communicate. Each specialist can retrieve decisions of interest from the blackboard regardless of which specialists recorded them. A specialist can combine earlier decisions with its own decisionmaking heuristics to generate new decisions.

We partition the blackboard into five planes containing conceptually different categories of decisions. Each plane contains several levels of abstraction of the planning space. Most specialists deal with information that occurs at only a few levels of particular planes. Fig. 1 shows the five planes of the blackboard and their constituent levels of abstraction. It also shows the activities of several illustrative specialists. We discuss these below.

Meta-plan decisions indicate what the planner intends to do during the planning process.

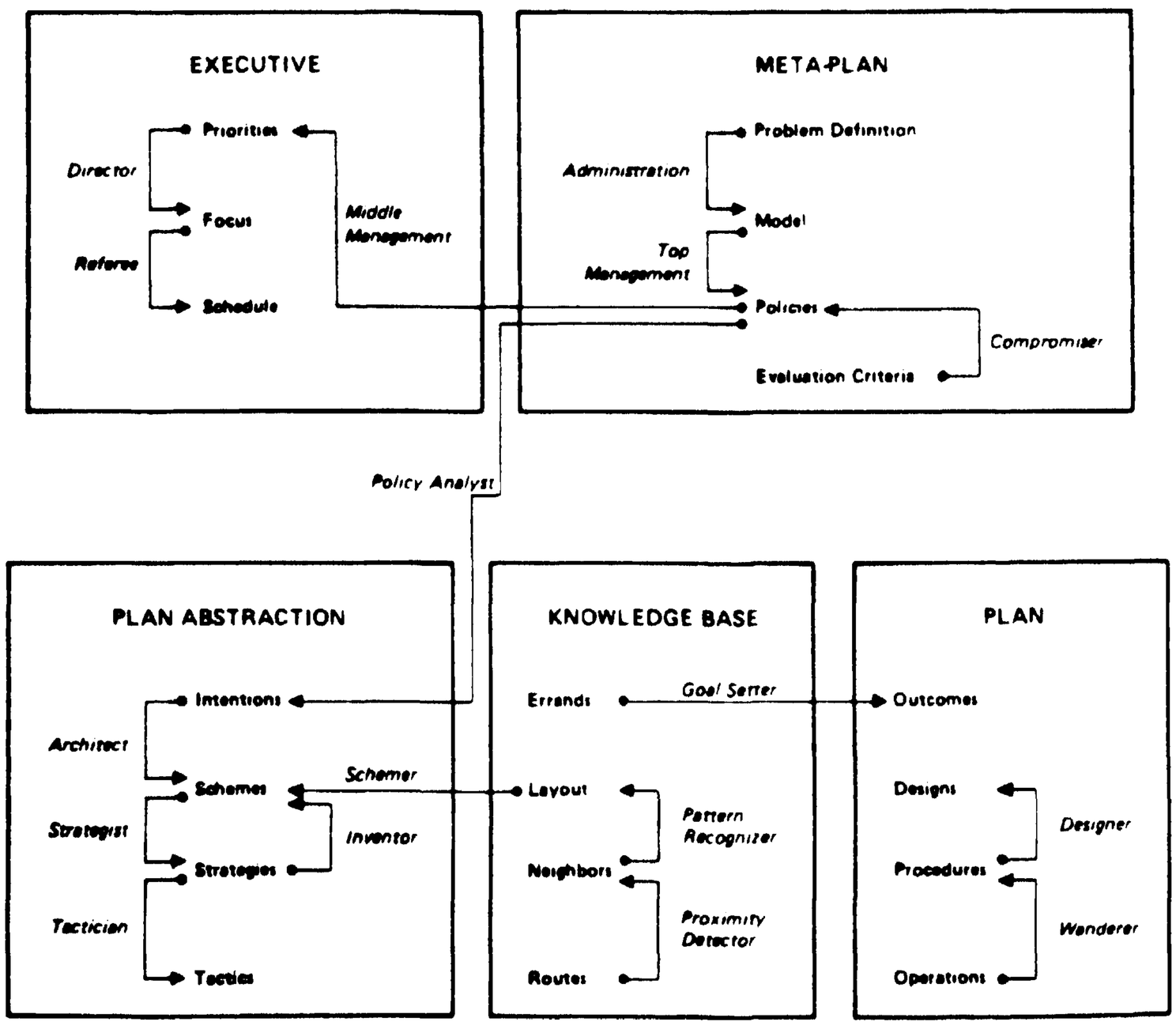


Figure 1. The planning blackboard and the actions of illustrative specialists

This plane has four levels. Beginning at the top, the problem definition describes the planner's conception of the task. It includes descriptions of the goal, available resources, possible actions, and constraints. In the errand-planning task, for example, the problem definition would include the list of errands, contextual information, and associated instructions. The problem-solving model indicates how the planner intends to represent the problem symbolically and generate potential solutions. For example, the planner might view the errand-planning task as an instance of the familiar traveling salesman problem [2], searching for the most efficient route among the errands. Alternatively, the planner might view the task as a scheduling problem, deciding which errands to perform before deciding when to perform them. Policies specify general criteria the planner wishes to impose on his problem solution. For example, the planner might decide that the plan must be efficient or that it should minimize certain risks. Solution evaluation criteria indicate how the planner intends to evaluate prospective plans. For example, the planner might decide to speculate on what could go wrong during execution and ensure that the plan is robust over those contingencies.

Plan decisions indicate actions the planner actually intends to take in the world. Decisions at the four levels form a potential hierarchy, with decisions at each level specifying a more refined plan than those at the next higher level. Beginning at the most abstract level, outcomes indicate what the planner intends to accomplish by executing the finished plan. In the errand-planning task, for example, outcomes indicate what errands the planner intends to accomplish by executing the plan. Designs characterize the general approach by which the planner intends to achieve the outcomes. For the errand-planning task, designs characterize the general route the planner intends to take to accomplish the intended errands. Procedures specify specific sequences of actions. For the errand-planning task, procedures specify sequences of errands. Operations specify sequences of more specific actions. In the errand-planning task, operations specify the route by which the planner will proceed from one errand to the next.

In addition to the levels of abstraction, the plan plane has a second dimension corresponding to the time period spanned by proposed decisions. It also permits representation of competing alternative decisions and simultaneous and event-contingent decisions.

Plan-abstraction decisions characterize desired attributes of potential plans. These abstract decisions serve as heuristic aids to the planning process suggesting potentially useful qualities of planned actions. Each level of the plan-abstraction plane characterizes types of decisions suggested for incorporation into the corresponding level of the plan plane. For example, the planner might indicate an intention to do all of the critical errands. This intention could stimulate efforts to partition the errands into critical and non-critical sets. At a lower level, the planner might generate a scheme to fabricate a design employing gross spatial clusters of errands. This scheme might motivate a search for coherent clusters. At the next level, the planner might develop a strategy suggesting that errands in the current cluster be completed before moving on to errands in another cluster. This strategy would presumably constrain procedural sequences eventually incorporated into the plan. Finally, the planner might adopt a tactic that suggested searching for a short-cut between one errand and the next. This tactic might lead to the discovery and use of one particular short-cut.

The knowledge base records observations and computations about relationships in the world which the planner generates while planning. This knowledge supports two types of planning functions: situation assessment, the analysis of the current state of affairs; and plan evaluation, the analysis of the likely consequences of hypothesized actions. Again, the levels of the knowledge base form a hierarchy and correspond to the levels of the plan and plan-abstraction planes. Each level of the knowledge base contains observations and computations useful in instantiating decisions at the corresponding level of the plan-abstraction plane or generating decisions at the corresponding level of the plan plane. Thus, the levels of the knowledge base are problem-specific. At the errand level, for example, the planner might compute the time required to perform all of the currently intended errands to evaluate the plan's gross feasibility. At the layout level, the planner might observe that several errands form a convenient spatial cluster and, as a consequence, formulate a design organized around clusters. At the neighbor level, the planner might observe that two planned errands are near one another and, as a consequence, adopt a procedural decision to sequence those two errands. At the route level, she or he might detect a previously unnoticed short-cut and then exploit it in an operation-level

decision to establish a route between two planned errands.

Before describing the executive plane of the planning blackboard, we must discuss planning specialists. Specialists generate tentative decisions for incorporation into the plan in progress. Decisions become final only after the planner has accepted an overall plan. This ordinarily requires that she or he has formulated a complete plan and determined that it satisfies solution evaluation criteria recorded on the meta-plan plane.

Most specialists work with decisions at only two levels of the blackboard. One level contains decisions (previously generated by other specialists) that stimulate the specialist's behavior. The other is the level at which the specialist records its own modifications to the blackboard. The circle and arrow ends of the arc associated with each specialist in Fig. 1 indicate these two levels, respectively. For example, the strategist (on the plan-abstraction plane) responds to prior scheme decisions by generating strategies useful in implementing those schemes. Suppose, for example, one specialist had generated a scheme to travel around among spatial clusters of errands, doing the errands in one cluster before moving on to the next. The strategist would generate a strategy for sequencing individual errands according to this scheme. One such strategy would be to perform all pending errands in the current cluster before performing errands in any other cluster.

Note that the arcs in Fig. 1 indicate that both bottom-up and top-down processing occur and that the two levels indicated by an arc need not be adjacent or even on the same plane of the planning blackboard.

We operationalize specialists as pattern-directed condition-action modules [15]. The condition component of a specialist characterizes decisions whose occurrences on the blackboard warrant a response by the specialist. The occurrence of any of these decisions invokes the specialist. For example, the occurrence of a new scheme on the plan-abstraction plane invokes the strategist. The action of a specialist module defines its behavior. For example, the strategist generates strategies for implementing schemes. In addition to recording new decisions, each specialist records relational linkages among the decisions with which it deals. For example, the strategist records support linkages connecting the scheme decision that invokes it to the strategies generated for implementing that scheme.

We have selected the specialists shown in Fig. 1 for illustrative purposes. The mnemonic names of the specialists and the preceding discussion of levels make most of the specialists self-explanatory, so we will not discuss them in detail here [but see 6 for elaboration).

During planning, each of the independent specialists monitors the blackboard for the occurrences of decisions specified in its condition. Invoked specialists queue up for execution, and an executive decides which will execute its action.

We have formalized executive decisions as the fifth plane of the blackboard. Decisions made at the three levels on this plane form a hierarchy, with decisions at each level potentially refining ones at the level above. Starting at the top, priority decisions indicate preferences for allocating processing activity to certain areas of the planning blackboard before others. For example, given a traveling salesman model, the planner might decide to determine what errand sequences he could do conveniently, rather than deciding what errands he ought to do. Focus decisions indicate what kind of decision to make at a specific point in time, given the current priorities. For example, the planner might decide to focus attention on generating an operation-level refinement of a previously generated procedure. Finally, schedule decisions indicate which of the currently invoked specialists, satisfying most of the higher-level executive decisions, to execute. If, for example, given current priorities and focus decisions, both the architect and the pattern recognizer had been invoked, the planner might decide to execute the pattern recognizer first.

Like the other planes of the planning blackboard, the executive plane includes decisions motivated by prior decisions on the same or other blackboards. For example, middle management responds to policies on the meta-plan plane by generating appropriate priorities on the executive plane. The referee uses focus decisions in deciding which of the currently invoked specialists to schedule. The executive plane differs from the other four planes of the planning blackboard because decisions recorded there do not motivate decisions recorded on other blackboards. Instead, they determine which invoked specialists can execute their

actions on their designated planes of the blackboard.

Under the control of the executive, the planning process proceeds through successive invocation and execution of the various operational specialists. The process continues until the planner has decided that the existing plan satisfies the evaluation criteria recorded on the meta-plan plane of the blackboard.

3. IMPLEMENTATION OF THE PLANNING MODEL

We have implemented a simulation of the planning model in INTERLISP. We describe the data structures, specialists, and control structure for the simulation below. We then note the main differences between the present implementation and Hearsay-II and assess the current performance of the simulation.

Data Structures. The simulation has four global data structures: the map, the blackboard, the agenda, and the event list.

The map is an internal representation of the map our human subjects use in performing the errand-planning task. It is a two-dimensional grid, with 38 cells and 30 cells on the east-west and north-south dimensions, respectively. Each cell contains a number indicating the object it represents. For example, all cells representing a particular street, store, park, or intersection have the same number. Thus, the system refers to an object on the map as the area covered by the corresponding number.

The blackboard contains all decisions generated during the planning process. Each decision appears as a node, residing at a particular level of abstraction on a particular plane of the blackboard (see above discussion). In addition, each node holds an arbitrary number of attribute-value pairs. Different nodes may have different attributes. However, all nodes have the TAG attribute which serves as a type designation. Once a node appears on the blackboard, its attributes may change, but it never disappears.

The following node might appear at the procedure level of the plan plane:

NODE N17	
PLANE	plan
LEVEL	procedure
TAG	thread
ELEMENTS	(errand (x) errand (y))
POSITION	last

This node represents a decision to create a procedure thread (an ordered sequence of errands) in which errand y follows errand x. It further specifies that this errand sequence will occur last in the plan.

The agenda contains all currently invoked specialists and complete descriptions of the nodes that triggered them. This information is used in scheduling specialists, as discussed below.

The event list provides a history of all blackboard activities. It maintains a complete description of each node creation or modification, in the order in which these changes to the blackboard occurred. We currently use the event list for tracing and debugging.

Specialists. Specialists add new nodes to the blackboard or modify the attributes of existing nodes. Each specialist has a two-part condition component and an action component, as discussed below.

The condition component of a specialist determines whether it gets invoked. It has two parts, a trigger and a test. Both are predicates which get applied to various nodes on the blackboard. They differ in complexity and time of application. A specialist gets invoked only after both its trigger and test have been satisfied.

The trigger provides a preliminary test of the specialist's relevance. Ordinarily it requires only that the focus node (the most recently added or modified node on the blackboard) reside at a particular level of the blackboard and that it have a particular TAG. The system tests all specialists' triggers for each new focus node. It adds to the agenda each specialist whose trigger has been satisfied.

The test specifies all additional prerequisites for the applicability of the specialist. It may require that the focus node have particular attributes or particular values of attributes. It may require the existence of a specific configuration of decisions on the blackboard. The system performs tests only for specialists on the agenda.

The action component of a specialist defines the modification it makes to the blackboard when executed. The actions of most specialists produce new nodes with particular attributes at particular levels of the blackboard. A few simply modify attributes of existing nodes.

The cluster recognizer illustrates the specialists in our simulation. It notices clusters of errands in the same geographic neighborhood. The trigger for the cluster recognizer requires that a node whose TAG is "location" should appear at the neighbors level of the knowledge base. Such a node indicates that the simulation has located a particular errand on the map. The cluster recognizer is relevant in this context. The test requires that two other nodes whose TAGs are "location" should also appear at the neighbors level of the knowledge base. It also requires that all three nodes have a common value (NE, NW, SE, or SW) of the attribute REGION. Satisfying both the trigger and the test of the cluster recognizer indicates that three errands are in the same neighborhood--i.e., a cluster exists. The cluster detector's action records a new node whose TAG is "cluster" at the layout level of the knowledge base. It also records MEMBERS and REGION attributes whose values are the names of the errands in the cluster and the region of the cluster, respectively.

Control Structure. Like Hearsay-II, our simulation is event-driven. On each cycle, the current focus node triggers some number of specialists, which the system adds to the agenda. At this point, the agenda contains relevant specialists whose actions the system might be able to execute. The system processes these pending specialists in three phases: invocation, scheduling, and execution.

During the invocation phase, the system evaluates the test of all specialists on the agenda. Specialists whose tests have been satisfied are invoked. If there are no invoked specialists, the simulation terminates. If there is exactly one invoked specialist, the system executes that specialist's action. In general, however, there will be several invoked specialists and the system will have to schedule these specialists for execution.

During the scheduling phase, the system recommends one of the invoked specialists for immediate execution. It currently bases this recommendation on two considerations: recency of invocation and the current focus decision. Other things being equal, the system will recommend a recently invoked specialist in favor of one invoked earlier in the planning process. Similarly, the system will recommend a specialist whose action would occur in an area of the blackboard currently in focus, in favor of one whose action would occur elsewhere. (Recall that decisions at the focus level of the executive plane designate areas of the blackboard as in focus.) If more than one specialist satisfies either of these criteria, the system chooses one of them at random. (The other specialists remain on the agenda for possible scheduling and execution on subsequent cycles.)

During the execution phase, the system executes the action of the scheduled specialist, adding a new node or modifying an existing node on the blackboard. The system immediately evaluates the trigger of each specialist against the new focus node and adds those specialists whose triggers are satisfied to the agenda. At this point, the agenda contains all of the newly triggered specialists along with any previously triggered but unexecuted specialists. Then the next cycle begins with the invocation phase, and so forth.

Major Departures from the Hearsay-II Framework. Our simulation differs from Hearsay-II in several ways. Obviously, the planning model embodies different specialists (knowledge sources) and different blackboard partitions. Our specialists are much more molecular than the Hearsay-II knowledge sources. While Hearsay-II comprised about ten very powerful knowledge sources, our model will eventually comprise about fifty much simpler specialists. In addition, we have enumerated a much larger number of levels for the planning blackboard than Hearsay-II used for speech understanding, and we have found it useful to group these levels in conceptual planes [see also 3]. The proposed model's most important departure from the Hearsay-II framework lies in its elaboration of executive decisionmaking. The model treats executive decisionmaking as it treats other kinds of decisionmaking within the planning process. Thus, it permits a potential hierarchy of executive decisions, each recorded by an independent specialist [see also 6 and 11].

Performance of the Simulation. Our main purpose in creating this simulation is to test the sufficiency of the planning model as a psychological theory. Toward this end, we wish to use the simulation to replicate a thinking aloud protocol [10] produced by a typical subject while performing the errand-planning task. In its current form (with about thirty specialists), the simulation can produce the exact sequence of decisions in the first half of a 2000-word protocol. We expect to be able to replicate the complete protocol with the addition of about twenty more specialists to our operational set. We will then attempt to replicate other protocols produced by other subjects for other versions of the errand-planning task.

We also want an experimental environment for evaluating different planning strategies. Accordingly, the simulation permits the user to override the executive and directly control the scheduling of invoked knowledge sources for execution. Thus, while the simulation can reproduce the exact sequence of decisions in the protocol, it can also produce other sensible decision sequences. We intend to evaluate the differences in decision sequences and resulting plans under alternative executive decisions.

4. PSYCHOLOGICAL SUPPORT FOR THE PLANNING MODEL

We have collected a variety of data which suggest that the proposed model provides a reasonable description of human planning. We summarize these data below.

General Features of Planning Behavior. We have collected thirty thinking-aloud protocols from subjects performing the errand-planning task. These protocols exhibit statements from each of the levels of abstraction of each of the five planes of the blackboard. In addition, these protocols exhibit decision sequences which do not conform to any obvious systematic pattern. Instead, the decision sequences appear fairly opportunistic--each decision is motivated by one or two immediately preceding decisions, rather than by some high-level executive program. Thus, the general features of these protocols confirm the basic assumptions of the model [see 5 for additional evidence).

Details of Planning Behavior. As discussed above, our simulation can replicate the thinking aloud protocol of one of our subjects. The protocol we chose to replicate is one of the most complex of the thirty we collected. It includes decisions at each level of abstraction on each of the five planes of the blackboard. It includes instances of both top-down and bottom-up decision sequences. It includes a considerable amount of opportunism. The ability of the simulation to replicate this protocol demonstrates the sufficiency of the model to account for these features of planning behavior as well as for the other more general features.

Levels of Abstraction. The model assumes that people make decisions at different levels of abstraction and that the levels of abstraction have functional significance in the planning process. This assumption implies that theoretically naive subjects should recognize that various decisions made during planning represent particular levels of abstraction. In order to test this hypothesis, we drew statements from the thinking-aloud protocols described above and presented them in a random order to a second group of subjects. We asked them to group statements that communicated similar kinds of information. These subjects reliably grouped the statements to correspond to the postulated levels of abstraction.

Multi-Directional Processing. The model assumes that decisions at a given level of abstraction can influence subsequent decisions at either higher or lower levels of abstraction. We tested this assumption by effectively placing subjects in the middle of the planning process and examining their choices of subsequent decisions. We gave subjects errand-planning problems, required them to make particular prior decisions and asked them to choose one of two alternative subsequent decisions. By carefully specifying required prior decisions, we could predict which subsequent decision a subject would choose. The manipulation had comparable effects on subjects' choices regardless of whether the subsequent decisions were at higher or lower levels of abstraction than the prior decisions.

Alternative Executive Decisions. The model assumes that subjects can make different executive decisions and that these decisions determine the order in which other kinds of decisions occur. For example, subjects can treat the errand-planning task as a scheduling problem or a traveling salesman problem. The former constitutes a roughly top-down approach to the task, while the latter constitutes a roughly bottom-up approach. In addition to the differences in decision order, these different approaches should introduce differences in the plans subjects form. The traveling salesman approach should produce plans for performing all of the desired errands. The scheduling approach should reduce the number of planned errands, preserving only the most important errands.

We have been able to induce subjects to take these alternative approaches to the errand-planning task with three different methods. In one experiment, we gave subjects explicit instructions to use one or the other approach. Most subjects followed the instructions successfully and produced plans with the expected characteristics. In another experiment we instructed subjects to adopt one or the other approach on several priming tasks and then gave them a transfer task with no

instructions. In this situation, most subjects adopted the approach they used on the priming tasks. In a third experiment, we instructed subjects to use each approach on some of the priming tasks and then gave them various transfer tasks with no instructions. Most of these subjects adopted the traveling salesman approach on the transfer task. However, some subjects discriminated transfer tasks for which the scheduling approach was more appropriate (tasks with time limitations) and adopted it instead.

5. CONCLUSIONS

As discussed above, our primary goal is to develop a computationally feasible and psychologically reasonable model of planning. We believe that the current performance of our simulation and the empirical results reported above provide good support for the proposed model. Our future work will focus on experiments with the simulation to evaluate its generality over specific planning tasks and planning strategies. We will also conduct additional psychological experiments to evaluate predictions derived from the simulation.

Our success in modeling planning also attests to the utility of the Hearsay-II framework as a general model of cognition. Several researchers have adapted the Hearsay-II framework to a variety of tasks, including image understanding [12), reading comprehension [13), protein-crystallographic analysis [11), and inductive inference [14). Note, however, that all of these tasks are interpretation problems: problems which present the individual (or computer system) with the lowest level representation of the problem content (e.g., the speech signal) and require interpretation of the highest level representation (e.g., the meaning). Our application of the Hearsay-II framework to planning takes it into a qualitatively different task domain--generation problems: problems which present the highest level representation (e.g., the goal) and require generation of the lowest level representation (e.g., the sequence of intended actions).

Interpretation and generation problems differ in important ways. For example, interpretation problems lend themselves well to initial bottom-up strategies, while generation problems lend themselves well to initial top-down strategies. Interpretation problems generally permit only one (or a small number) of solutions, while generation problems permit an arbitrary number of different solutions. Further, interpretation problems typically have correct solutions, while the correctness of solutions to generation problems varies under different evaluation criteria. Despite these differences, the Hearsay-II framework appears robust enough to guide solution of both interpretation and generation problems.

REFERENCES

[I) CMU Computer Science Research Group. "Summary of the CMU Five-Year ARPA Effort in Speech Understanding Research." Technical Report, Carnegie-Mellon University, Pittsburgh, Pa., 1977.

[2] Christophides, N. Graph Theory: An Algorithmic Approach. New York: Academic Press, 1975.

[3] Engelmore, R. S., & Nil, H. P. "A Knowledge-Based System for the Interpretation of Protein X-Ray Crystallographic Data." Report No. STAN-CS-77-589, Stanford University, 1977.

[4) Erman, L. D., & Lesser, V. R. "A Multi-Level Organization for Problem Solving Using Many Diverse Cooperating Sources of Knowledge." Proc. IJCAI-75. Tbilisi, USSR, 1975, pp. 483-490.

[5) Feitelson, J., & Stefik, M. "A Case Study of the Reasoning in a Genetics Experiment." Heuristic Programming Project, Working Paper 77-18, Department of Computer Science, Stanford University, April 1977.

[6) Hayes-Roth, B., & Hayes-Roth, F. "Cognitive Processes in Planning." R-2366-ONR, The Rand Corporation, Santa Monica, Calif., 1978.

[7) Hayes-Roth, F., & Lesser, V. R. "Focus of Attention in the Hearsay-II Speech Understanding System." Proc. IJCAI-77. Boston, Mass., 1977, pp. 27-35.

[8) Lesser, V. R., & Erman, L. D. "A Retrospective View of the Hearsay-II Architecture." Proc. IJCAI-77. Boston, Mass., 1977, pp. 790-800.

[9] Lesser, V. R., Fennell, R. D., Erman, L. D., & Reddy, D. R. "Organization of the Hearsay-II Speech Understanding System." IEEE Transactions on Acoustics, Speech and Signal Processing, ASSP-23, January 1975, pp. 11-23.

[10] Newell, A., & Simon, H. A. Human Problem Solving. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

[11] Nii, H. P., & Feigenbaum, E. A. "Rule-Based Understanding of Signals." In D. A. Waterman and F. Hayes-Roth (Eds.), Pattern-Directed Inference Systems. New York: Academic Press, 1978.

[12] Prager, J., Nagin, P., Kohler, R., Hanson, A., & Riseman, E. "Segmentation Processes in the VISIONS System." Proc. IJCAI-/7. Boston, Mass., 1977.

[13] Rumelhart, D. E. "Toward an Interactive Model of Reading." Technical Report 56, Center for Human Information Processing, University of California at San Diego, La Jolla, 1976.

[14] Soloway, E. M., & Riseman, E. M. "Knowledge-Directed Learning." Proc. Workshop on Pattern-Pirected Inference Systems, special edition of the SIGART Newsletter, Association for Computing Machinery, New York, slune 1977.

[15] Waterman, D. A., & Hayes-Roth, F. Pattern-Directed Inference Systems. New York: Academic Press, 1978.

A Prover for Parallel Processes

Masahiro Hirata
Departmemt of Mathematics
University of Tsukuba
Sakura-mura Niihari-gun
Ibaraki-ken 300-31, JAPAN

Toshio Nishimura
Department of Mathematics
University of Tsukuba
Sakura-mura Niihari-gun
Tbaraki-ken 300-31, JAPAN

We give an automatic prover for verifying logical properties of parallel processes. The prover bases on a subsystem of the system given in [8]. We mechanize it so that a proof-tree is automatically constructed. The prover reduces a parallel program into possible serial ones by appLying rules of inference 'interruption' etc.. This paper includes some examples processed by the prover.

1. INTRODUCTION

The need for proofs of properties of parallel program is very greater. A number of research results for parallel program have been given. [1],[2],[3],[4],[5],[8],[9],[10] In parallel processes, several processes which communicate each other run on several processors. So parallel processes cause complicated states. However, as shown in [8], we can consider them the compositions of indivisible operations. So, by decomposition of parallel programs, we can analyze their movements and effects on states. Syntactic analyses of this type are powerful for those of properties of parallel programs. Our prover processes these analyses.

Since the formal system given in [8] includes infinitely long expressions such as F* (the abbreviation of $F^0 \vee F^1 \vee F^2 \vee \ldots$), we mechanize its subsystem which contains decomposition of F* with respect to 'interruption (2.3.4)' but does not contain the rules of inference which decompose G(F*) into $G(F^0), G(F), \ldots, G(F^n), \ldots$.(cf. [8] 3.1 (i) *-left and *-right) We implement it by the KLISP[6] system with about 12K free cells on PDP11/55 computer in a similar way to [7].

We roughly explain our basic idea in the simplest cases. In our system, we introduce the operator F;G (G is executed after F), but we do not introduce the 'clock'. Let f, g and h be indivisible operations. f and g, sometimes, run in parallel on different processors (denoted by f//g), but f//g causes the same result as f;g or g;f (denoted by $(f;g) \vee (g;f)$). For example, (f;g)//h or (f;g)*//h causes the same result as $(h;f;g) \vee (f;h;g) \vee (f;g;h)$ or $(f;g)^*;h;(f;g)^* \vee (f;g)^*;f;h;g;(f;g)^*$, respectively. The general form of a rule of inference will be given in 2.3.4 below. In such a way, we could reduce parallel processes into possible serial ones.

In our system, we prove the equivalence of parallel programs in the form of sequent. In order to show the equivalence between two programs F and G, our prover proves two sequents $F \rightarrow G$ and $G \rightarrow F$. For example, let a process be able to enter the critical section C if it satisfies the condition Q. It can not enter C eternally, if it satisfies 'not Q' (denoted by $\neg Q$) eternally (denoted by $(\neg Q)^\infty$). So, in order to show that a parallel program P is deadlock free, it is sufficient to show that P is equivalent to a program E, where E does not contain infinite loop such as X^∞. And, a parallel program P which contains two critical sections CS1 and CS2 are mutually exclusive, if P is equivalent to a program E, where E does not contain any parallel run of CS1 and CS2 such as CS1//CS2.

2. IMPLEMENTED FORMAL SYSTEM

2.1. An expression and an indivisible expression.

The symbols used in the propositional calculus have the usual meanings for logical formulas (e.g. $\vee$, $\wedge$, $\supset$, $\equiv$ or $\neg$ represents 'or', 'and', 'implies', 'equivalent' or 'not' respectively). And we introduce symbols of expression-constants . Especially, the expression-constant "I" plays the role of the identity function and means the truth value "true" for a formula. And "$\mathbf{F}$" represents "impossible", and the truth value "false" for a formula. A formula in the propositional calculus, extended by adding the clause

"if A and B are formulas, so are A;B and A//B", is an expression. An expression-constant is an expression. A figure of the form x:=E is an expression, called an assignment statement to x, where x is a variable and E is a term in the predicate calculus. If F and G are expressions, so are F//G, F∨G and F;G. If F is an expression, so are F^m, F^ω and F*, where m is a variable for integers. If F is an expression, then [F] is an indivisible expression. An indivisible expression is an expression. Only those obtained in the above are expressions.

We briefly explain the intuitive meaning of the above definition. An expression means an instruction, a decision, a series of them or a formula in the propositional calculus. And a decision is denoted by a formula. F;G means that G is executed after the execution of F. F//G represents that F and G are executed in parallel. If F and G are formulas, F;G, F//G and F∧G have the same meaning as "F and G" in the propositional calculus. An expression F∨G represents that at least one of F or G is executed. If F and G are formulas, F∨G has the same meaning as "F or G". An expression F* means that F will be executed n-times for an arbitrary n. F^ω represents the infinite loop of F.

Generally, an expression F is of the form $F_1;F_2;\ldots;F_n$, where every F_i is an indivisible expression, an expression-constant or an expression of the form ¬H, H∨K, H//K, H^m, H* or H^ω. This is called the canonical form of F.

In the prover, F;G, F∧G, F∨G, F//G, F⊃G, F≡G, ¬F, F^n, F*, F^ω or [F] is denoted by (CONC F G), (*AND F G), (*OR F G), (PAR F G), (*IMPLIES F G), (*EQUIV F G), (*NOT F), (LABEL F n), (* F), (INF F) or (IND F), respectively.

2.2 A sequent, an axiom and a proof

A figure of the form

$$A_1,\ldots,A_m,\ E\longrightarrow E_1,\ldots,E_n$$

is called a sequent. This means that, under the assumptions $A_1,\ldots,A_m$, the results of the execution of E are included in the results of those of $E_1,\ldots,E_{n-1}$ and E_n. Sometimes we denote a finite series of expressions $E_1,\ldots,E_n$ by a Greek capital letter Γ, Δ etc.. Axioms are sequents of the form $\Phi\longrightarrow\Delta$, where Φ is a special expression-constant, or $\Gamma\longrightarrow\Delta$, where $\Gamma\cap\Delta\neq\phi$. A proof-tree is the figure of a tree-form which satisfies the following conditions.

(1) The uppermost sequent is an arbitrary sequent.
(2) Every lowermost sequent is an axiom.
(3) The upper sequent and the lower sequents are connected by a rule of inference given in the paragraph 2.3 below.

Let S be a sequent. Then a proof-tree, which has S as the uppermost sequent, is called the proof-tree to S. If we have a proof-tree to S, we say that S is provable.

Let $A_1,\ldots,A_h$ and $\neg B_1,\ldots,\neg B_k$ be assumptions. For the prover, a sequent

$$A_1,\ldots,A_h,\neg B_1,\ldots,\neg B_k,\ E\longrightarrow E_1,\ldots,E_n$$

is divided into two parts. One is the set of assumptions given in the form $(A_1$ I.1)... $(A_h$ I.h) $(B_1$ Φ.h+1)...$(B_k$ Φ.h+k). Other is $E\longrightarrow E_1,\ldots,E_n$ and this is given in the form (*ARROW (E) (E_1 ... E_n)).

2.3 Rules of Inference

Rules of inference of the subsystem are divided into the following parts.

2.3.1 Rules of inference for the propositional calculus.

We omit those here. (cf. [7])

2.3.2 Rules of inference with respect to ∨

Let F(e) be an arbitrary expression which contains the indicated expression-variable e in at most one place in F(e) except for the inside of any subexpression of the form K*, K^ω or ¬L.

∨-left Given a sequent $F(B\vee C)\longrightarrow\Delta$, we can give two sequents $F(B)\longrightarrow\Delta$ and $F(C)\longrightarrow\Delta$.

∨-right Given a sequent $\Gamma\longrightarrow\Delta,F(B\vee C)$, we can give the sequent $\Gamma\longrightarrow\Delta,F(B),F(C)$.

2.3.3 Replacement of expression

(i) Replacement by assumption

Let F(e) be an arbitrary expression. Here the indicated e may be contained in several parts of F(e). Let a sequent $F(A)\longrightarrow G(A)$, be given, where A is a formula. Then, according that the set of assumptions contains A or ¬A, we can give the sequent $F(I)\longrightarrow G(I)$, or $F(\Phi)\longrightarrow G(\Phi)$, , respectively.

(ii) Inherent replacement

In our system, the followings are built in as

schemes of assumptions.

$G^0=I$, $\neg\Phi=I$, $\Phi^*=I$, $I^*=I$, $I//G=G$, $G//I=G$, $I;G=G$, $G;I=G$, $\Phi\vee G=G$, $G\vee\Phi=G$, $\Phi^\infty=\Phi$, $\Phi;G=\Phi$, $\Phi//G=\Phi$, $G//\Phi=\Phi$, $\neg I=\Phi$,
if G does not contain an expression of the form K^∞, $G;\Phi=\Phi$,
$K^\infty;H=K$, $(K^\infty//G);H=K^\infty//G$, $(G//K^\infty);H=G//K^\infty$,
$(G^*)^\infty=(G^*;I^\infty)\vee G^\infty$ and $G^m;G^\infty=G^\infty$.

Given a sequent, every figure of the same form as the left-hand side of '=' in the sequent is replaced by the right-hand side by matching.

We roughly explain the meanings of some of them. Let the following parallel program be given:

perbegin P1: begin L1: if Q then goto L1; S end;
P2: begin G end
parend.

In P1, one of the following (i) and (ii) occurs: (i) Q is satisfied finite times and after that $\neg Q$ is satisfied and then S is executed (denoted by $Q^*;\neg Q;S$) and (ii) Q is satisfied eternally (denoted by Q^∞). So in P1//P2, either $(Q^*;\neg Q;S)//G$ or $Q^\infty//G$ is executed. If $(Q^*;\neg Q;S)//G$ is executed, Q is satisfied finite times (i.e. Φ^0 or I^*), the value of Q finally becomes Φ. So the value of Φ^0 or I^* is I. And then the infinite loop Q^∞ (i.e. Φ^∞) is impossible, i.e. the value of Φ^∞ is Φ. Since (i) is executed, the execution of $Q^\infty//G$ is impossible, i.e. the value of $\Phi//G$ is Φ. If $Q^\infty//G$ is executed, then the value of $\neg Q$ is Φ and the execution of $Q^*;\neg Q;S$ is impossible, i.e. the value of $I^*;\Phi;S$ $(=\Phi;S)$ is Φ. Since K^∞ is the infinite loop of K, everything after K^∞ is unable to be executed. So the value of $K^\infty;H$ is K^∞. In $K^\infty//G$, G may eternally wait for the completion of K^∞ after its completion. Then the value of $(K^\infty//G);H$ is $K^\infty//G$.

2.3.4 Interruption

In order to describe the rule of inference for interruption, we define two ordered sets comp(E) and lete(E) for an arbitrary expression E, which does not contain //, as follows.

If F is an indivisible expression or an expression-constant, then

comp(F) = (I,F) and lete (F) = (F,I).

Let F and G be arbitrary expressions, where $comp(F)=(F_{10},\ldots,F_{1m})$, $lete(F)=(F_{20},\ldots,F_{2m})$, $comp(G)=(G_{10},\ldots,G_{1n})$ and $lete(G)=(G_{20},\ldots,G_{2n})$.

Then

$comp(F;G)=(F_{10},\ldots,F_{1m},F;G_{11},\ldots,F;G_{1n})$,

$lete(F;G)=(F_{20};G,\ldots,F_{2m};G,G_{21},\ldots,G_{2n})$,

$comp(F\vee G)=(I,F_{10},\ldots,F_{1m},G_{10},\ldots,G_{1n},F\vee G)$,

$lete(F\vee G)=(F\vee G,F_{20},\ldots,F_{2m},G_{20},\ldots,G_{2n},I)$,

$comp(F^*)=(I,F^*;F_{10},\ldots,F^*;F_{1m})$,

$lete(F^*)=(F^*,F_{20};F^*,\ldots,F_{2m};F^*)$,

$comp(F^\infty)=(I,F^*;F_{10},\ldots,F^*;F_{1m})$,

$lete(F^\infty)=(F^\infty,F_{20};F^\infty,\ldots,F_{2m};F^\infty)$.

Let F//G or G//F be a subexpression in a sequent, where G does not contain //. Let an expression $F_1;F_2;\ldots;F_m$ be the canonical form of F. Let F_i be the leftmost indivisible expression. Denoting $F_1;\ldots;F_{i-1}$ or $F_{i+1};\ldots;F_m$ by H_1 or H_2 respectively, F is of the form $H_1;F_i;H_2$. When $comp(G)=(G_{10},\ldots,G_{1m})$ and $lete(G)=(G_{20},\ldots,G_{2m})$, the indicated F//G is replaced by $(H_1//G_{10});F_i;(H_2//G_{20})\vee(H_1//G_{12});F_i;(H_2//G_{22})\vee\ldots\vee(H_1//G_{1m});F_i;(H_2//G_{2m})$. The case of G//F is similar. In the actual programming, the sets comp(G) and lete(G) are not constructed, but it is processed in procedural way. (cf. 3.(6), (8))

2.3.5 Execution

(i) Execution of assignment statement

By G' we denote the result obtained by executing x:=E for G. Then G' is obtained as follows: Let $G_1;G_2;\ldots;G_m$ be the canonical form of an expression G, and G_j the leftmost expression which contains an assignment statement to x. Then $G_1',G_2',\ldots,G_{j-1}'$ are the results obtained by replacing E for x in $G_1,G_2,\ldots,G_{j-1}$ respectively. If G_j is of the form x:=K, then G_j' is x:=K', where K' is the result obtained by replacing E for x in K. If K does not contain x, then G_j' is G_j.

Let F be an arbitrary expression and $F_1;F_2;\ldots;F_m$ its canonical form. And let F_i be the leftmost assignment statement of the form x:=E where E does not contain x, and let F_j $(i<j)$ be the next leftmost expression which contains an assignment statement to x. Then the result obtained by execution of F_i is $F_1;\ldots;F_i;F_{i+1}';\ldots;F_j';F_{j+1},\ldots,F_m$.

(ii) Elimination of assignment statement

Let F and $F_1;F_2;\ldots;F_m$ be the same as the previ-

ous one. And let F_i be the leftmost assignment statement to x. If we have F_j such that F_j is an assignment statement to x and $F_{i+1};\ldots;F_{j-1}$ does not contain the variable x, then F is replaced by $F_1;\ldots;F_{i-1};F_{i+1};\ldots;F_j;\ldots;F_m$.

3. ALGORITHM

Given an arbitrary sequent $E \longrightarrow E_1,\ldots,E_m$ and a set of assumptions, the prover constructs the proof-tree to the sequent in the following order.

(1) Whether the given sequent is an axiom or not, is checked. If the left-hand side of the sequent is Φ, it is an axiom. If an expression in the left-hand side occurs in the right-hand side, it is also an axiom. When we have no remaining sequent, the proof is completed, and the obtained proof-tree is printed out. If we have a remaining sequent, the prover has the last one of them as the given one, and goes to (1). If it is not an axiom, then the prover goes to (2).

(2) The prover executes assignment statements contained in E, $E_1,\ldots,E_m$, if any, from left to right and the control goes to (1). Otherwise it goes to (3).

(3) If a rule of inference 'inherent replacement (2.3.3 (ii))' is applicable to the given sequent, then the prover executes it, and the control goes to (1). Otherwise, it goes to (4).

(4) If a rule of inference 'replacement by assumption (2.3.3 (i))' is applicable to the given sequent, then the prover executes it, and the control goes to (1). Otherwise, it goes to (5).

(5) If a rule of inference '$\vee$-left' is applicable to the given sequent, then the prover executes it. Let the given one be $F(B \vee C) \longrightarrow \Delta$, where the indicated $B \vee C$ is not in any inside of $\neg$, * or $^{\infty}$. Then for the next step, $F(B) \longrightarrow \Delta$ is the given sequent and $F(C) \longrightarrow \Delta$ is the last remaining one. And the control goes to (1). Otherwise, it goes to (6).

(6) If a rule of inference 'interruption' is applicable to the left-hand side of the given sequent, then the prover executes it. Let the given one be $F(G//H) \longrightarrow \Delta$, where the indicated G//H is not in any inside of $\neg$, * or Φ. And let the result be $F(A_1 \vee \ldots \vee A_n) \longrightarrow \Delta$. Then the prover constructs the following n sequents:

$F(A_1) \longrightarrow \Delta$(the given sequent for the next step),
$F(A_2) \longrightarrow \Delta$(the last remaining sequent),
$F(A_3) \longrightarrow \Delta, \ldots, F(A_n) \longrightarrow \Delta$,
and the control goes to (1). If an interruption is not applicable to the left-hand side, then the control goes to (7).

(7) If a rule of inference '$\vee$-right' is applicable to the sequent, then the prover executes it. And the control goes to (1). Otherwise goes to (8).

(8) If a rule of inference 'interruption' is applicable to the right-hand side of the given sequent, then prover executes it. Let the given one be $\Gamma \longrightarrow \Delta_1, F(G//H), \Delta_2$, where the indicated G//H is not in any inside of $\neg$, * or $^{\infty}$. And let the result be $\Gamma \longrightarrow \Delta_1, F(A_1 \vee \ldots \vee A_n), \Delta_2$. Then the prover constructs the sequent $\Gamma \longrightarrow \Delta_1, F(A_1), \ldots, F(A_n), \Delta_2$, and the control goes to (1). If an interruption is not applicable to the right-hand side, then it goes to (9).

(9) We fail the proof. Then we can select one of the following two modes. One, we stop the prover and print out the proof-tree obtained so far. Two, we return the control to the last remaining sequent, and continue to construct the proof-tree. If we have no remaining sequent, the proof-tree obtained so far is print out.

In our prover, we can change the order of (2),...,(8) by presenting the table of the order of execution in advance.

4. EXAMPLES

4.1 An example of mutually exclusive but deadlocked program

E.W.Dijkstra gave an example of a program which satisfies the mutual exclusion but causes the "deadlock"[1]. His program is the following:

```
begin integer C1, C2; C1:=1; C2:=1;
  parbegin
    process 1: begin A1: C1:=0;
                  L1: if C2=0 then goto L1;
                  critical section 1;  C1:=1;
                  remainder of cycle 1; goto A1

               end;
    process 2: begin A2: C2:=0;
                  L2: if C1=0 then goto L2;
                  critical section 2;  C2:=1;
```

remainder of cycle 2; goto A_2
end
parend
end

Let P_1 or P_2 be $C_1:=0;(C_2\neq 0 \vee (C_2=0)^{\infty});CS_1;C_1:=1$ or $C_2:=0;(C_1\neq 0 \vee (C_1=0)^{\infty});CS_2;C_2:=1$, respectively. Then the program except for the last "goto A_1" or "goto A_2" can be expressed by the following:

$C_1:=1;\ C_2:=1;\ (P_1//P_2)$,

provided that the program does not halt. By Δ, we denote the following:

$CS_1;C_1:=1;CS_2;C_2:=1$, $CS_2;C_2:=1;CS_1;C_1:=1$;

$C_1:=0;C_2:=0;(I^{\infty}//I^{\infty})$, $C_2:=0;C_1:=0;(I^{\infty}//I^{\infty})$.

Then the sequent which we want to prove is

$C_1:=1;C_2:=1;(P_1//P_2)\longrightarrow\Delta$,

with assumptions 0=0, 1=1, 1≠0, 0≠1.
Because it is not an axiom and the replacement of assumptions can not be applied, the prover applies $\vee$-left, and we have

(1) $C_1:=1;C_2:=1;((C_1:=0;C_2\neq 0;CS_1;C_1:=1)$
$//\ (C_2:=0;(C_1\neq 0\vee(C_1=0)^{\infty});CS_2;C_2:=1))\longrightarrow\Delta$

and

(2) $C_1:=1;C_2:=1;((C_1:=0;(C_2=0)^{\infty};CS_1;C_1:=1)$
$//\ (C_2:=0;(C_1\neq 0\vee(C_1=0)^{\infty});CS_2;C_2:=1))\longrightarrow\Delta$

From (1), applying $\vee$-left, we have

(3) $C_1:=1;C_2=1;((C_1:=0;C_2\neq 0;CS_1;C_1:=1)$
$//\ (C_2:=0;C_1\neq 0;CS_2;C_2:=1))\longrightarrow\Delta$,

and

$C_1:=1;C_2:=1;((C_1:=0;C_2\neq 0;CS_1;C_1:=1)$
$//\ (C_2:=0;(C_1=0)^{\infty};CS_2;C_2:=1))\longrightarrow\Delta$.

The prover applies interruption to the left-hand side of (3) for $C_1:=0$, so we have

(4) $C_1:=1;C_2:=1;(I//I);C_1:=0;((C_2\neq 0;CS_1;C_1:=1)$
$//\ (C_2:=0;C_1\neq 0;CS_2;C_2:=1))\longrightarrow\Delta$,

...,

$C_1:=1;C_2:=1;(I//(C_2:=0;C_1\neq 0;CS_2;C_2:=1));C_1:=0;$
$((C_2\neq 0;CS_1;C_1:=1)\ //\ I)\longrightarrow\Delta$.

Applying to (4) elimination of assignment($C_1:=1$) and replacement by assumption(G//I=G and G;I=G), we have

(5) $C_2:=1;C_1:=0;((C\neq 0;CS_1;C_1:=1)$
$//\ (C_2:=0;C_1\neq 0;CS_2;C_2:=1))\longrightarrow\Delta$.

By the application of interruption to the left-hand side of (5) for $C_2\neq 0$, we have

(6) $C_2:=1;C_1:=0;(I//I);C_2\neq 0;((CS_1;C_1:=1)$
$//\ (C_2:=0;C_1\neq 0;CS_2;C_2:=1))\longrightarrow\Delta$,

...,

$C_1:=1;C_2:=1;(I//(C_2:=0;C_1\neq 0;CS_2;C_2:=1));$
$C_2\neq 0;((CS_1;C_1:=1)//I)\longrightarrow\Delta$.

Applying to (6) replacement by assumption(G//I=G and G;I=G), execution of assignment((C≠0)=(1≠0)) and replacement by assumptions((1≠0)=I and G;I=G), we have

(7) $C_2:=1;C_1:=0;((CS_1;C_1:=1)$
$//\ (C_2:=0;C_1\neq 0;CS_2;C_2:=1))\longrightarrow\Delta$.

Applying interruption to the left-hand side of (7) for $C_1:=1$, we have

(8) $C_2:=1;C_1:=0;(CS_1//I);C_1:=1;$
$(I//(C_2:=0;C_1\neq 0;CS_2;C_2:=1))\longrightarrow\Delta$,

(9) $C_2:=1;C_1:=0;(CS_1//C_2;=0);C_1:=1;$
$(I//(C_1\neq 0;CS_2;C_2:=1))\longrightarrow\Delta$,

....

Applying execution and elimination of assignment, replacement of assumption to (8), we have

(10) $CS_1;C_1:=1;CS_2;C_2:=1\longrightarrow\Delta$.

This is an axiom. Then the prover stores the sequents (8) and (10) in the disk storage and returns to (9). Repeating these operations, at length, the control returns to the decomposition of the sequent (which is one of the results of $\vee$-left to (2))

(11) $C_1:=1;C_2:=1;((C_1:=0;(C_2=0)^{\infty};CS_1;C_1:=1)$
$//\ (C_2:=0;(C_1=0)^{\infty};CS_2;C_2:=1))\longrightarrow\Delta$.

Applying replacement by assumptions($K^{\infty};H=K^{\infty}$) (11) becomes

(12) $C_1:=1;C_2:=1;((C_1:=0;(C_2=0)^{\infty})$
$//\ (C_2:=0;(C_1=0)^{\infty}))\longrightarrow\Delta$.

By interruption for $C_1:=0$, we have

$C_1:=1;C_2:=1;(I//I);C_1:=0;$
$((C_2=0)^{\infty}//(C_2:=0;(C_1=0)^{\infty}))\longrightarrow\Delta$,

(13) $C_1:=1;C_2:=1;(I//C_2:=0);C_1:=0;$
$((C_2=0)^{\infty}\ //\ (C_1=0)^{\infty}))\longrightarrow\Delta$,

(14) $C_1:=1;C_2:=1;(I//(C_2:=0;(C_1=0)*));C_1:=0;$
$((C_2=0)^{\infty}\ //\ (C_1=0);(C_1=0)^{\infty})\longrightarrow\Delta$,

(15) $C_1:=1;C_2:=1;(I//(C_2:=0;(C_1=0)*;C_1=0));$
$C_1:=0;((C_2=0)^{\infty}\ //\ (C_1=0)^{\infty})\longrightarrow\Delta$.

Applying execution and elimination of assignment and replace by assumption, to (13), we have

(16) $C_2:=0;C_1:=0;(I^{\infty}\ //\ I^{\infty})\longrightarrow\Delta$.

This is an axiom and represents the deadlock. From (14), we have the same sequent as (16). Applying execution of assignment to (15), we have

C_2:=0;(1=0)*;1=0;C_1:=0;

$((C_2=0)^{\infty}$ // $(C_1=0)^{\infty}) \longrightarrow \Delta$.

By replacement of assumptions, we have

$\Gamma \longrightarrow \Delta$,

which is an axiom.

In this problem, the number of leaves of the proof-tree is 38. And the time for proving is 112 seconds.

4.2 Dijkstra's solution for two processes

E.W.Dijkstra gave the famous solution [2] p.58. In order to prove the correctness of the solution, it is sufficient to show the following sequent in our system. In the following, h=1 or h=2.

C_1:=1;C_2:=1;t:=h;

(C_1:=0;

((C_2=0, t=1)* V (C_2=0, t≠1, C_1:=1, t≠2, C_1:=0)* V (C_2=0, t≠1, C_1:=1, t≠2, C_1:=0)* V (C_2=0, t≠1, C_1:=1, t≠2, C_1:=0)* V (C_2=0, t≠1, C_1:=1, t≠2, C_1:=0)$^{\infty}$

C_2≠0 C_2≠0 C_2=0, t≠1, C_1:=1, $(t=2)^{\infty}$ $(C_2=0, t=1)^{\infty}$

CS_1;t:=2;C_1:=1

//

(C_2:=0;

((C_1=0, t=2)* V (C_1=0, t≠2, C_2:=1, t≠1, C_2:=0)* V (C_1=0, t≠2, C_2:=1, t≠1, C_2:=0)* V (C_1=0, t≠2, C_2:=1, t≠1, C_2:=0)* V (C_1=0, t≠2, C_2:=1, t≠1, C_2:=0)$^{\infty}$

C_1≠0 C_1≠0 C_1=0, t≠2, C_2:=1, $(t=1)^{\infty}$ $(C_1=0, t=2)^{\infty}$

CS2;t:=1;C_2:=1)

$\longrightarrow CS_1;C_1:=1;CS_2;t:=1;C_2:=1,$

$CS_2;C_2:=1;CS_1;t:=2;C_1:=1.$

In this proof, the number of leaves was 690 and it took 3074 seconds long in which 2976 garbage collections were included.

5. CONCLUSION

Our prover is very useful for verifying logical properties such as the mutual exclusion or deadlock free problem. But at the present, this prover is basic and the construction of a proof-tree is staightforward. So it has many branches and leaves. In the next step, we intend to supply it with the following facilities: (1) analyses of syntactic patterns, for example, application of rules of inference with attention to a fixed variable, (2) restricted mathematical induction (cf. [7]) for processing infinitely long expression F*, and (3) man-machine interaction.

REFERENCES

[I] DiJkstra,E.W., "Co-operating sequential processes," Programming Languages, edited by F.Genuy, Academic Press, 1968,43-112.

[2] Gries,D., "An Exercise in Proving Parallel Programs Correct," CACM, vol.20, Dec. 1977, 921-930.

[3] Hoare,C.A.R., "Towards a theory of parallel programming," Operating System Techniques, edited by C.A.R.Hoare and R.H.Perrott, Academic Press, 1972, 61-71.

[4] Hoare,C.A.R., "Parallel Programming: An axiomatic approach," Computer Languages, vol.1, 1975, 151-160.

[5] Knuth,D.E., "Additional Comments on a problem in concurrent programming control," CACM, vol.9, NO.5, May 1966, 321-322.

[61 Nakanishi,M., KLISP Reference Manual (Revised Version), Keio Univ., 1970 (in Japanese).

[7] Nakanishi,M., Nishimura,T. and Nagata,M. "Gentzen-type Formal System Representing Properties of Function and Its Implementation," Proc. of IJCAI 75, 1975, 57-64.

[8] Nishimura,T., "Formalization of Concurrent Processes," Information Processing 77, 1977, 929-937.

[9] Owicki,S. and Gries,D., "An axiomatic proof technique for parallel programs I," Acta Imformat ica, vol.19, No.5, May 1976, 279-285.

[10] Owicki,S. and Gries,D., "Verifying properties of parallel programs: An axiomatic approach ," CACM, vol.19, No.5, May 1976, 279-285.

CONVERSATION AS PLANNED BEHAVIOR

Jerry R. Hobbs
SRI International
Menlo Park, California

In this paper, planning models developed in artificial intelligence are applied to the kind of planning that must be carried out by participants in a conversation. A planning mechanism is defined, and a short fragment of a free-flowing videotaped conversation is described. The bulk of the paper is then devoted to an attempt to understand the conversation in terms of the planning mechanism. This microanalysis suggests ways in which the planning mechanism must be augmented, and reveals several important conversational phenomena that deserve further investigation.

1. Introduction

Perhaps the most promising working hypothesis for the study of conversation is that the participants make use of planning mechanisms much like those developed in artificial intelligence. The purpose of this paper is to explore some issues that arise in attempting to use this working hypothesis as the basis for a theory of conversation.

An empirical theory is an elegant description of a selected body of data. This characterization indicates three steps, if we are to construct such a theory:

1. to specify the data;
2. to define the language of description;
3. to describe.

These simply stated steps hide subtleties, some of which are explored below. Specifically, in section 2, we take up the question of Just what the data of a theory of conversation can be. In section 3. the language of description, viz. a language for the operations of a planning mechanism, is presented.

The bulk of the paper, sections 4 and 5, is then devoted to an exercise in description — the microanalysis of an actual conversation. This is a way of investigating two mutually recursive problems. The first is — given an adequate planning mechanism with access to adequate conversational strategies and world knowledge, show that such data could be produced. This, in a sense, is a hand simulation. The second problem Is — assuming the data jiaa. produced by a planning mechanism, what sort of planning mechanism would it have to be? That is, the data is used to put pressure on the formalism. The computational mechanisms suggested by this exercise are discussed in section 6. In section 7, some general properties of conversation are discussed in the form of questions for future research.

None of this can tell us what went on in the actual production of the conversation, only what could have gone on. But in this, it is no different from other explanations in cognitive science. The best we can do is to tell a story that contradicts nothing we know.

2. What is the Data?

A theory of conversation concerns itself with utterances which are appropriate in particular contexts to particular conversational goals. Among the goals are goals external to the conversation, such as a task jointly engaged in (Grosz 1977), the speaker's "coherence goals" to structure the conversation in a way that will ease comprehension (Hobbs 1978), and "image goals", the speaker's desire to project a favorable image (cf. Goffman 1974, Labov and Fanshel 1977). But appropriate utterances do not exist as data independent of a competent observer's judgments as to which utterances are appropriate. We require competent judgments just to determine what could count as data.

Since the data is not exhaustively presented, the theory must make predictions to verify that it covers the data. But we need to be precise about what we can expect our theory to predict. We cannot expect predictions of the utterances, given only the context and the speaker's goals. A goal can be realized in many ways, and the mystery of human choice intervenes. The most we can hope for is to predict the set of possible utterances. But this set exists as data only in the form of its characteristic function, the appropriateness judgments of a competent observer. It is this that the theory should predict.

The best observer is someone with the greatest possible access to the context of utterance and the speaker's conversational goals. But this Is Just the speaker herself. In studying real conversation, we may assume utterances to be appropriate in context unless

there is strong evidence to the contrary. The fact that a competent speaker uttered a sentence and did not retract it is generally the best appropriateness Judgment we have.

To predict an utterance, we would have to show why a derivation of the utterance from the conversational goals was chosen over derivations of all other possible utterances. To predict the appropriateness judgment, we need only show that some derivation exists. Thus in the microanalysis of section 5, we assume the utterances were appropriate, insofar as the speaker could tell, and seek to show that they could have been produced by a planning mechanism of the sort described in the next section.

3. The Planning Mechanism

Our working hypothesis is that an utterance is appropriate in a context if and only if it could be generated by an adequate planning mechanism operating in that context. Hence, our language of description is simply the language for describing the operations of the planning mechanism, a language of goals, plans, and beliefs. In order to begin our construction of a theory, we have to assume an initial version of such a planning mechanism.

We assume a planner of the sort familiar in robot planning (Sacerdoti 1977, Sussman 1975). It has goals and has access to general knowledge, including certain causal axioms encoding knowledge about what causes or enables what. Among these, for conversation, must be axioms that encode knowledge of conversational strategies, such as the facts that humor generally causes the listener to have a favorable image of the speaker and that descriptions of mishaps are frequently humorous.

The planner is able to decompose a goal into subgoals by using its knowledge of what actions will bring about the goal and what prerequisites enable these actions. It generates a conversational plan by decomposing its top-level goals into subgoals, and these subgoals into further subgoals, until the subgoals are directly realizable as utterances or gestures. When actions are executed, it uses its causal axioms to recompute the state of the world, by noting new conditions caused by the action and old conditions eliminated.

There has been much important work applying similar planners to the planning of speech acts (Searle 1969, Cohen 1978, Allen and Perrault 1978, Levy 1978, Moore 1978), and a theory of conversation will have to incorporate this work as a subpart. But it has not gone beyond the planning of single utterances. When we do, we can expect to encounter plans that require a number of steps to execute and that can go awry at any point. To deal with these problems, we need to borrow and modify two further mechanisms from research in robot planning — a monitor and a debugger.

In a natural language processor, a major component for interpreting utterances is an operation which finds a place for the utterance in an overall ongoing plan. Such an operation has been described elsewhere (A.Robinson 1978, Hobbs and J.Robinson 1978), and is similar to work on plan recognition (e.g. Schank and Abelson 1977, Wilensky 1979, Bruce 1978, Beaugrande 1979, Genesereth 1978). This operation serves to monitor the execution of a participant's plan. In the conversation analyzed, Y's reaction to X's moves (D5) and (D12) are interesting examples of monitoring.

The debugging situation in conversation is unlike debugging programs where we change the code and start again. It is more like a single run where we break at critical points, check certain conditions, patch up our errors, and go on. We have to live with the mistakes we've made. In our planning, we are using causal axioms that are at best only plausible, and sometimes actions don't cause what they are expected to cause. If the monitor has learned new information that contradicts what was expected, the debugger must attempt to determine which of the causal axioms happened not to be true, to account for this by searching deeper into the knowledge base for factors not previously considered, and to call on the planner to generate a repair and a new plan.

As the microanalysis of section 5 reveals, what has been sketched here is inadequate in at least two significant respects. These are discussed in section 6.

The planning metaphor provides an attractive vocabulary for describing conversation, for it seems to accord with the way we feel about our conscious moves and with what we are willing to attribute to our unconscious moves. The non-determinism of the planning process allows room for our sense of free choice. Unlike more rigid formalisms, e.g. flow charts, behavior outside the norm is not outside the system; rather it is a result of a less frequent path being taken by the planner. Unlike the rule systems proposed by ethnomethodologists (e.g. Sacks, Schegloff, and Jefferson 1974, the planning metaphor allows us to be explicit about the motives that lie behind the strategies we use. Among the various mechanistic metaphors of cognitive psychology, this one seems to detract the least from our humanity.

4. The Data to be Analyzed

For the rest of the paper, we will be analyzing a fragment of a free-flowing conversation, to exercise the planning mechanism. The fragment comes from the beginning of a videotaped conversation between a man X and a woman Y. The man enters the room first and sits down. Several minutes later the woman enters carrying a manuscript that happens to be her dissertation and four large manlla envelopes. She sits down and they begin the conversation shown below.

(D1) [Y displays dissertation.]
(D2) [Y displays four bulky envelopes.]
(D3) X: What's all this mail?
(DM) Y: My child is entering a Q-tips art contest.
You see, you haaa-
(D5) [X grins.]
(D6) Y: YOU don't have any children, obviously. You must...
(D7) You have to either draw or make things with the little Q-tips.
(D8) SO she thinks she's going to win an $8000 first prize.
(D9) So I have to send in this trash for her.
[One second pause.]
(D10) All these nice things made out of Q-tips.
(D11) And of course all the Q-tips will fall off.
[One second pause.]
and... in the mail....
(D12) X: And it's all to be sent to Blair Nebraska, huh?
(D13) Y: Yeah. This sounds really flakey though.
(D14) I... I never heard of Blair Nebraska
(D15) and you send it to a P.O. box.
So what happens too if I
(D16) What happens if you have dishonest mailmen
[X leans back in chair and crosses legs.]
and they see all these things going to an art contest, so they open it up and change it so that it's being sent from them?
[Y leaning forward. Three and a half second pause.]
(D17) X: How would they change it?
Y: Well... Instead of...
(D18) Instead of the return address being my address they would put down their address, so they would win, you see.
(D19) [Two and a half second pause. Y picks up envelopes, revealing dissertation for the first time since (D1).]
(D20) Y: Not that my poor child is going to win.
(D21) But anyway.
(D22) X: I don't think anybody, except for a child, would want to enter a Q-tips art contest.
(D23) [Both laugh. Y picks up dissertation and begins to leaf through it. Leans back. Shoulders relax.]
(D24) Y: Well, maybe the postman has children.
(D25) You never can tell.
(D26) This is my dissertation. It's Just been approved.

Like all transcripts of everyday conversation, this appears incoherent at first. However, when we examine it closely in terms of what X and Y are trying to accomplish, we discover an intricate structure.

The purpose of the conversation was Just to talk. It therefore provides an excellent minimal example of how conversation gets planned and of the structure that results, with the least intrusion from an accompanying task or event. The two people have met each other only briefly before, and this is their first lengthy conversation, so in our analysis we need not worry about shared knowledge that we lack access to.

5. Planning Topics of Conversation

Since most of the action in this conversation is Y's, we concentrate on her plans. We may assume she has two principal goals — To project a favorable image and to cooperate with the experimental setup by maintaining the conversation.

The conversation can be divided into four episodes, each characterized by a different problem that faces Y. In the first, (D1)-(D2), Y attempts to introduce first her dissertation, then the mail, as a topic. In the second, (D3)-(D9)» Y elaborates on the Q-tips. In (D10)-(D18), she tries to continue the conversation by fishing for a productive subtopic, finally hitting on the dishonest mailmen. In (D19)-(D26), due to the failure of this subtopic, she attempts to close the topic and again tries to introduce the dissertation.

Each of these episodes forms what we may call a "subtopic", a segment with a high degree of internal coherence. Each provides a different example of the speaker's ability to manipulate the topic of conversation.

5.1. Attempting to Introduce Topics

Y's initial problem is to introduce a topic that will cast her in a favorable light. She has the material for it: Her dissertation has

|""
I am indebted to Will Leben and Dave Evans for making this videotape available to me.

just been approved, and if they could talk about that, X would conclude she was at least intelligent enough to earn a Ph.D. degree.

Here's where the Catch *22* comes in, however. If X believes Y is intelligent, then X will think favorably of Y. But if X believes Y has uttered something with the intention of causing X to believe Y is intelligent, the utterance will be interpreted as boasting, and will make X think unfavorably of Y. Thus for Y to introduce a topic that will lead to a positive image too directly is dangerous.

However, if she can get X to introduce the dissertation, she will have achieved the goal of talking about it without the side effect of boasting about it. To get X to introduce it, she can display it prominently, and at last, we have arrived at an action which is executable. Y waves the dissertation about a bit, X does not pick up on it, and the plan fails.

Another way to convey a favorable image is to project the image of a good mother, and the good work of one's child is one way to do this. The problem as before is to introduce the topic, and the same hitch as before presents itself — how to avoid boasting. The solution is the same as before. Y displays the envelopes, and the plan works as X asks, "What's all this mail?"

5.2. An Answer Perturbed

At first glance, Y's answer seems somewhat incoherent. But let's examine it more closely. To describe mail, one should describe its contents and destination, so an answer might be

(1) CONTENTS: The envelopes contain Q-tips designs.

(2) DESTINATION: I'm sending them in to a contest.

In fact, (2) appears, almost as is, in (D9). But (1) is a bit unusual; the Q-tips designs require some explanation. Y must tell of the situation that gave rise to them -- the Q-tips art contest. Since this is also unusual, she has to elaborate on the nature of the contest and might, among other things, specify what the contestant must make or do (the entry) and something about the prize structure:

(3) My child is entering a Q-tips art contest.

ELABORATION:

(4) ENTRY: You have to draw or make things with Q-tips.

(5) PRIZE: There is an $8000 first prize.

Y begins this orderly answer. She says (3) and then begins her elaboration (4). But she is interrupted, in a way that changes the rest of her answer significantly.

While Just beginning (4) she looks up, the smile that X has been trying to suppress breaks into a grin, and they both laugh. His reaction to the notion of a Q-tips art contest is a negative evaluation of sorts. Y must therefore justify her involvement if she is going to maintain a favorable image. She does so by saying

(D6) You don't have any children, obviously.

The implicit line of reasoning is — if X had children, then X would understand Y's situation, and hence not evaluate negatively. So (D6) is an accusation of ignorance as a defense against the negative evaluation.

The next utterance (D7) is unaffected by the interruption, since it was entirely planned out before. There are several indications of this in her gestures. The rest of her answer, however, does seem affected in subtle ways.

The next utterance

(D8) SO she thinks she's going to win an $8000 first prize,

is quite problematic. It does convey the information in (5). But (5) is not really an essential part of the background information for the answer to X's question, for it does not explain anything that is out of the ordinary.

One possible explanation for Y saying (D8) is that the daughter's high expectations provide a very strong motivation for Y to take the trouble to mail the entries. One does not like to shatter one's child's dreams. For this reason, (D8) functions as a further retort to X's negative evaluation.

The next utterance, "So I have to send in this trash for her," completes the answer. But it also defends against X's evaluation. We will examine how in section 6.1.

5.3. Searching for Something to Say

It is now X's turn to talk, but he doesn't, so Y must continue in a way that coheres with what has just been said. Her first attempt involves an inappropriate elaboration (D10), uttered in a forceless, off-hand manner. But it is also coherent to say "what happens next." Sending in the Q-tips designs provides the

occasion for them to fall off. Hence, (D11) continues coherently.

She has now tapped into a productive topic, so she thinks — possible mishaps to the designs on their way to contest headquarters. At this point X interposes with the remark that it is all going to Blair Nebraska. It is likely that this is no more than a follow-up to his question about the nature of the mail. But Y, rather than deducing the place of this remark in X's conversational plan, such as it is, incorporates it into her own. She takes Blair Nebraska to be an example of sending the packages out into the unknown, and states the generalization of which Blair Nebraska is one example, namely, that the situation is flakey. Then she gives a further example, that the destination is a post office box. At this point, she pops up to the general topic of mishaps, of which the Q-tips falling off and the strange destination are two examples, and gives her third example, which she apparently believes will turn out to be a productive subtopic — the dishonest mailmen.

Figure 1 illustrates this development.

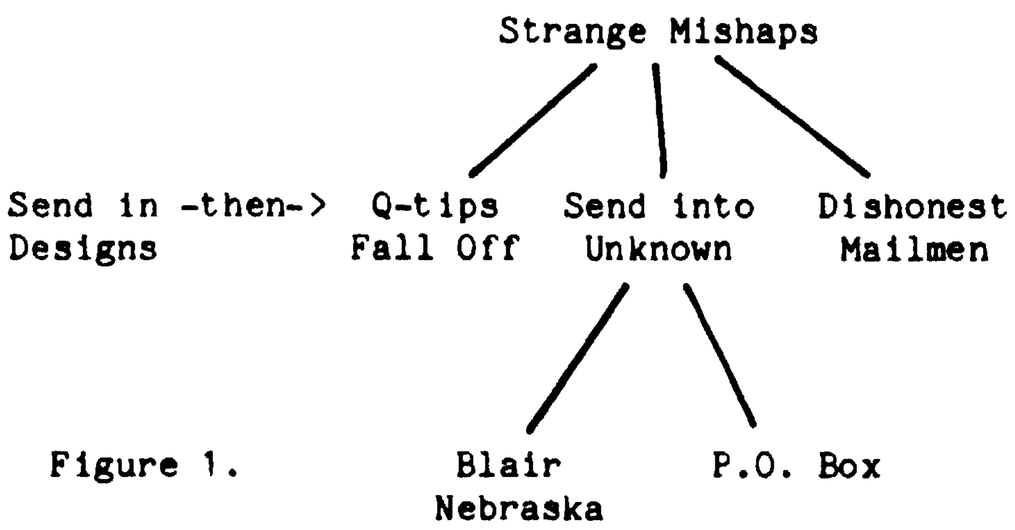


Figure 1.

Y confronts X in (D16), demanding a response with her direct "what if" question. There is a pause of 3 1/2 seconds. This is very long for a conversation like this, and it has a humorous effect on most viewers of the videotape. Finally, X does about the worst thing he could do with this topic — he takes it seriously, and consequently dismisses it as a possibility. Y responds by trying to construct a serious means by which the disaster could happen, and finally it becomes apparent that the topic has failed on all counts — it isn't productive of further conversation and isn't making her look good. She decides to cut bait and introduce a new topic, her original choice, the dissertation.

5.5. Escaping from a Failed Topic

She now faces the final topic-manipulation problem that we will examine — how to escape the current topic. When asked a question, she must answer if she is to cohere. If what is said to her reflects unfavorably on her, then she must retort. Finally, it is incoherent to suddenly switch topics — or insofar as it is coherent, it is an admission of the failure of the previous topic.

Y is thus faced with three subgoals in pursuit of maintaining the conversation in a way that will make her look good — she must salvage the current topic by arguing for the scenario's plausibility, close the current topic, and introduce the dissertation as a new topic. These three goals interweave in her next sequence of utterances and actions. She first displays the thesis for the first time since (D1), by removing the envelopes from on top of it. She has already defended plausibility in (D18), so she is free to close the topic. One way to do this is to deny the relevance of the topic to practical affairs, which she does with "Not that my poor child is going to win." At this point however, X won't let go. He responds to the whole idea with "I don't think anybody, except for a child, would want to enter a Q-tips art contest." This challenge puts Y back in the position of having to retort and then to close again before introducing the new topic. She retorts with "Maybe the postman has children", thereby denying the relevance of his argument, and then says "You never can tell," indicating that it is beyond their means at present to settle the question. She has thereby closed the topic again. In (D23) she has already picked up the dissertation and started to leaf through it, making the introduction of it as a topic less of a break with ongoing events. Then, in what represents a triumph of sorts, she says (D26), "This is my dissertation", thus succeeding in her original goal and closing the portion of the conversation that we torment with our microanalysis.

6. Computational Mechanisms

6.1. Multiple Acts in Single Utterances

In contrast to robot planning, where a single goal is realized by a sequence of actions, in conversation a single utterance frequently effects multiple goals. For example, utterance (D9) simultaneously answers a question, explains the motivation for an action, disavows the same action, and is humorous. We may approach this problem by viewing an utterance not as a single act but as a composite of several acts, each of which can realize separate goals. (D9) is a good example.

There seem to be at least three goals operating at this point. First, if Y is to cohere, she must complete her ANSWER of the question (D3) by conveying the information in (2). Moreover, the goal of defending against the negative evaluation remains, leading to two subgoals: She wants to show that her involvement results from some inexorable external circumstances (call this MOTIVATE), and to DISTANCE herself from the events by indicating that they are not a serious concern of hers.

Realizing all these goals in a single utterance requires something like the following mechanism: View a sentence as a conveyor belt with slots for each of its elements. The goals compete to fill these slots with lexical or syntactic material that will aid their own realization. Filling a slot is a matter of finding a match between what the sentence requires and what resources a goal has available. (A more pedestrian description would speak of looping through the slots, and for each slot, looping through the goals, and so on.)

Figure 2 illustrates this process:

DECISIONS:	UNMARKED:	ACTUAL:	SOURCE:
Conjunction	—	so	MOTIVATE
Subject	I	1	ANSWER
Aspect	Prog	have to	MOTIVATE
Verb	send	send	ANSWER
Destination	in	in	ANSWER
Object	these	this trash	DISTANCE
Beneficiary	—	for her	MOTIVATE

Figure 2

The goal ANSWER has determined the unmarked propositional content of the utterance at what Thompson (1977) has called the strategic level. At Thompson's tactical level, certain unmarked lexical choices are displaced with material provided by other goals. Thus, MOTIVATE supplies the conjunction "so" to indicate that Y's sending in the envelopes is due to her child's high expectations. MOTIVATE also replaces the present progressive tense with "have to"; Y's obligation excuses her for carrying around an armful of Q-tips. "For her" in the beneficiary slot indicates the circumstances behind the obligation.

Since those things one takes seriously, one necessarily values, one way to create DISTANCE is to evaluate the Q-tips designs negatively. Noun choice is a rich resource for such evaluations. The word "trash" means material with no value, and fits the bill perfectly.

6.2. Bidirectional Planning for a Next Utterance

Notice something about the segment from (DM) to (D16). First the daughter makes something to put into the envelopes (DM, D7), then Y sends them in (D9), they are in the mail (D11), they arrive in Blair Nebraska (D14), and are put into a P.O. box (D15) by mailmen (D16). What we have is the most mundane story imaginable of mail going to its destination. But Y has infused humor into this framework at every point, transforming dull raw material into the stuff of a good conversation. This is very suggestive about how utterances get planned.

A bidirectional search for a plan works not only from the goal, but also from whatever moves are currently possible. The tendency, once a schema is tapped, to follow the natural flow from one event to the next may constrain the possible next moves enough to make a bidirectional search feasible.

For example, we can imagine the initial stages of planning utterance (D11), "And of course all the Q-tips will fall off ... in the mail," going as follows: The next step in the mail scenario after the sending is that the envelopes are "in the mail" for a while. The design's fragility was a concern in packing, so associated with this step in the scenario is the possibility that the Q-tips might pop off. This is recognized as a mishap, and possible mishaps are a source of humor. Since the beginning, Y has had the goal of being humorous as a way of projecting a favorable image.

Once tapped into, the "mishaps" strategy gives a productive way of transforming further steps in the mundane scenario into conversational material. Bottom-up, we ask what happens next; top-down, we ask how this can be made interesting.

7. Some Questions for the Study of Conversation

Among the most useful ideas in our microanalysis was the notion of a subtopic as something that has structure and can be manipulated. Each subtopic served some overall purpose for the speaker and had some internal structure, and they flowed together naturally. This leads to three areas we might pinpoint for further investigation:

1. Can the notion of a subtopic be made precise, and does it represent a useful intermediate level between a person's top-level goals and what he ultimately says. In planning conversations, is there a useful distinction between deciding upon a subtopic and developing the subtopic?

2. What internal structures can a subtopic have? We have seen three examples in the microanalysis. The Answer (D3)-(D9) showed a structure of Background ♦ Primary Information. The Strange Mishaps (D10)-(D18) proceeded through parallel examples. In the Escape (D19)-(D26), there was an intermixing of utterances aimed at three different goals. What other organizing principles are there for subtopics?

3. Adjacent subtopics flow together smoothly. We have seen two interesting transition devices that accomplish this: The link between the Answer and Strange Mishaps was the causal link between (D9) and (D11). In what was spoken, the move from the mail to the dissertation in (D26) was abrupt, but accompanying actions prepared the way. What other transition devices are there?

Starting with the hypothesis that conversation is planned behavior, in a sense that artificial intelligence has made precise, we have begun to see both what sort of planning it has to be and what kinds of plans result.

ACKNOWLEDGMENTS

Much of this work was done with Dave Evans. He is responsible for many of the insights and none of the errors. The work was supported by the National Science Foundation under Grants No. MCS-76-2200M and MCS-78-07121 and by the Advance Research Projects Agency under Contract No. N00039-79-C-0118.

REFERENCES

Allen, J. & R. Perrault 1978. Participating in dialogues: Understanding via plan deduction. Proceedings, Second National Conference. Canadian Society for Computational Studies *of* Intelligence! Toronto.

Beaugrande, R. de, 1980. The pragmatics of discourse planning. To appear in Journal of

Bruce, B. and D. Newman. 1978. Interacting plans. Cognitive Science, Vol. 2, 195-233.

Cohen, P. 1978. On knowing what to say: planning speech acts. Technical Report No. 118, Department of Computer Science, University of Toronto. January 1978.

Genesereth, M. 1978. Automated consultation for complex computer systems. Ph.D. thesis, Harvard University. September 1978.

Goffman, E. 1974. Frame analysis. New York: Harper.

Grosz, B. 1977. The representation and use of focus in dialogue understanding. Stanford Research Institute Technical Note 151, Stanford Research Institute, Menlo Park, Cal. July 1977.

Hobbs, J. 1979. Why is discourse coherent? To appear in F. Neubauer (Ed.) Coherence in natural language texts

Hobbs, J. & J. Robinson 1978. Why Ask? SRI Technical Note 169, SRI International, Menlo Park, California. October 1978.

Labov, W. and D. Fanshel. 1977. Therapeutic discourse. New York: Academic Press.

Levy, D. 1978. Communicative goals and strategies: Between discourse and syntax. To appear in T. Givon (Ed.), Syntax and semantics. Vol. 12. New York: Academic Press. Moore, J. 1978. Modelling communication as a problem solving activity. Unpublished manuscript.

Robinson, A. 1978. Investigating the process of natural-language communication: A status report. SRI Technical Note 165. SRI International, Menlo Park, California. July 1978.

Sacerdoti, E. 1977, A structure for plans. and behavior. New York: Elsevier.

Sacks, H., E. Schegloff, and G. Jefferson. 1974. A simplest systematics for the organization of turn-taking for conversation. Language. Vol. 50, No. 4, 696-735.

Schank, R. and R. Abelson. 1977. Scripts, plans, goals, and understanding. Hillsdale N.J.: Laurence Erlbaum Associates.

Searle, J. 1969. Speech acts; An. essay In the. philosophy of language. Cambridge, England: Cambridge University Press.

Sussman, G. 1975, A computer model of *S*kl*i*l acquisition. New York: Elsevier.

Thompson, H. 1977. Strategy and tactics: A model for language production. Papers from the Thirteenth Regional Meeting. Chicago

A KNOWLEDGE-BASED APPLICATION DEFINITION SYSTEM

Clifford R. Hollander
Harry C. Reinstein
IBM Scientific Center
1530 Page Mill Road
Palo Alto, California 94304

The design of an experimental consultation system, here called ADS, aimed at providing expert-level assistance to (nonprogrammer) specialists in defining medically-oriented, sensor-based applications is considered. The heart of the system is a knowledge base containing extensive static descriptions of the problem domain and mechanisms for instantiating elements of a subject application.

1. INTRODUCTION

One very promising direction for facilitating more direct involvement by the (non-programmer) problem area specialist in the application definition process is the use of an intelligent computer-based assistant, capable of offering expert-level guidance. This paper considers the design of an experimental knowledge based consultant [1], here denoted as ADS, which incrementally elicits (and presents in flowgraph form) a detailed specification of the subject application. The task domain selected for exercising these concepts involves gathering, analyzing, and displaying data obtained from sensor-based medical instrumentation.

ADS has been heavily influenced in its design by a number of research AI systems [2,3,4,5,6,7,8]. It maintains a substantial knowledge base, containing both static descriptions of the problem domain, obtained from human experts, and dynamic descriptions of the subject application, generated during the consultation. The knowledge base is used to direct a simple, yet powerful, set of procedural mechanisms.

The emphasis here is on achieving superior problem-related expertise, ease of use, and extensibility. A prototype (coded in APL) was only recently begun, therefore most of this discussion pertains to design issues.

2. AN EXAMPLE APPLICATION

Fig. 1 depicts a typical ADS application, namely to display systolic (i.e., maximum arterial) blood pressure. Starting from this partial specification, ADS and the user cooperate in discovering the semantic relationships which complete the diagram:

(1) A 'windowing* criterion is needed to govern the number of values input to MAXIMUM on each iteration. The user decides to process the sample points gathered during a single cardiac cycle. This entails introducing an additional sensor device and an event handler.

(2) For enhanced readability, ADS suggests that several seconds of MAXIMUM-readings be averaged before updating the display screen.

3. EXTERNAL APPEARANCE

The dialog is conducted wholly in terms of primitives familiar within the problem domain, e.g., sensors, displays, sampling rates, averaging windows, events. It is materialized as a succession of interactive panels, each dynamically tailored to reflect the needs of the current dialog state. A panel appears as a collection of screen areas (embodying menus, lists, prompts, messages, graphs, and key-in slots) together with usage restrictions (e.g., value constraint, selection criteria, view-only, scrollability, panel chaining). Utilizing full-screen panels as the units of communication

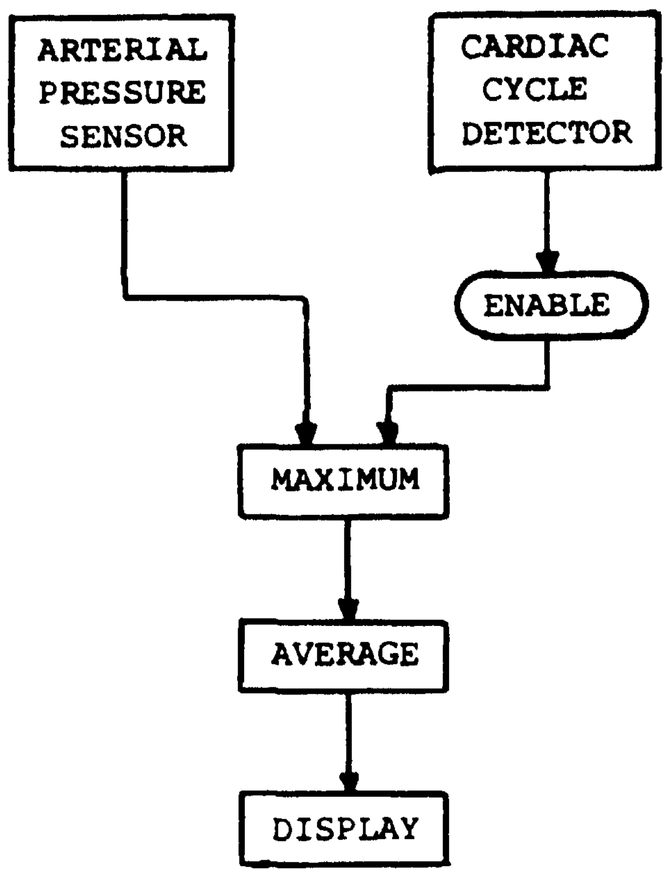


Fig. 1. Systolic Pressure

should make for a more comfortable dialog.

Throughout the definition process, a flowgraph, reflecting the current application description, is developed on a separate graphic screen. Extensive feedback and contextual information is available both from this diagram and from the knowledge structures associated with its components.

Also, an Explanation facility [2] is planned, in order to give the user auxiliary information about the dialog session and the knowledge base.

4. CHOICE OF REPRESENTATION

Knowledge in ADS is divided into two major categories based upon differences in origin, usage, and lifetime. The Resident Knowledge Base (RKB) is built by human experts and contains the static descriptions by which the system models the conceptual primitives which underlie the problem domain. Each such domain entity, has a number of identifiable components, each with certain attributes, and participates in various relationships with other entities. The Application Knowledge Base (AKB) is built during the consultation and contains dynamic descriptions which characterize the subject application in terms of the RKB model. Each application entity is an instantiated AKB version of a relevent domain entity, created as a result of traversing a portion of the RKB.

The RKB and AKB share a single object-centered representation scheme. Entities are expressed in KRL-like [9] notation and embedded within a semantic network L10], using several explicit edge types for semantic linkage. In the RKB, links are used primarily to express specialization (SPEC edges) and reference (REF edges) relationships among domain entities. Whereas, in the AKB, links tend to denote actual attachments (ATT edges) between the application entities involved. Other kinds of relationships are represented implicitly by introducing relation entities into the network and by partitioning it into subnets L10 J.

An entity contains a set of named slots, which elaborate its attributes and components via sets of (nested) descriptive clauses. A clause consists of a (1) value or value constraint; (2) datatype, e.g., heartbeats/minute, time series; (3) reference to another slot, entity, or subnet; (4) source designation, e.g., user-supplied, filled from another slot; (5) specialization constraint. The order of clauses within a slot is irrelevent. Some examples of clauses;

356, in SYMPTOM-LIST	(values)
NUMERIC, WAVEFORM	(datatypes)
ref-to PRESSURE-DATA	(REF)
fill-from MINVALUE of RANGE	(source)
spec-of SENSOR with MODE=ANALOG	(SPEC)

Considering only SPEC edges, the RKB is a set of hierarchies, one for each major class of primitives, e.g., sensors, displays, functions, connections, data. With respect to these hierarchies: (1) An entity inherits all slots known to its parent. (2) A leaf is a terminal node; it corresponds to an actual domain object. (3) A prototype is a nonterminal node; it stands for an entire (sub)class of objects.

5. BEHIND THE DIALOG

The dialog is an elaboration process in which the elements and flow of the application are recursively analyzed, using the RKB as a model, extracting the details necessary to synthesize the AKB. In each segment of the analysis, a SPEC path is traced from a prototype down to one of its leaves. Along the way, the slots of each entity encountered are used to propagate the information needed to instantiate a new application entity. The entity paths for the

systolic pressure example would be:

```
DEVICE                    FUNCTION
  SENSOR                    ARITHMETIC
    PRESSURE                  *MAXIMUM
      *ARTERIAL               *AVERAGE
  ...                       *EVENT-HANDLER
  DETECTOR                  ...
    *CARDIAC-CYCLE          DISPLAY
                              FIELD
                                *NUMERIC
```

where indentation indicates specialization and an asterisk precedes each leaf.

The clauses comprising a slot are processed according to their type. Value, datatype, and source related clauses are used to fill AKB slots (including user interactions). Specialization clauses control selection among alternative SPECs. Reference clauses (to other domain entities) are logged on a work list and serve as the origins of new traversal segments.

6. KNOWLEDGE ACQUISITION

The knowledge acquisition facility is organized around an Editor and a Checker. The Editor offers the problem domain expert full-screen interactive capabilities for examining, building, and modifying the RKB and (during a definition session) the AKB. Attention may be focussed upon knowledge structures at the subnet, entity, or slot/clause levels. The Editor is the sole overseer for activities performed at the subnet and entity levels.

The Checker is called to syntactically validate the contents of any new or altered slot. It makes use of a formal grammar for clauses together with a simple parser. A modest amount of semantic validation is also performed at this time, e.g., checking conformance between a slot-name and an entity-name in a reference clause.

7. AN ASIDE

Application definition may be viewed as a first step in a larger automatic programming process. In this light it is reasonable to consider extending ADS so that it generates executable programs corresponding to an elicited application description.

8. CONCLUSIONS

We have tried to design a consultant which is capable of offering expert-level assistance to nonprogrammers in specifying applications. The heart of the system is a knowledge base with both static and dynamic components, which drives a high-level problem definition dialog.

REFERENCES

[I] Feigenbaum, E., The art of AI: themes and case studies in knowledge engineering, Proc. 5th IJCAI, M.I.T., 1977, 1014-1029.

[2] Shortliffe, E., Computer-based Medical Consultations: MYCIN, Elsevier, New York, 1976.

[3] Buchanan, B.G. and Feigenbaum, E., DENDRAL and meta-DENDRAL: their applications dimension, Stanford Heuristic Programming Project HPP-78-1, Feb. 1978.

[4] Martin, N. et al, Knowledge management for experiment planning in molecular genetics, Proc. 5th IJCAI, M.I.T., 1977, 882-887.

[5] Lenat, D., AM: an artificial intelligence approach to discovery in mathematics as heuristic search, Stanford Heuristic Programming Project HPP-76-8, 1976.

[6] Duda, R. 0. et al, Development of a computer-based consultant for mineral exploration, SRI Intl., Ann. Report, Oct. 1977.

[7] Bennett, J. et al, SACON: A knowledge-based consultant for structural analysis, Stanford Computer Science Dept. CS-78-699, Sept. 1978.

[8] Green, C. and Barstow, D., On program synthesis knowledge, Artificial Intelligence 10, 3 (Nov. 1978), 241-280.

[9] Bobrow, D. and Winograd T., An overview of KRL, a knowledge representation language, Cognitive Science 1, 1 (Jan. 1977), 3-46.

[10] Fikes, R. and Hendricks, G., A network based representation and its natural deduction system, Proc. 5th IJCAI, M.I.T., 1977, 235-246.

Interactive Theorem Proving on Hierarchically and Modularly Structured Set of Very Many Axioms
- Extended Abstract -

Michio Honda*
Kagawa University
Dept. of Information Science
Takamatsu

Reiji Nakajima*
Kyoto University
Research Institute for Mathematical Sciences
Kyoto

A large number of axioms are often involved in the proof of a single theorem in many realistic applications of mechanical theorem proving such as formal verification of programs whose program domains are determined by user-defined axioms. There, fully automatic proofs are unrealistic due to the obvious constraints though a powerful machine support is highly desired. It is suggested that some meaningful structuring of theories can ease the difficulties. Several strategies are proposed to enhance efficient interactive non-resolution proofs on hierarchically and modularly structured theories with many axioms. Use of such strategies is illustrated in their application to verification of hierarchical programs with abstraction mechanisms.

1. Introduction

For many practical applications of mechanical theorem proving, a large number of axioms are often involved in the proof of a single theorem. An example is formal verification of programs whose program domains are determined by user-defined axioms. Due to the obvious time and memory constraints, fully automatic proofs are unrealistic. Thus, interactive non-resolution proofs are inevitable so that the user can keep well aware of what is being done in each stage of the ongoing proof, understanding the meaning of the formulas generated during the proof. With human assistance,it still can be highly difficult to conduct proofs on many axioms. Thus some organized methods are needed. Here we suggest that some meaningful structuring of theories or axiom bases can be useful and introduce some strategies for interactive proofs on structured theories. The motivation of this work derived from a software development called I system!5,6] at Kyoto University. This is going to be an interactive system for developing and verifying programs in language 1 which is designed to support hierarchical and modular programming. It is not possible to detail the language and system here, but in short, program development and verification in i involves hierarchical and modular development of theories with many user-defined axioms and requires theorem proving on them. (We are as much concerned with how such theory development should be done interactively as how proofs should be done on it. Often need of elaboration or modification of user-given axioms is found during proofs, which means that axiom-writing and theorem proving must go together to some extent, and this is another reason to make the system interactive.) The proof strategies have been implemented in i prover, subsystem of \ system, which is invoked during program development and debugging as well as verification. (The first version of i prover currently runs on DEC system~20[4].) The i prover contains an automatic proof system in addition to the proof checking facilities. The man-machine interaction (i.e. proof checker) exploits the strategies in order to reduce the proof that is beyond the limit of the capabilities of the automatic subsystem to one within the limit. This paper is not self-sufficient or comprehensive due to space limitation. [6] is a suitable reference to the background while [4] details the prover with work-out examples.

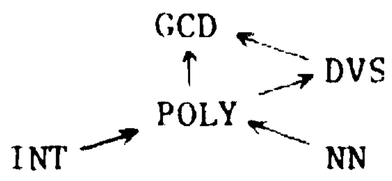
Related works: It seems that no previous work has attempted to exploit the structuring of theories to facilitate mechanical proofs. Clear [2] seems to be somewhat in a similar direction though it uses algebraic axioms while we use first order logic. (It is beyond the scope of this paper to compare the two approaches.) The concepts of user-developed reduction rules (Section 4) are introduced earlier in LCF[3], while an idea similar to theory-focusing (Section 3) is used in [7] but with a different objective. [1] surveys numerous works on non-resolution theorem proving.

2. Hierarchical and modular theory development

The proof system is based on a deviation of the many-sorted first order logic called i logic[6]

* Order is not significant.

which is tailored for the specification of hierarchical programs with abstraction mechanisms. Language \ provides a syntax with which one can structuredly build up theories in 1 logic. [6] gives a formal and complete description of the logic and the language and here we only give a rough sketch. Each sort in \ logic is characterized by a finite number of functions and axioms which are said to be the basic structure of the sort. There are two kind of sorts; types and sypes. The types are associated with the notion of generator induction rules while the sypes are not (Section 5). A syntactic unit of language i is called module which defines a chunk of theory in i logic by presenting a finite set of functions as well as axioms that characterize the functions, in the figure, NN, INT and HOLY are type modules presenting the basic structures of the (abstract) types of integers, natural numbers and polynomials with integer coefficients, respectively. DVS is a procedure module which extends the theory of POLY by adding a function DIV etc. Upon these modules a procedure module GCD is written to extend the joint of the theories defined by them. GCD introduces g.c.d. function on POLY. (Logical circularity among modules is prohibited, i.e. module development must be hierarchical.) Using language i, the module GCD is implemented; namely g.c.d. function is realized by an Algol-like routine which is written on the theory defined by DVS, POLY, NN and INT. In order to verify that the routine satisfies the axioms given by GCD, the following formula (i.e. one of the verification conditions) must be proven. $\forall w.(DIV(x0,w) \wedge DIV(y0,w) \equiv DIV(x,w) \wedge DIV(y,w)) \wedge x\neq 0 \wedge y\neq 0 \supset (DEG(COEF(y,DEG(y)).x - COEF(x,DEG(x)).TM(DEG(x)-DEG(y),y)) < DEG(y) \supset \forall w.(DIV(x0,w) \wedge DIV(y0,w) \equiv DIV(y,w) \wedge DIV(COEF(y,DEG(y)).x-COEF(x,DEG(x)).TM(DEG(x)-DEG(y),y),w))$ [COEF: POLY→INT, DEG: POLY→NN, TM:(NN,POLY)→POLY and .: (INT,POLY)→POLY are given by POLY. DIV: (POLY,POLY)→BOOL is by DVS. TM(n,Q) is Q multiplied by x^n, r.Q is scalr product, DIV(P,Q) means that P is divisible by Q, etc.]



3. Theory-Focusing

A major difficulty with a proof on a large theory derives from the wide selection of axioms to invoke at each step. As seen in the example of Section 2, the validity of a single formula to be proven generally depends on many axioms from different modules. If the theory is cleanly and naturally modularized, however, one could well expect some desirable property in the proof, which we call proof locality. Namely, a consecutive portion of proof steps, if appropriately chosen, tends to depend on axioms from only a few or preferably a single module. This property permits the man-machine interaction to focus the attention on a particular module for a portion of the period during the proof. We collectively call such strategies theory-focusing. A successful use of theory-focusing can enhance the efficiency in proofs because it largely narrows the selection of axioms at each step and facilitates the effective applications of reduction and simplification rules on a specific module (Section 4,5). For instance, hierarchical relations among sorts in i logic provides a useful technique. Generally a formula, which is generated as a goal in the course of the proof, can contain terms of different sorts. For example $x\neq O \wedge y\neq O \wedge DIV(x,w) \wedge DIV(y,w) > DIV(COEF(y,DEG(y)).x-COEF(x,DEG(x)).TM(DEG(x)-DEG(y),y),w)$ contains terms COEF(x,DEG(x)).TM(DEG(x)-DEG(y),y), COEF(y,DEG(y)).x, TM(DEG(x)-DEG(y),y), x, y, z of sort POLY, DEG(x), DEG(y), DEG(x)-DEG(y) of sort NN and COEF(y,DEG(y)), COEF(x,DEG(x)) of sort INT. A straightforward way for theory-focusing is to replace all non-variable terms of a designated sort by variables of the sort, where the same terms are replaced by a same variable. Thus the structure of the sort is concealed, facilitating focusing on the structures of the other sorts. Here, replacing the terms of sorts NN and INT, we have : $x\neq O \wedge y\neq O \wedge DIV(x,w) \wedge DIV(y,w) \wedge DIV(r1.x-r2.TM(n,y),w)$ which is free of the structures of INT and NN. Applying axioms of DVS and POLY, we reduce the formula above to true. This is an example of theory-focusing which goes upwardly in the theory hierarchy (i.e. focusing on modules which are higher in the theory hierarchy, hiding the lower modules). There are cases in which focusing goes downwardly. [4] illustrates use of other techniques for theory-focusing.

4. Module-wise development of reduction and simplification rules

Normally, the reduction rules used in our proofs are from among the logical reduction rules i.e. the converse of the logical inference rules of the i logic. The logical reduction rules which ought to be sound and complete tend to be inefficient. Thus, it is desirable to generate reduction rules on a specific theory from the (non-logical) axioms of the theory. (Notice that the soundness of such non-logical rules must be established, for which machine support is desired. This point seems to have received little attention so far.) But when the theory is large, it is not necessarily easy to generate efficient reduction rules. An immediate application of theory modularization is module-wise development of reduction rules. It will be convenient to develop powerful reduction rules on the subtheory

defined by a module because the axioms given by a single module are supposed to relate closely to each other.

Example Given the following axioms on the array of integers, the man-machine interaction generates the reduction rule R1.

```
axiom1: m=n ∧ m ≤ LENGTH(x)
          ⊃ FETCH(STORE(x,n,i),m)=i
     2: m≠n ∧ m ≤ LENGTH(x) ∧ n ≤ LENGTH(x)
          ⊃ FETCH(STORE(x,n,i),m)=FETCH(x,m)

rule R1(P) /P is a syntactic variable/
goal    : P(FETCH(STORE(x,n,i),m))
subgoal1: m=n ∧ n≤LENGTH(x) ⊃ P(i)
       2: m≠n ∧ m≤LENGTH(x) ∧ n≤LENGTH(x)
            ⊃ P(FETCH(x,m))
```

More examples as well as the rigid syntax for reduction rule generation are given in [4].

5. Theory extractions

Often several different theories contain a common substructure, and so do \ modules. For example the theory by INT and POLY share the structure of ring. A sype is obtained by extracting and isolating such a substructure. Here we give a sype module RING to define the theory of ring. Like the other kinds of modules a sype module introduces finite functions and axioms. The relation between RING and POLY is given by a mapping Σ which maps the functions on RING one to one to functions on POLY. Each axiom on RING is transformed by a natural extension of Σ into some formula which is provable in theory defined by POLY. ([6] defines formally the relation and the way to establish it. In language 1 we considered only the relation between a sype and a type (or a sype) under a strong syntactical restriction but here for theorem proving purposes we utilize the relation between a sype and a module of any kind without the restriction[4]. For instance a sype with two functions which satisfy commutativity and associativity is useful toward many modules of any kind.) Then if we generate powerful simplification and reduction procedures on RING, they are automatically used on INT and POLY and many others. There are many candidates for sypes. The sype of ordering is useful for example. The user defines such a sype S and establishes interactively the relation between it and any appropriate module M. When the proof is focusing on M, the procedures on S are invoked. Note that such a strategy is useful because we consider user-defined theories. From Programming view point, the sypes are new data type concept to generalize the so-called type-parametrization mechanisms. It is useful for structuring programs and specifications as well as simplifying verification. The sypes are simply another sorts in i logic and differ from the types in that there is no generator induction on them. (Note that this condition is essential in the above discussion of map Σ. Details on sypes as well as their formalization in first order logic is given in [6].)

6. Subformula reductions

In practice, fairly large formulas are involved in proofs, especially in program verification. They must usually be decomposed into smaller formulas to process. The usual technique is to reduce such a formula to a certain normal form, or to apply reduction rules exclusively to the outermost level. However blind applications of such normal form reductions often destroy the semantic inevitability in the structure of the formula, resulting formulas very hard to read and to apply reductions further. Here careful applications of reductions to some appropriate subformulas can be useful to enhance the applicability of the strategies given in the previous sections. For example

```
rule R2(P,A,B,C)
 goal:     P((A⊃B) ∧ (~A⊃C))
 subgoal1: B ≡ C
        2: P(B)
```

This rule is useful in program verification because subformulas in the form of (A >B) ^ (-AC) often appear due to if-statements. There can be many useful methods for subformula reduction [|4] illustrates use of some more). Similarly to user-given reduction rules, it is desirable to be able to develop such methods interactively to guarantee the soundness.

References

[1] Bledsoe,W.W.: Non-resolution theorem proving. Artificial Intelligence 9 '77

[2] Burstall,R., Goguen,J.: Putting theories together to make specifications. 1JCAI '77

[3] Gordon,M., Milner,R., Morris,L., Newey,M., Wadsworth,C.: A metalanguage for interactive proofs in LCF. Proc. Priciples of Prog. Lang. '78

[4] Honda,M., Nakajima,R.: An interactive theorem proving system for large axiom bases. Res. Inst, for Math. Sci., Kyoto Univ. '79

[5] Nakajima,R., Honda,M., Nakahara,H.: Describing and verifying programs with abstract data types. Formal description of Programming Concepts, (ed. Neuhold) North-Holland '77

[6] Nakajima,R., Nakahara,H., Honda,M.: Hierarchical program verification -a many sorted logical approach-. Res. Inst, for Math. Sci., Kyoto Univ. '78

[7] Nelson,C., Oppen.D.C.: Simplification by cooperating decision procedures. AIM-311, Stanford A.I. Labo '78

RECOGNITION OF HANDPRINTED ALPHANUMERICS, SYMBOLS AND "KATAKANA" CHARACTERS

Yukio Hoshino[1], Hiroyuki Kami[1], Ritsuko Ohmori[1], Hideaki Ueda[2] and Yasuhumi Mitsuzawa[2]

[1]Central Research Laboratories
Nippon Electric Company Ltd.
1-1, Miyazaki Yonchome, Takatsu-ku,
Kawasaki, Kanagawa 213, Japan

[2]Peripheral Engineering Devision
Nippon Electric Co. Ltd.
10, Nisshin-cho Itchome,
Fuchu, Tokyo 183, Japan

This paper describes a new recognition system for 106 handprinted characters. The physical image of a written character is converted to an original binary pattern. This binary pattern is smoothed and thinned to a skeleton pattern. An input pattern feature string, described by symbols of singular points and direction codes of segments, is extracted from the skeleton pattern. The successful candidate categories are determined by comparing the input pattern feature string to the standard pattern feature strings. The desired category from the candidate categories is selected by the tournament structured decision tree. The system has been developed as the recognition system for a commercial optical character reader, OCR N6370, system. The recognition is carried out by custom ordered LSIs for preprocessing and a micro-computer for feature extraction and decision.

1. INTRODUCTION

In handprinted character recognition by feature extraction, measured features, such as concavities, loops, or stroke direction etc. are compared to the standard patterns. However, it is necessary to overcome the problems concerned with feature extraction and decision making technique in order to recognize the large number of symbols, as shown in Fig.1.

(A) The first problem is concerned with noise smoothing. There are many individual patterns, such as シソツネホン" : ; . , and ? in Fig.1. Accordingly, if short lines or small points are erased by the noise smoothing operation, some misrecognitions will occur.

(B) The second problem is that there are many similar character pairs among the symbols shown in Fig.1. Especially, there are many pairs of topologically the same symbols, such as (0,ロ,゜), (5, S), (7, ク, フ, 7), (9, タ), (I, エ), (ソ, ン), etc. Therefore, it is necessary to detect many detail features from the characters so as to discriminate properly between all these pairs.

(C) The third problem is the necessity to use the contextual information. Since symbol and character sizes depend on individual writers, for example, the size of numeral 0 is sometimes the same as the size of special symbol * (P sound used in combination with katakana). Accordingly, it is difficult to discriminate between numeral 0 and symbol *

Since the second problem was included, even in the recognition of freely handwritten numerals, auxiliary features were utilized to assist in emphasizing the main recognition features which have a bearing on determining which symbol is being recognized. However, this recognition algorithm cannot conveniently handle more than 100 character symbols. The authors developed a new recognition system which is capable of solving the above mentioned problems for handprinted Japanese katakana characters. After that, the recognition system was extended for special symbols, the mixed use of alphanumerics and special symbols, and the mixed use of alphanumerics and katakana characters.

2. RECOGNITION ALGORITHM

The digitized character is represented by a binary matrix whose maximum dimensions are 64x48 bits. This is the original pattern. The original pattern is smoothed and transfered to a skeleton pattern. Each consecutive sequence of black dots (ones), between a terminal point and its connected terminal point or branch point, is called a "stroke". If the length of the stroke is shorter than a predetermined value, the short stroke is eliminated from the skeleton pattern. Instead of elimination, their position and direction are sequentially recorded in the memory. These features are mini features. Terminal points and branch points are easily detected from the skeleton pattern. Those positions are stored in the memory according to their detection order. The stroke is traced in order to detect inflection points, and divided into segments which are defined by consecutive sequence of black dots between the terminal points, the inflection points and the branch points on the stroke. The order of strokes is determined by the detection order of terminal point included in each stroke. Direction code,

as shown in Fig.2-1, is given to the segment. The origin of Fig.2-1 is corresponding to the preceding singular point detected by tracking. Figure 2-2 is an example of obtained segments, direction codes, and input pattern feature string. Each symbol, used in the input pattern feature string, has one of the following meanings. 0,1,2,————7 : Direction code. E : Stroke tracking ended at a terminal point. *K* : Inflection point. B : Branch point.

First decision is performed by sequentially comparing the input pattern feature string (IPS) to the standard pattern feature strings (SPSs) with category name. The structure of each SPS is similar to that of IPS. After comparing, if no SPS is found, the character is rejected, if unique category SPSs are found, the character is recognized as the SPS category. If SPSs of more than two categories are found, second feature extraction and decision are actuated.

In the second stage, a category, which is most similar to the input character, is determined by a tournament structured decision tree. Each pair of two categories is sequentially combined from the candidate categories. When the winning category is determined by sequentially executing several statements regarding feature extraction and decision (hereafter a feature extraction and decision pair is expressed by FED), if there are remaining candidate categories, the winning category and first one of the remaining candidate categories become the next pair. When the winning category is not determined, these drawn categories are stored in the drawn categories list and deleted from the candidate categories. If only one candidate remains, the remainder is treated as the winner. Otherwise, the character is rejected. After all the sequential candidate category combinations, the winning category is confirmed to determine whether the winning category is more similar than all the drawn categories to the input character.

The configuration of the tournament structured decision trees depends on the input character shape. Also, the auxiliary feature to be detected and the parameters, such as the predetermined threshold, depend on the category pair. The auxiliary feature name and its parameters are described in an FED. The format of each FED is as follows.

C1 C2 F p1 p2...pm d1 d2...dn t1 t2 s nb nt

Each symbol has the following meaning.
C1: First category name. It is always necessary.
C2: Second category name or logical mark. The second category name is necessary when pair categories are distinguished by the current single FED or at the first FED for a logical combination of FEDs. Logical mark is used when the current FED is logically combined with the preceding FEDs.
p1 p2...pm: Singular point name (Bi, Ti, Ki, BI, TI). Bi, Ti, or Ki is i-th branch, terminal, or inflection point respectively. TI or BI is a terminal or branch point connected with another directly assigned point, respectively.
d1 d2...dn: Parameters for feature extraction area size or directions for measuring the feature, etc.
F: Auxiliary feature name. Example of auxiliary features are as follows, a Distance between assigned two points vs. the distance between two other assigned points ratio, *b* Examine whether an assigned direction mini feature is in the assigned region. c The concavity or convex value in the area determined by two assigned points, d Current character height vs. preceding character height ratio. In execution time, a feature extraction routine is called by an auxiliary feature name in an FED. The measured feature value f is recorded in a temporary memory.
t1 t2 s: Functions of these parameters are defined in Table 1
nb nt: Numbers of branch points and terminal points respectively.

A series of FEDs for pair categories Ci and Cj, as shown in Table 2, constructs a decision tree as shown in Fig.3. Feature Fi value is represented by fi in Fig.3.

(3) HARDWARE IMPLEMENTATION AND CONCLUSION

The recognition algorithm was implemented in an actual OCR N6370 system. The original pattern is preprocessed in LSIs, which were developed for the OCR. Feature detection and decision are carried out by a micro computer. The feature detection program and decision program are recorded in 20k bytes of the micro program memory. The SPSs and the FEDs are recorded in 64k bytes of the data memory. Design work was laborious because the design concept depended on the heuristics of the designer and recognition experimentation. Instead, an economical OCR system has been successfully developed. The OCR can recognize 100 characters/sec.

REFERENCES

(1) Suen, C.H., etal, Proc. 4th Int. Jt. Conf. Pat. Recog., pp30, Nov. 1978.
(2) Hoshino, Y. and Kiji, K., Procedings of the 1st Int. Jt. Conf. Art. Intel., pp153, May, 1969.
(3) Miyazaki, T. and Hoshino, Y., Institute of Electronics and Commun. Japan, PRL74-62, Feb., 1975. (in Japanese)
(4) Yamamoto, I. etal, NEC Technical Journal, No. 11, pp29, Feb. 1977 (in Japanese)

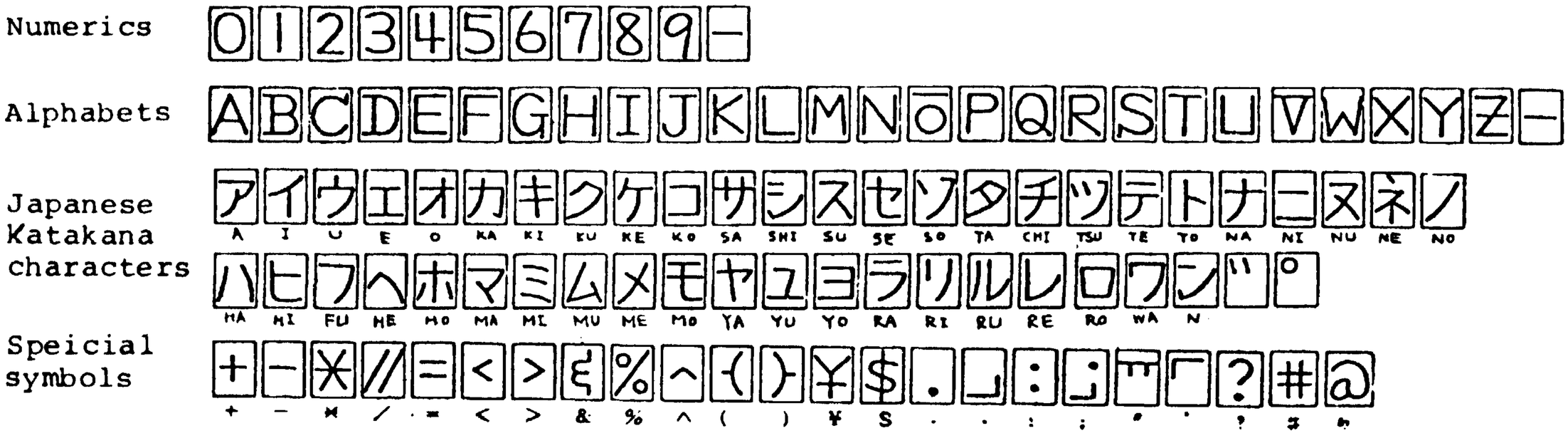


Fig.1. Readable symbols
These character shapes, without some special symbols depend on the Japanese handprint standard (draft)

When			If				Classified To	
S	$t_1 > t_2$	$t_1 < t_2$	$f > t_1$	$f < t_1$	$f < t_2$	$f > t_2$	C_1	C_2
1	1		1				1	
					1			1
		1		1			1	
						1		1
2	1		1					1
					1		1	
		1		1				1
						1	1	
3	1		1				1	
		1		1			1	
4	1		1					1
		1		1				1

TABLE 1. Detected value f and parameters s, t1 and t2.

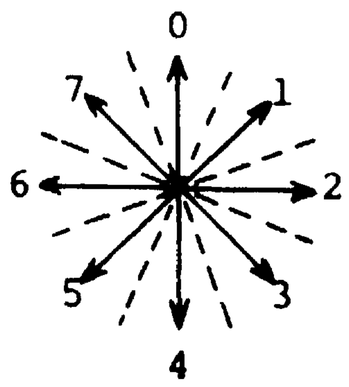
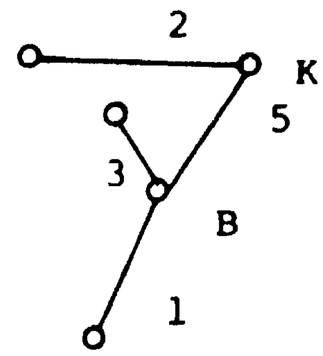


Fig.2-1.



2K5B3B1B.

Fig.2-2.

Fig.2-1. Direction code

Fig.2-2. Segments, direction codes and feature string for 丁(A)

:: ----- AND : ----- OR

Cl	Cw	F	p1---pm	d1---dn	t1	t2	s	nb	nt
Ci	Cj	F1			7	2	4		
Ci	::	F2			8	5	4		
Ci	:	F3			2	6	4		
Ci	::	F4			3	9	4		
Ci	:	F5			1	3	3		
Ci	Cj	F6			2	6	3		
Ci	::	F7			8	6	3		
Ci	Cj	F8			5	2	1		

TABLE 2 A Series of FEDs

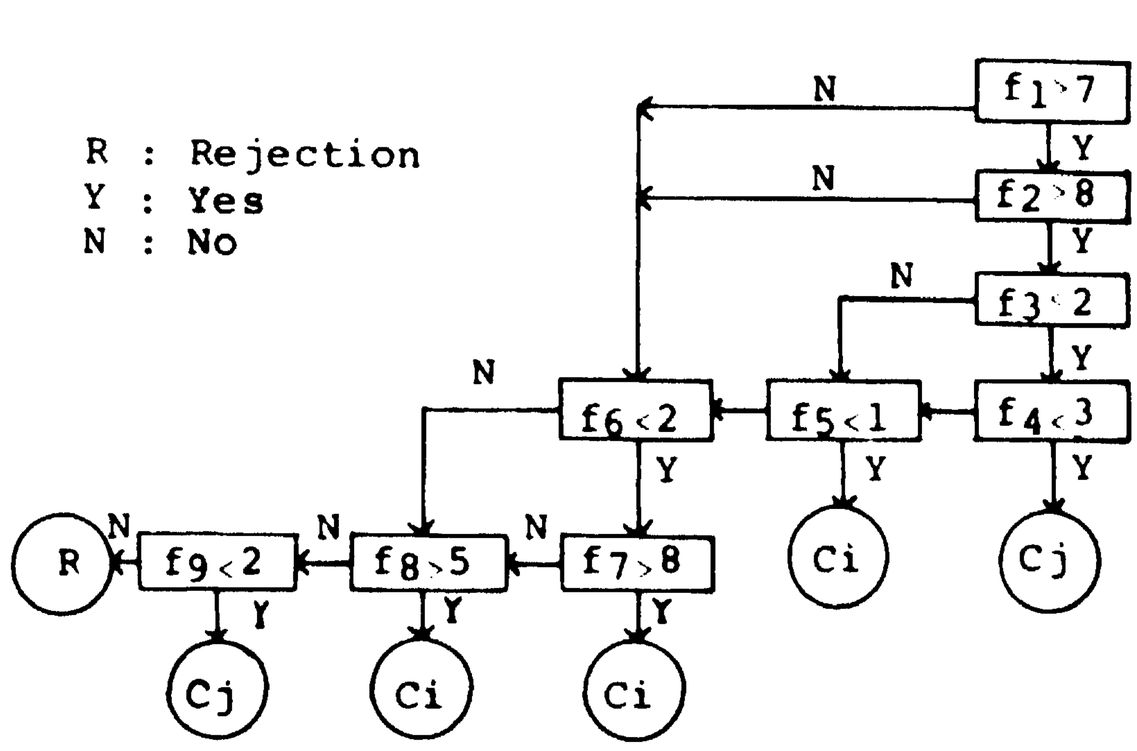


Fig.3. TABLE 2 diagramatical illustration

ASSOCIATIVE COMMUTATIVE PATTERN MATCHING

J.M. HVLLOT
IRIA-LABORIA
Domaine de Voluceau
78150 - ROCQUENCOURT
FRANCE

Plotkin MO] has opened the way to the design of unification algorithm incorporating equality axioms and their use in complete proof systems. In particular he gives complete unification algorithms for terms containing associative function symbols and for an arithmetic theory. Huet gives in [18] a unification algorithm for X-conversion and discusses in ["19] general properties of complete set of unifiers, A complete and finite algorithm for unification with associative commutative function symbols is given in Stickel [14] and Lankford shows in [22] how to extend it to abelian group theory. Various other specific unification problems are discussed in [9,15, 23]. In the case of equational theories which possess a complete set of reductions as defined in Knuth-Bendix [7], Fay has given in [20,21] a general unification algorithm.

We are interested in this paper in the special case of commutative associative function symbols. We first study the equivalence problem which is reduced to regognizing the equality of two multiset representations. In the same way, matching reduces to generating ordered partitions of a multiset. This generation is carried out by two processes operating on tree data structures spanning the lattice of partitions. Cardinality conditions inherited from the matching problem are used to cut the search drastically : our processes operate bidirectionnally in a coroutine manner, each one trimming sub-structures from both search spaces. The two processes are formally defined by a unique recursive definition, and their computations correspond to a call-by-need interpretation of this definition. The correctness of the method is proved formally. This basic organization is used to derive a new algorithm for associative commutative unification improving over Stickel [14], We have programmed in LISP all these algorithms, using University of Vincennes's V-LISP system [3]. We have applied them to the Stickel-Peterson's generalization of the Knuth and Bendix extension algorithm r7,8,16]. Their algorithm relies on associative-commutative unification for the computation of critical pairs, but also most critically on associative-commutative match, in order to reduce terms to their normal forms, Stickel and Peterson proposed to use unification for matching, but this leads to unnecessary inefficiencies that preclude the practical use of their method. We give in the Appendix running times on a DEC KL10 using the VLISP system, corresponding to some of their examples.

Most proofs are omitted in this version, the full paper is available as a technical report [24].

I. FIRST ORDER TERMS AND EQUATIONAL THEORIES

Let V be a denumerable set of elements called variables, C a finite or denumerable set of elements called constants with $C \cap V = \emptyset$. Elements in C are graded by an arity function $\alpha : C \to N$. The set of terms T is the smallest set containing V and closed by :

$t_1, \cdots, t_{\alpha(f)} \in T \to f(t_1, \cdots, t_{\alpha(f)}) \in T$ for every $f \in C$

A substitution is a mapping σ from V to T, with $\sigma(x) = x$ almost everywhere. For any $t \in T$, $V(t)$ will denote the set of variables of t. An equational theory A is a theory in which every axiom is the universal closure of an equality. Two terms t_1 and t_2 are equivalent in A iff : $A \vdash t_1 = t_2$. In the remainder we will write $t_1 \underset{A}{\equiv} t_2$ or $t_1 \equiv t_2$ when A is implicit. Let $A \subset C$ with $A = \{f_1 \cdots f_p\}$ where $\forall i \ 1 \leq i \leq p$ f_i is binary and f_i is an associative and commutative function i.e. $\alpha)$ $f_i(x,y) = f_i(y,x)$

and β) $f_i(x,f_i(y,z))=f_i(f_i(x,y),z)$. The equational theory of $f_1\cdots f_p$ is the theory with the 2p axioms α and β. In the remainder of this paper A will always denote the theory of $f_1,\cdots,f_p$. All terms will be in right associative form. A term $t=f_i(t_1,f_i(t_2,\cdots,f_i(t_{n-1},t_n)\cdots))$ where t_k does'nt begin with f_i is called NF-term. We shall abreviate such a term by $f_i(t_1,\cdots,t_n)$. If $t=f_i(t_1,\cdots,t_n)$, $\mathcal{O}(t)$ will denote n.

II. EQUIVALENCE

II.1 Equivalence lemma

Let T,T' be two NF-terms with head symbol f_i i.e. $T=f_i(t_1,\cdots,t_n)$ and $T'=f_i(t'_1,\cdots,t'_n)$. Then $T\equiv T'$ iff n=m and there exists a permutation Π of m such that $t_{\pi(i)}\equiv t'_i (1\le i\le m)$.

Proof : see [24].

II.2 Equivalence algorithm-EQUIAC (t,t')

Let us now describe an equivalence algorithm between terms in A. We call unificand a finite set of pairs of terms. Unificands are denoted by $N,N_1\cdots$.

We describe this algorithm as the construction of a tree whose nodes are either a unificand consisting of pairs of NF-terms or ∅ for success or Ⓕ for failure. We first define SIMPL(N) which simplifies a unificand N

SIMPL(N)

- SIMPL(∅)=∅
- SIMPL(<e,e'>∪N)=
  if $e\in V$
  then if e=e' *then* SIMPL(N)
  else Ⓕ
  else let $e=F(e_1\cdots e_n),e'=G(e'_1\cdots e'_n)$
  if F≠G *then* Ⓕ
  else if F=G∈A *then* <e,e'>∪SIMPL(N)
  else SIMPL($\{<e_i,e'_i>\mid 1\le i\le n\}\cup N$)

Termination : by induction on $\sum_{<e,e'>\in N}\theta(e)$. □

Let $e=f_i(t_1,\cdots,t_n),e'=f_i(t'_1,\cdots,t'_n)$, the function Permu(e,e') given below returns a finite set of unificands :

$$\text{Permu}(e,e')=\{\{<t_1,t'_{\pi_1(1)}>,\cdots<t_n,t'_{\pi_1(n)}>\}\cdots\{<t_1,t'_{\pi_{n!}(1)}>,\cdots,<t_n,t'_{\pi_{n!}(n)}>\}\}$$

Where$\{\pi_1\cdots\pi_{n!}\}$= set of all permutations of $\{1,\cdots,n\}$.

The tree for <t,t'> is grown as follows :

- the root node is N_0 = SIMPL(<t,t'>)
- node Ⓕ is a leaf
- if a node is ∅, the construction is interrupted *exit* True
- to grow the successors of node N, select in N an arbitrary pair : <e,e'>

If $\theta(e)\neq\theta(e')$ the only successor of N is Ⓕ, else Permu(<e,e'>) returns a finite set of unificands N_i. For each N_i grow an are from N to N'_i where :

- N'_i = Ⓕ if SIMPL(N_i) = Ⓕ
- N'_i = N-{<e,e'>}∪SIMPL(N_i) otherwise

If the construction stops and all leaves are Ⓕ *exit* False . The equivalence lemma proves the correctness of the method.

II.3 The heuristic part

In the implementation of the equivalence algorithm we do not generate all permutations for two given terms. Some structural remarks allow us to reduce the search space. The following example shows the behavior of our algorithm.

Let us assume

$A=\{f_1,f_2\}$, $t=f_1(x_1,f_2(x_2,a),f_2(b,x_3))$ and

$$t'=f_1(f_2(b,x_3),f_2(a,x_2),x_1)$$

where $a,b\in C$ and $x_1,x_2,x_3\in V$.

We have N_0 = SIMPL{<t,t'>} = {<t,t'>}.

Without loss of generality we generate now only

those permutations π such that if $<t_i,t_{\pi(i)}>$ is part of a resulting unificand, then $SIMPL(<t_i,t_{\pi(i)}>)\neq$ (F). Thus, since $SIMPL(<x_1,f_2(b,x_3)>)=$ (F) and $SIMPL(<x_1,f_2(a,x_2)>)=$ (F) there are only two successors :

$N_1=\{<f_2(x_2,a),f_2(b,x_3)>,<f_2(b,x_3),f_2(a,x_2)>\}$ and

$N_2=\{<f_2(x_2,a),f_2(a,x_2)>,<f_2(b,x_3),f_2(b,x_3)>\}$.

To grow the successors of node N_1 select the first pair of N_1, say $<f_2(x_2,a),f_2(b,x_3)>$. Since $SIMPL(<x_2,b>)=$ (F) and $SIMPL(<x_2,x_3>)=$ (F), the only successor of N_1 is (F). To grow the successor of N_2 select $<f_2(x_2,a),f_2(a,x_2)>$ since $SIMPL(x_2,a)=$ (F) the only successor of N_2 is $N_3=<f_2(b,x_3),f_2(b,x_3)>$. In the same way $\emptyset$ is the only successor of N_3 and the construction terminates successfully.

III. MATCHING

III.1 Definition

Let M be a finite sequence of subsets of a set N (resp.a multiset N), $M=(A_1,\cdots,A_n)$, $A_i\subset N$, $m\geq 1$, $A_i\neq\emptyset$. We say that M is an ordered partition of N iff $\bigcup_{1\leq i\leq n} A_i = N$ and $A_i\cap A_j=\emptyset$ for $i\neq j$ (resp. $\biguplus_{1\leq i\leq m} A_i = N$ where $\biguplus$ denotes union for multisets).

III.2 A complete matching algorithm

We can extend $\equiv$ to substitutions by defining $\sigma\equiv\sigma'$ iff $\forall t\in T\ \sigma t\equiv\sigma' t$. S is a complete set of matches of $<t,t'>$ iff S is consistent ($\forall\sigma\in S, t'\equiv\sigma t$) and S is complete ($\forall\sigma$ such that $t'\equiv\sigma t$, $\exists\sigma'\in S$, $\exists\lambda$ substitution with $(\sigma\lambda\restriction V(t))\equiv\sigma'$) where $(\sigma\restriction V)(x)=\sigma(x)$ if $x\in V$ and x otherwise). As for equivalence we shall construct a tree whose nodes are either a unificand or $\emptyset$ for success or (F) for failure. We first define $SIMPL_1(N)$ which simplifies a unificand N.

$\underline{SIMPL_1(N)}$

- $SIMPL_1(\emptyset)=\emptyset$
- $SIMPL_1(<e,e'>\cup N)=$

if $e\in V$ *then* $<e,e'>\cup SIMPL_1(N)$

else if $e'\in V$ *then* (F)

else let $e=F(e_1,\cdots,e_n)$

$e'=G(e'_1,\cdots,e'_m)$

if $F\neq G$ *then* (F)

else if $F=G\in A$ then $<e,e'>\cup SIMPL_1(N)$

else $SIMPL_1(\{<e_i,e'_i>\mid 1\leq i\leq n\}\cup N)$

Termination : as for equivalence. □

Let $e=f_i(t_1,\cdots,t_n)$ and $e'=f_i(t'_1,\cdots,t'_m)$ with $m\geq n$. The function PART(e,e') given below returns a finite let of unificands.

Let $\Sigma_m=\{$ordered partitions of the set $\{t'_1,\cdots,t'_m\}$ in n non empty blocks$\}=\{P_1\cdots P_q\}$ where P_i are the partitions. Thus $P_i=\{\tau_i(1)\cdots\tau_i(n)\}$ where $\tau_i(j)$ are the elements of the partition. Then we define $t^1_k=\tau_k(1)$ if $|\tau_k(1)|=1$ and $t^1_k=f_i(T_1,\cdots,T_r)$ if $\tau_k(1)=\{T_1\cdots T_r\}$ with $r>1$. We define : $PART(e,e')=\{\{<t_1,t^1_i>,\cdots,<t_n,t^n_i>\}\mid 1\leq i\leq q\}$.

The matching tree MT for $<t,t'>$ is grown as follows :

- the root node is $N_0=SIMPL_1(<t,t'>)$. We set $\sigma_{N_0}=\emptyset$
- node (F) is a leaf
- to grow the successors of node N, select in N an arbitrary pair $<e,e'>$.

if $e\in V$ *then if* $\sigma_N e$ is undefined

then grow an arc labelled with $<e,e'>$ from N to $N'=N-\{<e,e'>\}$. Set $\sigma_{N'}=<e,e'>\cup\sigma_N$

else if $EQUIAC(e',\sigma_N e)$ *then* grow an arc from N to $N-\{<e,e'>\}$ labelled by the empty substitution $\varepsilon(N\overset{\varepsilon}{\to}N-\{<e,e'>\})$

else grow an arc $N\overset{\varepsilon}{\to}$ (F)

else if $\theta(e')<\theta(e)$ *then* grow are $N\overset{\varepsilon}{\to}$ (F)

else the function PART(e,e') returns a finite set of unificands $\{N_1 \cdots N_q\}$. For each N_i grow an arc labelled with ε from N to N_i' where :

$N_i' = \textcircled{F}$ if $SIMPL_1(N_i) = \textcircled{F}$

$N_i' = (N - \{<e,e'>\}) \cup SIMPL_1(N_i)$ otherwise

The set of matches of t,t' is $S = \{\sigma_N \mid N = \emptyset \text{ in } MT\}$

The correctness of the algorithm is given in [24].

Remark 1; As for equivalence, some structural conditions allow us to reduce the search space (using $S1MPL_1$. instead of SIMPL). Thus we use the function PART only when the left term has only variables as arguments.

Remark 2 : One may define the function PART in using the concept of ordered partitions of a multiset. Such an improvement can be found in [24].

Let us now give an efficient way to generate ordered partitions of a set.

IV, THE UNDERLYING DATA STRUCTURE

We give in this part a basic algorithm which can be used to :

- generate ordered partitions of a set (detailed in example 1) or a multiset (see r24]),
- generate ordered partitions of sets or multisets with cardinality constraints on the blocks (see [24]),
- improve over Stickel's unification algorithm (see example 2 and [241 for more details).

IV.1 The Basic problem

Let $x = x_1 \cdots x_n$ and $y = y_1 \cdots y_n$ be two binary vectors of size n. We define the partial ordering relation P_1 on binary vectors by xP_1y iff $\forall i \geq x_i \geq y_i$ and $\exists j$ such that $x_j \neq y_j$. We define P_0 by xP_0y iff yP_1x. A graph of the partial order P_0 or P_1 for n=3 is shown in Fig.1.

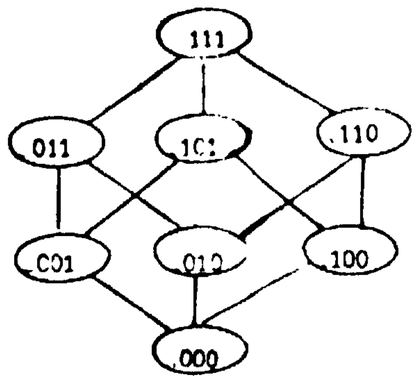


Fig.1

Throughout the remainder of this paper P_0 and P_1 will denote two predicates on binary vectors such that :

A_0. $P_0(y)$ = True and $xP_0y \Longrightarrow P_0(x)$ = True

A_1. $P_1(y)$ = True and $xP_1y \Longrightarrow P_1(x)$ = True.

Our aim is to design an algorithm for generating this subset of all binary vectors of size n such that $p_0(x)$ = True and $p_1(x)$ = True.

Example 1 : Let M be a binary vector of size m×n $M = h_1 \cdots h_n$ with $h_i = h_{i,1} \cdots h_{i,m}$ where $h_{i,j}$ is 0 or 1 Let v_j be the binary vector of size n, $v_j = h_{1,j} \cdots h_{n,j}$. We define predicates q_1, q_2, q_3 by :

$q_1(M)$ = True $\Leftrightarrow \forall i \ 1 \leq i \leq n \quad h_i \neq 0 \cdots 0$

$q_2(M)$ = True $\Leftrightarrow \forall j \ 1 \leq j \leq m \quad v_j \neq 0 \cdots 0$

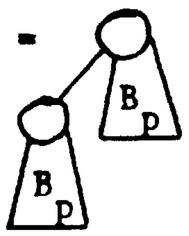
$q_3(M)$ = True $\Leftrightarrow \forall j \ 1 \leq j \leq m$ at most one element in v_j is 1,

and predicates P_0 and P_1 by $P_1(M) = q_1(M) \wedge q_2(M)$, $P_0(M) = q_3(M)$. The set of all ordered partitions of m elements into n blocks is represented by $\{M \mid p_0(M) = True \text{ and } p_1(M) = True\}$ in the following interpretation : the j^{th} element is in the i^{th} block iff $h_{i,j} = 1$. Furthermore p_0(resp.p_1) has property A_0(resp.A_1).

Example 2 : Let $M = z_1 \cdots z_n$ a binary vector of size n, $M_1 \cdots M_p$ some fixed such vectors and $M_{i_1} \cdots M_{i_q}$ some of the M_i ($q \leq p$). We define

L(M)= number of 1 in M, and predicates P_0 and P_1 by $P_0(M)=True \Leftrightarrow \forall i_j\ 1\le i_j\le q,\ L(M\wedge Mi_j)\le 1$ and $P_1(M)=True \Leftrightarrow \forall i\ 1\le i\le p\ M\wedge Mi\neq 0$, P_0(resp. P_1) has property A_0 (resp. A_1). A reader familiar with Stickel's unification algorithm [14] will recognize the basic situation of the second part of this algorithm (see [24] for more details).

IV.2. The algorithm is first conceived as a sequence of two passes that happen one after the other. It is then elaborated by stepwise refinement in order to produce the final version. In the first pass we search only those binary vectors such that $p_1(x)=True$. We may use the search method of Balas [1]. This method is equivalent to scan the combinatorial data structure called binomial tree (Vuillemin [17]) considered as a spanning tree of the graph for P_1 defined above. Binomial trees are defined by :

B_0 = ○ and B_{p+1} = 

The B_3 binomial tree T_1 associated with P_1 is pictured in Fig.2.

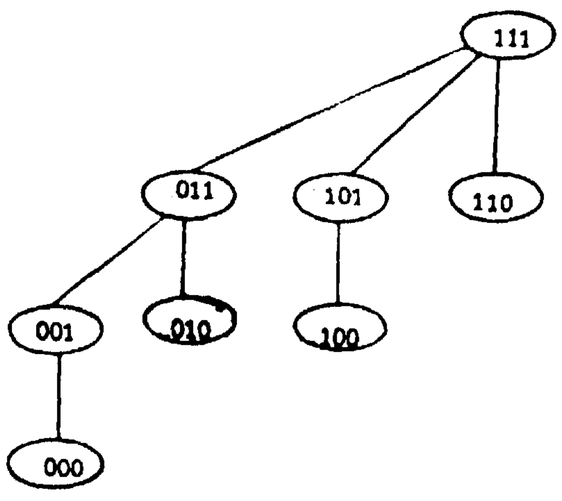


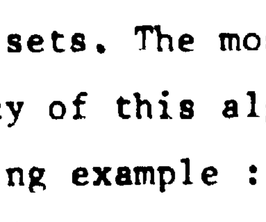
Fig.2

If for some x, $p_1(x)=False$ we do not have to scan the subtree rooted at x since $p_1(y)=False$ for all y in this subtree (property A_1). Similarly, we generate those binary vectors such that $p_0(x)=True$, in using the binomial tree T_0 associated with P_0 (T_0 is the same as T_1 except that if $x=x_1\cdots x_n$ is the label of node h in T_1, $y=(1-x_1)\cdots(1-x_n)$ is the label of node h in T_0). The final set of solutions S will be the intersection of these two sets. The most obvious source of inefficiency of this algorithm may be viewed on the following example : assume n=3, $P_0(100)=False$ and begin to scan T_1. Since, in this first pass, we do not know that the subtree of T_0 rooted at 100 must be eliminated we have to test P_1 for the 4 corresponding nodes.

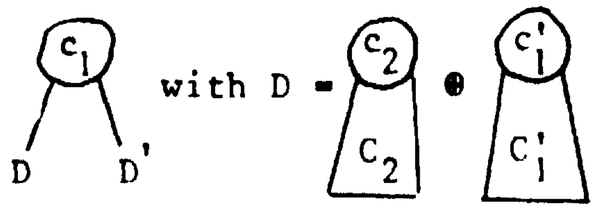
IV.3. Our refinement will consist in generating S by an algorithm which does not begin to test P_0 or P_1 for a node x until all fathers of x in T_0 (resp. T_1) have been tested for P_0 (resp. P_1). To achieve this aim, we shall represent the two binomial trees as one binary tree.

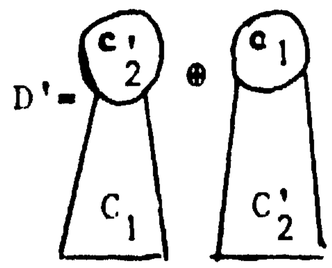
Let C and C' be two labeled B_p binomial trees. We define C⊛C' as follows :

Case 1 : p=0. Then C⊛C'=C

Case 2 : p>0. Let C= 

and C' similarly defined, then

C⊛C' = 

with D = 

$T_1 \circledast T_0$ for p=3 is pictured in Fig.3. [h] (resp (h)) means that n comes from T_1 (resp. T_0).

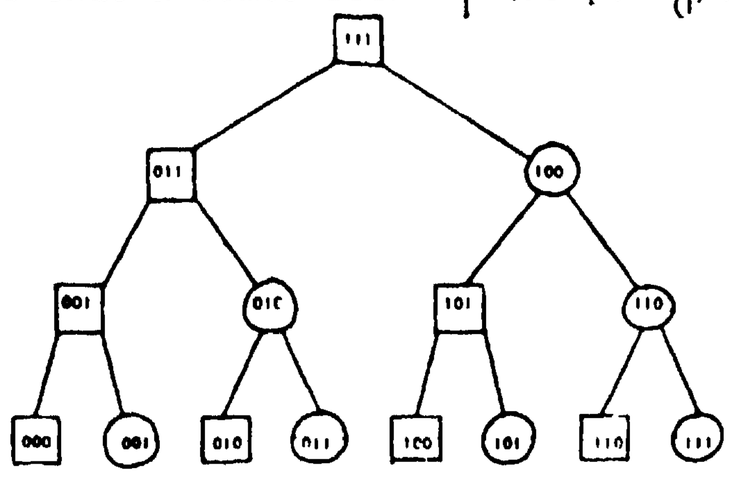
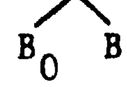


Fig.3

For any binary tree B, we define its set of occurrences O(B) as a finite subset of $\{0,1\}^*$ and the subtree of B at u denoted by B/u by :

- if B=O (only the root) then $O(B)=\{\wedge\}$ and $B/\wedge=B$
- if B=  then $O(B)=\{\wedge\}\cup\{0\cdot u \mid u\in O(B_0)\}\cup \{1\cdot u \mid u\in O(B_1)\}$ and $B/\wedge=B$, $B_0/u=B/0\cdot u$, $B/1\cdot u=B_1/u$.

Thus, any occurrence u is either $\wedge$ or u = father(u). last(u) where father(u) is an occurrence and last(u) is 0 or 1 identified respectively with False and True. Conventionnally we define $last(\wedge)=0$.

Lemma : There is a bijective mapping from the occurrences of length n onto the labels of the non terminal nodes of the binary tree of depth n+1 $T_1 \oplus T_0$. Its expression is given by : $label(u)=u\cdot b^h$ where $b=\neg last(u)$ and $h=n-|u|$.

Proof : in [24].

We define $R_n=\{w \mid w\in\{0,1\}^* \ |w|=n\}$. The definition of label is extended to R_n by : $\forall w\in R_n$ label(w)=w. Furthermore, we define :

$Set_n = \{w\in R_n \mid tree(w)\}$

where $\forall u$ such that $|u|\leq n$:

tree(u) *if* u=$\wedge$ *then* p(0^n,False)
else if tree(father(u))
then p(label(u), $\neg$last(u))
else False.

where $p(w,True)=p_1(w)$ and $p(w,False)=p_0(w)$.

Theorem : $\forall w\in R_n$ $tree(w)\Leftrightarrow p_0(w)$ and $p_1(w)$

Proof : see [24],

Our algorithm is conceived as a call by need interpretation of the recursive definition of tree. The theorem proves the correctness of this algorithm.

V. GENERALIZATION OF THE KNUTH-BENDIX ALGORITHM

It is not our purpose to present the basic Knuth-Bendix's algorithm (see [7] and [6] and its generalizations ([8],16]). Our aim is to apply our algorithms to the Stickel-Peterson's generalization of the Knuth-Bendix's extension . algorithm. Their algorithm relies on associative-commutative unification for the computation of the critical pairs, but also most critically on associative-commutative match in order to reduce terms to their normal forms. Stickel and Peterson proposed to use unification for matching, but this leads to unnecessary inefficiencies that preclude the practical use of their method. With our match algorithm a reasonable implementation is possible as shown in the Appendix where we give detailed running times on a DEC KL-10, using the V-LISP system ([3],[4]), corresponding to some of their examples.

VI. CONCLUSION

We have presented in this paper a new matching algorithm in term structures containing associative-commutative function symbols. Our experiments carried out on Stickel-Peterson's generalization of the Knuth and Bendix algorithm have shown that this algorithm together with our improvements over Stickel's algorithm provide an efficient way to implement large systems that deal with associativity and commutativity. For example we believe that the multiset matching algorithm obtained is efficient enough for the practical implementation of pattern directed invocation of procedures with multiset variables (Q-LISP [11, 1 2,15]).

Furthermore we hope that the techniques used here for the description of our partition generator can be extended to other graph algorithms.

Acknowledgements : This work is part of a thesis conducted at LRI, University Paris Sud. I would especially like to thank my supervisor G. Huet who has contributed to many ideas presented here. I would also like to thank J. Chailloux and

P. Greussay (University of Vincennes).

Appendix

We wrote a commutative associative extension algorithm as a 1500 line LISP program. It runs interpreted under the V-LISP system [3] on a DEC KL10 computer, with 1500 storage cells. We give the same examples as in Stickel-Peterson ([16]. We have two versions for simplification : inside-out and outside-in. The former is more efficient, but uses more memory and theretore the latter may be preferable for large examples. The version used is indicated in Fig.4, where U_1(resp.U_2) denotes Stickel's unification algorithm with (resp.without) our improvement and M_1 (resp.M_2) denotes matching where our algorithm (resp. unification) is used.

		superposition		simplification		other parts	Total			
Abelian groups	inside out	U_1	U_2	M_1	M_2		$U_1 + M_1$	$U_2 + M_1$	$U_1 + M_2$	$U_2 + M_2$
		8"	12"	12"	36"	1"	21"	25"	45"	49"
Abelian rings with unit		28"	45"	50"	4'	2"	1'20"	1'37"	4'30"	4'47"
Distributive lattices	outside in	1'30"	3'	3'30"	13'	5"	5'5"	6'35"	14'35"	16'5"

References :

[I] *E. BALAS* : "An additive algorithm for solving linear programs with zero one variables". Op. Res. 13 (1965).

[2] *V.J. BOWMAN & J.H. STAFF* : "Partial orderings in implicit enumeration" An. of discr. math. 1(1977).

[3] *J. CHAILLOUX* : VLISP 10.3 manuel de reference, Uni. Paris 8, Vincennes (1978).

[4] *P. GREUSSAY* : "Contribution i la definition interpretative et a l'implementation des lambda-langages". These n°78-2, Paris 7(1977).

[5] *G. HUET* : "An algorithm to generate the basis of solution! to homogeneous linear diophantine equations". Inf Proc. Let.7, 3 April 78.

[6] *G. HUET* : "Confluent reductions : Abstract properties and Applications to term rewriting systems" In Proc. of the 18th Symposium on FOCS (1977).

C7] *D.E. KNUTH & P.B. BENDIX* : "Simple word problems in universal Algebras". In Leech, J.(Ed.), Corp. problems in Abstract Algebras. Pergatmnon Press, 1970.

[8] *D.S. LANKFORD i A.M. BALLANTYNE* : "Decision procedures for simple equational theories with commutative-associative axioms : complete sets of commutative associative reductions". Tech. rep. Math. dept. U. of Texas at Austin, ATP 39 (1977).

[9] *M. LIVESEY A J. SIEMANN* : "Unification of A+C-terms (bags) and A+C+I-terma (sets)". Int. Ber. Nr 3176, Inst, fur Inf. I, Universitat Karlsruhe.

[10] *G. PLOTKIN* : "Building in equational theories" In Meltier & Michie(Ed) Machine Intelligence 7, Edinburgh University Press 1972.

[II] *R. REBOH S E. SACERDOTI* : "A preliminary Q-LISP Manual". Tech. note 81, SRI, Menlo Park, 1973.

[12] J.F. RULIFSON, J.A. DERKSEN S R.J. WALDINGER : "OA4 : a procedural calculus for intuitive reasonning". Tech. Note 71, SRI, Menlo Park, 1972.

[12] J.R. SLAGIE : "Automated Theorem proving for theories with simplifiers commutativity and associativity", J. ACM 21 ,4 (1974) .

[14] M.E.. *STICKEL* : "A complete unification algorithm for associative-commutative functions". Proc. of the 4th IJCAI Tbilisi (1975).

[15] *M.E. STICKEL* : "Unification algorithms for artificial intelligence languages" Ph.D. Carnegie Mellon, Pittsburgh, Penn (1976).

[16] M.F. *STICKEL & G.E. PETERSON* : "Complete sets of reductions for equational theories'' U, of Arizone, Techn. Report (1978).

[17] *J. VUILLEMIN* : "A data structure for manipulating priority queues" C, ACM Vol 21 (4)(1978).

[18] *G. HUET* : "A unification Algorithm for typed X-calculus[11] Theo. Com. Science 1 (1975).

[19] *G. HUET* : "Resolution d'equations dans les langages d'ordre 1,2,•••,w.' These d'F.tat U. Paris VII (1976).

[20] *M.J. FAY* : "First order unification in equational theories"Tech. Report N"78-5-002 U. of California Santo-Cruz (1978).

[21] *M J FAY* : "First order unification in equational theories!' Fourth Workshop on automated deduction Austin Texas(1979j)

[22] *D. LANKF0RD* : "A unification algorithm for ahelian group theory"Report MRPI. Dep. of Math. Louisiana Tech. Uni.

[23] *M. LIVESEY, J. SIECYMNN, P. SZAPO, F. UNVERECHT* : f'nification problems for combinations of associativity, commutativity, distributivity and idempotence axioms. Fourth Workshop on automated deduction. Austin Texas Feb.(1979)

[24] *J.M. HULLOT :* "Associative-commutative pattern matching'" Report IRIA-LABORIA to appear (1979).

AN APPLICATION OF THE PHOTOMETRIC STEREO METHOD.

Katsushi Ikeuchi
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
545 Technology Sq.
Cambridge, Massachusetts 02139

Berthold K. P. Horn
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
545 Technology Sq.
Cambridge, Massachusetts 02139

The idea of photometric stereo is to determine the surface orientation of the points of an object in a picture using several views of an object taken with varying illumination, but from the *same* viewing direction. We can implement this method using reflectance map techniques. In order to calculate the reflectance map, we consider primarily the specular component of surface reflectance, since most industrial materials are very specular. The technique has been demonstrated by experiment.

0 Conclusion

The technique of photometric stereo has been proven by an experiment whose schema is shown in Fig. 1. Orientations of surface patches on a sphere obtained using this technique are shown in Fig. 2.

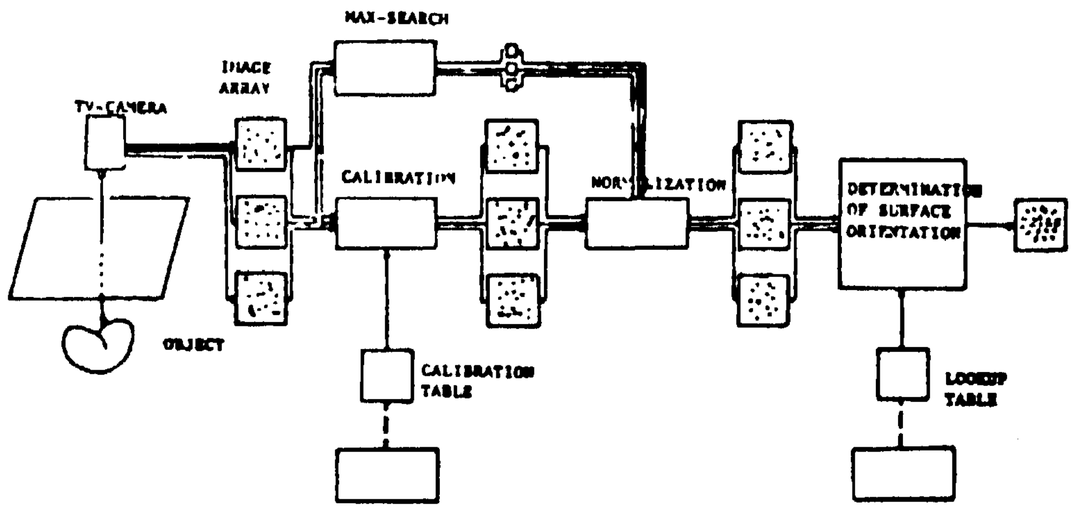


Fig. 1 The total schema of the experiment.

The schema has two parts of jobs, one is an off-line job, represented by broken lines. The other job is one-line. Image brightness is obtained by using a TV camera. We search for maximum values of brightness in each array. Brightness arrays are also calibrated and normalized. A lookup table has been constructed off-line from the reflectance map by using Newton's method. The lookup table is used to obtain surface orientation.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advance Project Agency of the Department of Defence under Office of Naval Research contract N00014-75-C-0643.

1 Reflectance Map and Photo-Metric-Stereo

We can express the geometric dependence of the reflectance characteristics of a surface in terms of the slope components p and q, used as axes in gradient space [1]

$$p = \partial z/\partial x \qquad (1)$$
$$q = \partial z/\partial y$$

If we take the direction from the surface to the viewer as the direction of the z-axis, then the reflectance properties of a surface patch depend on (p,q), the direction of the surface normal and (p_s,q_s), the direction to the source [1]. Each point in gradient space, corresponds to a particular surface orientation (based on the direction of the viewer).

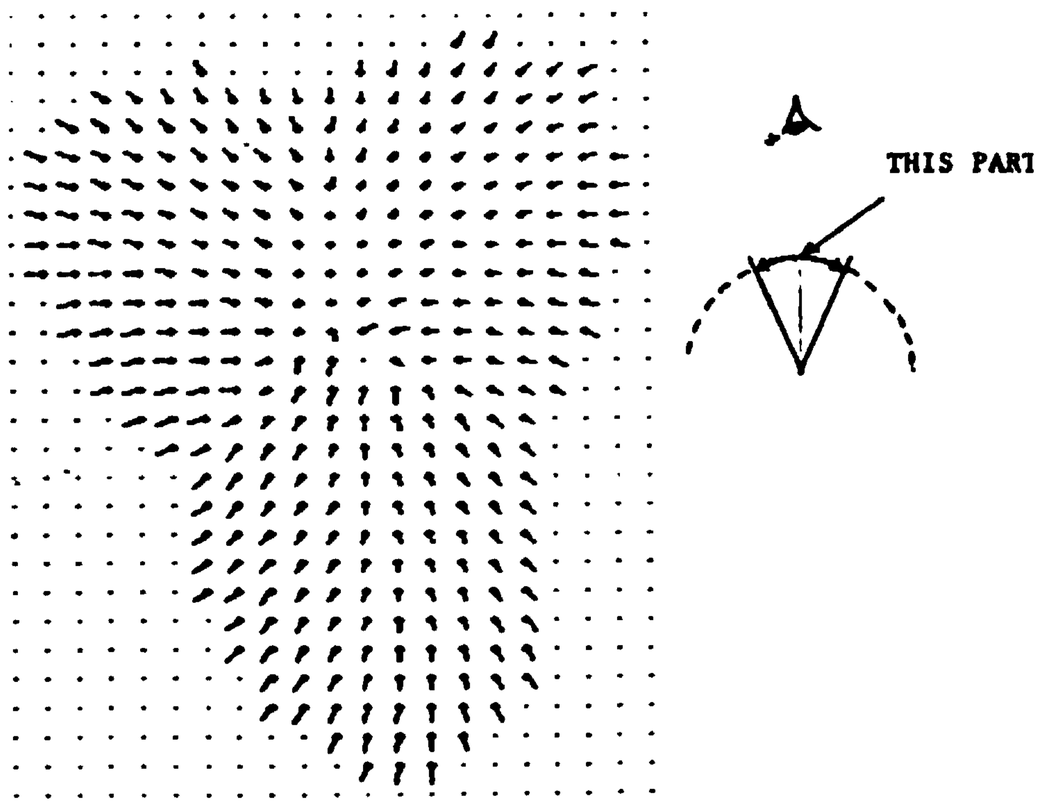


Fig. 2 Orientations of surface patches on a sphere obtained using the technique. Each needle, shown striding up out of a point, represents the orientation of a surface normal.

If we know the reflectance characteristics of an object, we can calculate how bright a surface element with that orientation will appear. This brightness distribution on the gradient space is denoted by R(p,q), and is called as "the Reflectance Map".

Using the reflectance map, the basic imaging equation is

$$E_p(x,y) = R(p,q,p_s,q_s). \quad (2)$$

where $E_p(x,y)$ is the brightness (image irradiance) in the image-forming system at the point (x,y) in the image plane, provided that we can use the orthographic projection; in other word, the object is small compared with the distance to the source and the image-forming system. This equation contains two unknown variables p, q If we change (p_s,q_s) keeping the other condition as same, we can solve for p and q from the resulting system of (non-linear) equations at the point (x,y), namely, if we take more than one image with varying illumination condition, but from the same direction, we can obtain a surface orientation at the point (x,y). This is the principle of photometric stereo [2]

2 Total Schema of the System

The technique contains two kinds of tasks; one is the off-hne(pie computing) job and the other is the on-line(real-time) job which is rather simple compared with the off-line job The simplicity implies rapid calculation, as desired in any hand-rye system The off-line |oh consists of making the reflectance map, constructing a lookup table, and a calibration table of brightness The on-line job consists of readme; the image brightness and determining orientations of a surface patch based on the lookup table, Fig. I shows information How between the on-line |ob and the off-line job

3 Calculation of the Reflectance Map

One of the main (joints of our discussion here is that we consider only specular components of reflectance when we calculate the retlectance map, since most industrial materials are made of metal and have strong specularity with very weak lambertain characteristics We cannot treat such kind of materials as a usual lambeman model Meanwhile, ratios of specular compornets to lambeman componnets are usually about 50 to 100 This situation allows us to consider only specula l commponets

For a specular surface and an extended source, we can obtain brightness distribution on the gradient space from the definition of the bi-directional reflectance distribution function (BRDF)and the transformation from the local coordinate system to the viewer coordinate system [1]

$$R(p_n,q_n) = L_s(p_s,q_s). \quad (3)$$

where L_s is the distribution function of the source radiance and

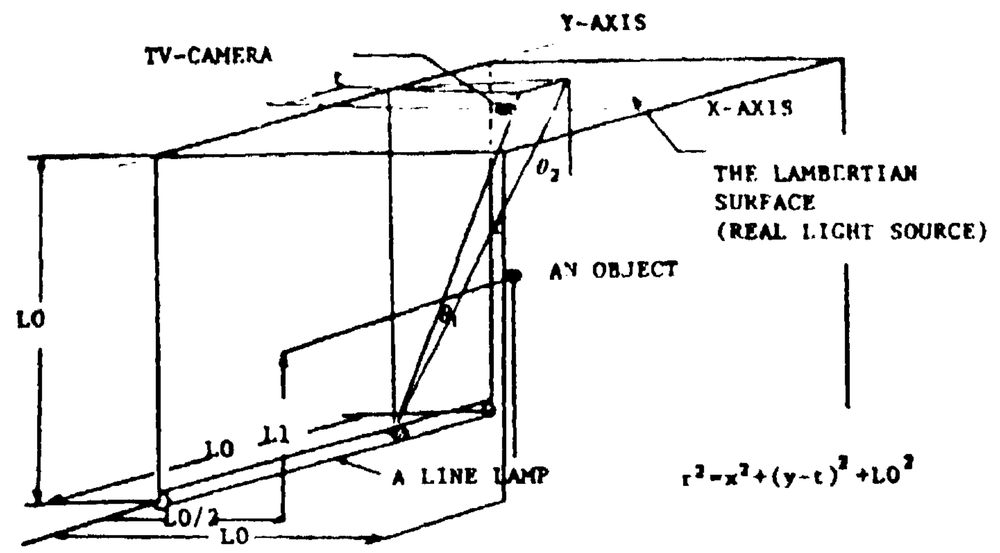
$$p_n = \partial z/\partial x$$
$$q_n = \partial z/\partial y$$

$$p_s = 2 p_n / (1-p_n^2-q_n^2)$$
$$q_s = 2 q_n / (1-p_n^2-q_n^2).$$

Thus, even though the source distribution may be complicated, only the contribution from a single direction (p_s,q_s) need be considered and one need not be concerned with the effects of other areas of the distributed light source

4 Consideration of Light Source

We use as a source plane a lambeman surface that is illuminated by a linear lamp shown in Fig 3 Though a spherical shape source easily can cover dnedions more than ninety clrgiees. it is difficult to make an ideal bnghtncss disn ibution. It is possible to cover angles of moie than ninety decree* by making a box like ^ouice In that case, however, we have to treat each plane sepaiately because the suiface normal is not differentiate at the intersection of two planes Thus, wr consider one plane suiface which has the lambeitian chaiactcnstics and is illuminated by a line source as a irpie<rnrativp ca<e



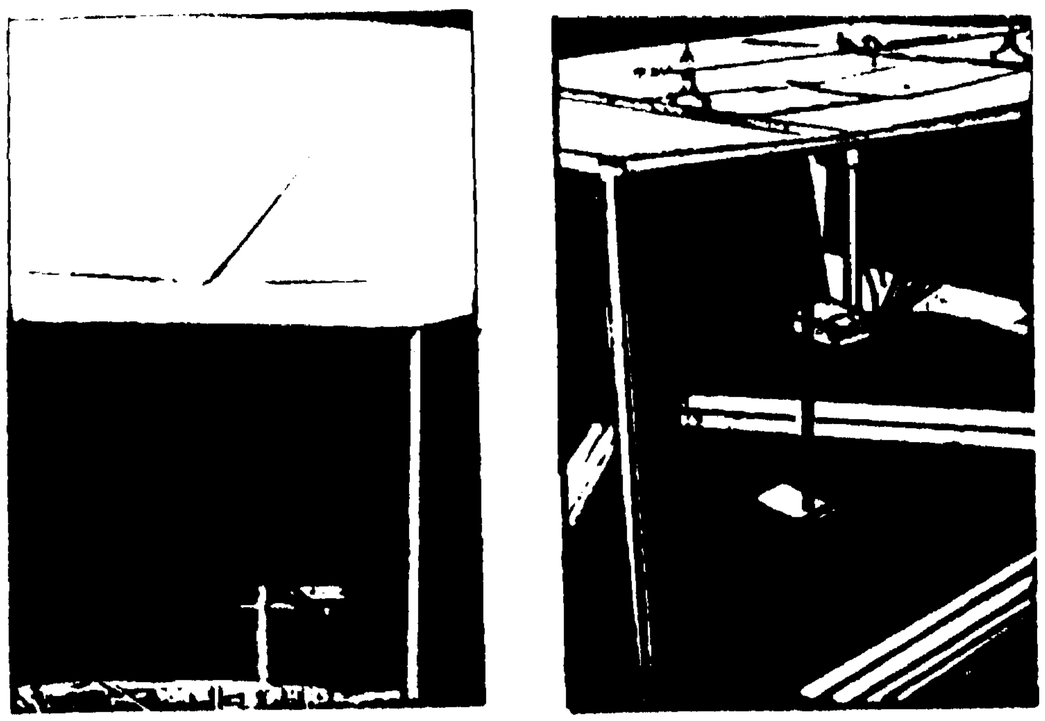


Fig. 3 The light source is a wide lambertian surface The surface is illuminated by a line lamp.

Brightness distribution on the surface is calculated by Eq. 4. Let f be the flux rate per unit source length [watt/(m sr)]. total irradiance E [watt/m^2] is

$$E=\int f\cos\theta_1\cos\theta_2/r^2 dt, \qquad (4)$$

where $\cos\theta_1=\sqrt{x^2+l_0^2}/l$ and $\cos\theta_2=l_0/l$ and l_0 is the distance from the lamp and the source plane.

We put an object just under the point $(x,y)=(0.5l_0,0.0)$. If we denote the distance from the surface to the object as l_1,

$$x=l_1p_s+l_0/2$$
$$y=l_1q_s \qquad (5)$$

Finally, by using Eqs 3,4 we can get a reflectance map

$$R(p_n,q_n) = [f l_0/2\{(l_1p_s+l_0/2)^2+l_0^2\}][$$
$$\tan^{-1}\{(l_1q_s+l_0/2)/\sqrt{(l_1p_s+l_0/2)^2+l_0^2}\}$$
$$-\tan^{-1}\{(l_1q_s-l_0/2)/\sqrt{(l_1p_s+l_0/2)^2+l_0^2}\}$$
$$+\{\sqrt{(l_1p_s+l_0/2)^2+l_0^2}(l_1q_s+l_0/2)\}$$
$$/\{(l_1p_s+l_0/2)^2+l_0^2+(l_1q_s+l_0/2)^2\}$$
$$\{\sqrt{(l_1p_s+l_0/2)^2+l_0^2}(l_1q_s-l_0/2)\}$$
$$/\{(l_1p_s+l_0/2)^2+l_0^2+(l_1q_s-l_0/2)^2\}]. \qquad (6)$$

where p_s and q_s are as defined above

5 Construction of the Lookup Table

The most convement method for converting; a triple of brightness to an orientation is fust to make a lookup table from the reflectance map We can lookup the table by using a tuple Each entry of the tablc contains a paticular surface orientation corresponding to a tuple.

We construct the lookup table by using the Newton method. In the case of a lambertian reflector and point sources an elegant method exists [2] In our case, however, we have to calculate it numercally. Namely, we solve three expressions of Fq 6 with lespect to (p.q) The three expressions arc different in then source directions (p_s,q_s)

From the lookup table we can get the surface orientation. The lookup table is actually two triangular two dimensional mattress Although it is possible to make a three dimensional lookup table in which each dimension corresponds to the brightness of a surface patch under three sources, the weakest brightness likely contains measurement error and we only use it to determine a solution between two altrinatives caused by the non-linearity. Thus, the lookup table can be two dimensional.

C Information Flow in the On-line Job

Image brightness is obtained from the TV camera. Actually, we took mote than one picture per light source, and arrays corresponding to the same light source were averaged. The resulting these brightness arrays, one for each light source, were input to the photometric stereo system.

We searchs each array for its maximum brightness If objects are compart and the visual angle is wide enough, the image array always contains the brightness corresponding to the maximum source brightness So. by using these values, we can cancel the eftect of varying albedo and can identify the maximum value in an image array with the value 1.0 in the lellectance map

Wc look up an entry using the two largest brightness values. To increase the accuracy of computation and economy of memory use. the first dimension represents the largest image brightness The second dimension represents to the second largest. Since the uon-linearity fives rise to two solutions for each brightness pair, each mesh of the matrix contains two alternative solutions. Each alternative solution contains the corresponding suiface orientation and the smallest image brightness In order to decide which one of the two alternatives is the teal one, we compare the distance between the actual thud image brightness and the element of the matrix. and choose the solution corresponding to smaller distance among the two solutions

7 Experiment and Discussion

Experimental results are shown in Fig 2 The algorithm reads the tablc twice by exchanging second brightness and thud brightness and determines the solution by averaging.

A direct application of out technique is ah industrial hand-eye system that picks up an object out of a jumble of material Although this technique cannot detect the surface orientation when there is mutual illumination, the technique recognizes this condition and does not produce an erroneous result. This is not so harmful is the result of mis-determination of the orientation. Because the misunderstood guidance to a hand may cause the hand or materials uncoverable able physical defects

Another application of our technique is to inspect surface condition of metals. If a suiface has a crack, stain, or finger print, the image brightness triple yields insistent values for the suiface oiientation in the area of the blemish

REFERENCES

[1] Horn. B and Sjobeig. R. " Calculating the Reflectance Map " AI-Memo 498, AI Lab. MIT, October, 1978.

[2] Horn. B. Woodham, R, and Silver, B. '' Determining Shape and Reflectance Using Multiple Images." AI-Memo 490. AI Lab. MIT, August, 1978.

A PARALLEL SEARCHING SCHEME FOR MULTIPROCESSOR SYSTEMS AND ITS APPLICATION TO COMBINATORIAL PROBLEMS

Masaharu Imai
School of Information Eng.
Toyahashi University of Technology
Toyohashi 440, Japan

Yuuji Yoshida
Computation Center
Nagoya University
Nagoya 464, Japan

Teruo Fukumura
Faculty of Engineering
Nagoya University
Nagoya 464, Japan

In this paper, we propose a parallelized computational scheme called *Parallelized Depth-First Algorithm* (PDFA) for Branch-and-Bound (B&B) method that works on multiprocessor systems. It Is shown theoretically that the space requirement of PDFA on p processing units is at most *p* times as much as that of the sequential B&B algorithm with the depth-first search function. Moreover, from our experimental results through simulation, it is known that the computation time of PDFA on *p* processing units can be resuced to less than 1/p that of the sequential B&B algorithm with the depth-first search function. We name this reduction effect in the computation time *Acceleration Effect,*

1. INTRODUCTION

Branch-and-Bound (B&B) method is one of the most general thchnique to solve combinatorial optimization problems; in particular, it is remarkable that B&B method is closely related to the problem solving methods used in Artificial Intelligence. [3]

In this paper, we propose a parallelized computation scheme called *Parallelized Depth-First Algorithm* (PDFA) for B&B method that works on multiprocessor systems. Next, we show that the space requirement of PDFA on p processing units is at most p times as much as that of the sequential B&B algorithm with the depth-first search function. And, we also show that the computation time of PDFA on p processing units can be reduced to less that 1/p that of the sequential B&B algorithm using the depth-first search function.

2. DEFINITIONS AND NOTATIONS

The multiprocessor system on which PDFA is assumed to be implemented consists of the following components (see Fig. 1):

(i) p Processing Units (PUs) with Private Momory Unit (PMU) each;
(ii) A shared Common Memory Unit (CMU).

In this model, PMU # i is restricted to be accessible only by PU#i, for each i, i ■ 1, 2, ... , p. On the other hand, CMU is accessible by any PU. Moreover, we also assume:

(i) Each PU computes independently of other PUs;
(ii) The communication between PUs is carried out only through CMU.

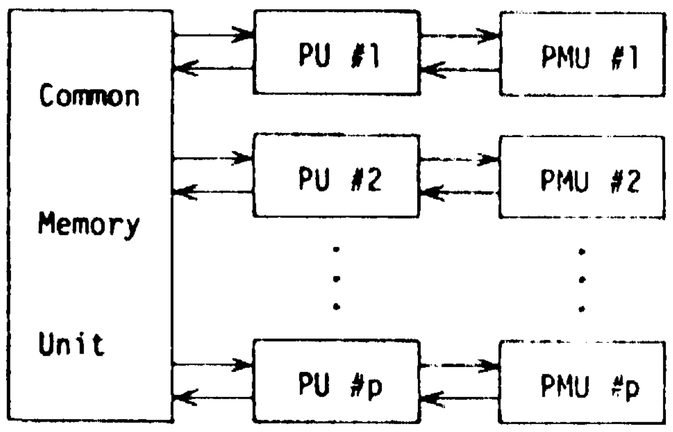


Fig. 1 Multiprocessor System
PU: Processing Unit
PMU: Private Memory Unit

In the followings, let P_0 denote a minimization problem with *n* *k*-valued variables. The goal of the problem is to find out one of the optimum solutions of P_0 and its objective value. Let *I* be the current set of open subproblems generated by a B&B algorithm. For each subproblem P. in I, we define the functions *cost, lb, h* and *depth* as follows:[1]

(i) Let *cost*(P_i.) be the optimal value of the objective function for P..
(ii) Since the actual value *ot* *cost*(P_i) cannot be known before P_i. has been completely solved, we use 1b{p_i.) as its lower bound under the assumption that *lb(?,)* should be less than or equal to *cost*(P_i.), for all P. in I. [1]
(iii) Wnen some additional information on *cost* (P_i) is available, we can construct a more powerful estimation than *lb.* We denote it h(P_i.). Without loss of generality, we assume h(P_i.) MP.) if i ≠ *j.* Function *h* is usually called heuristic *function.*

(ix) Finally, let *depth*(P_i) denote the number of variables whose values are already assigned in P_i.

There are several kind of search functions used in B&B method. One of the most popularly used search functions is the depth-first search function s_D. It selects the subproblem P_i that has the smallest h value among those with the largest depth in I.

3. PARALLELIZED DEPTH-FIRST ALGORITHM

In PDFA described below, following variables are stored in CMU for common use:

I: The set of open subproblems generated through PDFA;
Xopt: The currently best solution (a candidate of the optimal solution);
Copt: The cost of *Xopt*, i.e., *cost*(*Xopt*);
act: The number of "active" PUs, where "active" means that the PU is processing a subproblem.

All accesses to any of these variables should be controlled under mutual exclution.

PDFA is described as follows:
Phase 1: Set $I \leftarrow \{P_0\}$, $Xopt \leftarrow \phi$, and $act \leftarrow 0$.
Phase 2: Do Procedure P in parallel on each PU. Algorithm terminates when no PU is active.

Procedure P is described as follows:
Step 1: If $I = \phi$ and $act = 0$ then halt. Otherwise, if $I = \phi$ and $act > 0$ then repeat this step, else go to Step 2.
Step 2: Set $act \leftarrow act + 1$, $P_i \leftarrow s_D(I)$, and $I \leftarrow I - \{P_i\}$.
Step 3: If P_i is terminated then go to Step 5.
Step 4: If $lb(P_i) < Copt$ then decompose P_i into k subproblems and add them to I. Go to Step 6.
Step 5: If $cost(P_i) < Copt$ then set $Xopt \leftarrow P_i$ and $Copt \leftarrow cost(P_i)$.
Step 6: Set $act \leftarrow act - 1$ and go to Step 1.

In the followings, p-PDFA denotes PDFA with p PUs. It is obvious that 1-PDFA is identical to the sequential B&B algorithm using the depth-first search function.

4. SOME PROPERTIES OF PDFA

In this section, we discuss the space- and time-requirements of PDFA.

4.1 SPACE REQUIREMENT

Let M_p be the largest value of all $|I|$'s obtained during the computation of p-PDFA to solve P_0. Then, the following theorem holds.

Theorem 1: $M_p \leq p \times \{(k - 1) \times n + 1\}$

According to Theorem 1, the memoty space needed to implement p-PDFA is only $O(p \times k \times n)$, which is just p times that of the sequential B&B algorithm using the depth-first search function. It should be also noted here that the space requirement of the sequential B&B algorithm using any other search functions such as the best-bound, the breadth-first, or the heuristic are $O(k^n)$.

4.2 TIME REQUIREMENT

Let K_p be the number of subproblems processed by p-PDFA to solve P_0. It is reasonable to employ K_p to estimate the time requirement of p-PDFA. And, let T_p be the computation time needed to solve P_0 by p-PDFA. We assume that T_p satisfies $T_p = c \times K_p / p$, for some constant c.

One of the most significant results to be mentioned in this paper is:

T_p can be reduced to less than T_1/p, if p is properly selected.

The reason is briefly stated as follows:
(i) Let C_p be the cost of the first feasible solution of P_0 found by p-PDFA. Then, it is obvious that as p increses, C_p converges to the optimal objective value of P_0.
(ii) The sooner a good feasible solution is found, the lesser K_p becomes. And accordingly, T_p will become less than T_1/p.
(iii) But if p increses too much, K_p will also increses, since p-PDFA may process unnecessary subproblems to find an optimal solution.

We name this reduction effect in the computation time of p-PDFA compared to T_1, *Acceleration Effect*.

5. EXPERIMENT

Simulation experiment of p-PDFA was performed for a model of randomly generated 0-1 combinatorial optimization problems.[1] The purpose of this experiment is to know how *Acceleration Effect* appears in PDFA. In this simulation, we assumed that every subproblem can be processed in uniform time, for simplicity.

The problems were generated as follows. First, we set:
(i) $cost(P_0) = 1.0$ and $lb(P_0) = 0.0$.
And the followong values were randomly generated:
(ii) Let P_j and P_k be two different sons of P_i.

Then, we set: $cost(P_j) = cost(P_i)$ and $cost(P_k) = cost(P_i) + \alpha$, where α is a non negative random number taken from the exponential distribution, $\lambda exp(\lambda\alpha)$ with $\lambda = 0.2$.

(iii) Let P_j be a son of P_i. Then, $lb(P_j)$ was taken from uniform random numbers which distribute between $[lb(P_i), lb(P_i) + 2 \times (cost(P_j) - lb(P_i))/(n - depth(P_i))]$.

(iv) Let σ be a parameter which denotes the accuracy of the heuristic function h. Then, $h(P_i)$ was taken from the normal distribution, $N(cost(P_i), ((n-depth(P_i)) \times \sigma)^2)$. Thus, a small σ implies an accurate h.

The results are summarized in Fig. 2, in which $n = 30$. Each point in this figure represents the average of 20 different test runs. And max (K*,(n +1) * p) (shown by the dot-dash-line) denotes a theoretical lower bound of K_P. Where K* is the number of subproblems whose *lb* values are less than cost(P_0).

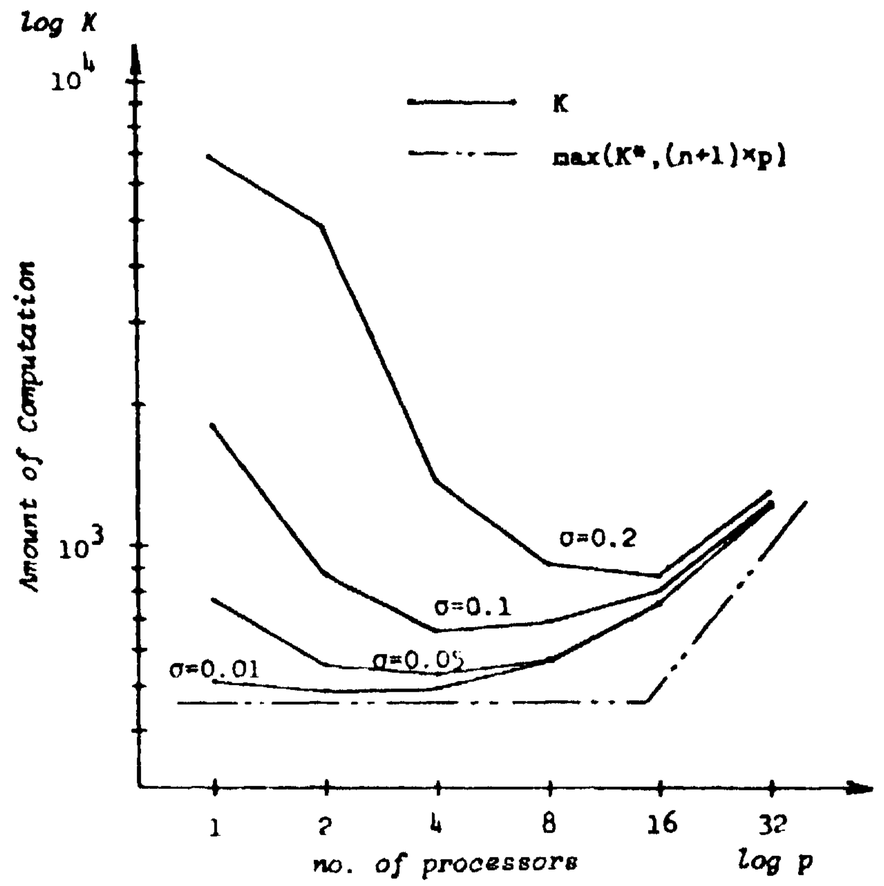


Fig. 2 Amount of Computation of PDFA

From this figure, we know:

(i) K can be reduced to less than K_1, if p is properly selected. Thus, *Acceleration Effect* appears.

(ii) *Acceleration Effect* appears more drastically when pooer heuristics are used.

6. CONCLUTION

PDFA is an attractive parallel algorithm for multiprocessor systems because of its small space requirement and especially of *Acceleration Effect.* *Acceleration Effect* appears through the parallerized B&B algorithm using the depth-first or the heuristic search function.

A similar effect as *Acceleration Effect* was also observed in the experiments on the Cm* multiprocessor system, which solved integer programming problems by B&B method.[2]

ACKNOWLEDGEMENT

Authors would like to thank Prof. N. Honda of Nagoya University for his helpful suggestions and discussions.

REFERENCES

[1] Ibaraki, T. "Theoretical Consideration on Computational Performance of Branch-and-Bound Algorithms" presented at IX_ Int. Symp. on Math. Prog. (1976).

[2] Fuller, S.H. et al. "Multi-Microprocessors: An Overview and Working Example" In Proc. of IEEE 66:2 (1978) 216-228.

[3] Nilsson, N.J. Problem-Solving Methods in Artificial Intelligence. Mcgraw-Hill (1971).

[4] Imai, M. et al. "A Parallelized Branch-and-Bound Algorithm - Implementation and Efficiency - Trans. IECE of Japan 62D:6 to be appeared.

A MODEL OF DIALOG BASED ON FUZZY SET CONCEPT

Haruki Imaoka
Department of Systems Science
Tokyo Institute of Technology
2-12-1, Oh-okayama, Meguro-ku
Tokyo 152, Japan

Michio Sugeno
Department of Systems Science
Tokyo Institute of Technology
2-12-1, Oh-okayama, Meguro-ku
Tokyo 152, Japan

Dialoq is considered as the exchange of information between two persons who speak natural language to understand each other. However natural language is so ambiguous both in expression and logic that for understanding it is very important to supplement the information through some reasoning or imagination. In this paper, we analyze this thinking process using fuzzy sets theory and approach a modelling of dialog. The dialog in the model is characterized by the set of objects talked about, the set of their attributes, etc. There are two important factors in the process of dialog: fuzzy information on the objects given by one person and the other person's fuzzy knowledge on the objects. Both fuzzy information and knowledge are expressed by fuzzy sets. The main problem in the paper is to compare these two fuzzy sets with each other. For this purpose the concept of truth qualification in fuzzy logic is useful.
The authors propose a generalized truth value instead of an ordinary one which appears in Zadeh's truth qualification rule. Then one of the two persons, say a listener in the dialog, can understano by referring the generalized truth value what the other wants to communicate.

1. INTRODUCTION

In real dialog the roles of a speaker and a listener changes alternately but in our model those are fixed for simplicity. Let us suppose the following situation. The listener acquires a sequence of information from the speaker by repeating indirect questions and guesses what the speaker is thinking. In order to supplement the obtained information by reasoning or imagination, the listener has a priori knowledge on objects which they talk about. Then listener evaluates this information by comparing it with his knowledge and repeats this process untill he is convinced that he has understood what the speaker wanted to communicate. Both the information and the knowledge are so ambiguous that fuzzy set concept is suitable for their mathematical expression.

2. PROCESS OF DIALOG

We first describe the situation of dialog between a speaker and a listener by X: a set of objects, A: a set of attributes, AV: sets of values of each attribute. For example X={Yamaguchi, Yoshinaga, Ken, ...}; names of Japanese female singers, A={Age, Appearance, Height, ..} AV(Age)={Young, Middle, Old, ...}, AV(Appearance)={Beautiful, Normal, ...}, etc.
Here it is assumed that both the speaker and the listener know the sets X and A. Let x_i be the i-th element of X, a_j the j-th element of A and $av_k(a_j)$ the k-th element of the set $AV(a_j)$ which is the set of values of the attribute a_j. Then dialog proceeds as follows.

The speaker chooses x_i out of the objects.
The listener: How about the Age of x?
The speaker: She is young.
The listener: .....

Here the listener is forbidden to directly ask the name of x_i and the speaker is allowed to answer only in the form "She is $av_k(a_j)$". Generally $av_k(a_j)$ in the form "She is $av_k(a_j)$" is a fuzzy information on the attribute a_j. Let us assume that this fuzzy information is expressed by a fuzzy set. We further assume that the listener has a priori knowledge on the attributes of the objects, e.g., "She, an object x, is about 20 years old". This type of fuzzy knowledge can be also expressed by a fuzzy set.

3. EVALUATION AND UNDERSTANDING

Let F be a fuzzy set associated with a membership function μ_F which implies a fuzzy information given by the speaker as stated in the previous section. Further let G_i be also a fuzzy set expressing the listener's fuzzy knowledge on an attribute of an object x_i.
Now we discuss the process of reasoning in which the listener finds, by using a given information F and a priori knowledges G_i, what the speaker has chosen out of the objects, where F and G_i are of course assumed to have the same universe of discourse concerned with an attribute

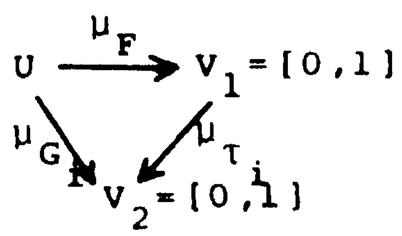
In order to select x_i out of the objects by comparing F with G_i, we can use truth qualification given by Zadeh [1].

$$x_i \text{ is F is } \tau_i = x_i \text{ is } G_i$$

where $\tau_i : [0, 1] \to [0, 1]$ is a linguistic truth value. For example "x_i is young (F) is more or less true (τ_i)" translates into "x_i is about 20 years old (G_i)". If τ_i is false, then the listener can conclude that the speaker is not thinking about the object x_i as far as the attribute under consideration is concerned. Thus the truth value of "x_i is F" with respect to "x_i is G_i" can be used as a criterion for selecting x_i.
According to truth qualification rule as is shown in Fig.1, we have

$$\mu_{G_i}(u) = \mu_{\tau_i}(\mu_F(u)) \qquad (1)$$

where u denotes a variable in the universe of discourse of F and G_i.



F is $\tau_i = G_i$

Fig.1 Truth qualification

If μ_F is one to one mapping, then μ_{τ_i} is given by

$$\mu_{\tau_i}(v) = \mu_{G_i}(\mu_F^{-1}(v)), \qquad (2)$$

where v denotes a variable in the unit interval [0, 1]. In the case that μ_F does not have the inverse, μ_{τ_i} can be obtained, for example, as follows:

$$\mu_{\tau_i}(v) = \sup_{u \in \mu_F^{-1}(v)} \mu_{G_i}(u), \text{ for } \mu_F^{-1}(v) \neq \phi \qquad (3)$$
$$= 0, \quad \text{otherwise}$$

That is, $\tau_i = \mu_F(G_i)$, the image of G_i through mapping μ_F. In general τ_i is not uniquely determined.
In this paper the authors define instead of $\mu_F(G_i)$ the following relation $R_F(G_i)$ on [0, 1] x [0, 1].

$$R_F(G_i) = \{(v_1, v_2) \mid v_1 = \mu_F(u), \; v_2 = \mu_{G_i}(u), \; u \in U\} \qquad (4)$$

The relation $R_F(G_i)$ is considered as a generalization of τ_i. Fig.2 shows the difference between $\mu_F(G_i)$ and $R_F(G_i)$. It is easily seen that the graph of the membership function of a fuzzy set $\mu_F(G_i)$ is a subset of $R_F(G_i)$.

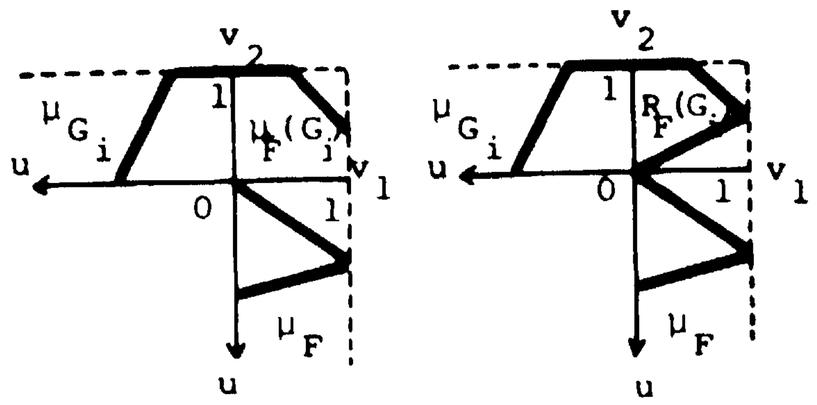


Fig.2 Difference between $\mu_F(G_i)$ and $R_F(G_i)$

Now our problem is how to evaluate $R_F(G_i)$ in comparison of F and G_i. To this end let us define the following values A, B, C and D with respect to $R_F(G_i)$.

$$A = \sup_{(v_1, v_2) \in R_F(G_i)} [1 - v_1 \wedge v_2]$$
$$B = \sup_{(v_1, v_2) \in R_F(G_i)} [1 - v_1 \wedge 1 - v_2] \qquad (5)$$
$$C = \sup_{(v_1, v_2) \in R_F(G_i)} [v_1 \wedge 1 - v_2]$$
$$D = \sup_{(v_1, v_2) \in R_F(G_i)} [v_1 \wedge v_2],$$

where A implies the distance in the sense of max-min from a point (0, 1) to its nearest point in $R_F(G_i)$ and B, C, D are those from (0, 0), (1, 0), (1, 1) respectively as is shown in Fig.3.

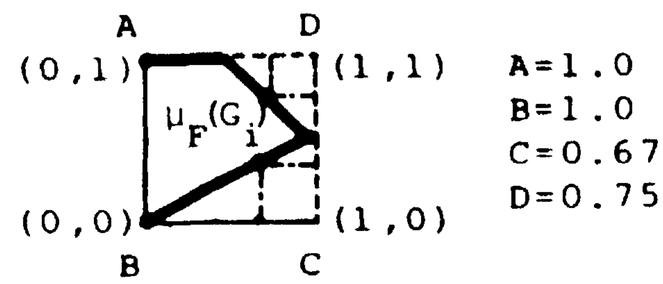


Fig.3 The meaning of A,B,C,D

Next we introduce a kind of order to the class of $R_F(G_i)$'s by a function transforming $R_F(G_i)$ to a real number such that

$$g(R_F(G_i)) = \alpha A' + \beta B + \gamma C' + \delta D, \qquad (6)$$

where A' = 1 - A, C' = 1 - C and α, β, γ, δ are constants.
We have to take into account that this order preserves the partial order concerned with τ_i since $R_F(G_i)$ includes τ_i in its special case.
The partial order $\preceq$ of the class of τ_i is defined by $\tau \sqcap \tau' = \tau$ and $\tau \sqcup \tau' = \tau' \Leftrightarrow \tau \preceq \tau'$, where $\sqcap$ is meet* operation and $\sqcup$ is join* operation [2].

---

* If A and B are fuzzy subsets defined by $A = \sum_i \mu_A(a_i)/a_i$, $B = \sum_j \mu_B(b_j)/(b_j)$, then

$$A \cup B = \sum_{i,j} (\mu_A(a_i) \wedge \mu_B(b_j)/(a_i \vee b_j)$$
$$A \cap B = \sum_{i,j} (\mu_A(a_i) \wedge \mu_B(b_j)/(a_i \wedge b_j),$$

where $\wedge$ stands for min, $\vee$ for max and Σ for union.

Let us take two fuzzy sets F_1 and F_2 and consider the special case where $R_{F_1}(G_i)$ and $R_{F_2}(G_i)$ are reduced to ordinary functions, more precisely the graphs of membership functions such as μ_{τ_i} in Eq.(2).

Here consider the following two equations.

$$R_{F_1}(G_i) \cap R_{F_2}(G_i) = R_{F_1}(G_i) \text{ and}$$
$$R_{F_1}(G_i) \cup R_{F_2}(G_i) = R_{F_2}(G_i) \qquad (7)$$

$$g(R_{F_1}(G_i)) \wedge g(R_{F_2}(G_i)) = g(R_{F_1}(G_i)) \qquad (8)$$

What we should do is to determine α, β, γ and δ in Eq.(6) so that Eq.(7) leads to Eq.(8). The operation in Eq.(7) illustrated in Fig.4 with the values A', B, C' and D.

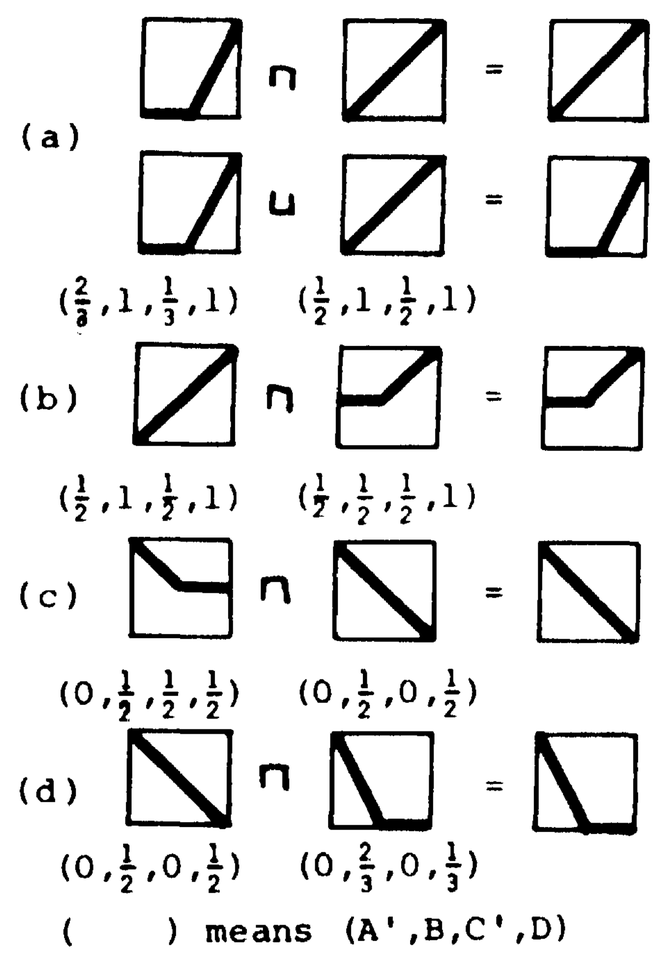


Fig.4 $\cup$,$\cap$ operation

From the above condition we have $\alpha \geq \gamma$ according to Fig.4(a). From Fig.4(b), (c) and (d) follow $\beta \geq 0$, $\gamma \geq 0$ and $\delta \geq \beta$. Finally we have the inequalites

$$\delta \geq \beta \geq 0$$
$$\alpha \geq \gamma \geq 0 \qquad (9)$$

Conversely, Eq.(8) is obtained from Eq.(7) using Eq.(9).

Now we can introduce an order to the class of $R_F(G)$'s as follows:

$$R_{F_1}(G) \prec R_{F_2}(G) \Leftrightarrow g(R_{F_1}(G)) < g(R_{F_2}(G)). \qquad (10)$$

Let us next consider the general case that the information in the attributes of the object x is sequencially obtained. Let F_j be the information on the j-th attribute and G_{ij} be a priori knowledge on the j-th attribute of the objext x_i. Let us define

$$h(x_i) = \bigwedge_j g(R_{F_j}(G_{ij})) \qquad (11)$$

Here $h(x_i)$ can be interpreted as the degree of the listener's confidence that the speaker is talking about the object x_i. Thus the listener can find, according to the value of $h(x_i)$, an x which the speaker might choose out of the objects.

4. CONCLUSION

A dialog spoken in natural language is the exchange of ambiguous information in many cases. In this paper we develope a mathematical model of a dialog in which one person understands what the partner wants to communicate through ambiguous information and knowledge. In our model of dialog, the behaviour of the listener plays the most important role. If this role is played by computer, then in a restricted sence a man can talk with computer in natural language. The key point of this paper is how to evaluate the given fuzzy information though a priori knowledge. We have presented a generalized truth value and introduced an order into its class for the evaluation.

REFERENCES

[1] L. A. Zadeh, "Fuzzy Logic and Approximate Reasoning", Syntheses 30 (1975) 407-428.

[2] M. Mizumoto and K. Tanaka, "Some Properties of Fuzzy Sets of Type 2", Inform. & Control, 31 (1976) 312-340.

DYNAMIC SPEECH DISCRIMINATION USING AN ARTICULATORY MODEL

Shun Ishizaki

Pattern Information Division
Electrotechnical Laboratory
2-6-1 Nagata~cho, Chiyoda-ku
Tokyo , JAPAN 100

The author proposes a dynamic processing method corresponding to the human one. It uses an articulator model representing the tongue and the lip movements to extract speech features from speech waveforms. He uses a linear enhancement of the articulatory movements in order to estimate their target articulatory positions, when the target positions are used for vowel discrimination, the correct rate is improved fairly well For example, ten males' symmetrical vowel V_1, V_2V_1 are analyzed and V_2 discrimination correct rate is improved from 85% to 100%. Next, hearing tests show that such dynamic process corresponds well to the human auditory system.

1. INTRODUCTION

In a study on coarticulation using formant frequencies, there are various researches[1-5], including observation of vowel's formant movements, their model and coarticulation normalization using the first and the second formants. On the other hand, coarticulation is considered to be a phenomenon of various changes in articulatory conditions of the phoneme by the preceding and the succeeding phoneme influence. It seems that the changes in articulation will depend directly on the articulatory organ's movements, such as the tongue and the lips, and that a talker allows "movement reduction of articulatory organs" so that he might not spoil the spoken content clearness by hearing his speech through his auditory organs. Such hearing system shares a feedback loop with the speaking system.

In an articulatory domain, there are various studies based on X-ray photograph observations 16-8]. But very few studies apply them to the actual speech waveform analysis data. The author starts from approximately estimated vocal tract area functions from speech waveforms, and estimate the articulatory parameters expressing the tongue and the lip movements by using an articulatory model. Then, he proposes a dynamic processing method of coarticulation and discusses its correspondence with hearing tests of the sound segmented from natural voices.

2. ARTICULATORY PARAMETER EXTRACTION[10]

The speech wave analysis is made according to the procedure shown in Fig.]. Vocal tract area functions are approximately estimated from the speech waves using an adaptive inverse filtering method of the second and the third order-critical damping system[9]. Next, the vocal tract length is obtained by the "an information criterion" based on the maximum likelihood estimation by use of an autoregressive model[11], Sizes of speaker's own articulatory organs are quantitatively estimated by using the estimated

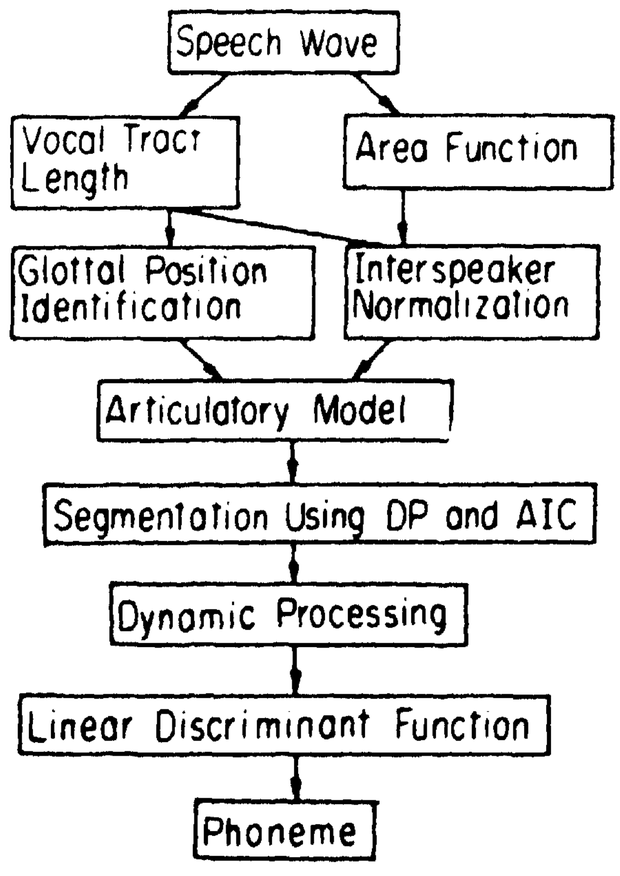


Fig. 1 Speech analysis system flow chart

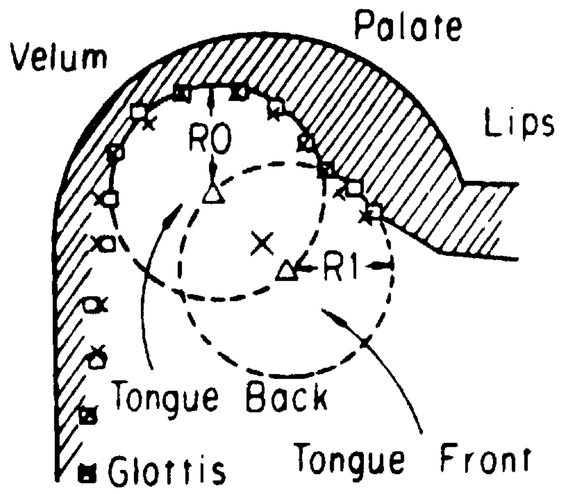


Fig. 2 Two-cylinder articulatory model
x; area function estimated from speech wave
□; area function identified with the model

vocal tract length, and individual differences are approximately normalized by uniforming the sizes to a certain constant value[10).

A vocal tract midsagittal cross sectional view is modeled as shown in Fig. 2. The "constriction position" between the upper jaw and a part of the tongue, is uniquely determined by this model, and is defined as the length measured from the glottis. By adding the "constriction area" and the "lip area" to the "constriction position", these three articulatory parameters are defined. The constriction position and the constriction area correspond well to the values measured from X-ray photographs, while the lip area corresponds well to the value measured from simultaneously recorded video tapes. Linear discriminant functions are calculated from these three articulatory parameters of isolated vowels, and discriminate them at 100% correct rate.

3. DYNAMIC PROCESS OF COARTICULATION

Coarticulation appears in almost all the phonemes of a continuous speech, and the phonetic characteristics are changed. These changed characteristics cause large error in speech recognition. When these phonemes are discriminated by only extracting static characteristics, discrimination algorithm cannot but become extensive. As mentioned later, it is known that hearing confusion phenomena are presented and caused by coarticulation. Namely, the same data can become a different phoneme by influence of the preceding and the succeeding phonemes.

Let a vowel seguence be - • • , V_1, , V_2 , V_3 , • • • and suppose that the influence of coarticulation is expressed by a linear combination of each articulatory parameters and is equal in the forward and backward directions of time. Let the values of articulatory parameters in quasi-stationary parts of triple continuous vowels V_1,V_2V_3 be P_1, , P_2 and P_3 , respectively, and the target value $\hat{P}_2$ of P_2 based on the dynamic process is approximated with the following linear formulas.

$$\hat{P}_2 = P_2 + P_{21} + P_{32} ,$$
$$P_{21} = \alpha(P_2 - P_1),$$
$$P_{32} = -\alpha'(P_3 - P_2).$$

The α (α') is a function of both time constant of the lip or the tongue dynamic characteristics and a voicing time. For the articulatory organ which is easy to move because of a small time-constant, α is set to a small value, while for the articulatory organ not so easy to move, α is set to a large value. At the same time, α is a function of a voicing time, If there is a sufficient time to voice, the articulatory organs sometimes reach near the target positions and it is, therefore, natural that α should be made small. As there is a strong correlation between the movement distance and the voicing time of each articulatory organ, α is made small when the movement distance is beyond a threshold. Examples of dynamic process are shown in Fig. 3. It shows step fucntion approximation obtained after the articulatory parameter segmentation, and Fig. 3(b) shows the target values after the dynamic process.

4. EXPERIMENTAL RESULT

Analyse are made on symmetrical triple-continuous vowel voiced at relatively fast

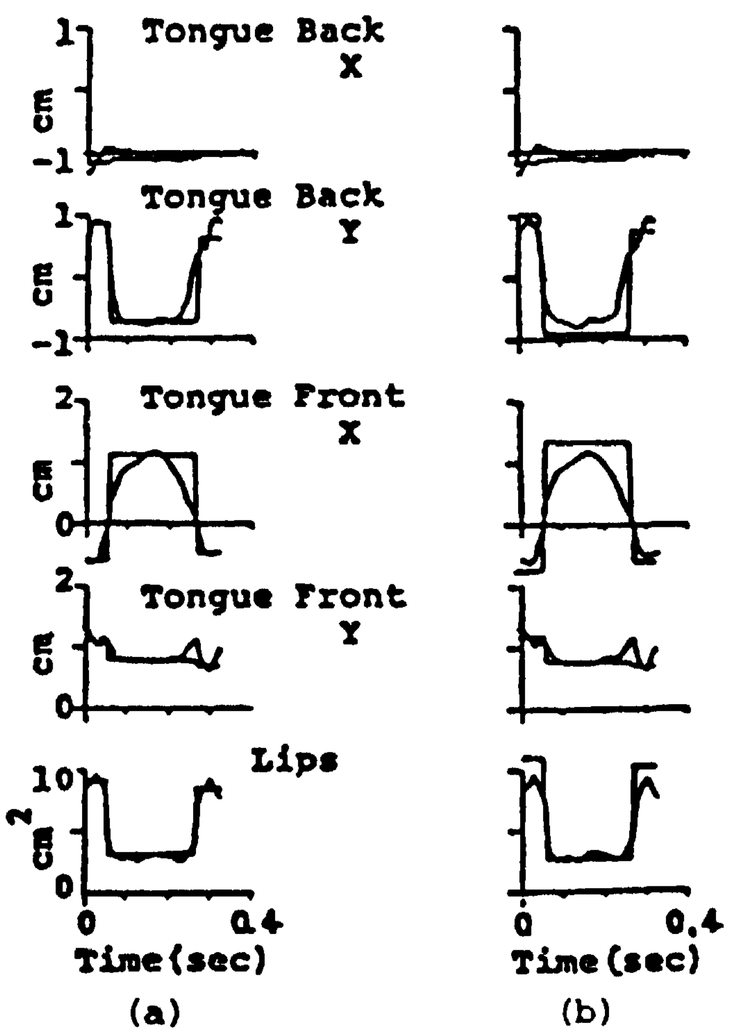


Fig. 3 /aia/'s articulatory parameters and
(a) step functions obtained after segmentation
(b) enhanced step functions with dynamic process

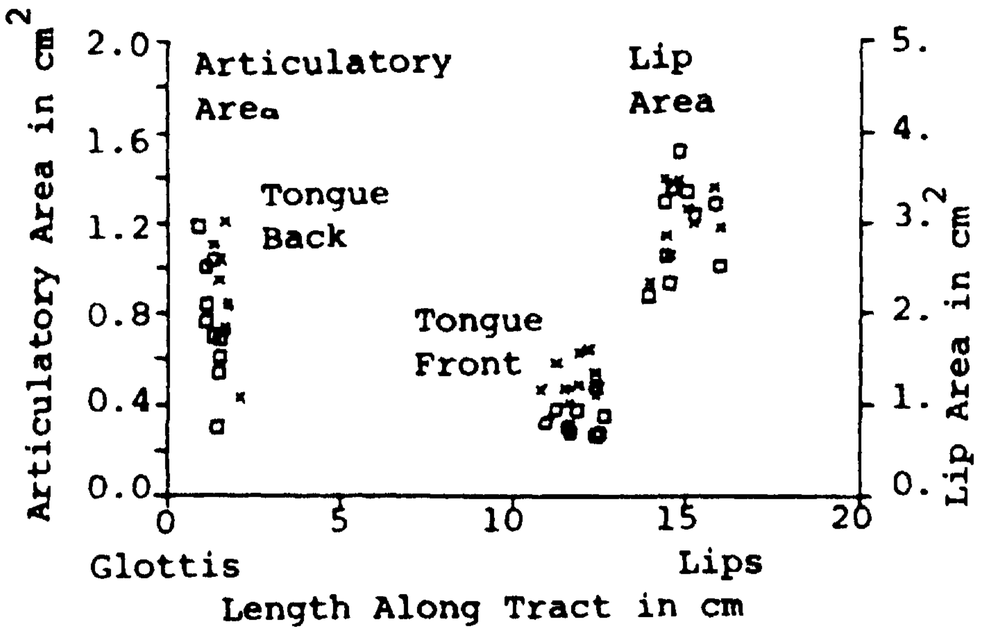


Fig. 4 Articulatory parameters of /i/ in /aia/
x ; static model, □ ; dynamic model

Table 1 V_2 confusion matrices in $V_1 V_2 V_1$ voiced with relatively fast speeds. (a) matrix before operating the dynamic process, (b) matrix after it.

(a)

	a	i	u	e	o
a	35			4	1
i		32		8	
u	1		30		9
e	1			39	
o	1		5		34
correct rate = 85%					

(b)

	a	i	u	e	o
a	40				
i		40			
u			40		
e				40	
o					40
correct rate = 100%					

speeds by ten male adults. Fig. 4 shows their articulatory parameters of /aia/. The mark "X" denotes the value before dynamic process. The mark "o" denotes the value after the dynamic process. In Fig. 4, three vowels, /i/, are erroneously discriminated as /e/, with large articulatory areas of the tongue front. However, each articulatory area can be made sufficiently small with the dynamic process mentioned in Section 3. V_2 discrimination results in V_1, V_2V_1, before and after the dynamic process are shown in Table 1.

5. CORRESPONDENCE WITH HEARING TEST

The fourth column from the left in Table 2 shows the hearing test where V_2 is segmented from V_1, V_2V_1, and presented five times to ten subjects. From the hearing tests, there are several almost complete confusion phenomena, and as compared with the fifth column from the left in the table, the human dynamic function at the perceptual process are found. When the hearing tests are compared with the discrimination results for segmented speech, most of the confusion tendencies agree with each other. In table 2, each /i/ after the dynamic process of /aia/, /uiu/ and /oio/ has common features corresponding well to [t] [13]. They are separated using a new category [i] and indicated as /i/ in Table 2.

ACKNOWLEDGEMENT

The author would like to thank Dr. Nakajima and Mr. Fuchi for their helpfull advice.

REFERENCES

[1] Lindblom,B, JASA 35, 11,p.1773 (1963)
[2] Ohman, S.E.G, JASA 39, pp.]51-168(]966)
[3] Sato, Y.and Fujisaki, H, JASJ 34, 3(1978)
[4] Kasuya, H,Tohoku Univ.Doctoral Thesis(1973)
[5] Kuwabara, H.and Sakai, H, JASJ 29,2(1973)
[6] Ohman, S.E.G, JASA 41, 2, p.310(1967)
[7] Lindblom,B. and Sundberg,J,JASA 50,4(1971)
[8) Kiritani, T, et al,JASJ 34, 3, p.132(1978)
[9] Nakajima, T, et al, Bul. Electrotech. Lab, 37, 4, p.462(1973)
[10] Ishizaki, S, Trans. IECE Japan, J61-A, 10, p.950(1978)
[II] Ishizaki, S, Trans. IECE Japan, E61,5(1978)
[12] Ishizaki, S,4-th IJCPR, p.1050(1978)
[13] Fant G, "Acoustic theory of speech production", Mouton(1960)

Table 2 Hearing tests and discrimination results of $V_1 V_2 V_1$ voiced with fairly fast speeds by a male.

Sample	V_2 Discrimination		Hearing Test	
$V_1V_2V_1$	Static Model	Dynamic Model	Stimulus V_2 (%)	Stimulus $V_1V_2V_1$ (%)
i a i	e	a	e: 98	a:100
u a u	o	a	a: 44 o:34	a:100
e a e	e	a	e:100	a:100
o a o	o	a	a: 76 0:24	a:100
a i a	e	i	e: 58 i:24	i:100
u i u	u	i	u:100	i: 94
e i e	i	i	i:100	i:100
o i o	e	i	i: 46 e:30	i: 94
a u a	u	u	u: 92	u:100
i u i	u	u	u:100	u:100
e u e	u	u	u:100	u:100
o u o	u	u	u: 54 0:46	u: 98
a e a	e	e	e: 92	e: 80 i:20
i e i	e	e	e: 68 i:32	e:100
u e u	u	e	e: 74 u:14	e:100
o e o	u	e	e: 46 u:22	e: 86 i:10
a o a	o	o	o: 78 a:22	o: 88
i o i	u	o	u: 92 o: 8	o:100
u o u	o	o	u: 84 o:16	o:100
e o e	u	o	o: 64 u:34	o: 78 u:18
Correct Rate	45%	100%	54.1%	95.9%

EXTRACTION OF MOVING OBJECT IMAGES THROUGH CHANGE DETECTION

Ramesh Jain
Department of Computer Science
Wayne State University
Detroit, MI. 48202

W. N. Martin, J. K. Aggarwal
Departments of Electrical Engineering
and Computer Sciences
The University of Texas at Austin
Austin, Tx. 78712

Abstract: This paper discusses a scheme for extracting the images of moving objects in dynamic scenes. Differencing operations are used to identify areas containing moving objects. The images of the moving objects can then be obtained by focusing the segmentation processes on these restricted areas. Thus motion is used as a cue to segmentation.

1. INTRODUCTION

Computer vision systems are usually composed of subsystems for segmentation and interpretation. Although motion has been suggested as a powerful cue for segmentation and has attracted increasing attention from researchers [6,7], little research has been concerned with the use of motion for segmenting images. Potter [8], and Fennema and Thompson [2], proposed schemes for segmentation using the velocity estimated at individual pixels. Jain et al.[3], and Jain and Nagel [4] showed that the image of a moving object may be extracted by using interframe differencing operations. Their scheme [4], however, has limited applicability because images can be extracted only when they have been displaced completely from their position in a specified reference frame, which is usually the first frame of the sequence.

We have developed an approach for recovering the images of moving objects, which exploits the image transformations due to motion. This approach has been implemented on a DEC-10 in Pascal. The input to the program is a sequence of registered frames of a dynamic scene. The output is in the form of binary images, i.e., two-dimensional masks, of the moving objects, or greylevel images obtained from the input frame using the mask. This paper describes our approach. The concepts and details of the method are given in [5].

This research was supported in part by the Air Force Office of Scientific Research under grant AFOSR 77-3190 and in part by the National Science Foundation under grant ENG 74-04986.

2. OVERVIEW

On receiving each raw image, a condensed frame is prepared [4,5]. Starting with the second frame, a difference picture, denoted DP, is prepared for each frame by comparing it with the preceding frame. Regions are found in the difference picture, and then the previous frame edge picture and current frame edge pictures are prepared. As has been shown by Jain and Nagel [4], it is possible in many cases to determine whether a DP region is due to the covering of the background, due to uncovering of the background, or due to both the covering and uncovering of the background. Let us denote these three types of DP regions as type 0, B, and X, respectively. For a given DP region, the type can be found by computing a ratio called CURPRE, which is defined as

$$\text{CURPRE} = CC/CP$$

where CC (CP) is the number of points which are both extreme points of a DP region and edge points in the current (previous) frame. The extreme points of a region are the leftmost and rightmost 1 for a row, and the topmost and bottommost 1 for a column. As has been shown in [4], this ratio is much less than 1 for type B regions, much higher than 1 for type 0 regions, and near 1 for type X regions.

After the classification of the DP regions, our approach comprises two main phases which are applied to each pair of consecutive frames in the input image sequence. In the first phase, depending on the class of the DP region, a process is applied to obtain a region in the previous (or current) frame which is an estimate of the mask of the moving object image. This phase is called the recovery process. The second phase modifies this estimate of the mask by testing whether or not the motion of an object having the image

corresponding to the mask could result in the transformations indicated by the difference picture. This phase is called the refinement process.

3. RECOVERY PROCESSES

We use region growing for 0 and B type regions to obtain masks in previous and current frames. This method is applied to the DP regions which are either 0 or B type, since a region which is classified as one of these two types is considered to be a core mask. A region is grown by taking each nonregion pixel which has a horizontal neighbor within the region and comparing its greylevel to that of the adjacent region pixel. If the pixels are similar then the nonregion pixel is added to the region. The greylevels are taken from the previous frame if the DP region is type B and from the current frame if it is type 0. A similar process is applied to nonregion pixels which have vertical neighbors within the region. Both processes are then iterated until no new pixels are added to the region. The region thus obtained represents the previous frame object mask when grown from a type B region and the current frame object mask when grown from a type 0 region.

4. REFINEMENT PROCESSES

In this section we consider various refinement processes, which are used to improve the masks obtained by the recovery process. The details of these processes may be found in [5].

SAME OBJECT. After recovering masks in the previous and current frames, for B and 0 type regions, respectively, the following method of refining masks is used. In this refinement, masks are modified by using the fact that a moving object has masks in the previous as well as current frames. Some parts of the masks may not be recovered due to noise or coincidences. The same object refinement modifies the masks to account for such lost parts.

Same object refinement is also important in processing DP regions formed by occluding objects. If region K is type 0 while regions I and J are type B and the masks of regions I and J in the previous frame have some overlap with the mask of region K in the current frame, then objects whose images do not occlude in the previous frame have occlusion in the current frame. Similarly, occlusion of objects in the previous frame may also be found.

Type X regions become important for occluding object analysis in the following case. If region K is type X while regions I and J are B type and if the masks of regions I and J in previous frames touch or overlap region K in the difference picture, then objects whose images do not have occlusion in the previous frame have occlusion in the current frame. The region K is the result of occlusion of the objects. This region is grown in the current frame to obtain a mask of the composite object in the current frame. In such a situation, if the regions I and J are type 0, then region K is grown in the previous frame.

TERMINATION REFINEMENT. Normally for a rigid object moving completely in the field of view, difference entries result at both its leading and trailing ends. This fact may be utilized for refining those regions which have been incorrectly grown by including non-object points (those points which do not belong to the object image) in the object mask during the region growing process.

GAP FILLING. For a variety of reasons the object masks resulting from the recovery processes may have holes or significant concavities. We use "gap filling" to check to see if such areas are representative of the object image or just side effects of the recovery processes when applied to the particular object shape and motion.

SAME FRAME REFINEMENT. When analyzing a sequence of frames, say frames F1,F2,...Fn, a difference picture is prepared for each successive pair of frames (i.e., F1 and F2, F2 and F3, F3 and F4, . . ., Fn-1 and Fn). The object masks *are* extracted in each frame. For each frame, F2 through Fn-1, object masks are extracted twice. For a moving object, the two masks extracted at different times but from a given frame should be the same. Due to noise and inexact recovery processes, this may not be the case in most practical situations. However, the fact that the masks are of the same object in the same frame is exploited for refining the mask.

5. RESULTS

Due to space limitation, we present the results for only one input sequence. The sequence shows the motion of two blocks in a scene containing many other objects. In Figs. 1, 2 and 3 we show three frames from the sequence taken in our laboratory. The difference picture for the first two frames is shown in Fig. 4, while the region types and CURPRE values are given in Table 1. The object masks recovered by the program are shown in Figs. 5 and 6 for the previous and current frames, respectively. After processing the third frame of the sequence, the program can apply the same frame refinement to yield the masks of the moving objects for the second frame. The moving objects start occluding in

this frame; thus a single mask is formed for the composite object, as shown in Fig. 7. The image of the objects is obtained from this mask as presented in Fig. 8. Note that all other objects have been neglected as they *dre* stationary.

6. CONCLUSION

The present version of our program was implemented to support our belief that the transformations in frames of a scene due to motion may help in the recovery of the images of moving objects. The results of the current evolutionary stage of the program indicate that it is possible to recover the images of moving objects using only low level knowledge, even from the frames of nontrivial scenes.

The processes implemented in the current version *are* based on the knowledge derived from simple idealized scenes as presented in [4,5]. It is clear that a deeper understanding of the image transformations due to motion will be useful in developing better recovery and refinement processes.

REFERENCES

[1| J.K. Aggarwal and R.O. Duda, "Computer Analysis of Moving Polygonal Images," *IEEE Trans:. . on Computers* vol. C-24, 966-976, 1975.

[2] C.L. Fennema and W.B. Thompson, "Velocity Determination in Scenes Containing Several Moving Objects," A contribution of the Central Research Laboratory of 3M Co., St. Paul, MN., and the Dept. of Computer Science, University of Minnesota, Minneapolis, MN., 1978.

[3] R. Jain, D. Militzer and H.H. Nagel, "Separating Non-Stationary Scene Components in a Sequence of Real World TV Images," *Proa, of the. 5th Int. joint Conf. on Artificial Intelli-gence,* 612-618, Aug. 1977.

[4] R. Jain and H.H. Nagel, "On a Motion Analysis Process for Image Sequence From Real World Scenes," Tech. Report, University of Hamburg, April 1978 (to appear in *1FFF Trans. pattern Analysis, and Machine Intelligence)* .

[5| R. Jain', W.N. Martin, and J.K. Aggarwal, "Segmentation Through the Changes Due to Motion," Computer *Graphices and Image Processing* (in press).

[6] W.N. Martin and J.K. Aggarwal, "Dynamic Scene Analysis, *computer Graphics and Image Processing,* vol. 7, 356-374, 1978.

[71 H.H." Nagel, "Analysis Techniques for Image Sequences," 78 Kyoto, Japan, Nov. 1978.

[8| J.L. Potter, "Scene Segmentation Using Motion Information," Computer *Graphics and Image Processing,* Vol. 6, 558-581, 1977.

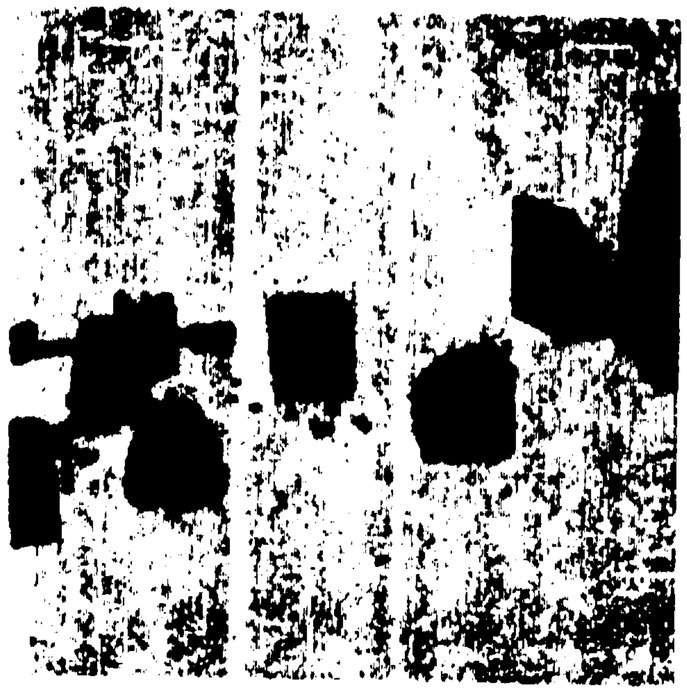


Figure 1. First frame of a laboratory sequence.

Figure 2. Second frame.

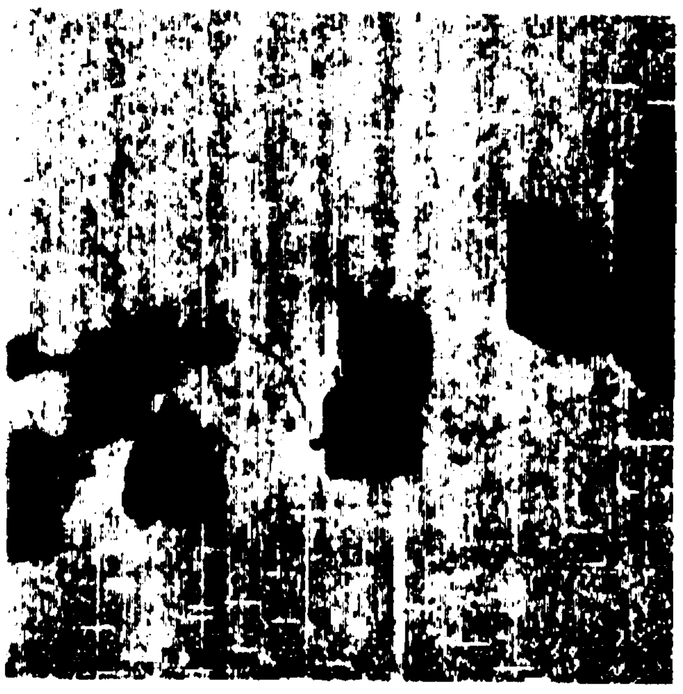


Figure 3. Final frame.

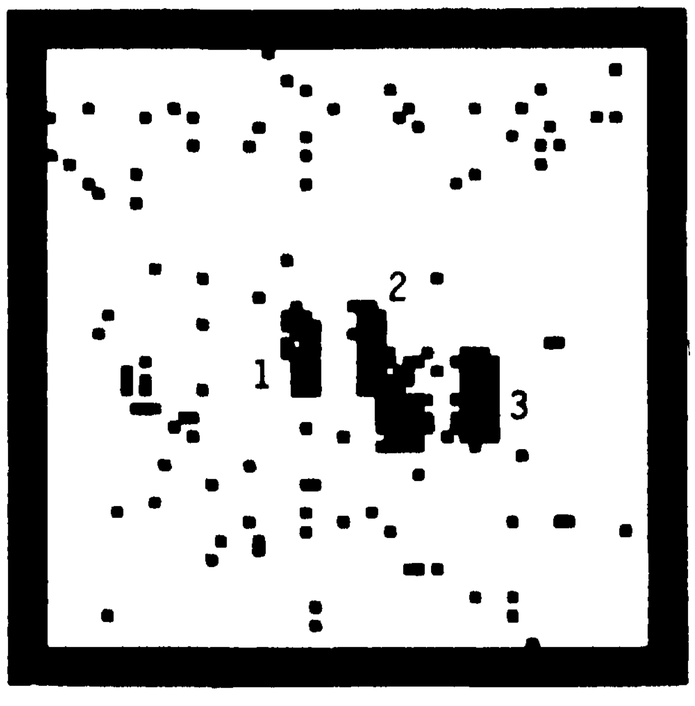


Figure 4. Difference picture for the frame pair of Figs. 1 and 2.

TABLE 1

Values for the difference picture of Fig. 4

REGION	TYPE	CURPRE
1	B	0.416
2	X	1.545
3	B	0.545

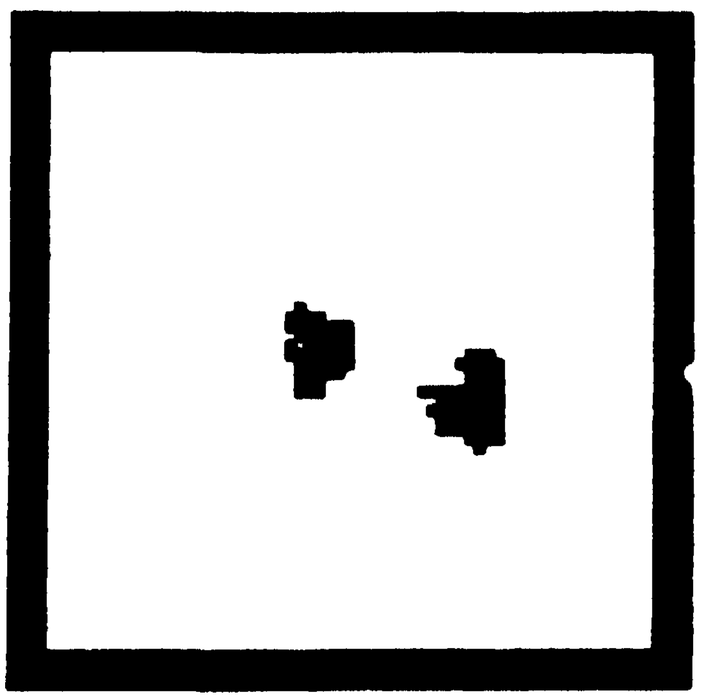


Figure 5. Previous frame object masks for the moving objects of Fig. 1.

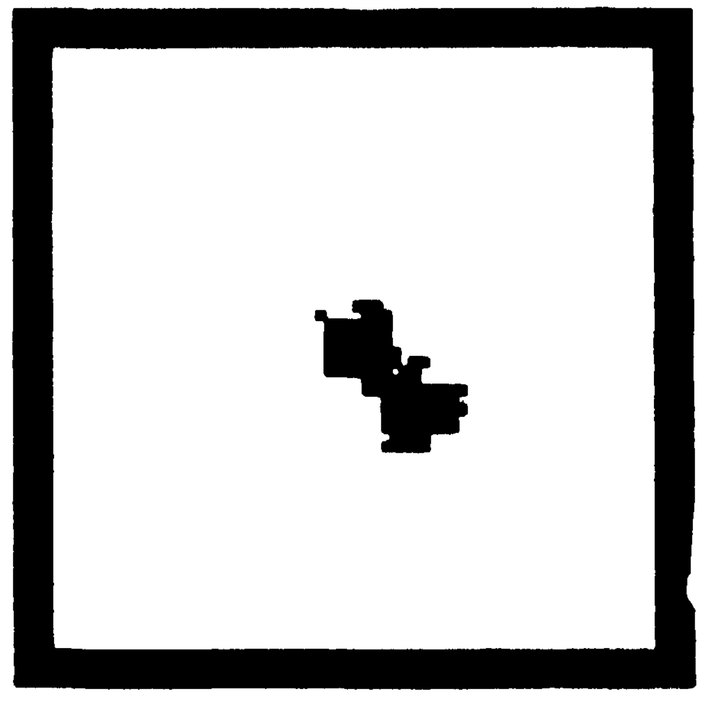


Figure 6. Current frame object mask for Fig. 2.

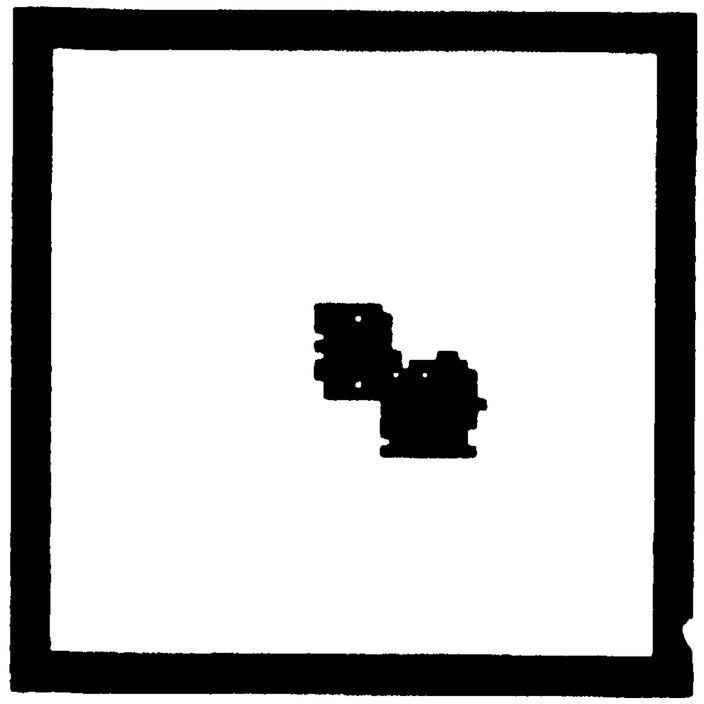


Figure 7. Refined object mask for Fig. 2

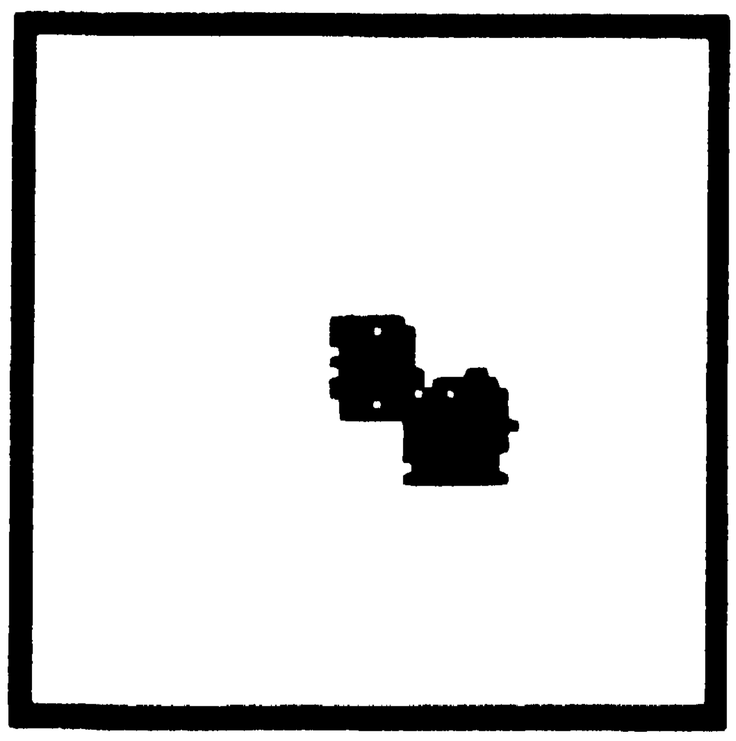


Figure 8 Moving object images extracted by the object mask of Fig. 7.

HOW TO NOT SAY "NIL" -
IMPROVING ANSWERS TO FAILING QUERIES IN DATA BASE SYSTEMS

Jurgen M. Janas
Hochschule der Bundeswehr Munchen
Fachbereich Informatik
Werner-Heisenberg-Weg 39
D-80l4 Neubiberg, Germany

If a data base query fails, i.e., if the required information is not available, then current data base management systems normally report this failure but do not provide a possible deeper reason for this failure. In this paper, a method is introduced by means of which for any query expressed in a predicate calculus based language the actual reasons for a failure may be detected; this is done by recursively inferring related queries from a failing query until no more failing queries are found, the queries obtained by this process then determine the desired answer to the original query. The method may be combined with an arbitrary query evaluation strategy and requires a minimum of additional computation. It is a good means to clear the dialog between a user and the data base system of unnecessary queries and thus contributes t*> tht usefulness and efficiency of the user interface.

1. INTRODUCTION

The relationship between a question and the appropriate answer(s) to this question has recently been investigated from several points of view; we only mention [3], [5], *[10]* . In this paper, we shall be concerned with a subproblem, namely how to find an appropriate answer to a failing data base query. We shall illustrate the purpose of this paper by the following example: Suppose somebody had asked for "all employees in the shoe department who are younger than 3o and earn more than $25.000" and the query failed, i.e., the answer was something like "nil", "0", or "there are none". If the user is seriously interested in the answer because s/he needs it for some decision s/he has to make, then s/he probably goes on asking a series of questions like the question about "all employees younger than 31 and earning more than $24.ooo" which all might be answered by "nil" as well; thus s/he will finally ask for a list of "all employees in the shoe department together with their age and salary" where another "nil" would indicate that there is no shoe department at all. Clearly it would be desirable that the system reports in its answer to the very first question that - in contradiction to the user's expectations - there is no such department. This becomes even more demanding since the number of questions which a user will have to put in order to discover the origin of a failure will rapidly increase as data base or queries become more complex.

This research was carried out while the author was still at the Institut fur Informatik der Technischen Universitat Munchen.

In this paper, we shall propose a method by means of which the reasons for the failure of a user's query may be reported to the user as the answer to his query. In section 2, we shall introduce a non-procedural predicate calculus based formal language which will be used to represent queries and answers throughout the rest of this paper. In section 3, we define how the so-called predecessors of a query (which are queries themselves) are inferred from a query; furthermore we show how recursive substitution of failing queries by their predecessors leads to an informative answer to the user's original query. In section 4, we shall discuss how the suggested method may be implemented and we shall demonstrate how it may be improved by considering the integrity constraints the data base is subject to. In a concluding section, we shall compare our approach to related research.

Throughout this paper, all examples refer to the following data base according to the relational model of data (cf. [l]) :

EMP (NAM, SAL, AGE, DN)
DEPT (DNO, DNAME)
CAR (LIP, OWN, COL)

An employee EMP has a name NAM, a salary SAL, the age AGE and works for a department DEPT which is identified by a department number DNO (or DN respectively) and which has a department name DNAME. Moreover there are cars CAR with a licence plate LIP, a color COL, and an owner OWN who is an employee identified by his name.

2. THE QUERY LANGUAGE

2.1 Conventions

Throughout this paper, we shall use the following notation:
- R and R' are non-empty relations ($R \neq R'$)
- A, A1, and A2 are attributes of R
- A' is an attribute of R'
- v is an attribute value of A'
- x and y are tuple variables for relations ($x \neq y$)
- ω is a relational operator, i.e., $\omega \in \{ =, \neq, <, >, \leq, \geq \}$

For convenience, we assume that all attributes in the data base are named differently also if they are from different relations.

2.2 Terms

The elementary constituents of our query language are terms, they are defined by the following axioms:

(T1) Any string of characters χ where
$$\chi=(x.A\ \omega\ v)$$
is called a term.

(T2) Any string of characters χ where
$$\chi=(x.A1\ \omega\ x.A2)$$
is called a term.

(T3) Any string of characters χ where
$$\chi=(x.A\ \omega\ y.A')$$
is called a term.

(T4) A string of characters is a term only if this is by reason of (T1), (T2), or (T3).

For a term χ, let $F(\chi)$ denote the set of variables which are free in χ; clearly $F(\chi)=\{x, y\}$ in the case of (T3) and $F(\chi)=\{x\}$ in the case of (T1) or (T2). Terms according to (T3) also will be called join terms, all other terms will be called simple terms if they have to be distinguished. Note that for any tuple variable occurring in a term, the relation it stands for may be uniquely determined because all attributes are named differently.

2.3 Expressions

Terms may be combined to form expressions by applying the following axioms:

(E1) If χ is a term then χ is an expression.

(E2) If α and β are expressions then
$$\chi=(\alpha \wedge \beta) \quad \text{and} \quad \chi=(\alpha \vee \beta)$$
are expressions.

(E3) If α is an expression and $x \in F(\alpha)$ then
$$\chi=\exists x\ \alpha \quad \text{and} \quad \chi=\forall x\ \alpha$$
are expressions.

(E4) A string of characters is an expression only if this is by reason of a finite number of applications of (E1) through (E3).

$F(\chi)$ may be expanded to expressions χ in a natural way: If χ is an expression according to (E2) then $F(\chi):=F(\alpha) \cup F(\beta)$ and if χ is an expression according to (E3) then $F(\chi):=F(\alpha) \setminus \{x\}$. Given an expression χ, we shall refer to the variables from $F(\chi)$ as to T-variables (target), whereas all other variables occurring in χ will be called Q-variables (qualification).

2.4 Well-Formed Sets

Expressions form the basis of our query language, but not every expression according to section 2.3 leads to a meaningful query; therefore, we restrict our further considerations to a subclass of expressions, namely the so-called connected expressions. An expression χ is called a connected expression if and only if for any Q-variable x in χ there are n+1 join terms $1_0, 1_1, \ldots, 1_n$ $(n \geq 0)$, n Q-variables $x_1, \ldots, x_n$ and a T-variable y, such that:

$$F(1_0)=\{x, x_1\} \wedge F(1_1)=\{x_1, x_2\} \wedge \ldots$$
$$\ldots \wedge F(1_{n-1})=\{x_{n-1}, x_n\} \wedge F(1_n)=\{x_n, y\}$$

Informally speaking, for any Q-variable of a connected expression there is a path of join terms which "ties" this Q-variable to a T-variable.

Let now χ be a connected expression with $F(\chi) \neq \emptyset$ and $x_1, x_2, \ldots, x_n$ the elements of $F(\chi)$ in an arbitrary but fixed order, then
$$\{ (x_1, x_2, \ldots, x_n) \mid \chi \}$$
is called a well-formed set. A well-formed set may be interpreted as the following data base query: "Find all those combinations of tuples from the relations specified by the variables $x_i \in F(\chi)$ for which the expression χ holds." Thus the well-formed set

$$\{x \mid (\exists y((y.DNO=x.DN) \wedge (y.DNAME='shoe')) \wedge ((x.AGE<30) \wedge (x.SAL>25.000)))\}$$

corresponds to the query "Find all employees in the shoe department who are younger than 30 and earn more than 25.000."

2.5 Properties of Well-Formed Sets

The query language formed by the well-formed sets corresponds to a great extent to the data sublanguage ALPHA (cf. [2]). The fact that well-formed sets cannot be used to ask for single attributes but only for complete tuples is not really a restriction but rather a tribute to a more convenient notation. The absence of explicit negation from the expressions does not mean a deficiency either, since the language contains also the inverse operator of any of its relational operators (cf. 2.1).

Our restriction to connected expressions seems to be of some consequence for the expressive power of the language; but in fact, queries formed from unconnected expressions do not seem to make much sense as one can see from the following example:

$$\{x \mid ((x.AGE>3o) \wedge \exists y(y.DNAME='shoe'))\}$$

The answer to this query would be either a list of employees older than 3o or "nil" depending on whether there is a shoe department or not; thus, if the answer to this query is "nil" one does not know whether there is no shoe department or whether there are no employees older than 3o and this seems to be a strong argument for excluding unconnected expressions from the formation of well-formed sets. Still we have to mention that for most (not all) unconnected expressions there are equivalent connected ones; this is illustrated by the following query which is semantically equivalent to the above one but formed from a connected expression:

$$\{x \mid ((x.AGE>3o) \wedge \exists y((y.DNAME='shoe') \wedge ((x.DN=y.DNO) \vee (x.DN\neq y.DNO))))\}$$

3. THE PREDECESSORS OF A QUERY

3.1 Predecessors of Natural Language Questions

If somebody asked for "all employees who are younger than 3o and own a red car" and there are no such employees then one possible situation is that there are employees who are younger than 3o and own a car but none of these cars is red and that there are red cars but all of them are owned by employees older than 3o. A user who obtains "nil" as the answer to the above question will be likely to interpret the answer in this way. Still there are several other situations which might have led to "nil":

(RI) There are no employees who own a red car.
(R2) There are no employees who are younger than 3o and own a car.
(R3) There are no employees who own a car.
(R4) There are no red cars.
(R5) There are no employees younger than 3o.

We neglect the pathological situations where there are no employees or no cars because in this case the question would have resulted from a wrong understanding of the structure of the data base.
Each of the situations (R1) through (R5) implies a negative answer to the user's question; moreover for each of these situations there is a question which corresponds to it in a natural way:

(S1) Find all employees who own a red car.
(s2) Find all employees who are younger than 3o and own a car.
(S3) Find all employees who own a car.
(S4) Find all red cars.
(S5) Find all employees younger than 3o.

Such questions are called the predecessors of the original question. In general, a question Q is called a predecessor of a question R if a negative answer to Q implies a negative answer to R.

A predecessor of a question may have predecessors itself and one can easily see that these have to be elements of the set of predecessors of the original question; thus the set of predecessors of a question is partially ordered. In the above example (S5) is a predecessor of (S2), (S4) is a predecessor of (S1), and (S3) is a predecessor of both (S1) and (S2). We make use of this property in the following definition: A question Q is called an immediate predecessor of a question R if Q is a predecessor of R but not predecessor of any of R's predecessors. The reason why we are concerned with these immediate predecessors is that stepwise substitution of a failing query by its predecessors finally leads to the actual reason for the failure like (R1) - (R5) in the example.

3.2 Predecessors of Well-Formed Sets

The questions from section 3.1 may be expressed as well-formed sets:

(Uo) $\{x \mid ((x.AGE<3o) \wedge \exists y((y.OWN=x.NAM) \wedge (y.COL='red')))\}$
(U1) $\{x \mid \exists y((y.OWN=x.NAM) \wedge (y.COL='red'))\}$
(U2) $\{x \mid ((x.AGE<3o) \wedge \exists y(y.OWN=x.NAM))\}$
(U3) $\{x \mid \exists y(y.OWN=x.NAM)\}$
(U4) $\{y \mid (y.COL='red')\}$
(U5) $\{x \mid (x.AGE<3o)\}$

Replacing a question by one of its predecessors here mainly means removing one term from the query's expression. On the other hand, not every removal of a term from a query results in a predecessor of that query; thus (y.OWN=x.NAM) must not be removed from (Uo) since the resulting expression would not be connected. Now we are going to introduce rules which specify how the predecessors of a well-formed set are obtained, i.e., which of the terms of a well-

formed set may and which may not be removed in order to yield a predecessor. Predecessors of well-formed sets correspond to a large extent to the immediate predecessors of natural language questions but they are not equivalent: The negative answer to a predecessor of a well-formed set does not necessarily imply the negative answer to the original query. Moreover predecessors are only similar to the related concepts "presumption" (cf. [6]) and "presupposition" (cf. [8]).

The above example showed that a join term may be removed from a query expression only if this does not "disconnect" the query expression; therefore, the first rule for obtaining predecessors is:

(X1) A join term must not be removed if both its free variables occur in other terms of the expression too.

This is the reason why (y.OWN=x.NAM) must not be removed from (Uo) but may be removed from (U2) thus yielding (U5). There is one exception to this rule, namely:

(X2) A join term may always be removed if there is a "parallel" join term in the expression which connects the same variables.

To give an example of this rule we add an attribute DRIVER to the CAR relation where a DRIVER is an employee identified by his name; we then might ask:

$$\{x \mid ((x.AGE<3o) \wedge \exists y(((y.OWN=x.NAM) \wedge (y.DRIVER=x.NAM)) \wedge (y.COL='red')))\}$$

i.e.,"find all employees who are younger than 3o and own and drive a red car"; according to (X2) we may remove (y.OWN=x.NAM).

Simply deleting join terms may have the effect that we obtain expressions without T-variables and hence no well-formed sets. Therefore we have to make sure that the resulting expressions always contain at least one free variable; this is achieved by the following rule:

(X3) Whenever we remove a join term which contains a Q-variable and a T-variable which does not occur anywhere else in the expression the Q-variable is made a T-variable.

A Q-variable is made a T-variable by removing the quantifier of that variable from the expression. An example of the application of (X3) is the way in which (U4) is derived from (U1).

3.3 How to Handle Disjunctions

So far we were only concerned with queries without disjunctions. When considering disjunction, we have to distinguish two situations, namely whether there is a disjunction at the top level of the query (i.e., the query expression is $\chi=(\alpha \vee \beta)$) or somewhere else in the query.

(X4) If a query is disjunctive at the top level of its query expression, then it has two predecessors which correspond to the two constituents of the disjunction.

Thus a query like:

$$\{x \mid (((x.COL='red') \wedge (x.OWN='Smith')) \vee ((x.COL='blue') \wedge (x.OWN='Brown')))\}$$

i.e.,"find all cars which are either red and owned by Smith or blue and owned by Brown" has the following predecessors:

$$\{x \mid ((x.COL='red') \wedge (x.OWN='Smith'))\}$$
$$\{x \mid ((x.COL='blue') \wedge (x.OWN='Brown'))\}$$

A negative answer to one of these predecessors does not imply a negative answer to the original query, rather a negative answer to the original query is equivalent to negative answers to both predecessors. Note, however, that a deeper reason for the failure of the original query, e.g., "there are no red cars and there is no Mr. Brown" would be detected by forming the predecessors of the predecessors.

If there is a disjunction at a deeper level of the query expression, then we need two more rules which further restrict the terms which may be removed from the query.

(X5) A simple term which is a constituent of a disjunction may be removed only if the other constituent is a simple term too.

(X6) If one constituent of a disjunction has been removed then the other constituent of that disjunction has to be removed together with it if it is a simple term.

Application of (X5) and (X6) is demonstrated by the following example:

$$\{x \mid \exists y((x.NAM=y.OWN) \wedge (((y.COL='red') \vee (y.COL='blue')) \vee (y.COL='white')))\}$$

(x.COL='red') may be removed because of (X5), then because of (X6) both other simple terms are removed too; so the only predecessors of this query are:

$$\{x \mid \exists y(x.NAM=y.OWN)\}$$
$$\{y \mid (((y.COL='red') \vee (y.COL='blue')) \vee (y.COL='white'))\}$$

Summarizing we can say that the predecessors of a well-formed set are obtained either by application of (X4) or - for every term of the query expression - by trying to remove it under consideration of (X1), (X2), (X3), (X5), and (X6). It can be shown that if B is a well-formed set representing a query which fails, then the answer "B=$\emptyset$" may be substituted by the answer

$$B_1=\emptyset \wedge B_2=\emptyset \wedge \ldots \wedge B_n=\emptyset$$

where B_i are those predecessors of B which are empty themselves. The advantage of this latter answer is that it is more informative than the original one because each Bi is either a super-

set of B or a set whose emptiness implies the emptiness of B; if B is disjunctive at the top level then both answers are equivalent. Since each B. is a well-formed set and thus a query itself, we may apply the process of answer substitution recursively until each B. in the answer has either non-empty predecessors only or no predecessors at all. For an example, we have to refer to [4].

4. IMPLEMENTATION

4.1 The Elementary Algorithm

In this section, we shall investigate how the predecessors of a given query may be computed. We start by taking the point of view that the evaluation of a query and the computation of the predecessors of that query are kept completely apart from each other; thus, as long as a query does not fail nothing has to be done. In the case of failure, the following algorithm will provide the appropriate answer according to 3.3. The data structures the algorithm operates on are three sets whose elements are well-formed sets; there is one set (PA) containing possible answers to the query, one set (DA) containing definite answers to the query, and an auxiliary set (AS). At the beginning PA contains the failing query whereas DA and AS are empty. The basic idea of the algorithm is to recursively replace the elements of PA by their predecessors as long as there are any and to move them to DA otherwise. In detail the algorithm looks like this:

(AI) Choose an element α from PA and put the predecessors of α into AS (a is the next element which either will be moved to the definite answers or will be replaced by its empty predecessors) .

(A2) If AS is empty then go to (A4) (If there are no more predecessors of *α* then we have to test whether α is a definite answer or not) .

(A3) Remove an element B from AS; if Bnot=Φ then immediately go to (A2) otherwise remove a from PA - if possible -, insert B in DA and then go to (A2) (As long as a predecessor of a is non-empty, nothing has to be done, otherwise the predecessor replaces a in the set of possible answers).

(A4) If α is contained in PA then remove it from PA and insert it in DA (If α is still a possible answer then it was not replaced by any of its predecessors and therefore becomes a definite answer).

(AS) If PA is non-empty then go to (AI), otherwise stop (The procedure does not terminate as long as there are still possible answers).

It can be shown that DA is non-empty after the algorithm has terminated and that it contains exactly those well-formed sets B_i which form the appropriate answer

$$B_1=\emptyset \wedge B_2=\emptyset \wedge \ldots \wedge B_n=\emptyset$$

to the user's query.

4.2 Performance Improvements

The above algorithm is a pretty obvious method but on the other hand it does not work very efficiently. One of the drawbacks of the method as it stands is that a well-formed set may be computed more than once if it is a predecessor of more than one possible answer. A simple way to overcome this redundancy is to keep a list of already computed predecessors together with the information whether they are empty or not. A better solution would be to support the algorithm by a strategy which makes sure that no well-formed set has to be evaluated more than once. An example for such a strategy would be to choose always that well-formed set with the "longest expression" in (AI); then no well-formed set could be inserted in PA again after it has already been removed from PA.

If we abandon the (naive) point of view that query evaluation and computation of predecessors are kept apart from each other, the algorithm may be improved considerably: Note that before our algorithm can start the whole query has already been evaluated, namely in order to find out that it fails. Evaluation of a query consists of a sequence of elementary operations such as projections, joins, and selections whose results are sets of tuples corresponding to certain well-formed sets. It is easy to see that - no matter what query evaluation strategy is applied - these well-formed sets occur somewhere among the predecessors of the failing query. Therefore, if information about the well-formed sets which already have been computed by the query evaluator is made available to the above algorithm, then the number of predecessors which have to be tested for emptiness will be further reduced.

4.3 Considering Integrity Constraints

Roughly speaking, integrity constraints are restrictions which the data base is subject to in order to be in a well-defined state. An example of an integrity constraint on our data base would be that for every tuple of the relation CAR there has to be a tuple of the relation EMP such that the values of their attributes OWN and NAME are identical. Such integrity constraints may be used to simplify the predecessor structure of a given query; we illustrate

this by means of the following example:

$$B=\{x \mid ((x.COL='red') \wedge \exists y((y.NAM=x.OWN) \wedge (x.AGE<30)))\}$$

i.e., "find all red cars owned by employees who are younger than 30." According to section 3 the predecessors of B are B1 and B2, those of B1 are B3 and B4, and those of B2 are B4 and B5

$$B1=\{x \mid \exists y((y.NAM=x.OWN) \wedge (y.AGE<30))\}$$
$$B2=\{x \mid ((x.COL='red') \wedge \exists y(y.NAM=x.OWN))\}$$
$$B3=\{y \mid (y.AGE<30)\}$$
$$B4=\{x \mid \exists y(y.NAM=x.OWN)\}$$
$$B5=\{x \mid (x.COL='red')\}$$

Now, if every car has to have an owner then obviously B4 is the set of all cars and therefore cannot be empty; moreover B2 is identical to the set of all red cars, namely B5. Consequently, we would like to obtain B1 and B5 as the predecessors of B and B3 as the only predecessor of B1. Integrity constraints may be used to remove certain join terms from the predecessor queries, we thereby reduce the average number of predecessors which have to be checked for emptiness in the case of failure and hence further improve the efficiency of our procedure. For a more thorough discussion of this point we have to refer to [4].

5. CONCLUSION

The work described in this paper is closely related to the research reported in [9] and in [7]. In *[9],* a formal query language called HI-IQ for a CODASYL-type data base is used to express queries and to compute deeper reasons for a possible failure of a query. Still, HI-IQ supports a much narrower range of queries than well-formed sets do; moreover the procedure that computes the predecessors of a query does not obtain all possible predecessors of an arbitrary query. The method for obtaining an appropriate answer to a failing query will on average require more computation than the procedure suggested here because small deviations of the user's expectations from the actual data require more computation than greater deviations do. Finally, the method proposed in [9] does not make any use of integrity constraints during the computation of predecessors.

In [7], the problem of how to find an appropriate answer to a failing query is investigated for natural language queries: A natural language query is translated into a so-called meta query language (MQL) which is a graphical representation of the query still close to natural language; from this MQL representation the predecessors of a query are computed. A query represented in MQL then has to be translated into the actual query language before it may be evaluated. Contrary to that approach, we have shown in this paper that informative answers to failing data base queries may be obtained from a purely formal query representation and that clues provided by natural language are not actually needed for this task. Nevertheless, we have to mention that our method will gain its full strength only if the informative answers will be rephrased in natural language.

REFERENCES

[I] Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." Communications ACM 13:6 (197o) 377-387

[2] Codd, E.F. "A Database Sublanguage Founded on the Relational Calculus." In Proc. 1971 SIGFIDET Workshop. ACM, New York, 1971, pp. 35-68

[3] Hobbs, J.R. and Robinson, J.J. "Why Ask?" Technical Note 169, SRI international, 1978

[4] Janas, J.M. "Towards More Informative User Interfaces." In Proc. 5th Int. Conf. on Very Large Data Bases. Rio de Janeiro, Brasil, 1979

[5] Joshi, A.K. and Weischedel, R. "Computation of a Subclass of Inferences: Presupposition and Entailment." AJCL microfiche no. 63 (1977)

[6] Kaplan, S.J. "On the Difference between Natural Language and High Level Query Languages." Proc. ACM'78. Washington, D.C., 1978

[7] Kaplan, S.J. "Cooperative Responses from a Portable Natural Language Data Base Query System." PhD thesis, Dept. of Computer and Information Sciences, University of Pennsylvania, 1979

[8] Keenan, E.L. "Two Kinds of Presupposition in Natural Language." In Studies in Linguistic Semantics (C.J. Fillmore and D.T. Langendoen, eds.). Holt, Rinehart, and Winston, New York, 1971

[9] Lee, R.M. "Conversational Aspects of Database Interactions." In Proc. 4th Int. Conf. on Very Large Data Bases. Berlin, Germany, 1978

[10] Lehnert, W.G. "The Process of Question Answering." Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1978

CENTERED LOGIC: THE ROLE OF ENTITY CENTERED SENTENCE REPRESENTATION IN NATURAL LANGUAGE INFERENCING*

Aravind K. Joshi
Department of Computer and Information Science
R.268 Moore School ,
University of Pennsylvania
Philadelphia, PA 19104 U.S.A.

Steve Kuhn
Department of Philosophy
Georgetown University
Washington, D.C. 20057 U S.A.

ABSTRACT: We will briefly describe the role of entity centered structure (ECS) of sentences in natural language inferencing. The basic structure of sentences in discourse, generally singles out an entity, to be called center, among all those which are the arguments of the main predicate. ECS makes n-ary predicates look like monadic by temporarily masking their structure, thereby affecting the relative ease with which certain inferences *are* made and information is retrieved. This short paper deals with a preliminary formulation of a system designed to capture these ideas and contains several examples of how some natural language inferences can be represented in the system. Formal properties of the system are under investigation.

1. Introduction: A uniform mechanism that subsumes all inference mechanisms involved in problem solving in general may be adequate to characterize inferences in language in some sense (analogy: Turing machines characterize all computable functions); however, it will not shed much light on those mechanisms that are language relevant and presumably contribute to the efficiency of the inferencing process. In a natural language inferencing system, we *are* concerned with not just what inferences are made, but also how they are made (with what ease, for example). This paper is motivated by these considerations. In particular, we will be concerned with the fact that the basic structure of natural language sentences in discourse, generally, singles out an individual (entity), to be called the center among all those which *are* the arguments of the main predicate. Our notion of center roughly corresponds to the linguistic notions of focus (in contrast to presupposition) and comment (in contrast to topic). We have deliberately used a new term in order to avoid precise identification with those notions and the possible resultant misunderstanding. These linguistic notions are somewhat vague and there is a great deal of confusion in the literature; further, the term "focus" is used used in the AI literature in yet another sense. For the interested reader, we recommend [4], "7], and [8] for the linguistic notions, and "5], [6] for the AI notions.

This work was partially supported by NSF Grant MCS 78-19466 and a grant from the Sloan Foundation to the University of Pennsylvania. We are indebted to the valuable comments of an unknown referee and a self-identified referee (Martin Kay). These have helped us greatly in imoroving both the style and content of this paper. We also want to thank Ellen Prince and Bonnie Webber for very profitable discussions.

The notion of center is a discourse construct; it may on occasion map on the subject of the sentence, but this need not be the case always. Such a representation can be regarded as an ascription of a property to a single individual, though the property itself may involve other individuals. For instance, in a particular context, JOHN may be the center of the sentence as in (1). JOHN HIT BILL. Underlining designates the center. It may help the reader to represent (1) in the extraposed form where the center is more clearly indicated as in (2) IT IS *JOHN* WHO HIT BILL. In another context, the center may be BILL, as in (3) JOHN HIT BILL.. (IT IS BILL WHOM JOHN HIT.) More formally, we will represent (1) and (3) respectively as (4) and (5): (4)(J0HN x) (HIT x BILL) or (jx) (H x b); (5) (BILL y) (HIT JOHN y) or (jy) (H j y).

This entity centered structure (ECS) of natural language sentences is in sharp contrast to the structure of the usual formal language sentences which express relations among several individuals without singling out any one in particular. (Our use of the term ECS is very limited, precisely as defined here. For a wider use of this

term in the context of knowledge representation (KR), see [2] and [1].) ECS makes it easier to see the rough logical form of the sentence. n-ary predicates seem to be monadic because part of their structure is temporarily hidden (see 4 and 5 above). An individual which is not centered is essentially ignored. It can be brought into consideration only by being made a center. ECS seems to affect the relative ease with which certain inferences are made and information is retrieved. For example, given (1) IT WAS JOHN WHO HIT BILL, (jx) (H x b), it is easier to answer (2) WHO HIT BILL? (∃x?) (H x b) than (3) WHOM DID JOHN HIT? (∃y?) (H j y). (See also Examples 1 and 2 in Section 5.)

It is well known that inferencing is much easier in monadic predicate logic (MPL) than in full predicate logic, and that it is decidable (see, for example, [3]). It is also obvious that MPL is inadequate to handle all inferences in natural language. Nevertheless, a great deal of inferencing in natural language seems to proceed as if we are dealing with monadic predicates. It is of interest to see what mechanisms can be added to MPL which would give the added power, yet maintain the essential flavor of monadic calculus. We have been investigating some systems of this nature from the points of view of (i) their ability to capture some key properties of inference mechanisms at work in language, and (ii) their formal properties, to the extent these formal results give some insignt into the structure and function of discourse constructs. In Sections 2, 3, and 4 we have presented a tentative formulation of Centered Logic (CL), and in Section 5 we have presented some sample derivations, followed by some discussion of the power of CL and some open questions.

2. Language: The formal language for centered logic differs from that of predicate logic in that the individual constants serve as variable binders in the "basic sentences," e.g., we write (jx) (H x m) for IT WAS JOHN WHO HIT MARY, and (mx) (H j x) for IT WAS Mary WHOM JOHN HIT. For the purposes of this paper, a simpler notation would have sufficed. Ours was chosen with an eye towards future treatments of intensional predicates. We want to be able to distinguish between, e.g., (ax) (Bxad) (Anastasia is such that she believes Anastasia is Grand Duchess of Russia) and (ax) (Bxxd) (Anastasia is such that she believes she is Grand Duchess of Russia.)

In addition, we allow complex sentences to be built-up using truth functional connectives and quantifiers. A quantified sentence is constructed by universally or existentially generalizing on a name, i.e., by replacing all occurrences of a name by a new variable v and prefixing the sentence with ∀ v or ∃ x. For example, we have (∀v)(vx)(∃y)(L x y): EVERYONE LOVES SOMEONE (OR OTHER); (∃y)(∀u)(ux)(L x y): EVERYONE LOVES SOMEONE (IN PARTICULAR); (∃y)(vy) (∀x)(L x y): SOMEONE IS LOVED BY EVERYONE. Note that the most natural reading for EVERYONE LOVES SOMEONE seems to be the one which does not require generalizing on names inside the predicate; the alternative reading is more naturally expressed by SOMEONE IS LOVED BY EVERYONE, which again does not require generalizing on predicate names. This jibes with our general view that names which are not centers are not taken as seriously as those which are.

3. Inference: We often speak of the "logical form" of a sentence as if it is unique. But we all know that the task of testing English arguments for validity can be considerably simplified if we chose the "right" representation. The trick is to expose only as much structure as is needed to see that the argument is valid: the less structure needed, the easier the inference. Predicates do have structure, but it can be provisionally masked and uncovered later only when necessary. Not all unravellings of the predicate are equally difficult. A major feature of the CL is that we ignore entities that are not centered.

Derivations are trees; each node follows from its predecessor or predecessors by one of the rules. The most important feature is that the means of introducing sentences with new centers is restricted. The centers of the premises can be thought of as the initial set of centered entities (SCE). A new entity can be introduced into SCE by making it the center of a sentence and there are only two ways in which this can be done. One is to use the change of center rule, which allows us to change a sentence into one which "says the same thing" but singles out different individual as the center. The other is the introduction of a temporary assumption with an entity that was not previously in SCE. The number of applications of change of centers and the number of individuals brought into SCE are measures of the difficulty of a derivation.

The rules are divided into four groups. 1:These rules allow the usual kinds of inferences, e.g., A∧B from premises A and B. Most rules in this group do not need the structure of predicates. Constraints on quantification: (a) universal instantiation will be allowed only with names for entities in SCE, i.e., we can infer (mx) (L j x) from (∀y)(mx)(L y x) and an additional premise of the form (jy)(P...y...).

(b) Existential generalization will be allowed only on names for entities in SCE. Thus, we can infer $(\exists y)(mx)(L\ j\ x)$ from $(mx)(L\ j\ x)$ if j∈ SCE. Of course, we can always infer $(\exists x)(L\ j\ x)$ from $(mx)(L\ j\ x)$ since the entity named m is centered.
2. These rules concern inferences which turn on the structure of the predicate, but which do not require us to recognize any names occurring in the predicate, e.g., we can derive $(jx)(Px) \wedge (jx)(Qk)$ from $(jx)(Px \wedge Qk)$. With these rules, we can bring out the structure of the predicate so that the sentential rules can be applied to it.
3. This group contains only one rule, the change of center rule, which allows us to infer, e.g., $(jx)(Gxma)$ which attributes a property to John (say, the property of giving Mary (the book) "Artificial Intelligence") from another sentence $(ax)(Gjmx)$ which attributes a property to "Artificial Intelligence" (say, the property of being given by John to Mary). 4. Finally, we have rules for changing bound variables and "instantiating" centers. In the usual formal system, changing bound variables requires an application of instantiation followed by another of generalization. In our system, these rules are restricted so that the strategy might require an additional application of change of center. Since the bound variable is an accidental feature of the representation, we feel that there should be rules which allow the bound variables to be changed directly. Center instantiation is needed to get the equivalence between $(ax)(Bxxd)$ and $(ax)(Bxad)$.

4. A Formal System (tentative version): The language of Centered Logic (CL) is defined as follows, given a formulation of first order predicate calculus (FOPC). A predicate of CL is a formula of FOPC containing at most one free variable. An atomic sentence of CL is of the form $(ax)(P)$ where P is a predicate containing no free variable other than x and a is an individual constant. A sentence of CL is a member of the smallest set X containing the atomic sentences and closed under the following conditions: (i) if A and B are members of X, then so are $(A \wedge B)$, $(A \vee B)$, and $(A \rightarrow B)$. (ii) If A and B are members of X and A^a_x is a result of substituting x for all occurrences of a in A, then $\neg A$, $(\forall x)(A^a_x)$, $(\exists x)(A^a_x)$ are all members of X. A pseudo-sentence of CL is either a sentence of CL or the result of substituting an individual variable for an individual constant in a sentence of CL. Some of the metamathematical variables we use are: A, B, C, D, ... for pseudo-sentences of CL; P, Q, R, ... for predicates of CL; a, b, c, ... for individual constants of FOPC; u, v, w, x, y, z, ... for variables of FOPC. The translations between CL and FOPC are obvious. (Note: CL could have been formulated as a kind of λ-calculus in which λ operators are used to form predicates from formulas subject to two restrictions:
(i) the predicates formed are monadic; we do not have predicates of the form $\lambda x_1 x_2 \ldots x_n A$.
(ii) The λ operators cannot be nested.)

Inference Rules: The notation is that or Prawitz Natural Deduction. Double lines indicate that th rule applies in either direction.

1.1 $\dfrac{A\ \ B}{A \wedge B}$ $\qquad \dfrac{A \wedge B}{A}$ $\qquad \dfrac{A \wedge B}{B}$

1.2 $\dfrac{A}{A \vee B}$ $\qquad \dfrac{B}{A \vee B}$ $\qquad \dfrac{A \vee B \quad \overset{(A)}{C} \quad \overset{(B)}{C}}{C}$

1.3 $\dfrac{\overset{(A)}{B}}{A \rightarrow B}$ $\qquad \dfrac{A \quad A \rightarrow B}{B}$

1.4 $\dfrac{\overset{(A)}{\neg A}}{\neg A}$ $\qquad \dfrac{\overset{(\neg A)}{A}}{A}$ $\qquad \dfrac{(ax)(P) \quad \neg(ax)(P)}{(ax)(Q)}$

1.5 $\dfrac{A \quad (ay)(P)}{(\forall x)(A^a_x)}$ $\qquad \dfrac{(\forall x)(A) \quad (ay)(P)}{(A^x_a)}$

1.6 $\dfrac{(A^x_a) \quad (ay)(P)}{(\exists x)(A)}$ $\qquad \dfrac{(\exists x)(A) \quad \overset{(A^x_a)}{B} \quad (ay)(P)}{B}$

Restrictions: In the first rule in (1.5), a must occur in any assumption on which A depends. In the second (1.6) rule, a must not occur in $(\forall x)(A)$, B, or any assumption on which the upper occurrence of B depends except A^x_a.

2. Predicate Decomposition

2.1 $\dfrac{(ax)(P \wedge Q)}{(ax)(P) \wedge (ax)(Q)}$ $\qquad$ 2.2 $\dfrac{(ax)(P \vee Q)}{(ax)(P) \vee (ax)(Q)}$

2.3 $\dfrac{(ax)(P \rightarrow Q)}{(ax)(P) \rightarrow (ax)(Q)}$ $\qquad$ 2.4 $\dfrac{(ax)(\neg P)}{\neg(ax)(P)}$

2.5 $\dfrac{(ax)(\forall y)(P)}{(\forall y)(ax)(P)}$ $\qquad$ 2.6 $\dfrac{(ax)(\exists y)(P)}{(\exists y)(ax)(P)}$

provided $x \neq y$.

3. Change of Topics

3.1 $$\frac{(ax)(P)}{(bx)(P^{x}_{a}{}^{b}_{x})}$$

4. Bound Variables

4.1 $$\frac{(\forall x)(A)}{(\forall y)(A^{x}_{y})}$$ 4.2 $$\frac{(\exists x)(A)}{(\exists y)(A^{x}_{y})}$$ 4.3 $$\frac{(ax)(A)}{(ay)(A^{x}_{y})}$$

4.4 $$\frac{(ax)(A)}{(ax)(A')}$$ where A' is the result of replacing one or more free occurrences of x in A by a.

5. Sample Derivations:

(1a) Everyone at the reception was thanked (T) by John (j). Mary (M) was at the reception (R). (therefore) Mary was thanked by John.

$$\dfrac{\dfrac{\dfrac{(\forall u)(ux)(Rx \to Tjx) \quad (mx)(Rx)}{(mx)(Rx \to Tjx)}}{(mx)(Rx) \to (mx)(Tjx) \quad (mx)(Rx)}}{(mx)(Tjx)}$$

(1b) John thanked everyone at the reception. Mary was at the reception. (therefore) John thanked Mary.

$$\dfrac{\dfrac{\dfrac{(jx)(\forall y)(Ry \to Txy)}{(\forall y)(jx)(Ry \to Txy)} \quad (mx)(Rx)}{\dfrac{(jx)(Rm \to Txm)}{(mx)(Rx \to Tjx)}}}{\dfrac{(mx)(Rx) \to (mx)(Tjx) \quad (mx)(Rx)}{(mx)(Tjx)}}$$

$$\frac{(jx)(Txm)}{}$$

(1b) is a more difficult inference than (1a) because it requires two applications of the change of center rule (marked by double lines).

Neither the inference in (1a) nor in (1b) requires bringing into SCE (Set of Centered Entities) a new entity which was not a center of a premise. This contrasts with the following derivation (2) which requires two new entities to be brought into SCE. The inherent difficulty of the inference in (2) below is due to this require ment, at least with respect to the CL.

(2) There is a house in which everyone lives
(L) (therefore) Everyone lives in houses.

$$\dfrac{\dfrac{\dfrac{(ay)(\forall x)(Lxy)^{*}}{(\forall x)(ay)(Lxy)} \quad \dfrac{(bx)(P)^{*}}{(bx)(P) \to (bx)(P)}}{(bx)(P \to P)}}{\dfrac{\dfrac{(bx)(P)^{*}}{(bx)(P) \to (bx)(P)}}{(bx)(P \to P)} \quad \dfrac{\dfrac{(ay)(Lby)}{(bx)(Lxa) \quad (ay)(\forall x)(Lxy)}}{\dfrac{(\exists x)(bx)(Lxy)}{(bx)(\exists x)(Lxy)}}}$$

$$\frac{(\exists v)(vy)(\forall x)(Lxy) \quad (\forall u)(ux)(\exists y)(Lxy) \quad (ay)(\forall x)(Lxy)}{(\forall u)(ux)(\exists y)(Lxy)}$$

* Temporary assumptions discharged later.

It is not difficult to prove that the rules given for CL are complete in the sense that A can be derived from ¬ whenever the * translation (from CL to FOPC) of A is classically derivable from the * translation of ¬ • To see this, note that (i) the o-translation (from FOPC to CL) of the classical natural deduction rules are all derivable from our rules, and * (ii) $A^{*\circ}$ is provably equivalent to A. However, we need both the change of center and the introduction of temporary assumptions with new centers to get this result. (The former is needed to establish the equivalence of A and $A^{*\circ}$ and the latter to show that translations of the classical quantifier rules *are* derivable.) If premises and conclusions have the same center, then no change of center rules are needed. For, if a is the center of premises and conclusions, then by translating each classical Φ as $(ax)\Phi)$ and using the predicate decomposition rules, the classical derivation of the *-translation of the argument can be converted to a CL derivation without center changes. Hence, the number of center changes need *never* exceed the number of premises whose centers are distinct from the centers of the conclusion. (The role of center change becomes more interesting, and more difficult to understand, when quantification into the predicate is prohibited. As we mentioned previously, there is some reason to believe that this restriction is appropriate.)

Although a restriction could be placed on the

number of center changes, no such limit can be placed on the number of new entities which must be brought into SCE. The premise in Example (2) contains no individual constant at all. It is clear from the rules that from such formulas alone we can only derive equivalent formulas and tautologous consequences. Hence, the introduction of new centers in Example (2) was unavoidable.

Thus, the logic resulting from a restriction on the number of centered entities is less powerful than one without, in particular, the legic with zero centered entities is less powerful than the logic with one centered individual. General questions about the number of new centers required for an inference and the decidability of the class of inferences with small number of centers remain open.

References (a few selected ones) :

[1] Brachman, R.J., "A Structural Paradigm for Representing Knowledge," Bolt Beranek and Newmwns.'. Report No. 3605, May 1978."

[2] Bobrow, D.G. and Winograd, T., "An Overview of KRL, a Knowledge Representation Language,' Cognitive Science, vol. 1, No. 1, Jan. 1977.

[3] Boolos, G.S. and Jeffery, R., Computability and Logic, Cambridge University Press, Cambridge, 1974.

[4] Chafe, W.L., "Givenness, Contrastiveness, Definiteness, Subjects, Topics, and Points of Views," in Subject and Topic (ed. C.N. Li), Academic press , New York, 1976.

[5] Grosz, B., "The Representation and Use of Focus in Dialogue Understanding," Stanford Research Institute Technician Report 15] Menlo Park,"California", 1977.

[6] Seidner, C, "The Use of Focus as a Tool for Disambiguation of Definite Noun Phrases,' TINLAP2, Urbana-Champaign, Illinois, 1978.

[7] Sgall, P., Hajicova, E., and Benecova, E., Topic, Focus, and Generative Semantics, Scriptor Verlag Gmbh", Kronberg Taurus, 1973.

[8] Prince, E., "On the Given/New Distinction," proc,Chicago Linguistic 1979.

CHARACTERIZATION OF A CLASS OF FUNCTIONS SYNTHESIZED FROM EXAMPLES BY A SUMMERS LIKE METHOD USING A "B.M.W." MATCHING TECHNIQUE.*

J.P. JOUANNAUD
Centre de Recherche en Informatique de Nancy
UER de Mathematiques
Universite de NANCY I
54037 NANCY CEDEX

Y. KODRATOFF
GR 22 du CNRS
Institut de Progranimation, T 55-65
Universite Pierre et Marie Curie (Paris VI)
4, place Jussieu
75230 PARIS CEDEX 05

Summary :

This paper describes first a program synthesis from examples method using a SUMMERS like methodology together with a sophisticated pattern matching technique, the BOYER-MOORE-WEGBREIT algorithm. We then characterize the class of functions synthesized from a restricted BMW algorithm. Our methodology is founded on three grounds.The first one is a precise characterization of the list domains fitting the synthesis from examples problem. The second one is SUMMERS' technique of input-output examples transformation into computational traces. The third one is the description by a restricted program scheme of the functions in the class. It seems to us that the proofs methods easily extend to the general class.

1. INTRODUCTION

The synthesis of functions from a set {x. -->F(x.)} df input-output examples has been extensively studied [1,2,4,5,6,7,8,9,10,14,15,16] our purpose here is to study a powerfull extension of SUMMER's method [16] using a pattern matching technique we shall call the BOYER-MOORE-WEGBREIT algorithm.

The principles of this technique are discussed in I 3,8,9,10,17] .

We recall briefly this method which is detailed in [8,9,10]. However we think that it is not enough to describe an algorithm, especially in our field : we must aim at the characterization of the class of functions which can be synthesized. A recent work of SMITH [15] describes such an investigation for a restricted SUMMER'S method. The goal of this paper is to design more adapted tools which allow simpler proofs of our theorems. We are able to characterize the functions synthesized by a restricted BMW algorithm and prove a linear growth property of the functions in this class. We mean that our results shall extend to the general case.

* Partly supported by IRIA-SESORI.

2. SYNTHESIS METHODOLOGY

We recall briefly, in an intuitive way, the methodology relative to one variable functions [8,9,10,16]. We are concerned with LISP unary functions defined on lists. Before the synthesis process itself takes place, two transformations of the input-output values are necessary. The first one transforms these values into a set of computational traces expressed with the set {cons, car, cdr, atom) of primitive functions. The second one transforms the computational traces into a partial function, defined on subdomains of F.

In order to make our notations and results clearer, we shall describe the synthesis of an ad'hoc function F described by the following set of examples :

x_o = (A(B)C) $F(x_o)$ = (B) ;

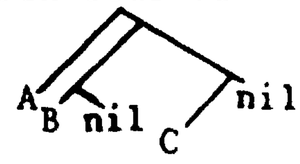
x_1 = (A(B(C(D)E))) $F(x_1)$ = (C C B) ;

x_2 = (A(B(C(D(E(F)G))))) $F(x_2)$ = (C C D C B) ;

x_3 = (A(B(C(D(E(F(G(H)I))))))) $F(x_3)$= (CCCEDCB)

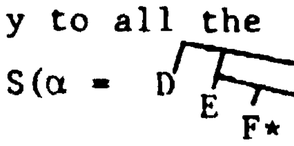
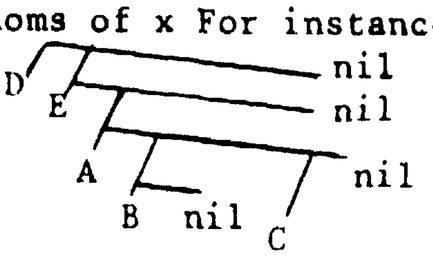
2.1. Notations and definitions

Let L bet the set of LISP lists. A list x of L may be viewed as a cons-tree, each leaf of which is an atom.

For instance, x_o= (A(B)C) or x_o = 

Let $|x|$ be the complexity of the list x, i.e. the number of leafs in the associated cons-tree.

Following KNUTH [11], we shall denote by $S(x, y, *)$ the list obtained by substitution of y to all the *-labelled atoms of x For instance

$S(\alpha =$  $, x_o, *) =$ 

It is well known that the order associated to the substitution relation is a partial ordering of L.

Definition : A reduction function is a finite composition of car and cdr. Let R be the set of reduction functions. The length $|f|$ of $f \in R$ is the number of car and cdr in f.

Notice that the triplet $< R$, composition rule, $I >$, where I denotes the identity function, is isomorphic to the free monoïd constructed on the alphabet {car, cdr}.

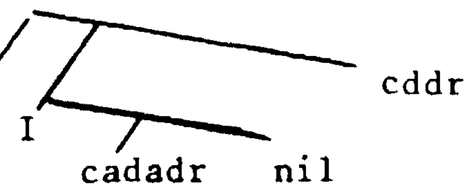
Definition : Let A_i be the ith occurrence of the atom A in the list x and u the unique reduction function which verifies $u(x) = A_i$. Then u is said to be the functional name of A_i in x.

For instance, the name of F_* in α is cadadr.

Definition : If $v \in R$ is a right factor of $u \in R$, then $u\ v^{-1}$ denotes the function of R such that $u = (u\ v^{-1})v$. Let u modulo v be the function $u\ v^{-k}$ provided $v^{k'}$ is a right factor of u iff $0 \leq k' \leq k$.

Example : u = caadadr, v = cadr, $u\ v^{-1}$ = caadr, u modulo v = car.

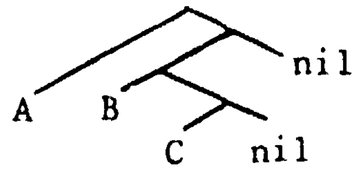
Definition : Given a set of terminal functions (Generally $R \cup \{nil\}$) a cons-expression is a cons-tree, each leaf of which is a terminal function or nil.

Example : f = 

Notice that nil may be viewed either as an atom or as a function.

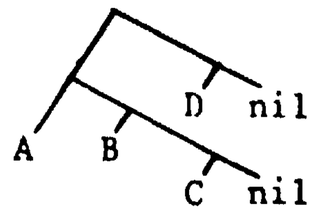
Let ω be the undefined atom, D(f) the domain of the function f, i.e. the set $\{x | x \in L, f(x) \neq \omega\}$

Definition : The adapted structure [15] ad(f) of a function $f \in R$ is the smallest $x \in D(f)$.

For example, ad(caddar) = 

Notice that the atoms of ad(f) do not have any importance at all, i.e. ad(f) is in fact a list structure.

Definition : The adapted structure ad(F) of a set $F = \{f_i\}$, $f_i \in R\ \forall i$, is the smallest $x \in D(F) = \bigcap_i D(f_i)$.

For example, ad {caddar, cddr} = 

In the following, we shall suppose that no atoms of ad(F) bear the same name.

2.2. Computational traces

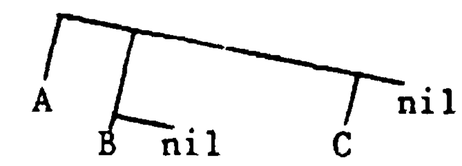
A main step in SUMMER's method is to transform the sequence $\{x_i \rightarrow F(x_i)\}$ of input-output examples in a sequence $\{p_i(x) \rightarrow f_i(x)\}$ of computational traces.

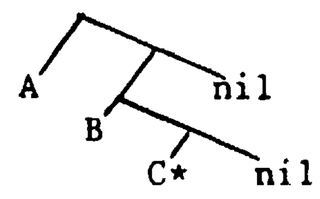
2.2.1. Ascending linear domains.

The further step of synthesis requires the existence of recurrence relations between the predicates p_i. It is thus natural to study the input domains that possess this property.

Definition : Given $x_o \in L$, $\alpha \in L$, α owning a single *-labelled atom, the set $\{x_o, x_{i+1} = S(\alpha, x_i, *)\}$ is called ascending linear domain $\mathcal{A}(x_o, \alpha)$.

In fact, we change the names of the atoms in order to avoid several occurrences of the same atom in x_i.

For instance, if $x_o =$ 

and $\alpha =$ 

$\mathcal{A}(x_o, \alpha)$ is the set

$\{(A(B)C), (A(B(C(D)E))), (A(B(C(D(E(F)G))))), \ldots\}$

Notice that the partial ordering on L is total on $\mathcal{A}(x_o, \alpha)$.

We now have to exhibit a sequence p_i of predicates which verify :

$p_i(x_i) = \underline{T}$, $p_i(x_j) = \underline{F}$ iff $j > i$, $p_i(x_j) = \omega$ iff $j < i$, where $\underline{T}$ and $\underline{F}$ denote the logical values true and false.

Let us call b the function name of the *-labelled atom in α and suppose that we know a reduction π such that atom. $\pi(x_o) = \underline{T}$.

It is obvious that $b(x_{i+1}) = x_i$ and it follows that the sequence $p_i = atom.\pi.b^i$ verifies $p_i(x_i) = \underline{T}$. We still need to find the "good" function π, i.e. the function such that p_i verifies the two remaining conditions $p_i(x_j) = \underline{F}$ iff $j > i$ and $p_i(x_j) = \omega$ iff $j < i$.

Definition : An atom in a list x is said to be in the direction of $f \in R - \{I\}$ iff its functional name has the form $f'.f^p$ where f' is a proper right factor of f, i.e : $f = f''.f'$, $f'' \neq I$.

For instance, $nil_\star$ in x_0=  is in the direction of cadr : its functional name is cdadr = cdr.cadr.

The intuitive meaning of this concept is the following : if we compute $cadr(x_o)$ we obtain  which is not yet an atom. We cannot compute cadr.cadr(x_o) which is not defined, but we start the computation and keep computing as long as we do not reach an atom : cdr.cadr(x_o) = $nil_\star$ is an atom ; we therefore stop the computation at this point.

Property of the ascending linear domains : given an ascending linear domain $\mathcal{A}(x_o, \alpha)$ and the name b of the $\star$-labelled atom of α, let π be the functional name of the atom of x_o which is in the direction of b. Then $\{p_i = atom.\pi.b^i\}$ is a sequence of predicates for the domain.

Proof : $\pi = b'.b^p$, where $b = b''.b'$ and $b'' \neq I$.
- As atom. $\pi(x_o) = \underline{T}$ by definition of π, we know that $p_i(x_i) = \underline{T}$, $\forall i$
- $j < i$: $p_i(x_j) = atom.\pi.b^i(x_j) = atom.b'.b^p.b^{i-j}(x_o)$
$= atom.b'.b^{i-j-1}.b''.b'.b^p(x_o) = atom.b'.b^{i-j-1}.b''.\pi(x_o) = \omega$

because $\pi(x_o)$ is an atom and $b'' \in R-\{I\}$.
- $j > i$: $p_i(x_j) = atom.\pi.b^i(x_j) = atom.\pi(x_{j-i})$.
Let us compute
$p_j(x_j) = \underline{T} = atom.\pi.b^j(x_j) = atom.b'.b^{j-i-1}.b''.\pi(x_{j-i})$

It is obvious that if $p_i(x_j) = atom.\pi(x_{j-i}) = \omega$ or $\underline{T}$ then

$p_j(x_j) = \omega$ which is false. Thus $p_i(x_j) = \underline{F}$.

Notice that the sequence of predicates verifying these previous conditions is not unique in the sense that π is not unique in some particular cases.

Example : Let us consider the previous set $\mathcal{A}(x_o, \alpha)$. We obtain b = cadadr and π = cdadr. Thus p_o = atom.cdadr, p_1 = atom.cdadadadr, p_2 = atom.cdadadadadadr. The match of p_i and p_{i+1} succeeds and we obtain :

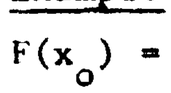
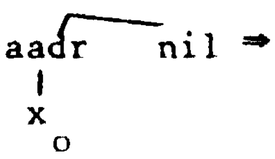
$$p_{i+1}(x) = p_i(cadadr\ x)$$

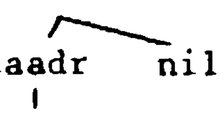
More generally, $p_i = atom.\pi.b^i$ and $p_{i+1} = p_i.b$

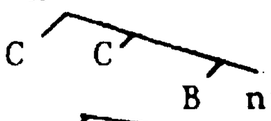
2.2.2. Fragments

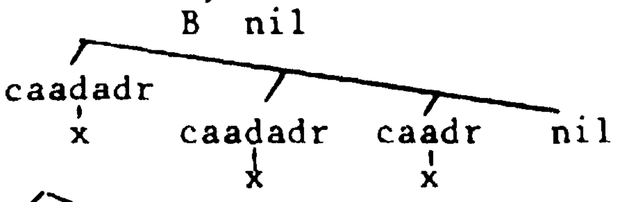
The output $F(x_i)$ is taken in its cons-tree form. Each atom, except nil, of this cons-tree is replaced by its functional name in x_i. The process is unambiguous iff there are not several atoms (except nil) bearing the same name in any x_i. Each obtained cons-expression $f(x_i)$ is then turned into a partial function $f_i(x)$ by the substitution of x to each x_i of $f(x_i)$. The function f_i is called the ith fragment of F. Notice that f_i is defined for any $x \geq x_i$ and equal to $f(x_i)$ when $x = x_i$.

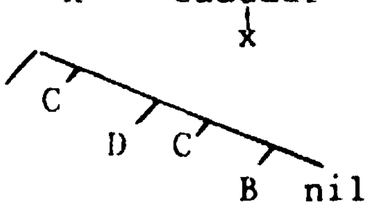
Example :

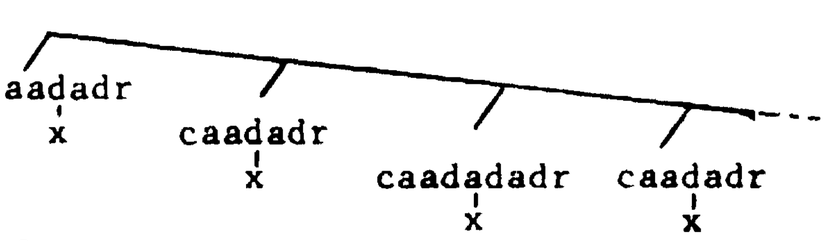
$F(x_o)$ =  ➡ $f(x_o)$ =  ➡

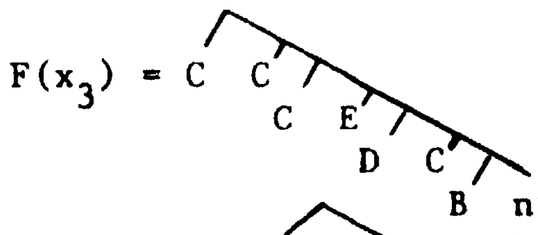
$f_o(x)$ = 

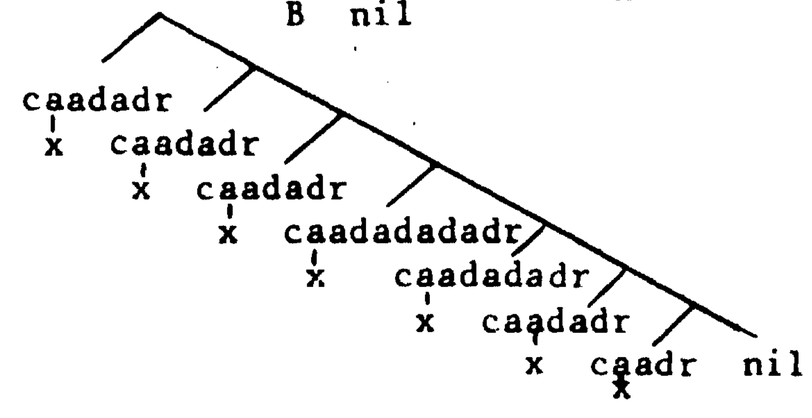
$F(x_1)$ = 

$f_1(x)$ = 

$F(x_2)$ = 

$f_2(x)$ = 

$F(x_3)$ = 

$f_3(x)$ = 

2.2.3. Approximations

The input-output sequence $\{x_i \rightarrow f(x_i)\}$ has been changed into a sequence of predicates and fragments $\{p_i(x) \rightarrow f_i(x)\}$ which is embedded into an if... then... else... function in order to obtain the following n-approximation of F :

if $p_o(x)$ then $f_o(x)$ else

if $p_1(x)$ then $f_1(x)$ else...

if $p_n(x)$ then $f_n(x)$ else ω.

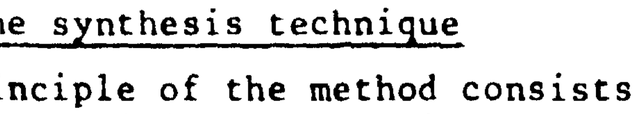
Notice that the names of the atoms are now lost we therefore cannot synthesize functions depending on these names, like the function which is true iff all the names of the atoms are different.
Our problem now is to get a finite expression of the ∞-approximation.

2.3. The synthesis technique

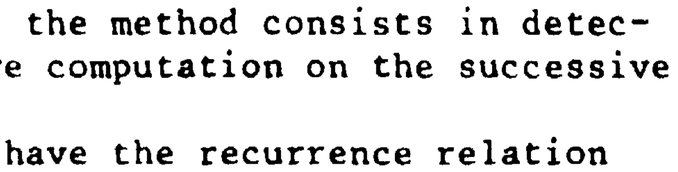
The principle of the method consists in detecting a repetitive computation on the successive traces.
Predicates : We have the recurrence relation
(1) $p_{i+1}(x) = p_i(\text{cadadr } x)$.
Fragments : We attempt to match $f_i(x)$ and $f_{i+1}(x)$ using the BMW algorithm. When it succeeds, we use its substitutions to write recurrence relations proved to be equivalent to programs by SUMMER's main synthesis theorem [16] or its extensions [9].

The classical pattern matching techniques [13] do not succed to match the preceeding sequence of fragments. In this case, a first order pattern matching attemps to use generalization techniques together with matches between subtrees, i.e. the BMW algorithm. It works as follows :
A first generalization leads to replace the nil atom by a variable z in each fragment :

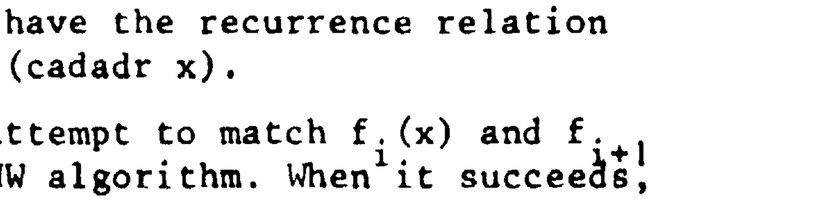
$f_o(x)$ becomes



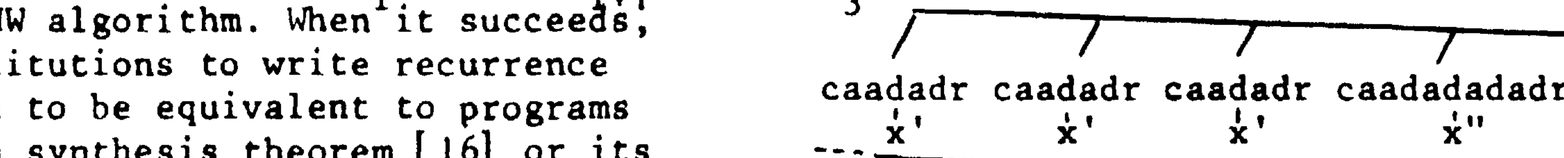
$f_1(x)$ becomes



$f_2(x)$ becomes

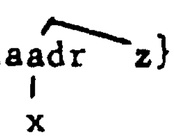


$f_3(x)$ becomes



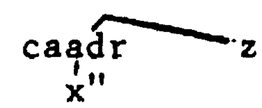
The matching is now possible : Each fragment is matched with the first subtree of the successive fragment, provided the following set of substitutions :

$\{x \rightarrow x,\ x \rightarrow \text{cadr } x,\ z \rightarrow \text{caadr } z\}$.

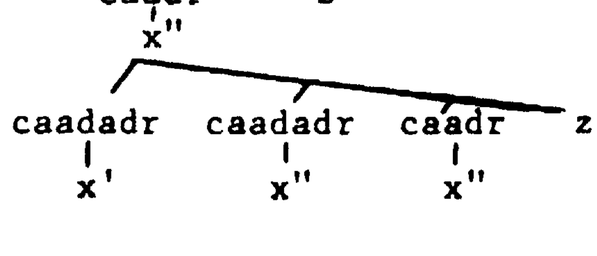


Since the variable x receives two different substitutions, a new generalization is necessary : the variable x becomes x' if it undergoes the first substitution and x" if it undergoes the second substitution, and the fragment $f_i(x)$ becomes $f'_i(x',x'',z)$:

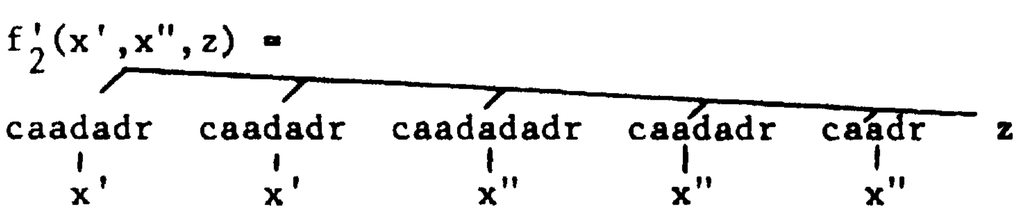
$f'_o(x',x'',z) =$



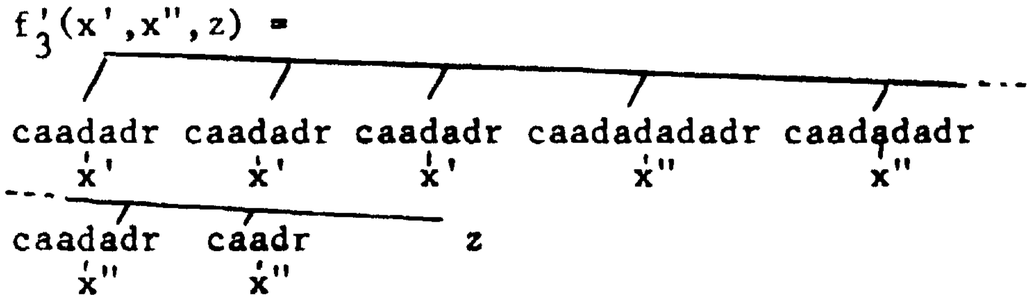
$f'_1(x',x'',z) =$



$f'_2(x',x'',z) =$



$f'_3(x',x'',z) =$



The matching between two consecutive fragments now succeeds and we find the following recurrence relations :

$$f'_{i+1}(x',x'',z) = h_i[x', f'_i(x', (\text{cadr } x''), g_i(x'',z))]$$

(2)

$$h_i(x',y) = (\text{cons}(\text{caadadr } x')y)$$

$$g_i(x'',z) = (\text{cons}(\text{caadr } x'')z)$$

Notice that $\{h_i\}$ and $\{g_i\}$ are two sequences of fragments associated to the sequence of predicates p_{i+1}. In the present case, h_i and g_i are constant cons-expressions. Sometimes h_i and g_i are not constant. When it is the case, we use repetitively the same algorithm until we get

constant h and g.
In the following, we shall restrict ourself to a one step BMW algorithm (the above example behaves in this way).

Since we use matches between subtrees, the succeeding match is generally not unique. One should not worry about this : we shall prove that all the matches lead to equivalent programs.

From the relations (1) and (2) a program is derived using the following extension [9] of SUMMER's Synthesis Theorem (MST) :

Theorem : If the infinite sequences $\{p_i\}$ and $\{f_i\}$ verify the recurrence relations :

$p_{i+1}(x) = p_i(b(x))$

$f_i(x) = f'_i(x^n, x, nil^m)$ with $x^n = x,\ldots,x$ n times and $nil^m = nil,\ldots,nil$ m times.

$f'_{i+1}(x,xx,\vec{z}) = h[\vec{x}, f'_i(\overrightarrow{b'(x)}, b(xx), \overrightarrow{g(x,z)})]$

where $\overrightarrow{b'(x)}$ is $b'_1(x_1),\ldots,b'_n(x_n)$, $g(x,z)$ is $g_1(\vec{x},z_1),\ldots,g_m(x,z_m)$ $\quad e \in R-\{I\}$, $b = e^q$, $b'_i = e^{r_i}$, $0 \leqslant r_i \leqslant q$, $0 < q$.

h and g are constant cons-expressions then the ∞-approximation associated to the computational traces sequence $\{p_i(x) \rightarrow f_i(x)\}$ is equivalent to the program :

$F(x) \leftarrow F'(x^n, x, nil^m)$
$F'(x',xx,\vec{z}) \leftarrow$ If atom.$\pi(xx)$ then $f'_o(\vec{x},\vec{z})$
else $h[\vec{x}, F'(\overrightarrow{b'(x)}, b(xx), \overrightarrow{g(x,z)})]$

The xx variable, defined as the domain variable, is introduced to take into account the recursion on $p_i(x)$. One realizes its importance when we look back to the previous example : neither x' nor x" have the same relation as xx :

```
F(x) ← F'(x,x,x,nil)
F'(x',x",xx,z) ← If atom(cdadr xx) then
(cons(caadr x")z)
                  h(x',x",xx,F'(x',(cadr x"),
(cadadr xx),g(x',x",xx,z))).
h(x',x",y) ← (cons(caadadr x')y)
g(x',x",z) ← (cons(caadr x")z)
```

In the following, the symbol F used as an example will always denote the above function.

Remark : Given a function, the recurrence relations are supposed to begin with the first example of the function. For the sake of readability, we shall rule out the trivial cases where the first examples of the function's behaviour (i.e. a finite number of them) have particular properties.

3. THE SYNTHESIS ALGORITHM AND THE RESTRICTED PROGRAM SCHEME.

Let us recall that we are only concerned with unary LISP functions given by a sequence $\{x_i \rightarrow F(x_i)\}$ of examples, the set $\{x_i\}$ of inputs belonging to an ascending linear domain $\mathcal{A}(x_0,\alpha)$.

Synthesis algorithm

1. Compute the functional name b of the unique * - labelled atom of α.
2. Compute the functional name π of the unique atom in x_o which is in the direction of b.
3. Compute the fragments f_i.
4. Look for recurrence relations between consecutive fragments using the BMW algorithm restricted to one step.
5. If recurrence relations are found then use the MST. Our present aim is to precise which functions can be synthesized and how many examples are needed.

Let us call $\mathcal{SA}$ the set of functions which can be synthesized. If $F \in \mathcal{SA}$, obviously F verifies the program scheme used in the MST. Our purpose is to prove the reciprocal. Following SMITH [15], we shall first introduce some very natural restrictions on the program scheme which come from the fragment computation technique.

Program scheme 0 :
F is a cons-tree whose terminal functions g_i belong to $R \cup \{nil\}$

Restriction : $\forall\ g_1,g_2, g_1 = v.g_2 \Rightarrow v = I$.

Notice that this restriction comes very naturally : It means that g_1 and g_2 are terminal functions of fragments (Since F^2 is constant, $f_i = F$, $\forall_i$) and thus $g_1(x_i)$ and $g_2(x_i)$ are both atoms. It follows that $v.g_2(x_i)$ is defined iff $v = I$.

Program scheme 1 :

```
F(x) ← F'(x,x,nil)
F'(x,xx,z) ← If p_o(xx) then f'_o(x,z)
                  else h[x,F'(c(X),h(xx),g(x,z))]
```

In fact x,c and z in the F' definition should be vectors as shown by the example. We do not use vectors in order to lighten the notations but this simplification implies no loss of generality.
We shall call $\mathcal{PS}_o$ the set of functions which can be computed by an instance of this restricted schema.

Restrictions :

1/ $\exists\, e \in R-\{I\}$, $\exists\, o < q, \exists o \leqslant r \leqslant q$ such that $b = e^q$, $c = e^r$.

2/ p_o = atom, π, $\pi = b'b^p$, $p \geqslant o$, $b = b''b'$, $b'' \neq I$.

3/ f'_o is a cons-tree whose terminal functions $f'_{o,i}$ belong to R-{I} $\cup$ {nil} when applied to x and equal I when applied to z.

4/ h is a cons-tree whose terminal functions h_i belong to R-{I} $\cup$ {nil} when applied to x and equal I when applied to F'.

5/ g is a cons-tree whose terminal functions g_i belong to {R} - {I} $\cup$ {nil} when applied to x and equal I when applied to z.

6/ Let X_o be the set of functions defined below which lead to a computation on x_o starting from a computation $F(x_k)$.:

$$\forall f,g \in X_o,\ f = v.g \Rightarrow v = I.$$

7/ Let $\mathcal{A}$ be the set of functions defined below which lead to a computation on α starting from a computation $F(x_k)$:

$\forall f \in \mathcal{A} \Rightarrow f \neq I$ and $\forall f, g \in \mathcal{A}$, $f = v.g \Rightarrow v = I$.

These two last restrictions mean that functions of X_o and $\mathcal{A}$ come from fragments. It follows that if g belongs to X_o, then $g(x_o)$ is an atom. Thus $v.g(x_o)$ is defined iff $v = I$.

$X_o = \{\pi\} \cup \bigcup_i \{f'_{o,i}.c^k.b^{-k} \mid o \leqslant k, b^k$ is a right factor of $f'_{o,i}.c^k\}$

$\cup \bigcup_i \{g_i.c^j.b^{-k} \mid o \leqslant j < k, b^k$ is a right factor of $g_i.c^j\}$

$\cup \bigcup_i \{h_i.c^j.b^{-k} \mid o \leqslant j < k, b^k$ is a right factor of $h_i.c^j\}$

$\mathcal{A} = \{b\} \cup \bigcup_i \{f'_{o,i}.c^j$ modulo $b \mid o \quad j,\ r \neq q\}$,

where r and q are defined in restriction 1

$\cup \bigcup_i \{g_i.c^j$ modulo $b \mid o \leqslant j\}$

$\cup \bigcup_i \{h_i.c^j$ modulo $b \mid o \leqslant j\}$

Example : In the above example (section 2) c was a two-dimensional vector (c_1, c_2). Its first component, c_1, is associated with the terminal functions which compute on x' and c_2 is associated with the terminal functions which compute on x".

- π = cdadr ; $\bigcup_i \{f'_{o,i}.c^k.b^{-k}\}$ = {caadr, car} ; $\bigcup_i \{g_i.c^j.b^{-k}\}$ = {car} ; $\bigcup_i \{h_i.c^j.b^{-k}\}$ = car ; .

  Thus X_o = {car, caadr, cdadr}

- b = cadadr ; $\bigcup_i \{f'_{o,i}.c^j$ modulo b$\}$ = {caadr, car}; $\bigcup_i \{g_i.c^j$ modulo b$\}$ = {caadr, car} ; $\bigcup_i \{h_i.c^j$ modulo b$\}$ = {car} ;.

  Thus $\mathcal{A}$ = {car, caadr, cadadr}.

Let us call $\mathcal{FS}_1$ the set of functions which can be computed by a program which is an instance of this restricted scheme and $\mathcal{FS}$ the set $\mathcal{FS}_o \cup \mathcal{FS}_1$.

Theorem : $\mathcal{FA} = \mathcal{FS}$.

Before proving this theorem, we shall first express the fragment f_k in terms of the functions f'_o, h and g which appear in the program schema.

Lemma 1 : $f_k(x) = h[x, h[c(x), \ldots, h[c^{k-1}(x), f_o[c^k(x), g(c^{k-1}(x), \ldots, g(c(x), g(x, nil))\ldots)]]\ldots]]$

This lemma is easy to prove by induction on k. As a consequence, the set of terminal functions of f_k is $\{h_i.c^j,\ f'_{o,i}.c^k, g_i.c^j \mid j \leqslant k\}$.

First part of the proof : $\mathcal{FA} \subseteq \mathcal{FS}$

Two cases may occur : the computed fragments are constant or not. If they are constant, the program scheme o is used and the restriction holds, else scheme 1 is used and we have to prove the restrictions 1 to 7.

Lemma 2 : Restriction 1 holds.

Proof : The matching of the fragments provides a function $c \in R$ such that $\forall i > o, j < i$, $c^j(x_i) \neq \omega$ This implies that either c = I and lemma 2 is proved with e = b, q = 1 and r = o, or $c \neq I$ and c and b "compute in the same direction". It follows that $b^{|c|} = c^{|b|}$.
As b and c belong to the free monoïd constructed on the alphabet {car, cdr} they are powers of the same $e \in R$. Thus $b = e^q$ and $c = e^r$.
Moreover if $q < r$, then for k large enough $c^{k-1}(x_k)$ would be ω since $b^{k-1}(x_k) = x_1$.
Restriction 2 holds because $\mathcal{A}(x_o, \alpha)$ is an ascending linear domain.
Restriction 3,4,5 come from the working technique of the BMW algorithm.

Lemma 3 : $\forall f \quad X_o$, atom $f(x_o) = \underline{T}$.

Restriction 6 follows easily.

Lemma 4 : $\forall f \in \mathcal{A}$, atom. $f(\alpha) = \underline{T}$.

Restriction 7 follows then easily and the first part of the theorem is proved.

Second part of the proof : $\mathcal{TS} \subset \mathcal{SR}$. Three steps are needed.

First step.
We have to exhibit an ascending linear domain $\mathcal{R}(x_o, \alpha)$. This domain should ve the "smallest" domain on which the synthesis of the known function F is possible. Given $F \in \mathcal{TS}$, we naturally take as x_o and α :

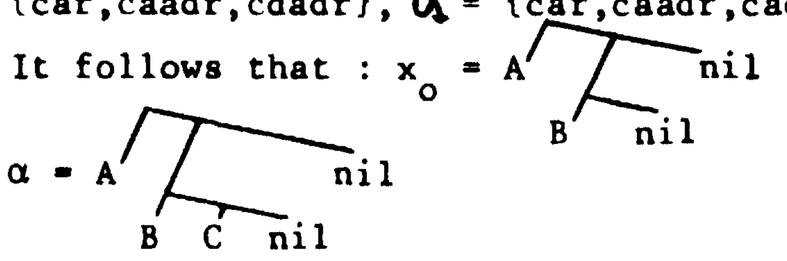
$$x_o = ad(X_o), \quad \alpha = ad(\mathcal{A})$$

This has a meaning, because :

Lemma 5 : X_o and $\mathcal{A}$ are finite sets.

The proof is simple.

Example : for the example F we have X_o = {car,caadr,cdadr}, $\mathcal{A}$ = {car,caadr,cadadr},
It follows that : x_o =



α =

Notice that the new x_o is simpler than the previous one which contained useless information.
To define $\mathcal{R}(x_o, \alpha)$, we need a *-labelled atom in α, the functional name of which is b : it is thus the atom C.

Second step.
Given the set $\{x_i\}$ of inputs, it is now possible to compute the set $\{x_i \rightarrow F(x_i)\}$ of examples and the set $\{p''_i \rightarrow f''_i\}$ of computational traces.
We shall now prove that the sets $\{p_i \rightarrow f_i\}$ and $\{p''_i \rightarrow f''_i\}$ are identical, where p_i = atom.π. b^i and f_i is defined in lemma 1.

Lemma 6 : Given a set G of reduction functions and x = ad(G), if atom.f(x) = $\underline{F}$ then there exists $g \in G$ such that g = v.f, $v \in R-\{I\}$.

Lemma 7 : If $G = \{g_i\}$ and $\forall g_1, g_2 \in G$, $g_1 = v.g_2 \Rightarrow v = I$, then atom.$g_i$ (ad(G))=$\underline{T}$.

Lemma 8 = p''_i = atom.π.b^i.

Lemma 9 : $\forall_k$, $f''_k = f_k$.

Third step.

We now may match the consecutive fragments. We know recurrence relations between these fragments : they derive from the given program F. Nevertheless, several matches may succeed and lead to different programs with program scheme o or 1. We thus have to prove that all these different matches lead to equivalent programs, although a finite number of examples are used to compute the fragments and the recurrence relations.
Notice that there is no problem if F is given by a program which is an instance of the program scheme o. In this case, F will be synthesized with the same program. Difficulties appear when F is given by a program which is an instance of the program scheme 1.
Before solving this problem, we shall prove a property of the class $\mathcal{TS}$.

Property 1 : $F \in \mathcal{TS}_o \Rightarrow |F(x_k)|$ is a constant.

Property 2 : $F \in \mathcal{TS}_1 \Rightarrow |F(x_k)|$ is a constant or a linear function of $|x_k|$.

Lemma 10 : If F is an instance of the program scheme 1 and F" an instance of the program scheme o which take the same values in x_o and x_1, then F and F" are equivalent. The synthesis process of F needs two examples : $x_o \rightarrow F(x_o)$ and $x_1 \rightarrow F(x_1)$.

Lemma 11 : If F and F" are two instances of the program scheme 1, which take the same values in x_o, x_1 and x_2, then F and F" are equivalent. The synthesis process of F needs three examples $x_o \rightarrow F(x_o)$, $x_1 \rightarrow F(x_1)$ and $x_2 \rightarrow F(x_2)$.

Proof : A detailed proof is rather technical and can be found in [7 bis]. Its principle is : characterize the links between the two sets $\{\pi, b, c, f'_o, h, g\}$ and $\{\pi, b, c'', f''_o, h, g\}$ (which define F and F") when F and F" have the same values on x_o, x_1 and x_2. These links enable us to prove easily that the sequences of fragments $\{f'_k\}$ and $\{f''_k\}$ are identical.

We have thus proved that any function in $\mathcal{TS}_o$ is synthesized from 2 examples and in $\mathcal{TS}_1$-$\mathcal{TS}_o$ from 3 examples.

4. CONCLUSION

We have obtained a characterization of the class of functions synthesized from examples, when a restricted BMW algorithm is used. Moreover, we have proved an important property of the functions in this class : the existence of a deep relation between the number of examples needed for their synthesis and their growth speed. These above results, obtained for the classes $\mathcal{TS}_o$ and $\mathcal{TS}_1$ may be extended to the class $\mathcal{TS}_n$ defined by the use of an n-step BMW algorithm. A key step for this proof is the

obtention of an equivalence result between an instance of the program scheme n and an instance of the program scheme $m \leq n$ wich take the same values for the inputs $x_0, x_1, \ldots, x_{n+1}$.

A further interesting problem should be to obtain an intuitive characterization of the class $\mathcal{TJ} = \bigcup_n \mathcal{TJ}_n$. We mean that there is a strong analogy between the properties of the polynomials and those of $\mathcal{TJ}$.

The BMW algorithm has been implemented in LISP by E. PAPON (unpublished "stage de D.E.A." Univ. Paris-Sud).

REFERENCES

[1] BIERMANN A.W. : The inference of regular LISP programs from examples, I.E.E.E. Trans, on Systems, Man and Cybernetics Vol. SMC - 8, (1978), 585-600.

[2] BIERMANN A.W., SMITH D.R. : "The hierarchical synthesis of LISP scanning programs", Information Processing 77, B. GILCHRIST ed. (North Holland, 1977) pp. 41-45.

[3] BOYER R.S., MOORE J.S. : "Proving theorems about LISP functions", JACM-22, vol 1, (1975) pp. 129-144.

[4] HARDY J. : "Synthesis of LISP functions from examples", Proc. 4th IJCAI (1975) pp. 268-273.

[5] JOUANNAUD J.P., GUIHO G., TREUIL J.P. : "SISP/1, an interactive system able to synthesize functions from examples", Proc. 5th IJCAI (1977).

[6] JOUANNAUD J.P., GUIHO G. :"Inference of functions with an iteractive system", Machine Intelligence 9, D. MICHIE ed., 1979.

[7a] JOUANNAUD J.P., KODRATOFF Y. : "A methodology for two variables functions synthesis from examples", Pub. CRIN Univ. Nancy (1979) (On request to the author).

[7b] JOUANNAUD J.P., KODRATOFF Y. : "Characterization of a class of functions synthesized from examples by a SUMMERS like method using a "BMW" mathing technique" Pub. CRIN Univ. Nancy (1979).

[8] JOUANNAUD J.P., KODRATOFF Y. : "Quelques methodes de synthese automatique de programmes a partir d'exemples", Journe'es IRIA-SESORI Synthase, Manipulation et Transformation de Programmes (1978) pp. 215-239.

[9] KODRATOFF Y. : "A class of functions synthesized from a finite number of examples and a LISP program scheme". Pub. Institut de programmation (1979) and Int. J. of Comp. and Inf. Sci. Vol 8, n° 6, (1979)

[10] KODRATOFF Y. FARGUES J. : "A sane algorithm for the synthesis of LISP functions from examples problems : The BOYER and MOORE algorithm", Proc. AISB meeting Hambourg (1978) pp. 169-175.

[11] KNUTH D.E., BENDIX P.B. : "Word problems in universal algebras" in Computational problems in abstract algebra, J. LEECH ed. (1970), pp. 263-297.

[12] CARTHY J. Mc, ABRAHAMS P.W., EDWARDS D.J., HART T.P., LEVIN M.E. : LISP 1.5. programmer's manual (M.I.T. Press, 1962).

[13] ROBINSON J.A. : "A machine oriented logic based on the Resolution Principle", J. ACM (1965), 12, 23-41.

[14] SHAW D., WARTOUT W.S, GREEN C. : Inferring LISP programs from example problems". Proc 4th IJCAI (1975), pp. 268-273.

[15] SMITH J.P. : "A class of synthesizeable LISP programs", Report CS-1977-4, Dpt. of Computer Sciences, DUKE University, 1978.

[16] SUMMERS P.D. : "A methodology for LISP program construction from examples", J. ACM (1977), 24, 161-175.

[17] WEGBREIT B. : "Goal directed program transformations", IEEE SE-2 (1976), 69-80.

Making Aesthetic Choices

Kenneth M. Kihn
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Massachusetts 02139, USA
Arpanet address: KEN@MIT-AI

A framework is presented for making choices that are primarily constrained by aesthetic, as opposed to pragmatic, considerations. An example of the application of this framework is a computer system called "Ani", capable of making simple computer animation in response to high-level incomplete story descriptions. Aesthetic choice is presented as a parallel computation in which each *choice point* gathers together and evaluates *suggestions.* When faced with difficulties these choices can be *postponed.* The order in which inter-dependent choices *are* made is influenced by the *focus* of the problem.

Introduction

People are often faced with choices that are under-constrained by considerations of utility, cost, simplicity and efficiency. In many of these cases, one can just choose arbitrarily between those alternatives which satisfy the pragmatic constraints. There remain many situations where this is inadequate however, where instead one wants to select the alternative that is the most beautiful, elegant, interesting, or that conforms to a particular style. Aesthetic considerations are important, sometimes even dominate, in tasks that vary from exploring mathematics to designing block diagrams, from writing an IJCAI paper to making an animated film.

An assumption of this paper is that making aesthetic choices is a knowledge intensive computational process. The interesting questions are what the knowledge is and how it is represented, organized, and used. These questions are addressed in [1]. This paper presents a general framework for how knowledge is *used* to create an aesthetic object (a detailed description of an object such as a film, a story or a proof) that is consistent and coherent. This means that ideally *every* choice should be justified by as much relevant knowledge combined in as reasonable a manner as possible.

Aesthetic choices cannot be made independently, since each decision *constrains* subsequent choices. The problem addressed here is really one of *generating* a reasonable set of constraints rather than trying to find a solution that *satisfies* a set of constraints. The control structure, the order in which choices are made, therefore becomes very important. Since early choices constrain the later ones, one should be careful that the early constraints *are* satisfiable and desirable.

The framework to be presented is based upon the notion of an *actor.* Actors are computational entities that communicate by passing messages. Each actor, containing both state and program, has the full power of a digital computer. Hewitt and others have argued for the usefulness of actors in the construction of large AI systems [2] Actors provide a modular and convenient means of implementing systems built on the framework about to be presented.

The model presented here begins with the exploration of a set of related choices. Each *choice point* starts by gathering up *suggestions* by asking the elements of its choice" for suggestions. The choice points evaluate the suggestions: combining closely related ones, classifying any conflicts, and noting any missing information. The choice points which run into trouble ask permission to *postpone* themselves until more has been decided, while the satisfied ones make their choices based on the suggestions gathered. Permission to postpone is granted depending upon the reason for desiring postponement, the state of other choice points, and whether the choice in question is part of the *focus* of the object being created. The hope is that when the choice point is awakened more information (or constraints) will be available *and* the choice will be easier. If permission to postpone is refused, more effort is expended despite the difficulty (e.g., more suggestions *are* gathered or conflicts are resolved). If no more progress is possible then a choice is made and its justification is recorded.

An Example

As part of my doctoral research I implemented a system called "Ani" that makes aesthetic choices ([1], [3]). It creates simple computer animation in response to high-level incomplete descriptions The user describes the personalities of the characters (e.g. shy), their physical characteristics (e.g. ugly, powerful), the relationship between the characters (e.g. hates), and their interactions (e.g., one prevents another from meeting a third). The user also describes the style of film desired (e.g. varied, simple, obvious). Ani's job is to decide where the characters should be placed, how they should move, and what they should do.

Ani was given a simple version of the story of Cinderella to animate. One of many of Ani's problems is to choose typical speeds for Cinderella, her stepmother, the fairy godmother, and the Prince which are in keeping with their personalities, physical characteristics, and the desired style of the film. The relative speeds of the characters should, in turn, be in keeping with the relationships and comparisons of the characters. A simple example of aesthetic choice is the determination of these typical speeds. This choice is aesthetic because the character of movement is important in assessing the quality of animation, because it is interdependent with other choices, and because it is only weakly constrained by pragmatics.

Choice points are created to represent the process of picking typical speeds for each character. The choice point for the stepmother's speed, for example, begins by asking each of the descriptors of the stepmother for suggestions for her speed. Only the description powerful" replies and suggests a high speed. The choice point is not happy with just that because there are not enough strong suggestions. So it asks permission to be postponed to wait for more information to become available and it is granted.

When the choice point for the stepmother's speed is reawakened, it inspects its record of its previous activations. It then asks the choice points for the relative speeds of the stepmother and the other characters for suggestions. These choice points are created in response to this request and they choose values (e.g., that the stepmother be faster than Cinderella because she dominates Cinderella and differs from her), but cannot make any concrete suggestions since none of the characters have speeds yet. The choice point for the stepmother's speed asks permission to postpone to wait for the speeds of the others to be determined and it is granted.

The choice points for the other characters also ask and are granted permission to postpone. This could potentially lead to a deadlock in which the four choice points wait for the others to make a decision. One of the reasons the choice points don't just postpone themselves, but instead ask permission first, is to avoid this type of situation. A postponement manager keeps track of the situation and will not grant someone permission to postpone for the same reason twice. A common exception to this is when the choice point is waiting for other choice points to finish and at least one of these is making progress. In this case, no one is making progress so the postponement manager must refuse permission to at least one of the choice points.

Ani is built upon the principle that as few decisions as possible be determined arbitrarily. The decision as to who should be refused permission to postpone has too many consequences to be determined by something like who asks first. Instead the postponement manager asks the *focus* which, in this case, indicates that conveying the personality of Cinderella is important. The choice point for Cinderella's speed is refused permission to postpone and the deadlock is broken. This means Cinderella's speed will be based on the description of Cinderella without being constrained to be faster or slower than the others.

The choice point for the stepmother's speed finally gets suggestions from the relative choice points. It discovers conflicts with one of these suggestions and the earlier suggestion it had received from "powerful" and postpones again. Upon being resumed the choice point asks the descriptions of the film's style for suggestions and receives them from the moderate variety level, high energy level, and low flashiness. Unfortunately they do not all agree and so the choice point postpones one more time.

When it is reawakened it discovers that there are no more sources of suggestions and proceeds with what it has. First it attempts to make compromises between the conflicting suggestions and makes one that in turn generates a new conflict. Excuses are found for rejecting some of the conflicting suggestions. The choice point finally picks a high speed for the stepmother and saves away a justification for this choice.

The Major Mechanisms

The model of aesthetic choice presented here consists of (a) suggestions (together with a means of gathering, combining, and resolving conflicts between them), (b) choice points to organize and record progress on choices, (c) a means of deciding when to postpone (and resume) work on a choice, and (d) a means of focusing upon the more critical elements.

Suggestions Suggestions are like rules, advice, or hypotheses that are rejected, modified, combined, and compromised. Suggestions differ from facts in that they typically conflict among themselves, are rejected or compromised, and possess degrees of reliability or strength. Suggestions, *are* often ignored or modified with minor consequences. Ani is constantly faced with contradictory suggestions and spends much time detecting, classifying, and resolving these conflicts. Conflicts are resolved by making compromises and by rejecting some of the suggestions.

Choice Points A choice point represents the exploration of a choice. Each choice point responds to messages asking it to accept suggestions, to combine suggestions, and to make choices. A choice point maintains in its database records of the best suggestions so far, conflicts between suggestions, postponements, and the sources of suggestions that have already been tapped. Choice points decide whether a set of suggestions is adequate, more work needs to be done, or permission to postpone should be sought.

Conflict Resolution If a choice point finds no conflicting suggestions for a value, the decision is simple and it just picks the value suggested. More typically there are many suggestions and they don't all agree. It is important that these conflicts be resolved as sensibly as possible if there is to be any coherence. The general approach is to use the heuristic with the strongest criterion of applicability. The criteria for making a choice between two conflicting suggestions are (a) the strength of the suggestions, (b) the degree of compatibility with the other suggestions, (c) the extent to which the sources of the conflicting suggestions have other suggestions followed, and (d) the number and kind of sources of the suggestions. If the difference between the conflicting suggestions is great along any of these dimensions, the associated rule is used.

Postponement Postponement is an important component in the making of choices that are interdependent. One choice adds new constraints that strongly influence later choices. Aesthetics only emerges when there is a set of inter-related choices; one hesitates to call an isolated atomic choice aesthetic. Because of this the order in which subproblems are attacked is very important. We want the relatively straight-forward choices to be made first because they already have a strong consistent justification. We want the choices with the least basis for a decision to be postponed as long as possible in the hope that by the time a choice finally has to be made additional constraints will have been added. Otherwise the choice would have to be made on a relatively arbitrary basis and the consequences (the additional constraints), if chosen badly, will cause trouble. Additional constraints often make the choice between the conflicting alternatives easier. The result of this control strategy is that the final product is more coherent and less arbitrary.

The postponement mechanism is designed so that the choices with the most justification are made first, then those that are difficult due to conflicting suggestions, followed by those with the least to go on. The order in which postponed choice points *are* forced to continue is under-determined by these criteria. The unordered choices could be made in parallel, however the interdependencies between them are such that the order of execution could adversely affect some of the decisions.

Focus To help avoid this arbitrariness, we have a structure called the *focus* of the object being created. It describes the parts or aspects that are primary or deserve emphasis (e.g., Cinderella's personality, her relationship with her stepmother, the second scene and so on). Those choices that relate to the focus (e.g., Cinderella's speed) tend to be made first and so are more likely to be self-consistent and effective because they are not constrained by choices yet to be made.

Minimization of Arbitrariness and Maximization of Coherence The goals of this process of choosing is that the arbitrariness of each choice be minimal and the coherence of a set of choices be maximal. In Ani's filmmaking this means that each choice of any consequence should be compatible with the description of the film and Ani's general animation knowledge. Arbitrariness is minimized by use of knowledge in the form of suggestions and from the guidance of the focus. Coherence in this context means that the choices for the relative dynamics of the characters be satisfied and that the choices of activities of the characters be self-consistent and compatible with the choices of the character dynamics. Coherence results from the control structure that postpones troublesome choices and that focuses on the relatively more important ones. The aesthetics of Ani's films a*re* a result of this striving for coherence, this minimization of arbitrariness, and the currently small amount of knowledge about animation and emotions that Ani brings to bear.

Relationship to Other Work

The most illuminating comparisons of this work with others a*re* briefly described below. A more complete discussion can be found in [1].

Meehan's Talespin One system that creates objects that are usually judged primarily on an aesthetic basis is Meehan's "Talespin" [4]. It makes up fables about talking bears, birds, and so forth. Talespin is told the initial conditions (e.g., a bear is hungry and a bird in a tree is sleepy) and spins a tale based upon a high-level simulation. The characters of the story generate plans to satisfy their needs and desires. Executing these plans causes the characters to interact forming the substance of Talespin's stories. The resulting stories are plausible but typically not very interesting or aesthetic since there is no higher-level structure to the stories and no notion of style or focus. Meehan's research, despite the problem domain, is primarily concerned with plans and symbolic simulation, not with aesthetics.

Talespin is often faced with aesthetic choices: what name to give the bear, where the crow should be, what kind of food should be available, and so on. These story aspects are aesthetic in our

culture because they are aspects that we try to interpret as the result of purposeful choices made by the author. Not every aspect of a story is aesthetic. The number of letters in the name of a bear, the locations of the word "crow" on a printed page, and the amount of ink used in a description of food *are* not typically considered aesthetic choices. And writers rarely make deliberate choices regarding these aspects.

Talespin, however, rarely makes deliberate choices for those aspects that are normally considered aesthetic. Instead it either is told by the user, "chooses" randomly, chooses based upon a symbolic simulation of the characters involved, or chooses that which will help give the story a particular moral. The first two cases *are* not choice-making at all. The third is *an* interesting alternative to (or supplement of) the model presented herein. The difficulty is that simulation is concerned with plausibility, with having the components (the characters in the story) behave in a reasonable way. Aesthetics, on the other hand, is concerned with creating the world that such a simulation occurs in, with determining the rules of interaction, the goals of the components, and the initial conditions. The moral-fulfilling aspect of Talespin is more relevant to aesthetic choice. Unfortunately, it is a small, under-developed part of a large system.

Lenat's Artificial Mathematician One of the more creative A. systems of late is Lenat's AM *[5]* AM starts with very elementary concepts such as sets, composition, and equality. AM *creates* many new concepts, makes conjectures, and discovers new aspects of the original concepts. AM's heurstics propose tasks to perform that are explorations of this space and these are placed upon an agenda. The tasks on the agenda that are most interesting are performed first.

AM is probably the research most related to the work described here. This may seem odd since AM's domain is elementary mathematics which is very formal and well understood — almost the antithesis of aesthetics and art. Mathematics is formal and good models of it do exist, but, as Lenat points out, the *exploration* of mathematics, the heurstics that guide one in making conjectures, in constructing new concepts, and in evaluating them, in other words, the *doing* of mathematics is neither formal nor well-understood. The problems that Ani and AM address are both weakly specified: making *good* animation and discovering *interesting* mathematics. Both systems construct structures out of a *very* large space of possibilities. Since the creations of AM and Ani *are* not judged as right or wrong — but as interesting or dull, plausible or implausible, good or bad — aesthetic choice plays a crucial role. Both AM and Ani are knowledge-oriented, in contrast with other approaches which are simulation-based, search-oriented, or based upon a few very general pieces of knowledge. Both systems spend a considerable portion of their time deciding what to do, in addition to doing it.

There are many differences between the two systems, of course. The proposed tasks of AM can be viewed as suggestions from various heuristics as to what should be explored, however they are not treated as described above (e.g., combining, compromising, relating, and so on). AM has a focus of attention which tends to keep AM from jumping from topic to topic. Ani's focus instead influences the relative priority of the elaboration of the different parts. The control structures of the two programs *are very* different. AM keeps executing the most interesting task on its agenda, while Ani jumps from choice to choice on the basis of their past difficulties and the focus. AM executes its tasks in a fairly straight-forward manner, white Ani works on a choice by gathering, combining, and rejecting suggestions, noting and classifying conflicts, making compromises, and searching for more suggestions.

Discussion

Generality To some it is plausible that the model of aesthetic choice presented here is applicable to any set of choices that are not primarily constrained by pragmatics, whether or not there are aesthetic considerations. Others think the framework is only appropriate for the kinds of choices that Ani makes in creating simple films and *are* skeptical of any claims of generality. This issue is separate from the question of how *easy* it is to apply the framework described here to a particular domain. It may be very difficult to apply it, say, to the production of oil paintings, not because it is a bad framework, but because it is very unclear how to structure the space of choices, what the sources of suggestions *are,* where to get the necessary knowledge in a complete and detailed enough form and so on.

Nonetheless, one should be suspicious of claims of generality when something has been applied only to one example. In progress is another test of the model of aesthetic choice. The model is being applied to the design of block diagrams that illustrate papers and lectures. The problem is to choose locations and dimensions for the boxes, to decide where links between boxes should originate and end, where to place labels and in what font, and so on. Preliminary results indicate that the notions of suggestions, postponement, *and* focus have their natural places in this application. This will be described in detail in a subsequent publication.

What is Missing The framework presented for making aesthetic choices is simple. Several components are missing for building high performance systems. No means of undoing poor choices is included, for example. Also the *structure* of the set of choices explored is given to the system, while ideally that loo should be determined by the system. The examples discussed are limited to the selection of a single value for an element of a complex description. Many aesthetic choices are not the selection of values, but the generation of complex descriptions. Much of Ani's time, for example, is spent deciding what should happen in the scenes of the film. The problem is to create a description of the activities that should occur in a particular portion of a scene. While there are many similarities and parallels between this process of choosing activities and the selection of values, some special mechanisms were needed and are described in [1]

Where to Go from Here One avenue of future research is to attempt to apply the framework presented here to other domains to discover its shortcomings and strengths. Part of this involves extensive testing of computer systems like Ani. The focus, conflict resolution rules, or postponement criteria can be modified and the effects can be observed.

Another avenue of research is to attempt to fill in the missing components of the framework. The inability to undo previous decisions, for example, can be quite serious. The decision making process needs to be able to recognize when things are not right and to generate directed criticism to the choice points responsible. Choice points need to be extended to take such criticism into account and redo just that portion which needs changing.

Finally, one last avenue of further research is to formulate a general theory of aesthetics and incorporate it into the model. If formal aesthetic judgments can be made about the description and justification of a creation, then their discovery and inclusion in a model of aesthetic choice is a fascinating possibility.

Bibliography

[1] Kahn, K. *Creation of Computer Animation from Story Descriptions,* MIT PhD thesis 1979 (AI Lab TR in progress)

[2] Hewitt, C. "Viewing Control Structures as Patterns of Passing Messages", *Artif. Intell.,* 8:3 (1977) 323-364.

[3] Kahn, K. "Ani: An Example of Computational Creativity", Proceedings of the AISB/GI Conference, Germany, July 1978

[4] Meehan, J. *The Metanovel: Writing Stories by Computer*; Yale University Computer Science Research Report 74, 1976

[5] Lenat, D. *AM: An AI Approach to Discovery in Mathematics as Heuristic Search,* Stanford AI Lab Memo AIM-286, 1976

CONCEPTUAL LATTICE: A UNIFIED MODEL FOR MEDICAL INFERECE PROCESSES

Tsuguchika Kaminuma
Medical Informatics Group
The Tokyo Metropolitan Institute of Medical Science
3-18-22 Honkomagome, Bunkyoku
Tokyo 113, JAPAN

The conceptual system in medicine can be ordered by a relation between two concepts. We can then define a lattice structure over the ordered conceptual system. Such a lattice whose elments are the aggeregates of concepts can be used to simulate the inferential process of medical decision making. This model can potentially unify all models so far proposed for computer-based medical decision making, from both the statistical and empirical (AI) approaches. When combined with the semi-empirical approach, the model is able to constitute the computer algorithm which is most suited to attack the diagnostic problems, even where neither number of sample data per disease categories are large nor reliable medical knowledge has been provided. The model is also a good guideline by which one can clarify relations between medical terms and concepts, and to organize logical thinking into a knowledge system.

1. INTRODUCTION

The history of attempting to use the computer as a tool for clinical decision making goes beyond two decades. Since the beginning there have been two approaches in designing the computer alogrithm for medical decision making: The statistical approach [1]—13] and the empirical approach [4]-[6], the latter has been customarily called the artificial intelligence (AI) approach by its school.

The statistical approach is successful whenever enough samples can be accumulated for each category of diseases, but quite useless for more complicated differential diagnosis where reliable data samples are small. The AI approach illustrates its strength when the decision categories are extensive but also where sound medical knowledge for reasoning exists. Here the statistical reasoning based on the objective probabilites calculated from data is replaced by a decision criteria based on an expert confidence measure. But these theories seem to show their weaknesses when applied to the areas where empirical knowledge is neither well organized nor reliable.

Since 1971, the author has been working on the computer diagnosis of heart diseases based on a third approach, called "semiempirical" [7]. In the semiempirical approach, the specialist's diagnostic reasoning is first analysized and broken down into more elementary processes. Then the decision process is reconstructed by a hierarchy of elementary decision compartments. The logic at each decision compartment may be given empirical knowledge at the beginning, but these empirical logics can gradually be replaced by probabilistic decision logics as statistically reliable data accumulates. The semiempirical approach thus builds a bridge between the statistical theories and the AI theories, and overcomes the limitations of the both approaches.

One of the gists of the computer-based clinical derision making is the modelling of dieases and simulating the expert's thinking processes based on the resultant model. In this paper a representation model for human inference proce-sses based on a structure disease model is proposed. This model is a refinement and extention of the algorithmic frame, developed in the semiempirical approach. The model is called conceptual lattice model, because it uses relation among medical concepts. The model is not only relevant for the models proposed up to now, but also useful to unify the semiempirical approach, from the statistical to AI approaches.

2. CONCEPTUAL SYSTEM FOR MEDICAL KNOWLEDGE

Medical knowledge, as a conceptual system, has two embedded structures: biostructural (i.e., etiological, pathological, physiological, and

anatomical) semantics and hierarchical human information processings. If we consider any disease etiologically, we see casual relations between concepts parallel to the biological hierarchy. But the present biostructural knowledge in medicine is far from complete, and the biostructural conceptual systems cannot directly be modelled into computer programs.

The structure of human information processing for medical decision making is better ordered, and suitable to a multi-stage and multi-level information reduction process model. Usually a patient takes several tests in sequence. At each test the primary information concerning the patient's body or his specimens are reduced into symptoms and signs, and the medical consultant determines probable diseases from the information.

The ideal situation is that of human guessing based on observations going side by side with the biostructural casual reasoning. Conversely, the biostructural conceptual system may suggest the necessity for new observations. However in reality the medical consultant's mind works neither purely by observation nor on a biostructural basis. His thinking is somewhere in between. The conceptual lattice model thus intends to abstract the conceptual system structure embedded in the human mind.

3. CONCEPTUAL LATTICE

3.1 Definition

Let concepts used in medical decision making be *a*, b, c ,• • • , *y*, and *z*, etc. In the case of necessity we distinctively use a, b, and c for definite expression with yes/no or with constant values included such as "having cyanosis" or "age less than 5 years old", while the x, *y,* and *z* are used to express variables, such as "the number of white blood cells," or "the systolic blood pressure" without specifing their values. In general the a, b and c symbols correspond to propositions appearing in medical textbooks. If a concept a appears as an element to explain concept b, we write a--> b , in other words, we need the concept *a* in order to define the concept b in some knowledge frame in current medicine. We assume *a* -----> *a* always nolds.

Let (x, B , Y ••• be the aggregate of concepts, we introduce lattice structure between the aggregate of concepts, which are called "conceptual lattices". For example, we may extract a partially ordered set of concepts from any given conceptual system having the relation -▶.

a disjoint set of subsets (aggregate of concepts) α , β, ..., and introduce a new realtion ➡ w hich is defined by " $\alpha \rightarrow \beta$ iff for any $a_i \in \alpha$ there exists a $b_j \in \beta$ such that $b_j \rightarrow a_i$ and for any $a_i \in \alpha$ there exists a $b_j \in \beta$ such that $b_j \rightarrow a_i$", we again obtain a new lattice under certain additional conditions. The operations $\cap$ and $\cup$ can be introduced as l.u.b.(α, β) and g.l.b. (α, β) respectively, as usual [8].

To illustrate that the conceptual lattice model originates from a common language in science, few examples of conceptual lattices will be given. From an extremely reductionistic view point a patient is considered a physical system, whose state may either be represented by a boolean propositional lattice (classical case) or a weak modular lattice (quantum case) [9]. In usual pattern recognition language, a patient is represented by a n-tuplet observation, and the diseases correspond either to regions or subspaces in the *n*-dimensional space. These pictures are again examples of a boolean and a modular lattice system [10].

3.2 Inference Compartments

Between any two conceptual lattice elements $\alpha = (a_1, a_2 \ldots, a,)$ and $B = (b_1, b_2 \ldots, b)$ which satisfy $\alpha \Rightarrow B^m$, we say that there exists an inter-nodal logic, which is a mapping between the two element concepts of the lattices. Since there may be several other lattice elements $\alpha'', \alpha'', \bullet .$, for the element B with the same relation =>, the values of b_i cannot uniquely be specified unless giveing a mapping which determines the values of b_i uniquely based on the outputs of the above mapping. This new input-output relation within the lattice element, when specified by an explicit function relation, is called the compartment logics.

When compartment logics are specified, the conceptual lattice is called a state lattice. Elements of the state lattice are called state nodes. Concepts included in the state nodes are called state variables. Thus a state lattice which corresponds to a computer algorithm of diagnosis, is the model of physicians inference process for diagnosis. However a conceptual lattice only gives an algorithmic frame, and not the algorithm itself.

In order to specify a compartment logic one needs an empirical logic. Usually these logics are given by "If..., then..." sentences written by a medical consultant [4]. But for checking logical consistency, and for improving and up dating the logic, a logic table is preferable, because the size of the logics is usually very

large. Moreover table logic can easily be transformed into a probabilistic logic, such as linear discriminant function, as data is gathered. Thus a translator, which generates a table logic matrix from conversational "If... then..." type sentences becomes useful [7].

3.3 Models for Progonosis

The applicability of the conceptual lattice model for progonosis problems is briefly explained. Prognosis is the prediction of patient state transitions. In our model, state transitions are represented by transitions between two state lattices. Symbolically we represent a patient's state at time t by a state lattice $\Psi(t)$ having $\chi_1(t)$, $\chi_2(t)$, ... as its nodes. As was explained, these nodes may either be the concept (physiological variables) or an aggregate of concepts. We then denote the transition probability between two states $\Psi(t_i)$ and $\Psi(t_j)$ by

$$\{ \Psi(t_i), \Psi(t_j) \}. \qquad (1)$$

We may also consider the two probabilities

$$\{ \chi_1(t_i), \Psi(t_j) \}, \qquad (2)$$

and

$$\{ \chi_1(t_i), \chi_m(t_j) \} \qquad (3)$$

If we take χ's in (3) to be a same physiological parameter such as blood pressure, (3) predicts its change. However such predictive probability may not be known in general.

4. DISCUSSIONS

As we have emphasized the conceptual lattice model is not an algorithm for compuer-based decision making, but it is a methodology to generate an algorithmic "frame" with which one can draw in the body of logic which is "squeezed" from human empirical thinking.
One of the salient feature of the above methodology is that the computer alogrithm accepts some incomlete data, which a usual probabilistic logic can not handle. Another big advantage of the conceptual lattice model is its evolutional feature, which uaual AI approachs do not emphasize.

Despite such partial success the model is still in its development stage. Several applications of the model has been attempted for heart, metabolic, and hematological diseases.

ACKNOWLEDGEMENTS

The author owes his medical knowledge to Dr.s K. Machii, Y. Yahata, and S. Kurashina. He also would like to thank Mr. I. Suzuki for his helpful discussions.

REFERENCES

[1] Warner, H.R., Toronto, A.F., and Veasy, L.G. "Experience with Bayes' Theorem for Computer Diagnosis of Congenital Heart Disease." Ann. N. Y. Acad. Sci., 115, (1964) 558-567.

[2] Gorry, G.A. and Barnett, G.O. "Experience with a Model of Sequential Diagnosis." Comp. Biomed. Res. 1, (1968) 490-507.

[3] Nordyke, R.A., Kulikowski, C.A., and Kulikowski, C.W. "A Comparison of Methods for the Automated Diagnosis of Thyroid Dysfunction." Comp. Biomed. Res. 4 (1971) 374-389.

[4] Shortliffe, E.H., Computer-Based Medical Consultation: MYCIN. N.Y.: Elsevier/NH, 1976.

[5] Weiss, S.M., Kulikowski, C.A., Amarel, S., and Safir, A. "A Model-Based Methods for Computer-Aided Medical Decision-Making." Artificial Intelligence. 11 (1978) 145-172.

[6] Svolovits, P. and Pauker, S.G. "Categorical and Probablistic Reasoning in Medical Diagnosis." Artificial Intelligence. 11 (1978) 115-144.

[7] Kaminuma, T. and Machii K. "Semiempirical approach to computer diagnosis." In Proc IJCPR-78 . Kyoto, November, 1978 pp. 889-891.

[8] Birkhoff, G. Lattice Theory. Providence: Am. Math. Soc, 1973.

[9] Piron, C. Foundations of Quantum Physics. Reading: Benjamin, 1976.

[10] Watanabe, S. Knowing and Guessing. N.Y.: John Wiely, 1969.

A THEORY OF THE ORIGAMI WORLD

Takeo Kanade

Department of Information Science
Kyoto University, Kyoto, Japan

1. INTRODUCTION

This paper presents a brief summary of a theory of the Origami world[4], a model for understanding line drawings in terms of plane surfaces and for finding their qualitative 3-D shapes by assigning the labels (+, -, ↑ or ↓) to each line. The labels signify the physical meaning of the lines (Huffman [2]): the label + stands for a convex edge, - for a concave edge, and ↑ or ↓ for an occluding edge.

The theory is an extension of the work of Huffman [2] and Mackworth [6], but the feature is that it is surface oriented, that the precompiled knowledge is all contained in the augmented junction dictionary, and that the labeling is all symbolically performed using the filtering procedure not only for a consistency check of junction labels as in Waltz [7] but also in the consistency check of surface orientations in the gradient space.

2. SUMMARY OF THE THEORY OF THE ORIGAMI WORLD

2.1 Surface-oriented Assumption

An important feature of the Origami world is that it is surface oriented. This idea can be best illustrated by Fig. 1. Though it appears perfect, the Huffman-Clowes-Waltz labeling scheme for the trihedral world cannot handle it. The reason for this failure is that the trihedral world essentially assumes solid objects, and thus the picture of a box would need to be "super" perfect, as in Fig. 2, in order for it to be handled.

The assumption concerning the geometrical configurations around a vertex in the Origami world is as follows. The planar surfaces meet edge to edge, *no more than three* surfaces of different orientations meet at a vertex, and the combination of the three orientations is general in the sense that they span the three-dimensional space. Let us call such vertices up-to-3-surface vertices.

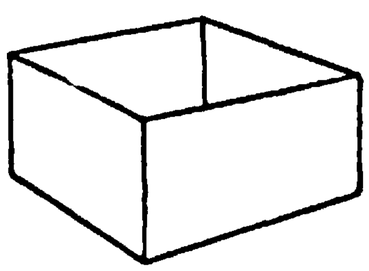


Figure 1. A line drawing of a "box" *scene.* Though it looks perfect, the trihedral labeling does not work for it.

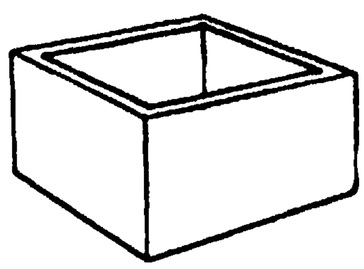


Figure 2. A "super-perfect" line drawing of the "box" scene for the trihedral world,

2.2 Legal Junction Labels for Up-to-3-Surface Vertices

Once we recognize the basic assumptions of the Origami world, we can generate the junction dictionary, as in the Huffman-Clowes-Waltz theory, which contains legal junction labels (i. e., possible line-label combinations) for each junccion type. Table 1 gives an idea of the degree of constraints imposed by the up-to-3 surface vertices compared with the trihedral vertices.

2.3 Augmented Junction Dictionary

Necessity j A legal junction label represents a possible configuration of surfaces at a vertex. Consistently labeling a line drawing, so that all the junctions are given legal junction labels, is nothing but checking the consistency of surface interconnections by passing information by means of line labels from one junction to another. The Waltz filtering [7] on junction labels is known to be a good method for doing this. However, the labeling in the Origami world cannot simply rely on the filtering on junction labels. Because of the weaker restriction at the vertices than the trihedral world, a lot of anomalous interpretations exist in which the labeling is consistent, but the whole configuration is not possible [6].

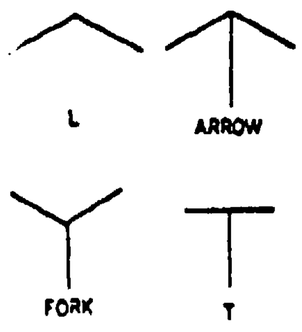
Links : The junction dictionary for the Origami world is augmented for more thorough and global consistency check concerning surface orientations. The surface orientations are conveniently represented by means of the gradient space[2][6]. To each legal junction label is attached the information as to what constraints in the gradient space should be satisfied by the surfaces incident at the junction. Fig. 3 shows two examples. As shown in Fig. 3(a), the constraints are represented by the links which connect a pair of related regions and which include information about the constraints on their gradients. The properties of dual lines of the gradient space are used here.

In such a junction label as shown in Fig. 3(b), which is legal in the Origami world, we have an occluded intersection. It is typicall/ a result of folding a planar surface along BC, thus RI occludes a part of R2. A little more generally, we can include such cases that the intersection line of RI and R2 lies within the angle ABC. Therefore, the associated link represents that the gradient G2 of R2 should be inside of the fan-snaped area

* This research was done while the author was visiting Department of Computer Science, Carnegie-Mellon University, Pittsburg, Pa.

Table 1. The sizes of junction dictionaries.

Junction Type	Huffman-Clowes Dictionary	Origami World Dictionary
L	6	8
ARROW	3	15
FORK	3	9
T	4	16



whose origin is at G1, and is bounded by the lines which are perpendicular to AB and BC.

2.4 Labeling Procedure

The labeling procedure of the Origami world uses the augmented junction dictionary. First, the Waltz filtering on junction labels is performed. Then, the procedure begins to assign a junction label, which has been filtered and left, to each junction one by one. When a junction label is assigned to a junction, the constraints represented by the associated links are instantiated by the directions of lines of that particular junction.

Consistency of surface orientations is tested by using these instantiated constraints. The test can be performed by an iterative "filtering" operation defined on a labeled graph called a Surface Connection Graph (SCG), in which a node represents a surface, and an arc represents the instantiated constraint between the surfaces. The feature of this labeling procedure of the Origami world is that all the constraints are maintained symbolically in the SCG during the computation.

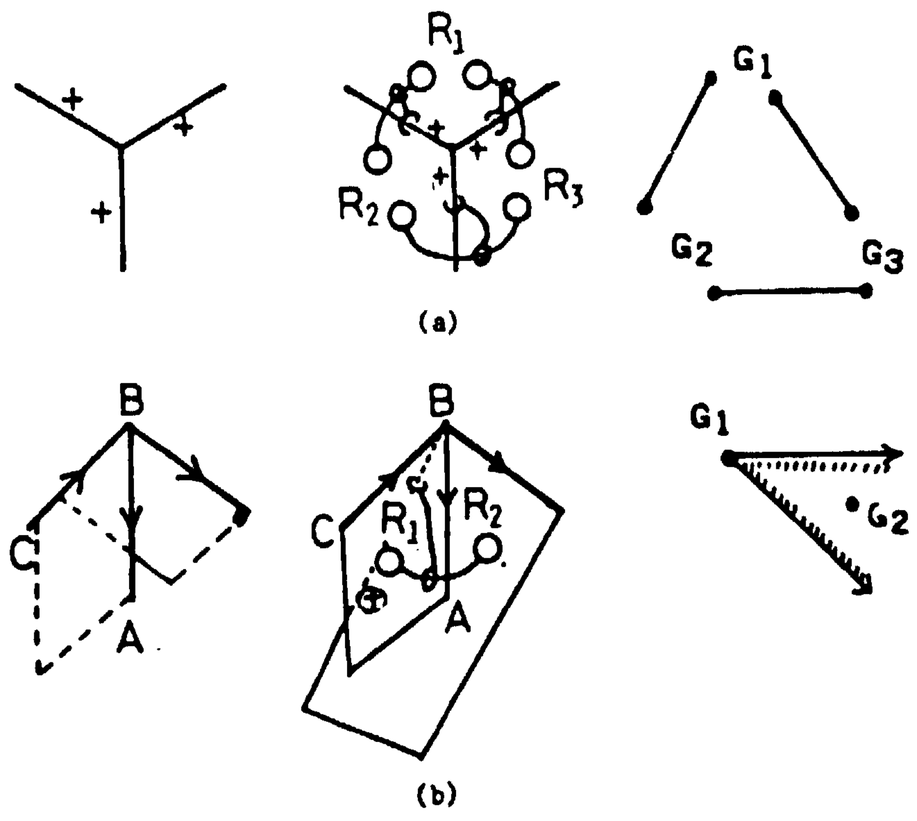


Figure 3. The augmented junction dictionary for the Origami world. Two examples of entries are shown. For each, the first column is the junction label, the second the associated links, and the third the illustration of the relationships of gradients represented by the links.

2.5 The Results of Labeling

As a result of the labeling procedure, we obtain not only a labled line drawing but also a filtered SCG, which is a summarized description of the constraints on the gradients of the involved surfaces

Let us see a few examples of labeling. Usually we obain multiple labelings. The "cube" scene has three labelings (Fig. 4). The "box" line drawing of Fig. 1 has eight labelings, two of which are shown in Fig. 5. The labeling in Fig. 5(a) corresponds to an "ordinary box"; the two front faces form a convex intersection, and partially occlude the rear two faces which form a concave intersection. Fig. 4(b) corresponds to a "squashed box"; the front two faces, as well as the rear two, form a concave intersection.

3. DISCUSSIONS AND COMMENTS

3.1 Origami World and Various Worlds

Assume we have a set of line drawings. We can consider a set of al 1 the combinations in assignments of line labels for those line drawings. A subset exists comprising those interpretations which can be realized by plane surfaces. Let us denote this subset as the Plane Surface World, Spsw. We can also think of a subset, the Consistent Gradient World Scgw, consisting of interpretations in which all the constraints on the graidents of surfaces are completely satisfied. Obviously ScgwDSpsw.

We can view a labeling procedure as a method consisting of a generator and a tester: given a line drawing, a generator generates interpretations in a certain manner, each of which a tester accepts or rejects based on a certain criterion. Table 2 summarizes various labeling methods [1][2][3][6][7] according to this taxonomy. Various subsets can be defined which are generated by generators or determined as legal by testers. Fig. 6 shows the set inclusion relationships among those subsets. The locations of several example interpretations indicated in it can serve to understand the relationships. (See [4] for details).

3.2 Qualitative Shape Recovery

Note that we have only qualitatively recovered the 3-D shapes by labeling drawings. For example, the labeling of Fig. 4(a) represents a convex corner, but it can be either a cube or a skewed rohmboid. In fact, a "cube" scene ( ), a "trapezoid-block" scene ( ) and a "house" scene ( ) all have the equivalent labelings. In order to recover the shape quantitatively we need to introduce more assumptions together with tools for exploiting them.

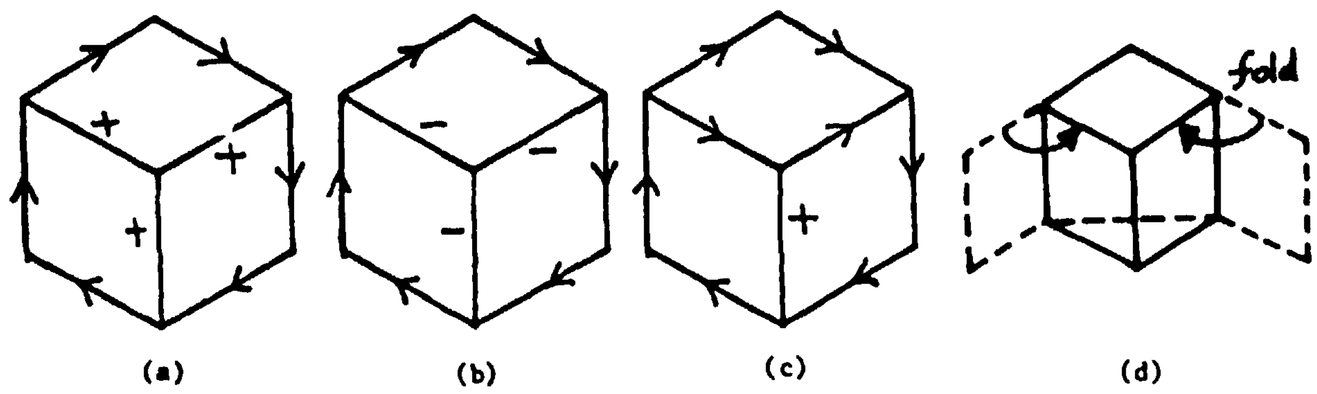


Figure 4 . The "cube" scene has three labelings: (a) a convex corner; (b) a concave corner. (c) This represents a shape which can be made as shown in (d)

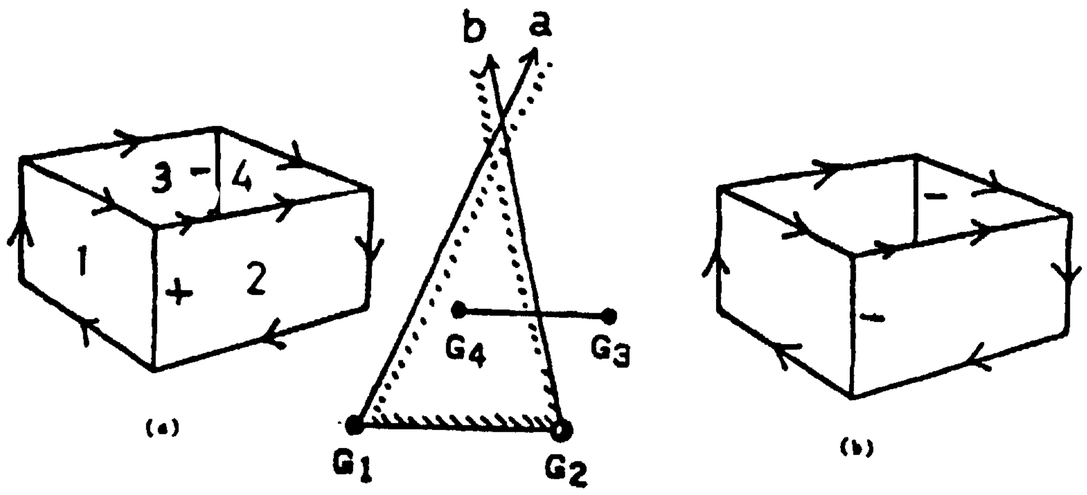


Figure 5. There are eight labelings in total for the "box" scene. Two of them are shown here: (a) corresponds to an ordinary box; (b) a "squashed" box The diagram represents the constraints in the gradient space for the labeling (a).

3.3 "Natural" and "Unnatural" Interpretations

One of the common impressions we tend to have about those multiple interpretations is that most of them are "unnatural", and therefore it is a disadvantage of the theory to allow them. It is true that we do not usually think of such peculiar shapes as Fig. 4(c) and 5(b) in interpreting these line drawings. It should be very difficult even to imagine those shapes as possible interpretations. However, any labeling is geometrically no more "natural" or "unnatural" than others, *unless* we use more assumptions.

The above two points (3.2 and 3.3) are actually closely related problems, for whose solution we need to identify more assumptions concerning image formation process and objects in concern. The recent paper [5] by the author presents a theory and a technique which can recover "natural" quantitative 3-D shapes of an object from a single picture, so that we can generate images of the same object as we would see it from other directions.

ACKNOWLEDGEMENT

I thank John Kender, Allen Newell, Raj Reddy, and Steven Shafer for having very stimulating discussions with me.

REFERENCES

[1] Clowes, M.B., "On Seeing Things", Artif. Intell. 2, p.79, 1971. [2] Huffman, D.A., "Impossible Objects as Nonsense Sentences", Machine Intelligence 6, Edinburgh University Press, p.295, 1971. [3] Huffman, D.A., "Realizable Configuration of Lines in Pictures of Polyhedra", Machine Intelligence 8, Edinburgh University Press, p.493, 1977. [4] Kanade, T., "A Theory of Origami World", Technical Rep. CMU-CS-144, Carnegie-Mellon University, Pittsburgh, Pa. 1978. [5] Kanade, T., "Recovery of Three-Dimensional Shapes of an Object from a Single View", CMU Tech. Report, 1979. [6] Mackworth, A.K., "Interpreting Pictures of Polyhedral Scenes", Artif. Intell. 4, p.121, 1973. [7] Waltz, D., "Generating Semantic Descriptions from Drawings of Scenes with Shadows", MAC AI-TR-271, MIT., 1972.

Table 2. Various labeling schemes as a method of a generator and a tester.

Method	Generator	Subset	Tester	Subset
Huffman Clowes Waltz	Trihedral junction dictionary Trihedral junction dictionary with cracks, shadows, etc	S_{tri}		
Mackworth	Sequential generation of most connected interpretations		Constructive test on coherence rules in the gradient space	S_{poly}
Huffman			$\phi(\phi')$ point test for all the cut sets in the line drawing	$S_{\phi(\phi')}$
Origami World	Up-to-3-surface junction dictionary	S_{up3}	Filtering of spanning angles on the SCG	$S_{origami}$

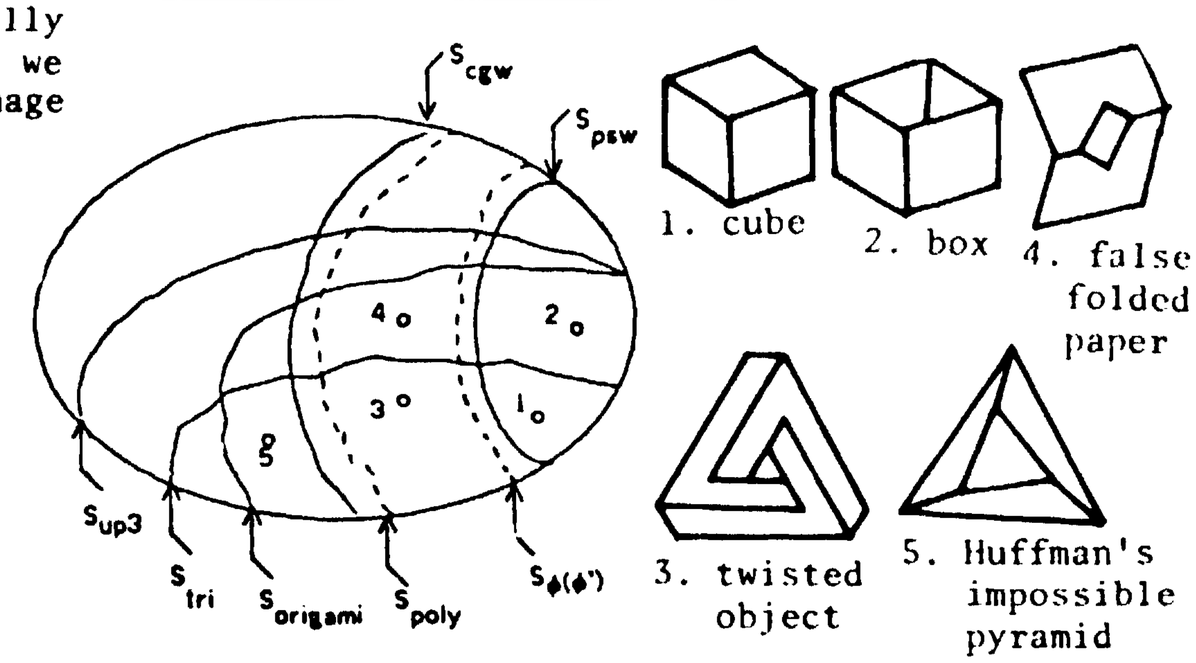


Figure 6. Relationship among various subsets of interpretations.

A KNOWLEDGE-BASED APPROACH TO USING EFFICIENCY ESTIMATION IN PROGRAM SYNTHESIS

Elaine Kant*
Artificial Intelligence Laboratory
Stanford University
Stanford, California 94305

This *paper* describes a system for using efficiency knowledge in program synthesis. The system, called LIBRA, uses a combination of Knowledge-based rules and algebraic cost estimates to compare potential program implementations. Efficiency' Knowledge is used to control the selection of algorithm and data structure implementations and the application of optimizing transformations. Prototypes of programming constructs and of cost estimation techniques ere used to simplify the efficiency analysis process and to assist in the acquisition of efficiency Knowledge associatod with new coding Knowledge. LIBRA has been used to guide the selection of implementations for several programs that classify, retrieve information, sort, and generate prime numbers.

I. INTRODUCTION

Efficiency considerations often impose conflicting demands on a program synthesis system. On the one hand, a synthesis system must produce an efficient target language program; on the other, It must produce that target code In a reasonable amount of time end without running out of storage. This paper discusses a system that takes a middle ground between the extremes of 1) constructing all possible programs that meet the specification and picking the most efficient, a*nd* 2) using default implementations. The system, called LIBRA, uses a knowledge base of *efficiency rules* to guide the construction of relatively efficient target language programs In a reasonable amount of time. LIBRA works from a more abstract specification and considers a wider range of target-language Implementations than optimizing compilers. Many choices must be made, a*nd* making a good decision depends on a global view of the program. The target programs 8re not guaranteed to be optimal, but the efficiency knowledge is designed to allow the flexibility of trading off target-program efficiency for speed and compactness in the synthesis process.

The basic paradigm Is heuristic search through a set of more and more complete program descriptions. Estimates of the execution costs of program implementations are used as evaluation functions in the search. Symbolic, algebraic program analysis is used to estimate the execution costs, knowledge about the time and storage costs of data structures a*nd* operations is used to choose combinations of algorithms and data representations *and* to control the application of optimizing transformations. Rules about plausible Implementations *are used* to prune the search tree. LIBRA has been been used to guide the construction of several variants of programs that retrieve Information, sort, classify, and generate prime numbers.

2 BACKGROUND

LIBRA is an extension of an interactive program synthesis system that generates implementations in a target language by a series of transformations and refinements of program descriptions, called *coding rules.* The knowledge base of coding rules was developed by Barstow [1]. The knowledge base allows programs In the area of symbolic processing to be specified in terms of constructs including sets, mappings, set operations, and enumeration. The knowledge in both the coding rules and efficiency rules permits the construction of programs using lists, arrays, hash tables, property lists, and several enumeration, sorting, and searching constructs. The target programs are written in a subset of INTERLISP.

Most of the rules are not specific to the target language. For example, there are 5 or 10 rules that gradually refine a set into a hash table, and then a few language specific rules for refining the hash table into LISP. Although the general paradigm is refinement from abstract to more detailed program descriptions, transformations such as combining nested blocks of code or nested loops are also allowed. LIBRA decides whether or not to apply such a transformation just as it decides which of several refinements to apply, by looking at the global execution cost estimates or by applying heuristics.

LIBRA and the coding rules function together both as the synthesis phase of the PSI program synthesis system [2] and as an independent synthesis system. Figure 1 shows a simplified view of the synthesis phase and its relation to the rest of PSI. The other modules of the PSI system allow the description of programs by English dialogue or by examples or traces, *and* translate the specification into a complete high-level language description. A specification in this high level language can also be given directly to the synthesis phase.

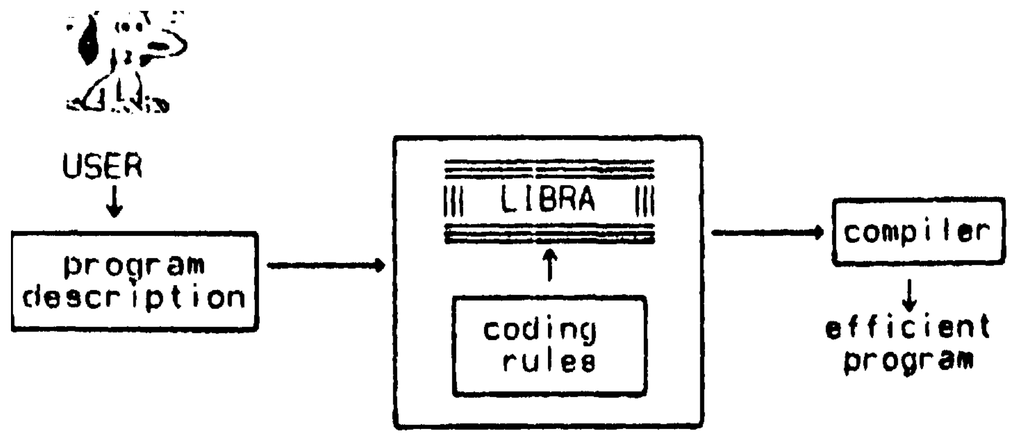


Figure 1. A LIBRA's eye view of program synthesis in PSI

3. PREVIEW

LIBRA chooses from among applicable refinements in the knowledge base of coding rules through additional sets of rules that can be easily modified. For example, rules about planning, derived from previous analyses of how to make particular implementation decisions, reduce the effort of explicitly constructing .and comparing alternative implementations. Related decisions are grouped to reduce the size of the search space and to make cost tradeoffs more obvious. Rules about scheduling and resource allocation set priorities that reflect the importance of a coding decision and the effort expended in making the choice.

* This research was supported in part by a Fannie and John Hertz Foundation Fellowship, in part by a National Science Foundation Fellowship, in part by Systems Control, Inc., under the Defense Advanced Research Projects Agency Order 3687, Contract N00014-79-C-0127, and in part by the Stanford Artificial Intelligence Laboratory *under* the Defense Advanced Research Projects Agency Order 2494, Contract MDA 903-76-C-0206.

When appropriate, alternate implementations are explicitly constructed and compared analytically. The comparisons use rlobal cost estimates to reflect the interdependence of decisions. The cost estimations can be *made* at *any* stage of the refinement process, although estimates of more completely refined programs are generally more accurate. LIBRA computes upper and lower bounds on the estimated execution cost and uses them for pruning program implementations with branch and bound. These bounds are also useful in identifying parts of the program that might lead to bottlenecks. Refinement resources are then concentrated on those parts of the *program.*

Since the knowledge-based is modular, it facilitates the acquisition of new programming knowledge. The same prototypes of programming constructs and of cost estimation procedures that simplify the efficiency analysis process are also quite useful in adding the efficiency information to match the coding knowledge that is in the system. A semi-automated process for adding new efficiency information has been developed.

The focus of this article is on the overall efficiency framework and on the knowledge-based aspects of LIBRA. More details on the analysis procedures and on other topics only covered briefly here can be found in [3].

4. THE PROBLEM

The question addressed here is how to select *an* efficient implementation for a high-level program specification, given a set of rules for constructing the possible implementations. It is assumed that there may be a *Very* large number of possible implementations and that it is not possible to construct and compare all possibilities explicitly. The desien goal was to produce a system that would *automatically* select implementations *and* that would be compatible with the refinement paradigm for program synthesis.

The following example illustrates the type of problem that LIBRA solves. The problem is to synthesize a good implementation of a simple database retrieval program.

> The program first inputs a database of news stories. It then loops, accepting a keyword command and printing a list of all stories in the database that contain that keyword, alphabetized by story name. The special keyword *"xyzzy** causes the program to terminate.

As part of the program specification, the user may specify information such as the estimated number of times a keyword command will be given, the expected number of stories in the database, and the average number of keywords per story. Some variations of this example are developed further in later sections.

4.1 Implementation issues

Given a high level program description, there are several types of implementation issues to be considered:

-- choosing data structure representations
— implementing high level operations
— applying optimizing transformations

Some of the major difficulties in resolving these issues arise from the need to consider:

— time and space trade-offs
— dependencies among decisions
— efficiency of target program versus efficiency of synthesis

Thus, in the program described above, a representation for the database must be chosen, and a method for finding the stories associated with the keyword must be chosen. If there is the an opportunity to apply a transformation such as combining two loops, it must be determined whether that transformation will actually improve the performance of the target *program.*

Often there is no ideal representation that minimizes both space and time. In the news retrieval example, the database can be represented as a mapping from stories to sets of keywords. Unless the database is relatively small, it will take quite some time to search for all the stories containing the given keyword *and* to sort that list. Another possibility is to use an additional representation of mappings from keywords to a sorted list of stories containing that keyword. If keyword searches are requested frequently, this would improve the running speed, but at the expense of additional storage space.

When more than one data structure is involved, it may not be possible to make implementation decisions independently. Given most cost functions, there will be cross-product terms involving the space from one representation and the time from an operation on another. For example, this could happen if the cost function were the product of 1) execution time of a statement, 2) number executions, and 3) total storage in use, summed over all statements in the program. These cross-product terms make it impossible to analyze the costs of the decisions independently. The best implementation choice also depends on the relative frequency of the retrieval operations and the sizes of the data structures.

4.2 Some subproblems

Some subtasks of this general problem of finding an efficient implementation include codifying the efficiency knowledge needed to:

1) symbolically estimate and compare execution costs
One way to choose a good implementation is to make several alternative refinements, estimate the costs of the resulting program implementations, and choose the best one.

2) store and apply previous efficiency analysis results
To avoid excessive analysis, it is helpful to be able to exploit the results of previous analyses. So there should be a mechanism for adding rules such as:

> "In refining a set that has more than 30 elements and that is used only to test membership and add and delete elements, the hash-table representation is a good choice."
>
> "In refining a sequentially represented set in which elements are frequently inserted and deleted, use a linked list rather than an array." (This avoids shifting.)

3) concentrate effort on important parts of the program
The synthesis system should determine whether the representation of the database has a greater effect on the global program cost than the choice of alphabetizing technique, *and* should use that information to focus synthesis resources.

4.3 Related research

Onfy some of the types of efficiency knowledge described in the previous section have been codified for machine use. The primary research has been in data-structure selection systems. Some verification and theorem proving systems can prove facts about the execution performance of programs, but tney do not use this information to guide program synthesis. The use of efficiency knowledge \n program synthesis has not been addressed by debugging or analogy approaches.

The data-structure selection systems all use cost estimation for comparison of implementations. Low [4] uses numerical cost estimates to choose data structures from among a library of implementations. To find branching probabilities, the system inserts statement counts into a default implementation that is *run* on sample data. Set sizes at different points in the program *are* determined by querying the user. Morgenslern's system, a part of PR0T0SYSTEM-1, [5], uses estimates of file input/output and sorting costs to choose file system organizations and order the flow of processing operations in management information system.

These systems include heuristics for avoiding complete search, but the heuristics are not always expressed explicitly. Low's system has a built-in rule for avoiding multiple representations by forcing all data structures to have the same representation throughout the program and by constraining all data structures that are arguments to a common operation to share an identical

representation. Rovner [6] extended Low's work to the selection of associative data structures *and* also allowed the selection of redundant representations. Heuristics about when to consider redundant representations and about other cost-tradeoff assumptions were carefully noted in the description of the system, but were not expressed as independent rules in the system implementation.

Several different search strategies have been tested. Low and Rovner use hill climbing among the estimated costs of the target programs to choose an implementation. Morgenstern uses a dynamic programming algorithm specifically tailored to choose structures for large files. Wegbreit [7] Rives some examples of the use of performance analysis To drive a program transformation process. LIBRA represents its resource-management strategy in rules. One of the rules, which suggests consideration of the high potential impact decisions first, is similar to the techniques used by Wegbreit and Morgenstern.

Several other approaches to the problem of data structure selection have been taken. The SETL project [8] uses a more traditional optimizing compiler approach to choose set representations based on a small set of alternatives. The svstems described in [9] and [10] attempt to match modelling structures with the users needs. An unsolved problem in this approach is how to combine several modelling structures into one representation.

5. A FRAMEWORK FOR EFFICIENCY ESTIMATION

LIBRA was designed to explore the feasibility of combining analytic and knowledge-based approaches to efficiency estimation. The basic idea in the framework is heuristic search through e tree of partially implemented program descriptions. Efficiency rules from LIBRA are used to control the search and to add efficiency-analysis information to the program description. Coding rules from Barstow's knowledge base are used to refine the program description into a more concrete description.

The root node of the search tree is the initial program specification and the leaf nodes *are* target language programs. Each of the intermediate nodes is a partially implemented version of the entire program. The order in which refinements are considered affects the subtree that is constructed. The focus of attention for refinement may be limited to a particular part of the program, but comparisons between nodes *are* based on global execution costs. The free of partial program implementations, each with an agenda of synthesis tasks, serves as a workspace for recording the state of the search (see Figure 2 below).

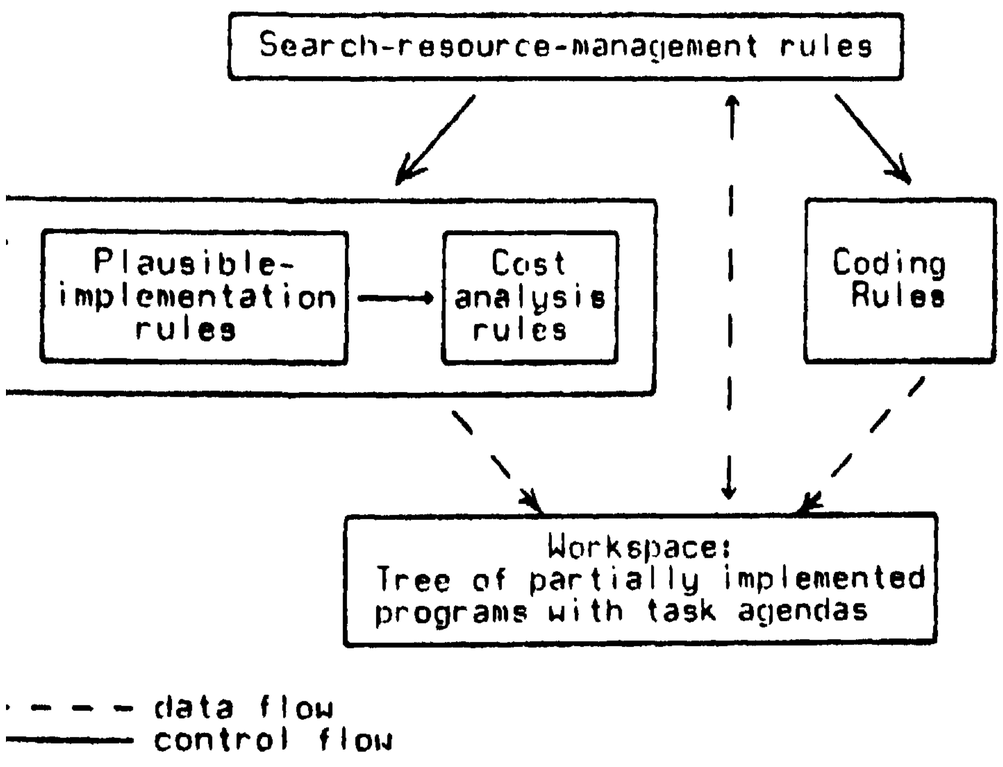


Figure 2. Overview of efficiency framework.

A somewhat simplified description of the search strategy is: pick a program implementation to work on, pick a refinement task within that implementation, pick • coding rule to achieve that task, and finally apply the coding rule and any associated efficiency rules.

Search-resource-management rules choose a program implementation and then a part of that program to work on. These rules assign priorities to tasks to ensure that the tasks *are* carried out within the limits of the resources.

When refining a part of a program, all relevant coding rules are retrieved and tested for applicability. *Plausible-implementation rules* are used to help decide which coding rule to apply. These rules contain precomputed analyses and are used to restrict the possible coding rules to those that seem reasonable in the given program situation, thus pruning the search tree.

Sometimes several coding rules seem plausible. Separate program descriptions are set up and refined, then compared using the cost estimates determined by *cost-analysis rules.* Search-resource-management and plausible-implementation rules may call on the cost-analysis rules for symbolic execution cost estimates to compare different implementations and identify potential bottlenecks in the target program execution.

5 1 Assigning priorities to decisions

Since all implementations cannot be considered in equal detail, the quality of the decisions depends on the order in which they are considered and the depth to which the consequences are explored before making a commitment. The search-resource-management rules use scheduling and resource allocation tc balance the final program performance with the cost of choosing and constructing the implementations.

Task-ordering rules determine the ordering for attempting different refinement tasks. Ordering principles include expanding complex programming constructs, such as "SUBSET" early to expose choices, and postponing choices of refinement rules *and* low level coding details until the *major* decisions have been made.

Choice-ordering rules find an order for considering the decisions that must be made. One of these rules suggests allocating the most resources to the decisions that are likely to lead to bottlenecks and making those decisions first. Section 5.3 describes how these *high potential impact* decisions *are* identified. LIBRA makes an adjustment to the potential impact of a decision to reflect the accuracy of cost estimates for the current level of program development and the expected cost of completing the refinement process. Without this, a highly refined implementation might be abandoned in favor of a very abstract description with a slightly better optimistic estimate that is probably not achievable.

5.2 Applying plausible-implementation rules

The plausible-implementation rules in LIBRA describe the situations under which data structure implementations are appropriate, when different sorting operations are plausible, and when to consider using more than one representation for a data structure. This knowledge is used to compare implementations without the expense of explicit construction and evaluation of execution costs of all alternatives.

The plausible-implementation rules are structured condition-action rules. The condition of a rule about data structures, for example, states all the critical uses of a data structure that make the rule relevant. Efficiency information such as the size of a data structure and the number of executions of a statement may be used in the rule condition. The rule action can set a Boolean combination of constraints for a set of program parts requiring that they be refined (or not refined) to a particular programming construct. A three valued logic (satisfied, impossible, possible) is used to check constraints.

5.3 Estimating execution costs

LIBRA includes a knowledge base of rules for estimating the execution cost of a program description at *any* stage of the refinement process and with varying degrees of accuracy. The user is expected to provide some basic information about the program, and then LIBRA Keeps the analysis updated for the rest of the refinement process. For example, i*n* the NEWS *program,* the basic information needed is the expected number of stories, the average number of Keywords per story, and the number of times the main loop in the program will be executed for a given database. LIBRA then makes analysis transformations in parallel with refinements so that more accurate cost estimates can be associated with succeeding nodes in the tree. Some analysis rules are associated with particular coding transformations. Many rules, such as those for analyzing Boolean combinations, are associated with coding constructs rather than transformations, information about parameters such as data structure sizes, statement running times and execution frequencies, and data structure usage information is maintained.

The top-down, incremental analysis allows programs to be analyzed that would be difficult to analyze automatically if only the target program were presented. An advantage of combining the stepwise refinement with this sort of analysis is that classes of implementations can be compared by considering the cost estimates for intermediate program descriptions rather than explicitly expanding the tree and comparing the target language programs.

Estimating execution costs is not an exact science. LIBRA attacks the problem by using both upper and lower bounds on the execution cost. The upper bound, or *achievable* estimate, is calculated by introducing a *standard implementation* for each of the programming constructs used and by assuming that standard Implementation choices are made for the rest of the refinement process. The lower bound, or *optimistic* cost estimate is based on a lower bound for implementations Known to the program, not a theoretical lower bound. Global optimistic cost estimates *are* estimated by assuming optimistic costs for each of the constructs in the *program* and by assuming that no representation conflicts occur.

The importance of a decision is *measured* by its *potential impact.* This is achievable bound cost estimate and the execution cost estimated when optimistic cost estimates are used for all parts of the program involved in the decision.

A general model of program constructs and specific models for each construct are used to organize the cost estimation process. Also, a standard cost-compulation process allows sharing of subroutines between estimation strategies for making quicK estimates and for performing more detailed (and usually more expensive) analysis.

6. AN EXAMPLE

This section will consider the implementation of a retrieval program in more detail. The problem to be implemented, called NEWS, is:

> Read in a database of news stories. The DATABASE is a mapping from stories to sets of KEYWORDS. Repeatedly accept a keyword and prints out a list of the names of the stories in the database that contain that keyword. When the special command "xyzzy" is given instead of a keyword, then halt.

LIBRA has directed the implementation of several versions of NEWS. Under different assumptions about the size of the database or the cost function to be used, different implementations are selected. Figure 3 below shows the tree of implementations that is generated and searched under certain assumptions about data structure sizes and branch probabilities. The major choices to be made in implementing NEWS are choosing representations for the DATABASE mapping *and* for the KEYWORDS set.

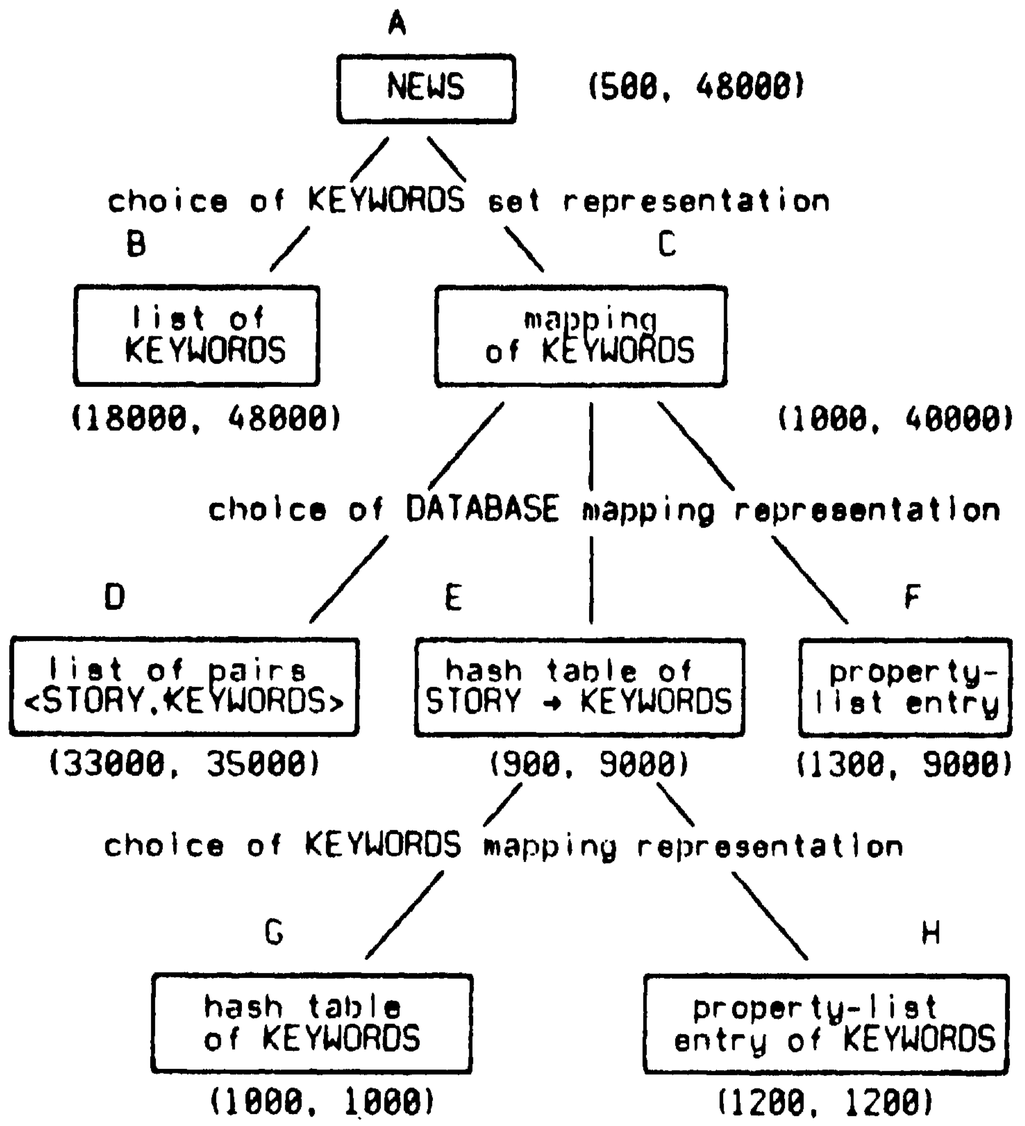


Figure 3. Overview of NEWS implementation.

6.1 Alternate implementation paths

A'number of ways to implement NEWS are possible with the current set of coding rules. One refinement path, node G in the search tree of Figure 3, is followed through in more detail in the following sections. It involves representing DATABASE internally as a hash table of stories, with each story in turn having a hasn table of keywords. The cost function used in this case is the product of running time and number of pages in use. LIBRA chooses a hash-table representation for KEYWORDS because there are many Keywords for each story. The time to convert the set of keywords into a hash table is balanced by the time savings from the membership test, which is faster as a hash-table look-up than as a search through the list of keywords (for large keyword sets). The DATABASE representation decision is similar. Both choices are reinforced by the fact that the main loop is executed many times before exiting with "xyzzy."

Under other assumptions, a path through node B is taken and a linked-list representation is selected. If the loop is executed only a few times or if the number of keywords associated with a story is small, then the time required to convert the database from the list of pairs (<story, keywords>) representation to a hash-table representation is not outweighed by the fast hash-table look-up operations. If space is a critical factor in the cost function, another path through B is taken in which the original representation of a list of pairs is preserved. This avoids using any additional space, but at a cost in time.

A different tree than the one pictured in Figure 3 may also be searched. Suppose there are only a few keywords per story, many stories, and a cost function dominated by running time. Then the representation of the DATABASE mapping is a more critical decision than the KEYWORDS set representation, because the lime for the membership test would not differ greatly for the different representations. If fewer resources are available for synthesis than in the examples described above, then some of the less reliable plausible-implementation rules are used. For example, nodes F a*nd* H are not considered when a plausible-implementation rule that prefers hash-table representations to property-list entries is applied.

The implementations that LIBRA chooses in this case are about the best possible with the current set of coding rules. People can do better on the NEWS example by using representations outside the scope of the coding rules. However, for any given sot of coding rules, allowing people to make the decisions would not produce better implementations.

6.2 Initial refinements in NEWS

The following sections show more details of the path leading to node G. By questioning the user, LIBRA determines that the expected number of stories in the database is 80, the average *number* of keywords *per* story is 100, the expected number of iterations of the loop is 300, and the probability that the command is a keyword of the average story is .01.

LIBRA first calls on the coding rules to make refinements that do not involve *any* decisions. For example, the input DATABASE is refined to the standard input format for mappings, a lift of pairs <story, keywords, and the set of KEYWORDS is refined into a linked list. LIBRA applies plausible-implementation rules to decide whether to consider multiple representations for the KEYWORDS set and the DATABASE mapping.

During refinement, a "for-all statement enumerating the domain of DATABASE is created. It is refined into an explicit enumeration of the items of domain, since only one coding rule is applicable. To decide how to refine the enumeration, more information about the representation of the domain is needed. LIBRA does not consider all possible representation of the domain set explicitly; the choice is made by the application of plausible-implementation rules. For example, two of the efficiency rules •bout sets are:

> If the only uses of a set A are for enumerations over that set, *and* if B is another representation for A that is easily enumerable, then use the same representation for A as for B.
>
> If all ur.es of a set are for enumerations, or as pointers to positions in set, or as tests of the state of the enumerations, and if the target language is LISP, then refine the set into a linked list.

These rules determine that domain set, which is used only for enumeration and is not an alternate representation of some other set, should be refined into a linked list, a linked-list should be used. Therefore constraints on the domain set *are* established, and it is refined into a sequence, *and* then into a list (rather than an array) with the choices between applicable coding rules resolved by the constraints.

Some of the details of constructing the domain list *and* the enumeration of the domain are postponed by search-resource-management rules because LIBRA predicts that no decisions will be involved and the cost estimate for that part of the program will not change significantly. Other choices that arise and cannot be resolved by plausible-implementation rules are also postponed until other useful refinements are finished.

6.3 Identifying the most important decision

All of the changes above take place in node A of Figure 3. During this refinement, several choices are postponed. These choices are 1) how to refine the DATABASE mapping used inside the for-all, and 2) how to refine the KEYWORDS set within that mapping. What is the effect of each of the two choices to be *made* in this example?

The *internal* representation of DATABASE, (DB1), is used for retrieving the *map value (keyword* sets) of stories once *per* story per command. Possible implementations for mappings range from a linked-list format that make retrieval linear in the number of stories to associative structures that have nearly constant retrieval time.

The keyword sets in DB1 (KEYWORDS 1), are used in a "member(command, KEYWORDS)" test. This test is executed one for each story for each iteration of the loop. Possible implementations give membership tests with times ranging from linear in the number of keywords to nearly constant.

Since the number of keywords is greater than the number of stories, the keyword representation has the largest cost differential and »s more likely to be a bottleneck in the final program if care is not taken in the representation choice. According to the choice-ordering rule about making high potential impact decisions first, the next step is to look at tne possible refinements of KEYW0RDS1.

Decision-making resources *are* assigned. Currently the resources measured are The CPU time used in carrying out the refinements and the number of nodes used in the refinement trees. The resources needed to complete a program implementation without making choices are estimated and subtracted from the total available resources. Decision-making resources from the remainder *are* assigned in proportion to the estimated importance of the decision. Then, separate program descriptions are set up (actually they share some substructure) in which each of the alternate coding rules *are* applied. In this decision, the applicable rules allow either refining the keyword set into an explicit set, leading to search node B, or into an explicit mapping, leading to search node C.

6.4 Exploring two implementations for KEYWORDS!

LIBRA'S goal Is to refine the alternatives (B and C) enough so that the comparison among implementations will be informative. The resources previously assigned give upper limits on the time *and* space to be spent on getting a more accurate estimate of the program cost of the implementation being explored. Each program description also has a "purpose" to be fulfilled, which serves as a test of whether the task has been achieved and is used to set some of the task and choice-orderine strategies. There is also a set of program parts that is to be the focus of attention of processing. In this case, the KEYWORDS 1 data structure and the representation conversion *and* the membership test are included in the focus set.

In the first program description, search node B, the explicit-set rule is applied and refinement proceeds until all relevant tasks *arc* satisfied -- the resources allowed for writing the program *are* generous in this example. At the conclusion, the keyword set for each story has been refined, after the application of several coding rules, into a LISP list, and the membership operation has been refined into a list search.

Refinement of search node C, the *program* description in which the explicit-mapping rule was applied, also halts because all relevant tasks have been accomplished. Here the keyword set is refined to a mapping and membership tested by seeing if there is mapping for the given key. There is also a representation conversion since the keyword set is represented as a list in the input.

LIBRA then computes optimistic and achievable bounds on the cost of the whole program for each program description. In the linked-list implementation, B, the optimistic estimate is 18000 millisecond-pages, and the achievable bound is 48000. The optimistic and achievable cost estimates for the *mapping* representation, C, are 1000 and 40000 respectively. Branch and bound is applied to eliminate any implementations with optimistic estimates worse than the achievable estimate of some other implementation. Neither implementation is eliminated in this case, though later in the refinement of NEWS this technique will be fruitful. Node C has the best optimistic estimate and is chosen for further refinement.

6.5 Refining (he rest of NEWS

The remaining decisions *are* choosing a refinement for the explicit mapping of KEYWORDS! and choosing a refinement for DB1. The database decision is chosen by the potential impact method. Three program descriptions are set up to consider the three applicable refinement rules -- one to consider refining the mapping to a list of pairs (search node D), one to consider a stored mapping (nodo E), and one to consider a distributed mapping (node F). The relevant parts of the program, those related to the DB1 decision, are then refined in each program description. For example, the stored mapping is refined to a hash table. The resulting program descriptions *are* then compared

with each other end with Other program descriptions that have been temporarily abandoned, such as the search node B. As Figure 3 shows, nodes B *and* D can be eliminated from further consideration because even their lower bounds ere worse than achievable bound on node E. The most promising implementation, search node E, is then chosen and refinement continues.

The final decision to be made is how to represent the KEYWORDS! set, which has been refined into • mapping. As in the refinement of node C, there *are* three applicable coding rules. However, there is an applicable plausible-implementation rule about mappings that eliminates one of the possibilities.

> If a mapping has *a\ready* been refined from a set, then do not refine it into a set of pairs.

Thus, only two coding rules *are* considered. These rules are both tested, in search nodes G and H. The stored mapping, leading to the hash table representation in node G proves to be the best choice. At this point, the cost estimate is precise enough to eliminate all the other possibilities. Thus, the best possibility is the implementation of both the keyword set and the mapping DB1 as hash tables. As refinement continues, several other choices of coding rules are presented, but they are all resolved by plausible-implementation rules. The decisions made include choosing to recompute rather than store values that *are* easy to compute. The program description is finally refined into a LISP program.

7. KNOWLEDGE ACQUISITION AIDS

LIBRA includes mechanisms to assist in the acquisition of new programming constructs, including the additions that are made to efficiency Knowledge when new coding Knowledge is added. When new high-level constructs are added, such as new types of sorts, or trees, new efficiency Knowledge is needed to analyze these constructs, their subparts, running times, and other efficiency properties. LIBRA'S prototypes of programming constructs are consulted by acquisition-aid routines when new constructs are added. Some of the necessary information can be deduced automatically, and the user is asked specific questions to obtain the rest.

Estimates of *running* time and space usage depend on the target language and target computer. LIBRA provides a semi-automatic procedure for deriving cost estimation functions from the set of functions for the target language constructs. This procedure can be used in to update efficiency rules when new coding rules are added. Currently only times estimating functions are derived, but a similar process could be used to check the accuracy of the plausible-implementation rules in the system when new coding Knowledge is added.

8. CONCLUSIONS ANO FUTURE DIRECTIONS

The use of efficiency estimation in program synthesis is a new but promising field. The issue of data-structure selection has been studied in some detail, but not the issue of estimating the effects of applying high level program transformations. LIBRA provides a framework in which both data-structure and algorithm selection can be treated. The heuristics that suggest orderings for considering refinement tasKs and decisions and that suggest plausible implementations and when to consider multiple implementations are expressed explicitly as rules. A start has been made on symbolic algorithm analysis, and incremental analysis is used to make the analysis process tractable. One of the goals in LIBRA is to breaK up the programming process into manageable chunks in order to learn more about the sequences of implementation choices available, how the choices interact, and when and how the choices should be made.

To extend LIBRA to complete automatic programming system, additional research would be needed. For example, to write more complex programs such as compilers or operating systems, more coding *and* efficiency rules about constructs such as bit-packing, machine interrupts, and multiprocessing would *need* to be added to the system. However, the efficiency techniques described here should be sufficient to control combinatorial explosion.

Higher level optimizations, extended symbolic analysis and comparison capabilities, and more domain expertise are some feasible extensions to LIBRA. Another possibility is to automate the checking of conditions in the heuristic rules by doing a complete search through the current set of coding rules. Automatic generation of heuristics based on analysis of symbolic cost estimates would be another important addition. Adding *an* inference process to both the coding and efficiency estimation process would also be useful, though not as straightforward.

More powerful symbolic comparison techniques are also possible. For example, the range of values for which one implementation dominates another ($c1.N^2$ over $c2.N$) could be determined. The user would then only have to say whether N was within a particular range, rather than giving a definite value. Another use of symbolic costs is in proposing alternate solutions, each with the conditions that make that solution the best choice. If, for example, the cost for primitive operations such as multiply are given as ranges, the system could produce the solution "implementation X is best if the target machine has a *Very* fast multiply, but implementation Y is best if multiplication takes about the same time as addition.**

LIBRA has demonstrated the feasibility of the approach described here, but has by no means exhausted the research topics in efficiency estimation for program synthesis.

REFERENCES

[1] Barstow, D. R. Knowledge-based *Program* Construction. Elsevier North-Holland, New York, 1979.

[2] Green, C. C. "The Design of the PSI *Program* Synthesis System.'* In *Proceedings of the Second International Conference on Software Engineering,* Computer Society, Institute of Electrical and Electronics Engineers, Inc., Long Beach, California, October 1976, 4-18.

[3] Kant, E. *Efficiency Considerations in Program Synthesis: A Knowledge-based Approach,* Forthcoming Ph.D. thesis, Stanford University, 1979.

[4] Low, J R. Automatic Coding: Choice of Data Structures. ISR 16, Birkhaeuser Verlag, Basel, Switzerland, 1976.

[5] Morgenstern, M *Automated Design and Optimization of Management Information System Software.* MIT Laboratory for Computer Science, Ph.D. thesis, September 1976.

[6] Rovner, P. D. *Automatic Representation Selection for Associative Data Structures.* Ph.D. thesis, Computer Science Department TRIO, The University of Rochester, Rochester, New York, September 1976.

[7] Wegbreit, B. "Goal-D«rected Program Transformation." In *Third ACM Symposium on Principles of Programming Languages,* January 1976.

[8] Schwartz, J. T. "Optimization of very High Level Languages." In *Computer Languages,* Vol. 1, *Permagon* Press, Northern Ireland, 1975, 161-194.

[9] Rosenschein, S., and Katz, S. "Selection of Representations for Data Structures." In *Proceedings of the Symposium on Artificial Intelligence and Programming Languages,* August,

[10] Rowe, L., *and* Tonge, F. M "Automating the Selection of Implementation Structures". *IEEE Transactions on Software Engineering,* Vol. SE-4, 6, November 1978.

A TECHNIQUE FOR MANAGING THE LEXICON IN A NATURAL LANGUAGE INTERFACE TO A CHANGING DATA BASE*

S. Jerrold Kaplan
Computer Science Department
Stanford University
Stanford, California 94305

Eric Mays
Aravind K. Joshi
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pennsylvania 19104

A difficulty in designing a Natural Language (NL) interface to a Data Base (DB) management system is insuring that DB updates do not obsolete the NL components. Of particular concern is the lexicon (the list of words that the system can process), mainly because NL queries can contain terms that appear as values in the DB, and hence are subject tc change as the contents of the DB changes. For example, to process the question "Does John Jones still work for the company?", it is necessary to identify the string "John Jones" as a (potential) value in the "EMPLOYEE-NAME" field (assuming a suitable DB). If such names must appear in the lexicon for the system to process the query, then the lexicon will go out of date as the company's personnel shifts. Using the DB itself as an extension of the lexicon is equally problematic: the system will be unable to parse and provide a negative answer to such a question if the name does not appear at all in the DB.

The approach suggested here is to infer a plausible field from the context of the query and semantic information about the domain that is implicitly encoded in the structure of the DB. This technique has been implemented in CO-OP, a NL DB query system that provides cooperative responses and operates with a typical CODASYL DB system. CO-OP treats the problem of selecting a plausible field as a special case of resolving semantic ambiguities. Examples drawn from the implementation are presented.

A. Introduction

When designing a Natural Language (NL) interface to a Data Base (DB) that is subject to updates (a "live" DB), it is useful to insure that changes in the contents of the DB do not require separate adjustments in the NL components. This eliminates the need for periodic intervention in the interface to keep it consistent with the DB. The property of remaining up to date on a live DB could be called transparency of DB update.

One component of NL DB query systems that is particularly prone to obsolescence is the lexicon - the vocabulary required by the system to parse and interpret the NL queries. The main reason for this is that the queries can contain terms that appear as values in the DB, and hence are subject to change as the contents of the DB shifts. Consider the question "Has William Smith checked out of room 423?"** To process this question, it is necessary to identify William Smith as a hotel guest (perhaps a potential value of the "GUEST-NAME" field of a record). If such names are explicitly coded in the lexicon, the query system will go rapidly out of date as the clientele of the hotel shifts. Clearly, providing lexical entries for DB values is a poor strategy: not only do DB updates require a corresponding change to the lexicon, but the information that William Smith is a guest is represented twice in the system, raising the possibility of anomalous states (considered as a whole, the system can be said to be improperly normalized).

B. The concept of DB image

An alternative strategy is to use the DB itself as an extension of the lexicon, as in the ROBOT system of (1). Thus, updates will automatically be reflected in the lexicon. A problem with this approach is that extensive searches of the DB may be required to locate the terms not explicitly present in the lexicon* (these will be called unknown terms). A more serious problem is that the values that appear in the DB are only a subset of the unknown terms that may be appropriately incorporated into a query. In the example above, if William Smith has already checked out and paid his bill, his name may no longer appear as a value anywhere in the DB, preventing the system from processing the query. In general, if the DB

The ROBOT system circumvents this difficulty by utilizing a fully inverted DB.

This work partially supported by NSF Grant MCS78-08401 and ARPA Contract MDA 903-77-C-0322.

** The examples given in the text are drawn from an imaginary DB about the operation of a hotel. Examples drawn from two real DBs are presented in the conclusion.

is used as an extension of the lexicon, the system will be incapable of responding negatively to queries containing unknown terms. The source of the difficulty is that unknown terms are words or phrases that could appear in the DB, not only words or phrases that do appear. Note that simply keeping a lexicon of "old" (deleted) terms in the DB does not solve this problem - it is possible to pose reasonable queries that contain terms that were never in the DB to begin with. The problem, then, is not to locate them in the DB, but rather to identify the field that could appropriately contain the unknown term, regardless of whether it actually occurs there or not. This field could be called the DB image of the unknown term.

C. Inferring the DB image of unknown terms

One technique for determing the DB image of an unknown term is to rely on the user to provide this information. Although effective, this strategy has several drawbacks. First, the user may not be sufficiently familiar with the logical structure of the DB to select the correct DB image. After all, one of the justifications for using a NL DB query system is that it frees a user from having to know precisely this type of information. Second, the correct DB image may seem obvious to the user, and being asked to provide an unnecessary clarification should be an annoyance ("Is 'William Smith' = GUEST-NAME?").

An alternative approach is to infer a plausible DB image from the context of the unknown term in the query and semantic information about the domain that is implicitly encoded in the structure of the DB (schema). This technique, described below, has been implemented with considerable success in CO-OP, a NL DB query system that provides cooperative responses to NL questions that request the retrieval of data, and operates with a typical CODASYL DB system. (2)

1. Treating unknown terms as semantically ambiguous

In CO-OP, selecting an appropriate DB image for unknown terms is reduced to the problem of selecting a sense (meaning) for each word in the input query. The various senses that a word can take on may be regarded as semantic ambiguities. Associated with each entry in the lexicon for terms that make reference is a set of definitions of the possible senses that term can take on in the domain of the DB. For example, a term like "orders" might have a list of senses designating room service orders, purchase orders, reception desk assignments, etc. Temporary lexical entries are created for unknown terms that designate as their potential senses each field in the DB that could reasonably occur as a DB image in a query. Thus, an unknown term like "William Smith" might be treated as a lexical item that could have the DB image of either an employee name, a supplier name, a guest name, etc. The word sense disambiguation routines then proceed to select a sense for unknown terms as they do for other terms, as detailed below.

2. Predictive value of selectional restrictions

A particular word sense is selected using two heuristics. The first exploits the predictive value of the words in the immediate syntactic context of the ambiguous term to constrain the set of potential senses. Specifically, verbs and prepositions are marked in the lexicon (when appropriate) to indicate the more likely subjects and objects they can take. If a subject or object has one or more senses that are consistent with these predictions, the alternative senses are eliminated from further consideration. In effect, simple selectional restrictions are used to reduce the semantic ambiguities.

For example, a word like "who" may refer to employees, suppliers, or guests; in the context of the query "Who works in the maintenance department?", the verb "work" may predict employees as a subject, and the possibility of "who" referring to suppliers or guests can be eliminated. Similarly, "William Smith" can be resolved to be a hotel guest, by the predictive value of the verb "checked out". Another potentially useful technique (not implemented) would be to take advantage of verb case roles, or semantic memory structures to help predict categories for the unknown term. As none of these richer knowledge structures were present in CO-OP, no such approach was adopted.

3. Schema distance and semantic relatedness

The above heuristic will not always result in an unambiguous selection. Indeed, a verb like "is" may have no predictive value at all. Remaining ambiguities are resolved by choosing the sense whose DB image results in the shortest distance through the DB schema to the DB image of the surrounding referring terms in the query*. This distance metric is a rough measure of semantic relatedness - semantically related terms tend to have DB images that appear close to each other in the DB schema. Thus "Who is the maintenance department?" may still select employees as the sense of 'who' because the DB image of employees may be closer in the schema to departments than suppliers (perhaps linked through inventory) or guests (perhaps linked through orders or complaints). Similarly, the unknown term "maintenance" (assuming it appears solely as a DB value, if at all) can be correctly identified as a

A network like structure (such as CODASYL; is required for the operation of this heuristic.

department name, through the close association in the schema of the "DEPT-NAME" field with the DB image of the more specific term "department".

These heuristics, while admittedly rather crude, tend to operate acceptably when the DB schema accurately models the structure of the real world. When CO-OP makes an incorrect interpretation, its mistake is made clear to the user via a paraphrase of the query (3), presented tc the user for approval. Experience indicates that it is easy to correct these misinterpretations by rephrasing the query in more specific terms, for example, "Which employees are in the maintenance department?"

D. Examples

Some examples of interpretations of unknown terms actually produced by CO-OP are reprodeced below "Q: " (below stands for query, "P:" for paraphrase (responses are deleted in the interest of brevity), commentary is in square brackets([]). The first examples are taken from a DB at the National Center for Atmospheric Research in Boulder, Colorado. The DB contains information on their users, divisions, projects, and computer resource utilization.

Q: Which projects in oceanography does NASA Headauarters sponsor?

P: (I am assuming that 'NASA Headquarters' is a SPONSOR NAME.) (I am assuming that 'oceanography' is an AREA OF INTEREST.) Which projects does NASA Headquarters sponsor? Look for projects that are in oceanography.

[This query contained the unknown terms 'oceanography' and 'NASA Headquarters'. The system has correctly identified the first because the preposition 'in' predicts a division, superdivision, or area-of-interest as an object in this domain, and area-of-interest is the nearest to projects in the DB schema. 'NASA Headquarters' was identified as a SPONSOR NAME because it is the subject of the verb 'sponsor'."]

Q: Which projects are advised by Thomas Harris?

P: (I am assuming that 'Thomas Harris' is an ADVISOR NAME.) Which projects are advised by Thomas Harris?

[Co-OP is capable of recognizing that 'Thomas Harris' is the logical subject of advise, and hence is probably an advisor, although the passive form is maintained throughout the balance of the processing for other purposes..]

[The following examples are drawn from the ONRODA DB, a DB about military equipment and logistics created as part of the Operational Decision Aids Program of the Office of Naval Research at the University of Pensylvania* .]

C; What is the fire rate of the M-61?

P: (I am assuming that 'M-61' is a GUN NAME.) Display the fire rate of the M-61.

[Here, 'M-61' has been identified as a GUN NAME because it is the nearest field in the DB schema to the 'FIRE RATE' field that could reasonably contain an unknown term. The semantic relatedness (schema distance) heuristic is the only one invoked in this case.]

Q: Which F-5s carry strut curve radar?

P: (I am assuming that 'F-5s' is a SHIP NAME.) (I am assuming that 'strut curve' is a RADAR NAME.) Which F-5s carry strut curve radar?

[Because of insufficient contextual cues, CO-OP <u>incorrectly</u> identifies 'F-5s' as a SHIP NAME (it is actually a type of aircraft) - radar can appear on ships, subs, or aircraft in the DB, and the selection defaults to ships. By contrast, 'strut curve' is correctly identified as a type of radar, because it is parsed as modifying 'radar'. The proper interpretation can be induced by re-phrasing the question in a more specific form, as follows.]

Q: Which F-5 aircraft carry strut curve radar?

<u>References</u>

(1) Harris, L.R., Natural Language Data Base Query Using the Data Base Itself as the Definition of World Knowledge and as an Extension of the Dictionary, Technical Report TR 77-2 Mathematics Dept., Dartmouth College, Hanover, N.H., 1977.

(2) Kaplan, S. Jerrold, Cooperative Responses from a Portable Natural Language Data Base Query System, Ph.D. dissertation, Department of Computer and Information Science, University of Pennsylvania, Phila., Pa., 1979.

(3) McKeown, K., "Paraphrasing Using Given and New Information in a Question-Answering System", Association for Computational Linguistics Meeting, San Diego, CA., August 1979.

Contract N00014-75-C-0440 with the assistance of CTEC, Inc. of McLean, VA.

JAPANESE WORD PROCESSOR

Tsutomu Kawada

Information Systems Lab.
Research and Development Center
Toshiba Corporation
1 Komukai Toshiba-cho, Saiwai-ku
Kawasaki,210,Japan

Shin-ya Amano

Information Systems Lab.
Research and Development Center
Toshiba Corporation
1 Komukai Toshiba-cho, Saiwai-ku
Kawasaki, 210, Japan

Japanese word processor(JWP)presents a typewriter which can easily handle Japanese documents with newly developped and quite different way from conventional Japanese typewriters. Japanese documents consist of more than two thousand letters (Kanjis:Chinese characters, Kanas: Japanese alphabet, alphanumerics, and so on). The conventional Japanese typewriter is equipped with all these letters. It means that typewriting is very difficult and typing speed is low. JWP overcame these difficulties with Kana-to-Kanji translation technology.

1. OVERVIEW

Japanese is a very unique language in its variety of letters. It has several kinds of letters: Hira-Kana (Japanese alphabet; あ, い, う, ---), Kata-Kana (Japanese alphabet used for foreign words; ア, イ, ウ, ---), Kanji (Chinese character; 山, 川, 水, ---), and alphanumerics (a,b,c,---,0,1,2,---).
Kanas consist of about fifty letters. Popular Kanjis amount to about two thousand. These letters appear in ordinary Japanese documents such as newspapers, business letters, and so on. The abundant letters invoke difficulties in typing Japanese documents. The conventional Japanese typewriter has to be equipped with all letters, and typists must look for letters one by one. Thus "letter-searching" is an intrinsic problem in the Japanese typewriting.
In order to overcome this problem, Kana-to-Kanji translation thechnology is applied in the Japanese Word Processor (JWP). A Kana sentence is just a sequence of phonetic symbols. So the Kana sentences are easy to type because of the small number of keys, but they are difficult to understand.
The new techniques to translate Kana sentences into ordinary Japanese sentences are discussed henceforth.

2. KANA-TO-KANJI TRANSLATION

Grammatical analysis is applied to get correct translation by JWP. Kana sentences are translated into Kanji sentences phrase by phrase.

The definition of the phrase is given as follows:

Phrase-(prefix)substantive word(suffix)
(functional word*)

Substantive word«noun/pronoun/verb/adjective/
adverb/conjunction/interjection

Functional word=particle/auxiliary verb

Here parentheses indicate optionality, the asterisk indicates one or more occurences, and the slant indicates alternatives.
Generally speaking, nouns are accompanied with a particle which indicates cases. Verbs are accompanied with conjugating auxiliary verbs, which indicate tenses, moods, and voices, without any separation between them (Fig.I).

みなければならなかった
minakerebanaranakatta
(had to see)

Each underlined part denotes a verb, auxiliary verbs, and a particle.

Fig.I An example of Japanese verb phrase

The grammatical analysis means analyzing the phrase grammatically. There are two main difficulties here, conjugation and concatenation of verbs and auxiliary verbs, and appearance of homonyms.

2.1 GRAMMATICAL ANALYSIS

Japanese sentences consist of a series of the phrases. The role of the Japanese phrases in sentences is approximately similar to English one (Fig.2), but structure of the phrases is different as shown by the above definition.

私は　学校へ　行く.　I go to school.
p1　p2　p3　p1 p3　p2
(watasiwa gakkoue iku.)

Here p1 is a noun phrase, p2 is a prepositional phrase, and p3 is a verb phrase.

Fig.2 An example of the role of phrases

The analysis of verb phrase will be cited below since it is the most intricate. Japanese verbs do not express tenses, moods,and voices by their conjugation. If it is necessary to express them, auxiliary verbs which denote them respectively are concatenated to the verb. The verbs are classified into five conjugational groups and there are 25 auxiliary verbs. They have six or less conjugational forms. These numerous conjugational forms of all verbs and auxiliary verbs make the verb phrases very complex.

The conjugational forms indicate the condition of the concatenations of verb and auxiliary verbs in the phrase. That is, they indicate what kind of word is concatenated to the verb or auxiliary verb. We made a connection table for all verbs and auxiliary verbs to check the concatenation. Figure 3 showes examples of the structure of the verb phrases.

走る　(hasiru)　run
走った　(hasitta)　ran
走りたい　(hasiritai)　want to run

Fig.3 Examples of the structure of the verb phrase

In the figure 3, each underlined part indicates a verb and an auxiliary verb.

2.1.1 Dictionary consultation

The first step of the grammatical analysis is to consult a dictionary for finding the substantive words. All possible candidates are extracted from the dictionary. We take the phrase "UUO'.it^" (hasiranaijdo not run) as an example to explain the grammatical analysis. Candidates for "はしらない" are shown by figure 4. Only stems are stored for verbs to reduce the number of items in the dictionary. The dictionary has the following contents for a word; Kanas, Kanjis, grammatical informations such as part of speech, conjugational type,etc., and frequency of use.

KANJI	KANA	PART OF SPEECH	(MEANING)
柱	はしら(hasira)	noun	(column)
橋	はし (hasi)	noun	(bridge)
箸	はし (hasi)	noun	(chopsticks)
走	はし (hasi)	stem of verb	(run)
葉	は (ha)	noun	(leaf)
歯	は (ha)	noun	(tooth)

Fig.4 Candidates for "はしらない"

2.1.2 Analysis of conjugation and concatenation

The conjugational part of verb "走"(hasi) must be ら(ra),り(ri),る(ru),れ(re),or ろ(ro) according to the grammatical information of the dictionary. As the third Kana of はしらない is (ra), the verb "走" satisfies the conjugational condition. Then it must be checked that the conjugating form of " 走ら "(hasira) can be accompanied with the string "ない"(nai). It is known by the connection table that the auxiliary verb "ない" can be attached to " 走ら ". "走らない"(hasiranai) satisfies the condition of the concatenation. Therefore "走らない" is one of possible translations. Nouns are accompanied with specific particles. These particles are stored in a table. The attached Kana-string to "柱"(はしら;hasira) is "ない" (nai). But this string is not found in the table, accordingly "柱ない "does not satisfy the condition of the concatenation as the phrase. By the same procedure, all the nouns of the candidates are also rejected. Thus only one translation"走らない" is got.

2.2 Homonyms

2.2.1 Reduction of homonyms by grammatical analysis

There are a lot of homonyms in Japanese. It results from the fact that plural Chinese characters have the identical pronunciation expressed by Kanas. Chinese character itself is a letter and not a word in Japanese. The words are composed of one or more Chinese

characters. The words have homonyms, too. That is, plural words have the identical Kanas. JWP reduces the homonyms by the grammatical analysis. For example, the phrase "きょうりょくな" (kyouryokuna; powerful) is translated uniquely into "強力な" by analyzing it grammatically. If the grammatical analysis is omitted and only consultation of the dictionary is applied two homonyms occurred: "強力な" and "協力な". Here "協力な" is meaningless, since "協力" is a noun meaning "cooperation" and "な"(na) is a conjugating part of the adjective with which noun can not be accompanied. The meaningless translations like this often happen without the grammatical analysis.
In case of the simplest translation technique, a table of Kanjis and their corresponding Kanas is used (Fig. 5).

あ (a) ;阿, 亜, 唖, ---
あい (ai) ;愛, 相, 哀, ---
あか (aka) ;赤, 垢, 朱, ---
-
-
-

Fig.5 Part of table of Kana-Kanji pairs

There are a lot of homonyms to a phonetic unit which is expressed by Kanas. For example, a phonetic unit "きょう"(kyou) corresponds to 137 Chinese characters; "協", "強", "京", and so on. "りょく" has six; "力", "緑", "物", and so on. Moreover "きょうりょく"has other possibilities of combination of phonetic units except "kyou-ryoku";that is, "kyo-u-ryo-ku", "kyo-uryo-ku", and so forth. If including these, homonyms increase enormously and many meaningless translations occur.

2.2.2 Reduction of semantical homonyms by learning

JWP has two kinds of memories to avoid semantical homonyms: a short term memory and a long term memory. The former is valid during typing one document and is reset when other documents are typed. When homonyms appear on a CRT display of JWP and one of them is selected by the typist, the word is memorized by JWP. When the same homonyms occur , the memorized word is indicated on the CRT display as the most possible candidate word in the context of the text. The latter is implemented as an administrator of the frequency of word use: JWP administrates its dictionary regarding to the frequency. When one of homonyms is selected as mentioned above, the frequency of the word increases by one. Accordingly the dictionary is becoming suitable to a certain field such as science, economics, literature, etc., since JWP refers the frequency during translation and the most frequent word is given priviledge to appear on the CRT display first. This method is effective for the semantical homonyms depending fields.

3. CONCLUSION

The automatic Kana-to-Kanji translation made the touch method possible in Japanese typing and released the typists from the burden of searching letters among the huge number of letters on the keyboard.
JWP occupies a remarkable status against the conventional Japanese typewriter and other related machines for its ability of automatic translation. Several new techniques applied here are quite effective to realize high performance translation. Figure 6 showes an example of translation which is a Japanese sentence of the abstract of this paper.

にほんごワードプロセッサはこれまでのわぷんタイプライタとはよったくことなるあたらしくかいはつされたほうほうによりにほんごのぶんしょをよういにあつかえるタイプライタをていきょうします。

--- Typed Kana sentence ---

日本語ワードプロセッサはこれまでの和文タイプライタとは全く異なる新しく開発された方法により日本語の文書を容易にあつかえるタイプライタを提供します。

--- Translated sentence ---

Fig.6 An example of Kana-to-Kanji translation

4. ACKNOWLEDGEMENT

We would like to thank Dr. Hiroshi Genchi and Dr. Ken-ichi Mori, our supervisors at Toshiba R&D Center for their encouragement and support in the course of the work reported here.
We also wish to acknowledge Kimihito Takeda for his cooperation to us in developping JWP.

AN UNDERSTANDING SYSTEM OF NATURAL LANGUAGE AND PICTORIAL PATTERN IN THE WORLD OF WEATHER REPORT

Eiji KAWAGUCHI, Masao YOKOTA, Tsutomu ENDO and Tuneo TAMATI

Faculty of Engineering, Kyushu University
Hakozaki 6-10-1, Higashi-ku, Fukuoka, 812 Japan

An Information understanding System Of BAsic weather Report(ISOBAR) is introduced first. It can adapt to both linguistic and pictorial input, and produce each output. The central part of this system is semantic processing, but ISOBAR's world is limited to weather report of Japan and Far East Asian area in terms of Japanese and corresponding weather charts. According to experimental results, the performance of the system proved to be satisfactory, except the computing time of picture processing procedures. Finally, the problems for the future research are pointed out.

1. Introduction

The present paper is based on the experimental research for understanding system of natural language and pictorial pattern. The prototype system is called ISOBAR(Information understanding System Of BAsic weather Report). The model was designed and developed after the following phylosophy:

1. In order to be able to understand natural language for human and machine, they need to have some correspondence to the knowledge or experience of external wor!d(in our case visual world),
2. And also in order to make them understand pictorial patterns, it is necessary to extract and recognize some characteristic features of pattern, corresponding to human concepts which reflect to the structure and meaning of natural language man spontaneously developed in his long history,
3. Through these interactive mechanism, man can explain the pictorial pattern by using natural language and vice versa.

The ISOBAR's world is limited to only Japanese sentences of weather report and weather chart of Japan and Far East Asian areas. But from the view point of semantics, the fundamentals of processing algorithm or mechanism of this limited world is applicable to more general world.

ISOBAR has two operating modes in principle. One is accumulation of meteorological database both from linguistic and pictorial inputs. The other is its retrieval. In the accumulation mode, man presents weather report sentences and charts to the machine. The machine interprets and understands the input information semantically by the aid of general knowledge about meteorology. It transforms the input into suitable data format and adds to the database if and only if the input is meaningful and conveying new information. If not, the machine either inquires about ambiguous points, or rejects the total input.

The semantics in ISOBAR in the final version is designed to have such hierarchy as in Fig.l. GKS is the symbol for General Knowledge about Semantics, and GKM for General Knowledge about Meteorology. SWM and KMP are Semantics in the World of Meteorology and Know Ledge about Meteorological Phenomena respectively. Generally speaking, it is difficult to distinguish GKM from GKS strictly.

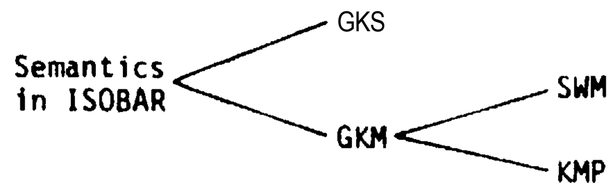


Fig.l Hierarchy of semantics in ISOBAR

The present ISOBAR is not completed as a total system with full semantic hierarchy as in Fig.l. SWM was already implemented in the present system because it is the core of the semantics in the system, and fortunately it has rather simple structure. In the meantime, GKS must cover all field of our linguistic activities in

This work was financially supported in part by Grant~in-Aid for Special Project Research of the Ministry of Education, Science and Culture, 1978. Project Number 310233.

Japanese. So in our case, GKS is limited only to general linguistic knowledge which is essential to the field of weather report. Present ISOBAR is partly containing GKS in the form of "Event of Constituent" which will be introduced in the next chapter. KMP is regarded as the total contents of an "Encyclopedia of Meteorology", but this part is limited and is not explicitly separated from SWM in the present system.

2. Outline of ISOBAR

First, the weather report sentences and weather charts for ISOBAR are outlined. Then some important semantic notions are presented here.

2.1 Sentences of weather report

We limited the words and sentence patterns for ISOBAR only to those of weather reports in Japan which are broadcasted every day from NHK on its second radio program. We termed them basic weather reports. NHK weather reports consist of four parts:

Part 1 General weather conditions
Part 2 Local weather conditions
Part 3 Reports from vessels
Part 4 Information for fishermen

Among these, we omitted the sentences in Part 1 from our consideration. All of these materials were collected from real NHK program between June, 1976 and May, 1977, and these were stored in the database of ISOBAR. Examples of sentences from Part 2 to Part 4 are shown in Fig.2.

2.2 Weather charts

Weather charts for ISOBAR are illustrated in Fig.3. The meaning of each pictorial component will be found in Table 1.

2.3 Notions in the semantic structure of weather report[1],(2],[3)

We introduce three basic semantic notions to describe the semantic contents of the world of weather report. The first is "Constituent" of meteorological world in ISOBAR. The second, "Attribute" of Constituent, and the last, "Event" in Attribute-space. "Attribute-space" in this context means an abstract space whose bases are all Attributes essential to that Event.

After careful analysis *of* weather report sentences and weather charts, we postulated i7 Constituents (denoted by $C_1, C_2, \cdots, C_{17}$) in all. These are tabulated in Table 2. Attributes ol Constituents are extracted to be 14 different kinds ($A_1, A_2, \cdots, A_{10}$) as in Table 3. Each Attribute is, in other words, a semantic phase or point of view of Constituents. Relation between Constituents and Attributes makes "Cross-Relation Table" of Constituents and Attributes. "Semantic Table" is defined to be such Cross-Relation Table that is filled with each "Attribute Value". A set of weather reports at a special data and time, or a sheet of weather chart, makes complete Semantic Tables, if all input sentences or pictorial components are meaningful, and all are recognized correctly. Table 4 shows an example of Semantic Tables.

"Event" in an Attribute-space is introduced as a semantic contents of "verb", "adjective", "adverb", etc.; those are all representing the conditions of Constituents or relation among Constituents. We visualize each event as a

Part 2)
O-SAKA DEWA NNE NO KAZE , FU-RYOKU 3 TEN-KI WA HARE DE , KIATSU 998MB , KION- 23°C . (OOSAKA area, NNE wind, wind force 3, fair, pressure 998MB and temperature 23°C.)

Part 3)
KAN-TO-TO-HO- NO N35°E146° DEWA W NO KAZE , FU-RYOKU 4 , TEN-KI WA HARE DE , KIATSU 05MB . (At N35°E146° east of the KANTO district, west wind, wind force 4, fair and pressure 1005MB.)

Part 4)
(a) KAN-TO-HIGASHIKAIJO- NO N36°E149° NIWA , 1004MB NO TEIKIATSU GA ATTE , E NI 75KM DE SUSUN-DEIMASU . (A low pressure of 1004MB at N36°E149° east of the KANTO district, proceeds east at 75KM/hr.)
(b) NIHON-HUKIN- O TO-RU 1020MB NO TO-ATSUSEN- WA , N60°E140° , N56°E136° , ··· , N19°E113° NO KAKUTEN- O TO-TTEIMASU . (An isobar of 1020MB in and around Japan passes through N60°E140°, N56°E136°, ··· and N19°E113°.)

Fig.2 Examples of weather report sentences

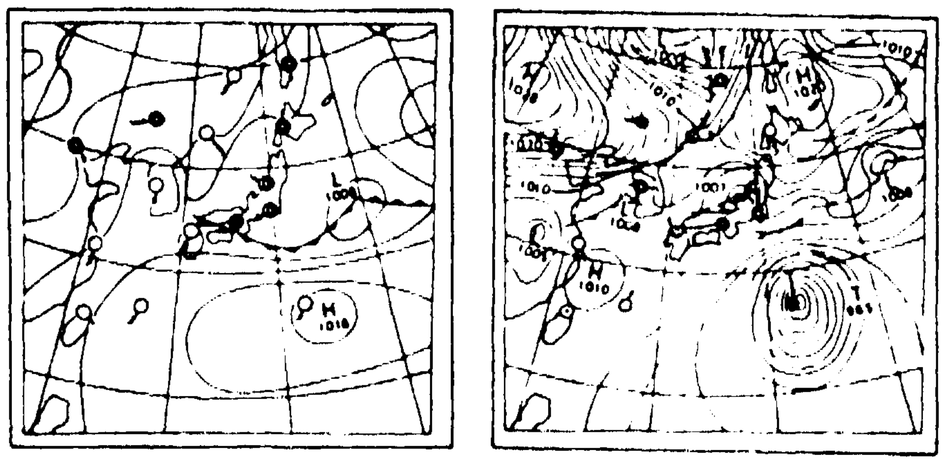


Fig.3 Examples of weather charts

Table 1 Picture components in weather chart

Component	Pictorial pattern	Informantion from the component
Weather symbol	⦶ ◎ ●	Location, variety of weather
Wind symbol		Location, direction, force
High pressure	H	Location, central pressure
Low pressure	L	
Tropical depression	TD	
Typhoon	T	Number, location, central pressure
Front line		Location, variety of front
Isobar		Locus, pressure value
Coast line		No information (constant components)
Latitude, longitude		
Numeral	01 ···· 9	Typhoon number, pressure value

locus of some Constituent in an Attribute-space. There are many patterns of Events in the meteorological world in ISOBAR. Some are temporal patterns, others are spatial patterns. In the linguistic analysis of weather report, we need an explicit expression of each event pattern. ISOBAR has all lists of variations of event patterns, which appear in the world of weather report. These event patterns work as both GKS and SWM in the present system. Semantic Tables, in the meantime, exclusively work as SWM.

Table 2 List of Constituents

Symbol	Constituents	Linguistic Expression	Pictorial expression
C_1	Low pressure	TEIKIATSU	L
C_2	High pressure	KO-KIATSU	H
C_3	Typhoon	TAIFU-	T
C_4	Tropical depression	NETSUTAISEITEIKIATSU	TD
C_5	Warm front	ON-DAN-ZEN-SEN-	
C_6	Cold front	KAN-REIZEN-SEN-	
C_7	Stationary front	TEITAIZEN-SEN-	
C_8	Occluded front	HEISOKUZEN-SEN-	
C_9	Isobar	TO-ATSUSEN-	
C_{10}	Ridge	KIATSUNOMINE	No direct correspondence
C_{11}	Wind	KAZE	
C_{12}	Fog	KIRI	
C_{13}	Surge	NAMI	
C_{14}	Storm	BO-FU-U	
C_{15}	Local weather	KAKUCHINOTEN-KI	
C_{16}	Weather at sea	KAIJONOTEN-KI	No direct correspondence
C_{17}	Time of observation	KAN-SOKUJIKAN-	

Table 3 List of Attributes

Symbol	Attribute	Symbol	Attribute
A_1	Location	A_8	Temperature
A_2	Pressure	A_9	Wind Direction
A_3	Number of Typhoon	A_{10}	Time
A_4	Direction	A_{11}	Degree of Influence
A_5	Area	A_{12}	Velocity of Motion
A_6	Wind Force	A_{13}	Wind Speed
A_7	Weather	A_{14}	Visibility

Table 4 Examples of Semantic Table

Constituent	Time	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}
$C_1(x)$	t_0	β	γ		α										
$C_1(y)$	t_0	p			α_1									β_1	
	t_1	q			α_1									β_1	
$C_6(z)$	t_0	α_1													
	t_0	α_2													
	t_0	α_3													

N.B. All above are the semantic structures of the following sentences (i.e. S1-S3) which appear in the weather report at t_0.
S1:KO-KAI(α) NO N34°E122°(β) NIWA 1000MB(γ) NO TEIKIATSU($C_1(x)$) GA ARIMASU.
S2:TEIKIATSU($C_1(y)$) GA MAIJI 20KM(α_1) DE HIGASHI(β_1) SUSUN-DEIMASU.
S3:KAN-REIZEN-SEN-($C_6(z)$) GA N34°E107°(α_1) , N35°E115°(α_2) , N37°E122°(α_3) O TO-TTEIMASU.
The tense of S2 is "Progressive present tense", which we denote by the pair of times, i.e. t_0(the present) and t_1(some immediate future).

2.4 General performance of the system

The total configuration of ISOBAR is illustrated in Fig.4. It consists of seven processing units and three databases. Linguistic input, i.e., weather report sentences in the accumulation mode, and questions in the retrieval mode, are sent to STA(Syntactic Analyzer) unit first. If the input is syntactically correct, it is stored in the LDB(Linguistic DataBase) only in case of weather report sentences, and at the same time, it is transferred to SSG(Semantic Structure Generator) unit. SSG generates the semantic structure of the input sentence. Then SAS(Semantic Analyzer and Synthesizer) analyzes the structure. The output from SAS makes a Semantic Table which is to be stored in SDB(Semantic DataBase) in the accumulation mode, or to be hold in SAS itself in the retrieval mode.

On the other hand, pictorial input, i.e., the weather chart, both in the accumulation and retrieval mode, are given to PPP(Pictorial Preprocessor) first. Pictorial processing consists of digitization of analog image, noise elimination, data compression, etc.. In the accumulation mode, all original data and their second data are stored in PDB(Pictorial DataBase) in a compressed form according to a special coding scheme. Output from PPP is sent to PR(Picture Recognizer) unit in every mode, where each component in the chart is recognized. Then its output is fed to SAS unit. SG(Sentence Generator) and CG(Chart Generator) work only in the retrieval mode. That is, linguistic answer to the question is synthesized by SAS using SDB and generated by SG, while pictorial answer is synthesized by SAS using SDB and PDB,

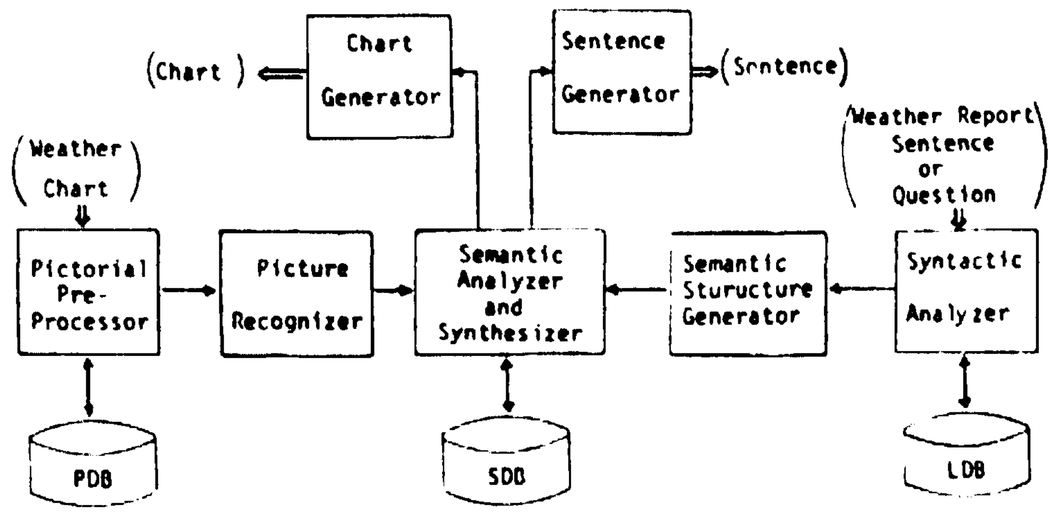


Fig.4 Configuration of ISOBAR

then generated by CG. More details about these units and databases will be presented in Chapter 3 and 4.

Fig.5 shows the hardware system of ISOBAR. The main system has special I/O devices, all of which are peripherals of either T-40(TOSBAC-40C) mini-computer or 0-4300(OKITAC-4300b) multi-CPU mini-computer system.

3. Sentence understanding

The outline of the processings in ISOBAR for linguistic input is presented in this Chapter.

3.1 Restriction for input sentences

Original weather report sentences have mostly regular sentence patterns. In ISOBAR, we admit any sentence which satisfies the following restrictions.

[Restriction 1] The vocaburary for input sentence is about 200 words, most of which are adopted from original weather report sentences, and some specific words are supplemented for interrogative and imperative sentences, e.g., "ITSU(when)", "DOKO(where)", etc..

[Restriction 2] The forms of input sentences are either (1)imperative form like, "•••GATA NO TEN-KIZU 0 SYUTSURY0KUSEY0(Output all weather chart of ••• type.)", or (2)such forms that are arbitrarily transformed from original kernel sentences in weather report by the optional rules R1-R5.

- RI: Alternation of words in kernel sentences.
- R2: Addition to such a word-group as "1976 NEN- 9 GATSU 11 NICHI 12 JI (WA)", which denotes a special data and time.
- R3: Substitution of interrogatives to words or word groups of kernel or transformed sentences.
- R4: Abbreviation of words.
- R5: Addition of interrogative particle "KA".

[Restriction 3] Input Japanese sentences are segmented word by word with spaces.

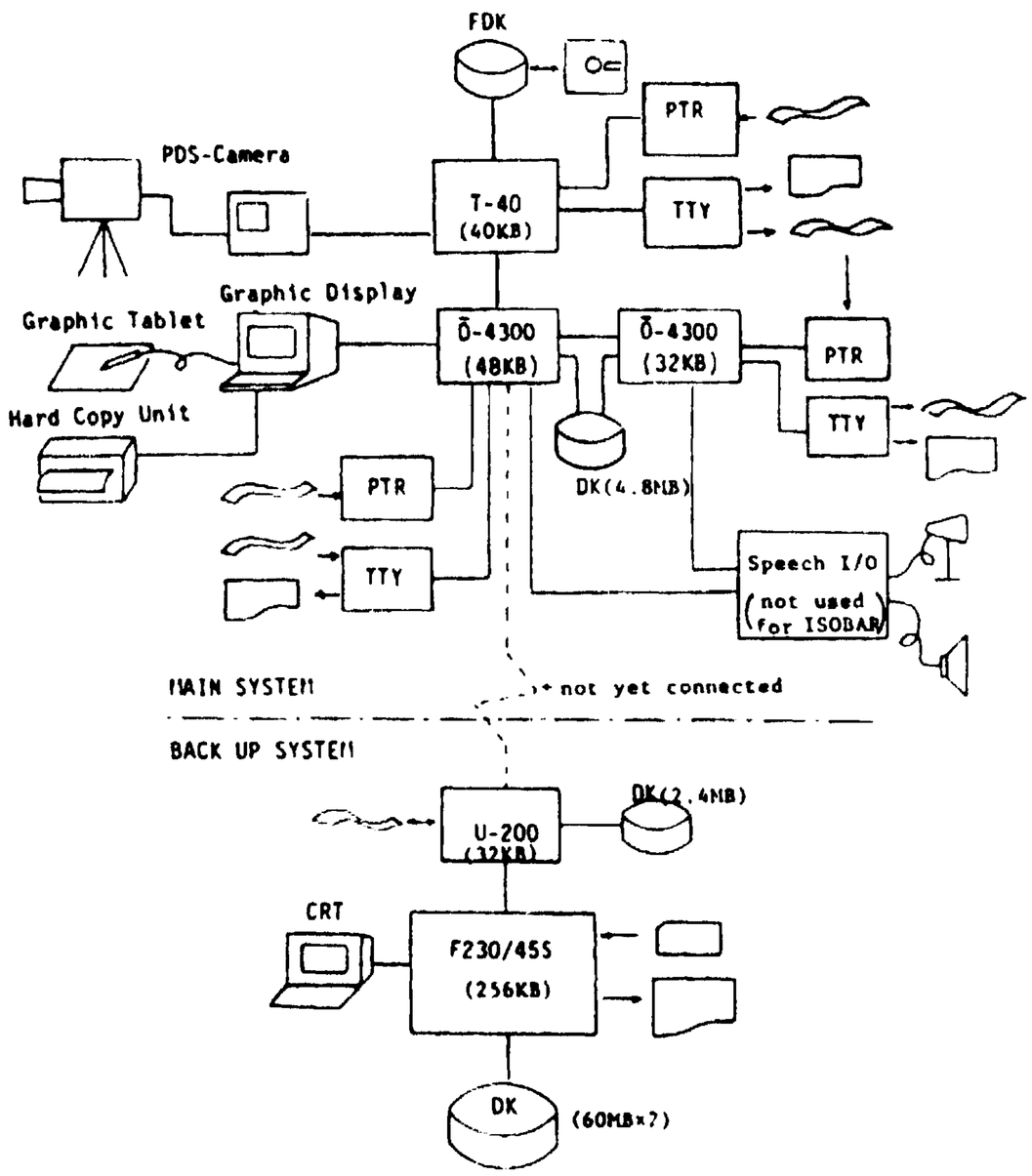


Fig.5 Illustration of the hardware system

3.2 Syntactic analysis, semantic structure generation and semantic analysis

The functions of STA and SSG in ISOBAR are as follows.

Input sentences are transformed in STA from surface structures into dependency structures by steps (I)-(4) below, and then into semantic-structures in SSG by step (5).

- Step(I) Parsing into individual words.
- Step(2) Forming word groups.
- Step(3) Construction of inter-word dependencies within each word-groups.
- Step(4) Construction of dependencies among word-groups.
- Step(5) Transforming dependencies into semantic structured.e., Semantic Table) by referring the Dictionary of Conceptual Connection(DCC).

At the end of step(4), each input sentence is identified with one of the template patterns, where verbs are essential keys. DCC as above is the most important dictionary that indicates which word-groups are representing "Constituents" or "Attribute Values", and in which slots in the Table they are to be stored.

In the course of above steps, syntactically anomalous sentences are rejected in STA. Several types of anomaly are detected in SSG, and in this case no semantic structures are generated. The semantic structures just generated are analyzed by SAS, and semantic contradictions are detected which can not be interpreted or understood in the world of weather report. This is realized as follows. SAS examines every slot of the Semantic Table whether stored Constituents and Attribute Values are correct. If not, the semantic structure is contradictory and rejected.

3.3 Access to SDB and output generation

The response of the system to a legal input sentence is determined according to kinds of input sentences. It is as follows. Cased) is the only case that opens the accumulation mode of ISOBAR, while Case(2)-(4) open the retrieval

mode.

Case(I) Declarative sentence

Every declarative sentence is assumed to be asserting some facts about weather. SAS compares the generated semantic structure of the input with every structure that was already stored in SDB, where comparison is executed on every Attribute. If the input is conveying new information to SDB, it is stored there, otherwise SG comments that it is already stored or contradictory.

Case(2) YES-NO question

Case(3) WH question

Case(4) Imperative sentences

If an imperative input sentence demands ail charts of a specific type of pressure configuration, the system interprets the input as "When did the pressure configuration of....type occur?". Examples are shown in Appendix A.

4. Picture processing

In this chapter we describe the operation of PPP and PR in ISOBAR.

4.1 Pictorial pre-processor(PPP)

An analog image is converted into binary-valued picture with l,024xl,02A pixels by PDS-camera. The noises caused by quantizing operation, are eliminated by simple masks. The pre-processed pictorial data arc sent to PR. At the same time, they are stored in PDB. In this case, we took a strategy to reduce the amount of storage. In our working system, a picture is represented by a series of black or white square regions of various sizes. The square size, its location, and the gray level(either black or white) are encoded in the structural information of "0", "1" and "(" series. We call this expression DF-expression[41,[5]. DF-expressions of original charts, together with their second data, are stored in PDB. The second data include such items as complexity of picture, spectrum of primitives, headings of weather patterns, etc.. These are used for chart searching operation in the retrieval mode.

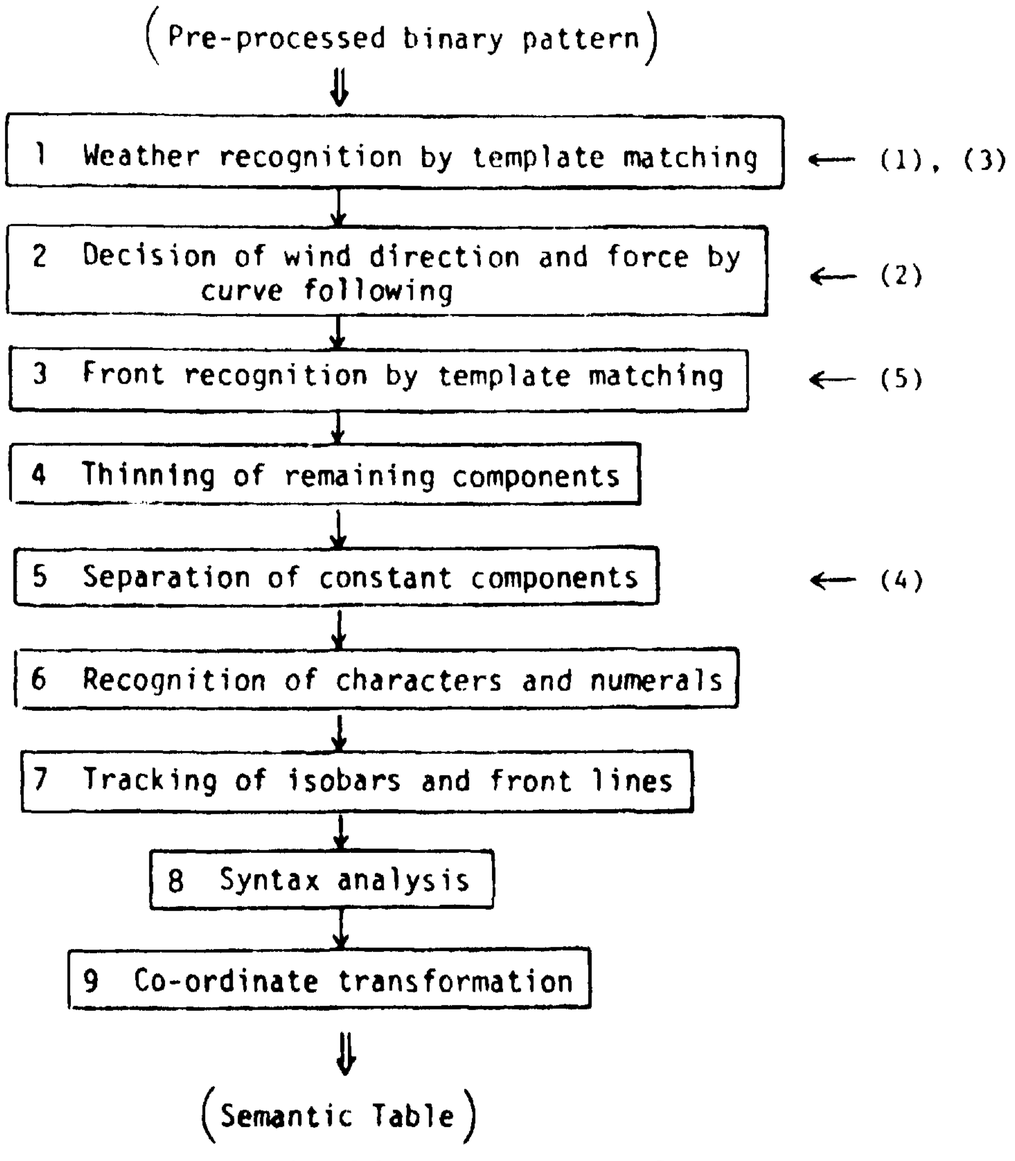


Fig.6 Flow diagram of PR

4.2 Picture recognition(PR)

The major function of this unit is to extract and recognize each picture component as shown in Table 1. We have some a priori knowledges about weather chart in PR. They are as follows.

(1) Every weather symbol is enclosed with a circle of a constant size. The variation of symbol is limited only to its interior pattern.

(2) Other components are not connected to the circle except wind symbol.

(3) There are 15 weather symbols in a chart. The location of respective symbol is fixed.

(4) The location of coast lines, latitudes and longitudes are fixed, but they are partly erased or interrupted by other components.

(5) Only front symbols and a few kinds of weather symbols have areas. The others are line drawings.

(6) Numerals are located near the characters or isobars. They specify the number of Typhoon or pressure values.

(7) The shape and size of numerals and characters are normalized.

The flow diagram of PR is shown in Fig.6, where each number within parentheses corresponds to above a priori knowledge which is used there. The final output from PR makes the Semantic Tables. Thus, all information from a weather chart is recognized as the semantic content of the world of weather report at the given data and time. We illustrate the results of step(I) -step(4) in Appendix B. These were experimented on a part of an entire picture covering Western Japan. The picture size is 256*256

5. Discussion

According to our experience, the performance of ISOBAR in the accumulation mode for linguistic input was very satisfactory. Also in the retrieval mode for linguistic input, the system worked successfully. In other words, we can say present restrictions for input sentences are rather severe. So do the vocaburary and sentence patterns. We should remove such restrictions as much as possible in the next version of

1SOBAR.

On the other hand, its performance for pictorial input was not so much successful. There are no fundamental problens left for the performance of pictorial output, that is, the retrieval operation. But in the accumulation mode, or in recognizing operation of the system, some problems are left unsolved. For instance, computation time is the most untolerable problem. This is partly because of our small(in terms of the size of the main memory) and slow(cycle time) computer system, and partly because of large size of input pictures in spite of rather high compression. Another problems left are improvement of curve following algorithm. Present ISOBAR works in the man-machine interactive mannar, but we must find some good idea of automatic gap tracer. Any solution, if ever, of those and other unsolved problems seems to require a high performance machine.

6. Conclusions

The conclusions obtained from this research are as follows.

(1) The semantic model in ISOBAR was proved to be appropriate.

(2) The system performance for linguistic input was satisfactory.

(3) The present ability of ISOBAR for pictorial input is not necessarily sufficient, because of the machine size and some unsolved problems.

(A) In order to make ISOBAR more flexible and powerful, it is desired to remove restrictions for input sentences as much as possible.

(5) All picture processing procedure should be improved to be more time saving ones. But now, authors have already started such improvement.

ACKNOWLEDGEMENT

The authors are deeply indebted to Mr. Y.Ogawa, R.Taniguchi, K.Tsuzuki and K.Noda for their considerable assistance with the experimental work.

APPENDIX A Examples of Q-A

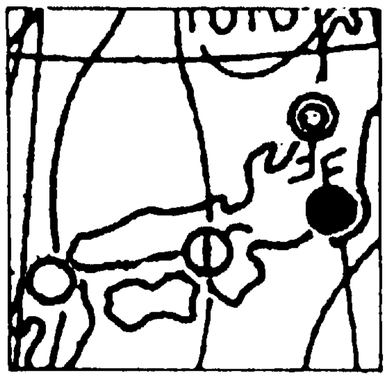
1978/12/30/12/ WA NIHON-KAIGAWA DEWA TEN-KI WA DAITAI HARE DESU KA. (Was the weather on the side of the Japan Sea almost fair at noon on Dec. 30, 1978?)
SPATTERN=7 QUEST=1 PLACE=12 WEATHER=2 DEGREE=2
HAI , NIHON-KAIGAWA DEWA TEN-KI WA DAITAI HARE DESU. (Yes, the weather on the side of the Japan Sea was almost fair.)

1977/1/12/12/ WA NAHA DEWA TEN-KI WA S KA. (In NAHA area at noon on Dec. 12, 1977, was the weather south?)
ZOKUSEI(TEN-KI) TO ZOKUSEICHI(S) TOGA HUTEKIGO-. (The weather has not the attribute of direction.)

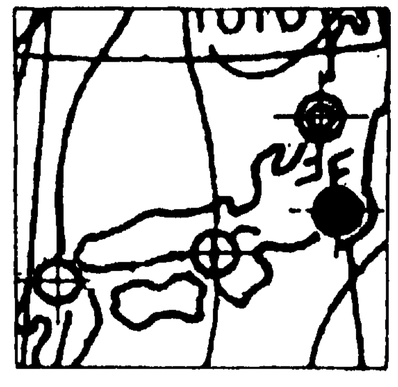
1976/9/12/12/ WA L NO CHU-SHIN- KARA ON-DAN-ZEN-SEN- GA DOKO NI NOBITEIMASU KA. (Where did a warm front extend from the center of the low pressure at noon on Sep. 12, 1976?)
SPATTERN=4 QUEST=1 TIME=1976/9/12/12/ ORG=L FNAME=5 LOC=#
L NO CHU-SHIN- KARA ON-DAN-ZEN-SEN- GA N26DE140D NI NOBITEIMASU. (A warm front extended to N26°E140° from the center of the low pressure.)

1TSU SEIKO-TO-TEI NO KIATSUHAICHI TO NATTEIMASU KA. (When did the pressure configuration of typical winter type occur?)
SPATTERN=6 QUEST=1 TIME= TYPE=1
1976/11/29/18/ WA NIH0N- HUKIN- WA SEIKO-TO-TEI NO KIATSUHAICHI TO NATTEIMASU. (At 6:00p.m. on Nov. 29, 1976, the pressure configuration of typical winter type occured around Japan.)

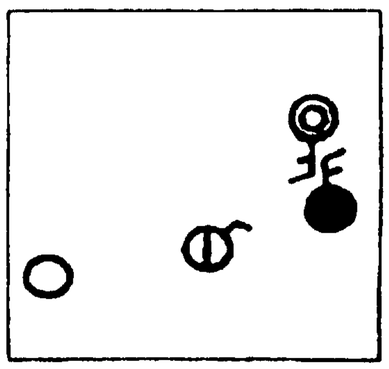
APPENDIX B Experimental result of PR



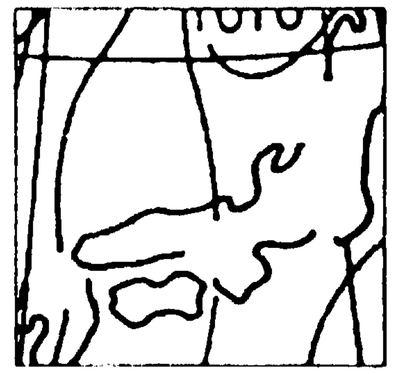
Original picture.



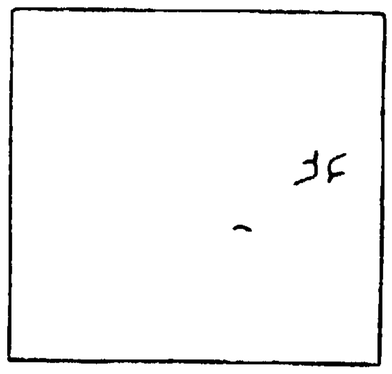
Cross shows exact position of weather circle.



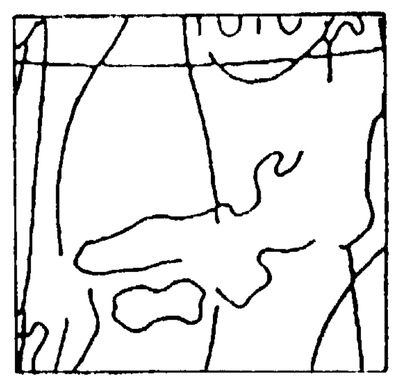
Extracted weather symbol.



Remaining part.



Result of thinning of wind symbol.



Result of thinning of remaining part.

REFERENCES

[1] YOKOTA,M. and TAMAT1,T.,"A Study on Formal Description of Word Meaning(I)",Technology Rep. of Kyushu Univ., Vol.48, No.6, pp.809-814, Dec. 1975.

[2] YOKOTA.M. and TAMATI,T./'Description of Word Meaning and Its Application", Tech. Rep. of 1ECE Japan, AL77-41, 1977.

[3] YOKOTA,M. and TAMATI,T.,''On Syntactic and Semantic Analysis of Weather Report Sentences", Tech. Rep. of IECE Japan, AL77-78, 1978.

[4] ENDO,T. and KAWAGUCHI,E.,"Some Properties of DF-Picture Expression and Its Application to Data Compression", Trans. IECE Japan, Vol.J62-D, No.2, Feb. 1979.

[5] KAWAGUCHI.E. and ENDO,T.,"On a Method of Binary Picture Representation and Its Application to Data Compression", IEEE Trans, on PAMI, to be published.

SHAPE FROM TEXTURE:
AN AGGREGATION TRANSFORM THAT MAPS
A CLASS OF TEXTURES INTO SURFACE ORIENTATION

John R. Kender
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, *P:* 15213

A new approach to obtaining shape information from textural information in static monocular Images la outlined. Also presented is a new aggregation transform which determines (under certain condItiohe) vanishing points and lines. A theorem is proved which relates this Hough-IlKe transform to the gradient space, showing that the transform also directly indicates local surface orlentatlona. Additionally, the transform is shown to have many properties that make it an appealling substitute for some other current image transforms. An example is given of its application to a synthetic textured Image.

1. INTRODUCTION

One central task of image understanding is the recovery of three-dimensional scene information from the two-dimensional perspective transformation that is the image. The recovery of the missing dimension can be achieved by the use of multiple views: either extensive in time, as in the determination of structure from motion [1], or extensive in space, as in deriving shape from binocular disparity [2, 3]. However, even a single image often contains powerful cues as to object definition and shape; for example, many properties of object surfaces can be derived from an understanding and exploitation of image intensities [4]. In restricting the input to a single Image, the task necessarily becomes a heuristic one, given the vast array of scenes that can generate identical images. But such heuristic approaches, especially to the degree that they are model-free, can provide basic theories and algorithms applicable to many Image tasks.

This paper begins with a very brief outline of one such relatively model-free approach to deriving shape information from a static monocular view. This method is based on the analysis of texture gradients and the application of principles of projective geometry. Just as an understanding of surface reflectivity enables shape to be derived from shading, so too can shape be derived from the textural properties of a scene.

This research was sponsored in part by an IBM Graduate Fellowship, and in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under contract F33615-78-C-1551.

The remainder of the paper is devoted to the presentation of a new image aggregation transform, in the style of the Hough transform. It is more efficient and "natural" than existing Hough-like transforms, and (under certain conditions) can determine the location of local or global vanishing points or lines. A theorem is proven which shows the intimate relation of such points and lines to the local surface gradient. It further demonstrates the natural correspondence that the transform establishes between perspectivity in the scene and the gradient space [5].

2. SHAPE-RELATED ASPECTS OF TEXTURE

"Texture" is an ill-defined term. However, in some respects, it can be considered an attribute of surfaces, like reflectance or color: its appearance is usually dependent on illumination and view angle. It is well known that blurred textures behave very much like gray scale tones. Similarly, texture gradients behave like intensity gradients.

But there are also important, exploitable differences. Intensities are usually identified one-to-one with picture elements ("pixels")} they have no shape. Under many conditions, the observed intensities of a surface's pixels are fully independent of their distance from the observer; the inverse square law of optics is cancelled by the square law of solid angle. Most discouraging of all, it is difficult to discriminate intensity differences due to illumination variations, reflectance differences, or changes in orientation.

In contrast are textures, especially those made up of identifiable texture elements ("texels"). Texel definition can be rather insensitive to illumination variation. Indeed, if individual texture elements have negligible

extent normal to the surface they define (the texture is "paint"), even shadows may not obliterate the fundamental textural pattern. Further, texture density is directly correlated with surface distance; the inverse square law holds exactly. In addition, given the necessarily larger area needed to define a texel, texels comprise a multi-dimensional family, compared to the one dimension which describes pixels. They are therefore potentially more discriminating at occlusions, and less sensitive to noise. Thus, intensity and texture are somewhat complementary; in fact, they often coexist within the same surface (e.g. the surface of a golf ball).

1.1 Physical Assumptions

At least three interrelated physical phenomena pertain to the derivation of shape information from textural Information. These phenomena in turn provide the necessary assumptions needed in recovering three-dimensional surface descriptions. First, there is surface textual homogeneity, which implies that like, nearby regions are really the same region. In the intensity domain this forms the justification for region-growing or -splitting approaches. The analogue for a textured object is based on the assumptions of local texel similarity. Looked at another way, region-growing and -splitting algorithms implicitly recover near-planar surfaces based on the similarity of very small texels: pixels.

Secondly, there is the phenomenon of surface uniqueness and smoothness. It implies the assumptions directly applied in shape-from-shading: the uniqueness of the local surface ("microplane") orientation with respect to an observer, *and* the continuity of the global surface. Smooth changes in surface orientation usually create textural (and intensity) gradients, but the opposite in not always true; heuristic rules are necessary to reverse the implication.

Lastly, there is the phenomenon of surface position. As position with respect to the observer varies, additional texture gradients arise due to perspective deformation. Such gradients have no direct counterpart in the intensity domain. However, they can be analyzed by using the implied assumption of viewpoint uniqueness. It is with respect to this one viewer position that surfaces have direction or distance.

Each of these three phenomena can be studied in isolation by carefully selecting images that minimize the effect of the other two. In the absence of curvature and perspective, texel similarity has been explored by, among others, [6], who segmented planar objects under orthographic projection. Large, simply-curved surfaces isolate the problem of determining global shape from local clues; processing would be analogous to the intensity-based work of [7]. Large, simply-textured planar surfaces (e.g. tilted checkerboards) single out the last aspect, perspectivity.

Perspectivity has received little attention. In fact, much research assumes orthographic projection and takes pains to compensate for, rather than utilize, perspective effects. As a consequence, model-free approaches have found it difficult to determine quantitative depth and orientation information: there is no fixed viewer position under orthographic projection. It is these issues that the remainder of this paper addresses, by exploiting the properties of a special, perspective-isolating domain.

3. A NEW AGGREGATION TRANSFORM

Suppose the task of determining shape from texture is simplified to the following very simple subtask. Texels are restricted to be one-dimensional and line-like; they are organized into textures in a regular mesh-like fashion. This is a "structural" texture, somewhat like a grid, except that the line segments can have arbitrarily long gaps, and they do not have to have a fixed spatial frequency. The scene is limited to large planes set at various distances and orientations. The planes are "painted", in the scene, with such a network of parallel lines; in the image, these lines are usually perspectively distorted. Although an abstraction, these conditions fairly well imitate many man-made surfaces: building faces, walls, floors, and so on.

The subtask has several exploitable properties. Surface planarity implies that any local surface orientation (relative to the observer) is the global one as well. Therefore, the determination of local vanishing lines is equivalent to finding global vanishing lines. The latter can be done once, in the large, with consequent improvement in accuracy. Segmentation is eased by the uniformities of the texture component directions. Individual texels are easily identified by an edge detector; no local region growing, etc., is necessary to define them.

The major problem is to aggregate the texels (in this case, edgels) into surfaces, mindful of the vanishing points. Most traditional textural transforms are of limited use here; most were developed with the implicit assumption that the image was the orthographic projection of a frontal two-dimensional scene. Using them, it would bo difficult to distinguish intrinsic textural variations from perspective-induced ones.

The new aggregation transform is initially designed, then, to group texels according to the two or more vanishing points they orient towards. (This is a *very* strong assumption. The occurrence of two or more vanishing points is a special case of the general problem of the vanishing line. Vanishing lines exist independently of texel orientation; they occur with

statistical textures, and even with no textures at all.)

Conceptually, it ought to be sufficient to infinitely extend each line segment, and detect points of mutual intersection. Texels can then be classified into implied surfaces by their respective vanishing points. In effect, this would implement the general image understanding heuristic that converging image lines often arise from parallel lines defining a surface within the scene.

Practically, the problem is a bit more complex. Many times vanishing points are *very* distant, if not infinite. Further, any solution should be computationally inexpensive, and finding all mutual intersections is not. Lastly, it would be beneficial if the aggregation operator grouped together like-oriented texels *in* an ensemble that can be easily defined and detected. It will be seen that this, and more, can be efficiently attained.

3.1 The Hough Transform

The vector version of the rho-theta Hough transform is a likely starting point for such a transform (see Figs. 1 and 2). Under it, image points are mapped into sine waves in the parameter space; edge vectors *are* mapped into points [8]. Colinear edge segments are represented (and recovered) by the accumulation of their transform values at unique points in the Hough space. It is easy to see that parallel line segments are indicated by accumulation points sharing the same thefa value. That is, parallel lines transform into a line of accumulation points in the parameter space; the line is theta - c. Any mutually converging line segments (possible parallel lines in perspective) transform into accumulation points lying on a sine curve. That sine curve is the transform of their vanishing point.

Unfortunately, sines in the Hough space are difficult to detect. It is likely one would need to apply a similar Hough-like transform to do so: mapping each potential sine point into a curve in a second parameter space, and detecting accumulation points there. Still, this aggregation has the advantage of implicitly representing and aggregating like-oriented edge segments by exactly one curve.

3.2 A First Modification

The following modification of the vectored rho-theta Hough preserves its local grouping property, but represents aggregates in a form that is easier to detect. In addition, It is computationally cheaper, conceptually and visually more forthright, and can be used to replace other Hough-like transforms used for vector grouping (for example, the transform used in the gradient intensity transform method ("GITM") of [9]).

This new approach plots the rho-theta transform space on polar coordinates (see Fig. 3). This has many desirable effects. Points now map into circles which

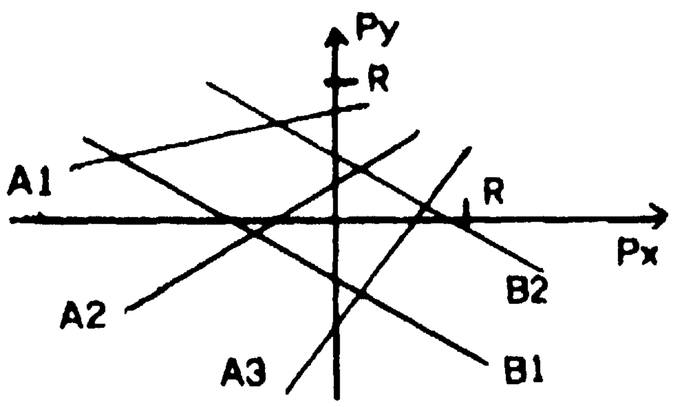


Fig. 1. An image containing parallel lines seen in perspective. Pixels are labelled (P_x, P_y).

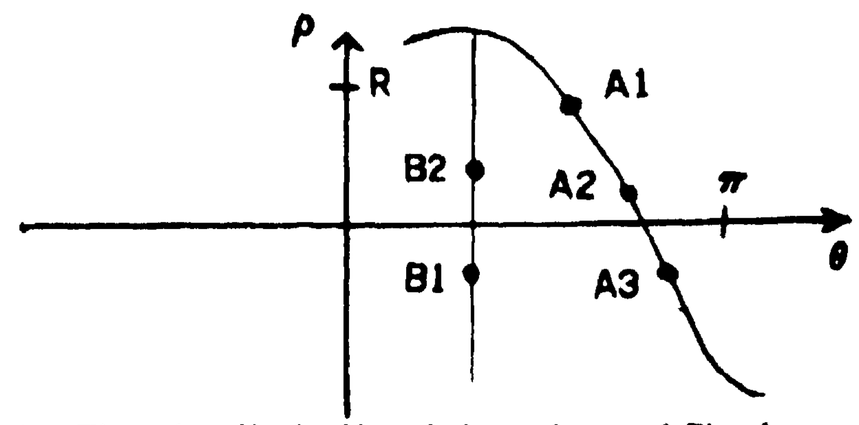


Fig. 2. The rho-theta Hough transform of Fig. 1.

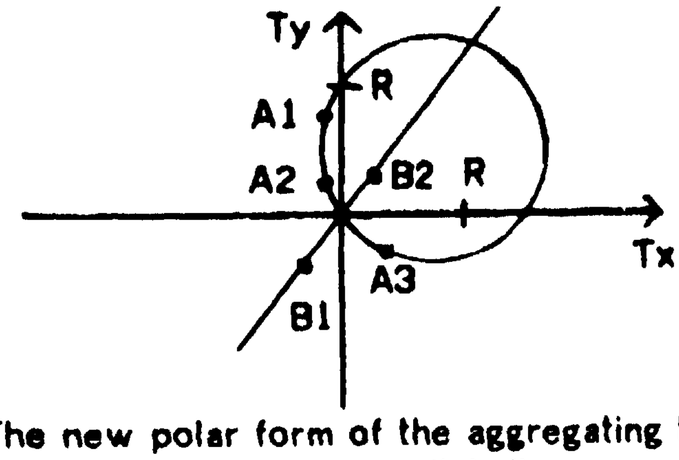


Fig. 3. The new polar form of the aggregating transform of Fig. 1. Both sets of parallel line segments are mapped into accumulation points that lie on circles.

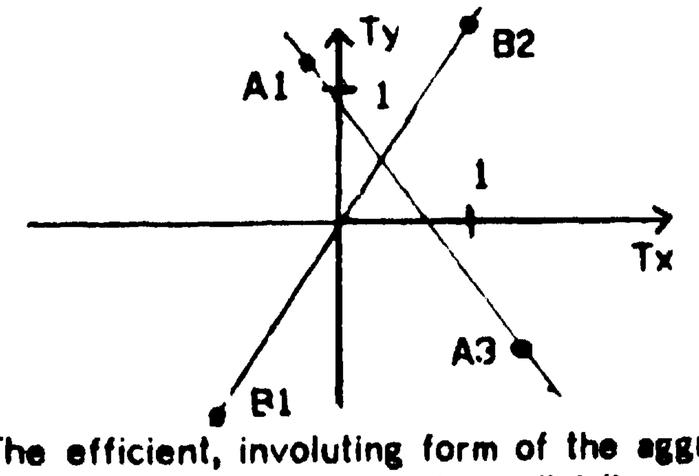


Fig. 4. The efficient, involuting form of the aggregating transform of Fig. 1. Both sets of parallel line segments e now mapped into lines.

pass through the origin. Edge vectors still map into points, but with two interesting new properties. Consider the transformed edge vector (a point) as a vector with respect to the transform origin. Now its direction is parallel to the direction of the edge vector itself. And, its length is equal to the distance (from the image origin) of the line determined by the edge itself. These two properties make the transform easier to view and imagine. As an elegant bonus, no trigonometry is required to calculate this new transform. If the edge vector is the vector $E = (E_x, E_y)$, and if its position in the image is considered the vector $P = (P_x, P_y)$, then the transformed point, represented as the vector $T = (T_x, T_y)$, is defined as:

$$T = ((E \cdot P) / ||E||^2) E$$

where "·" is the dot product and "|| ||" is the Euclidean norm.

Further, the transform maps mutually converging line segments Into accumulation points that lie on circles passing through the origin. The vanishing point is represented by that point on the circle farthest from the origin. In the case of parallel line segments (with infinite vanishing point), the accumulation points lie on a line through the origin perpendicular to the parallels. Local grouping is preserved; but now the aggregate representation (circles through the origin) is easier to detect, as shown below. Almost all of this discussion, Including the claims of computational efficiency, has been shown to apply analogously to the other uses of the vectored rho-theta Hough. As an example, in the GITM method, what once *mapped* into secant curves in the transform spaces now maps into straight lines.

3.3 The Second Form

Detecting the circular arcs can be done efficiently in the following *manner.* Consider a second transform that involutes the result of the first polar transform (see Fig. 4). That is, all transformed edge vectors, which are represented in the first polar space as (rho, theta) are *mapped* Into (K/rho, theta), for some K. This transform has even more desirable effects. Infinite vanishing points *are* now *mapped* into the origin (K/rho - 0). Lines through the first polar origin (along which lie the transforms of parallel lines) are unchanged. Most importantly, all circles passing through the first polar origin (along which lie the transforms of converging lines) are mapped into straight lines. Thus, in either case, a set of scenic parallel lines, perspectively deformed in the image or not, is mapped into accumulation points that lie along a straight line. The distance of this straight line from the newest origin is inversely proportional to the distance to the vanishing point of the line set. A normal to the straight line is parallel to the vanishing point direction. Another bonus: If this transformation is composed with the first, the combined operation is even cheaper than that of the first alone. The transform vector $T = (T_x, T_y)$ is then defined in terms of the edge vector $E = (E_x, E_y)$, the edge position $P = (P_x, P_y)$, and the scaling constant K as:

$$T = (K / (E \cdot P)) E$$

where "·" is again the dot product.

Lines are easy to detect, especially when compared to the original suggestion of searching for sines.

4. THE VANISHING LINE-GRADIENT SPACE RELATION

These lines, however, have yet another remarkable property. Under very natural conditions, they Intersect in the transform space at (p, q), the value of the surface gradient vector. That is, the transform simultaneously achieves three goals. It aggregates edgels into straight lines (represented by points in the transform space). It aggregates straight lines into sets of (perspectively distorted) parallel lines (represented by lines in the transform space). And it aggregates parallel line sets into planes (represented by the intersection of lines In the transform space at the unique point corresponding to the plane's orientation).

The following theorem formalizes the conditions under which the above claims are valid.

Theorem: Suppose an image contains the perspective projection of a planar surface defined by two or more coplanar sets of parallel line segments. Let the origin of the transform space correspond to the coordinates of the focal point in the image (that is, where the camera is "aimed"). Let R equal the focal distance. Then T - R E / (E ' P) (as defined above) transforms edges so that the Intersection of lines of accumulation points in the transform space is at (p, q), the gradient vector of the surface.

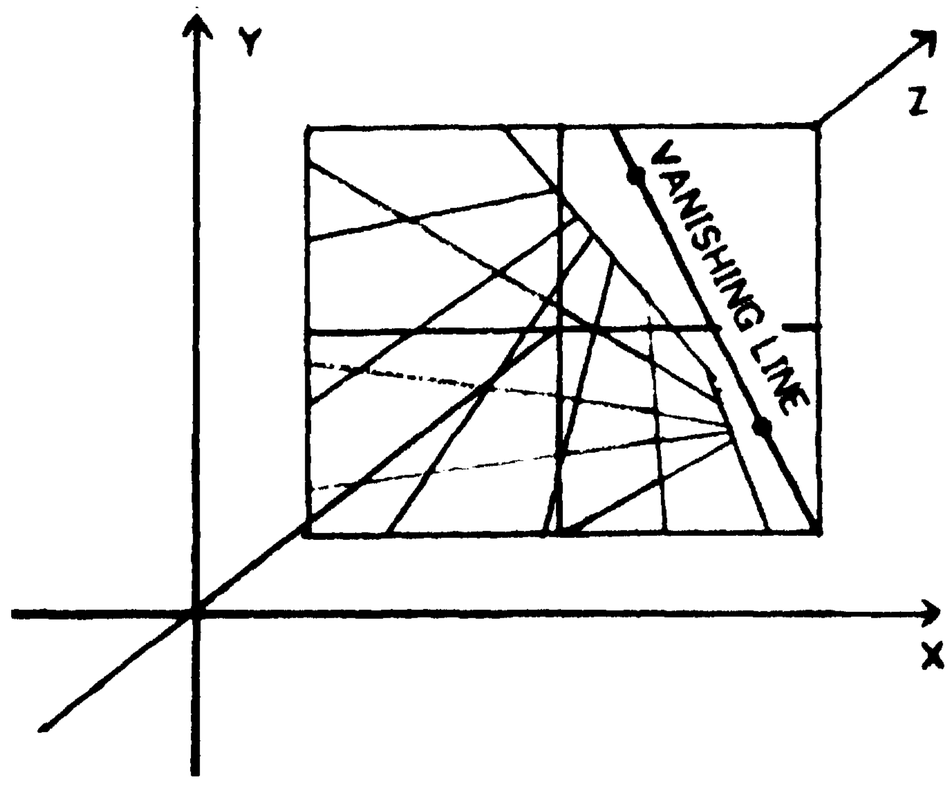


Fig. 5. The imaging model, showing a mesh-like textured plane with its vanishing points and vanishing line.

Proof: By the duality of the transform, the transform of the vanishing line (a point) lies on the intersection of the transforms of the vanishing points (lines). It is this intersection which occurs in the transform space. It is now only necessary to show the correspondence, under the transform, of the vanishing lines and surface gradients.

The imaging process can be modeled as in Fig. 5. The center of projection corresponds to the origin of the (x, y, z) coordinate system. The customarily upside-down focal surface (the "film") has been rotated through the center of projection so that it is right-side-up; it is at a focal distance of R. The projection of the focal point passes through the focal surface at (0, 0, R). On the "film" is the image of a mesh-like planar surface. Also depicted in the figure, for illustrative purposes only, is the plane's vanishing line and two vanishing points. It is important to note that such lines and points are usually only theoretic constructs. They rarely appear in the image itself.

Let the surface equation be given by $px+qy+c = z$, following Huffman. Its gradient vector is thus $(\partial z/\partial x, \partial z/\partial y) = (p, q)$. Points on the surface map onto the focal surface under central projection according to the relation $(x, y, z) \rightarrow (Rx/z, Ry/z, R) = (Rx/(px+qy+c), Ry/(px+qy+c), R)$. The equation of the vanishing line on the focal surface can be found by passing a line through two different vanishing points. However, a vanishing point is defined as the limit of the projection of an infinite line lying in the surface. Let (x, 0, R) and (0, y, R) be two such lines; the limits of their projections, as x and y respectively tend to infinity, are (R/p, 0, R) and (0, R/q, R).

Now, impose a coordinate system on the focal surface in the natural way: "film" points (x, y) are the points (x, y, R) in three-space. The vanishing line in the two-dimensional coordinate system then has the equation $y = -(p/q)x+R/q$. Its normals (its "edge vectors") have everywhere a slope of (q/p); that is, they are in the direction of the vector (p, q). Therefore, any point on the vanishing line is transformed into $R\,E / (E \cdot P) = R\,(p, q) / ((p, q) \cdot (x, -(p/q)x+R/q)) = R\,(p, q) / (px-px+R) = (p, q)$. This last is the surface gradient.

Thus, the theorem is proved: vanishing lines map into gradient vectors. Or, put another way, the transform creates a natural correspondence between the Hough transform and the gradient space.

5. EXAMPLES AND SUMMARY

The complete transform process is summarized and illustrated by the following figures, using a synthetic textured image (Fig. 6: a simulated building face). The focal point is at the center of the image; the focal distance is equal to the width of the image. The surface was created with gradient (p, q) = (0, -1).

Edge vectors (Fig. 7) are first mapped into accumulation points in a polar rho-theta Hough space. Mutually converging lines are thereby mapped into accumulation points that lie on circular arcs through the origin (Fig. 8). Involuting the transform space maps these arcs into straight lines (Fig. 9). This result was actually derived directly from the edge image by means of the more efficient second form of the transform. The lines in the involuted polar space are seen to intersect at (p, q) =

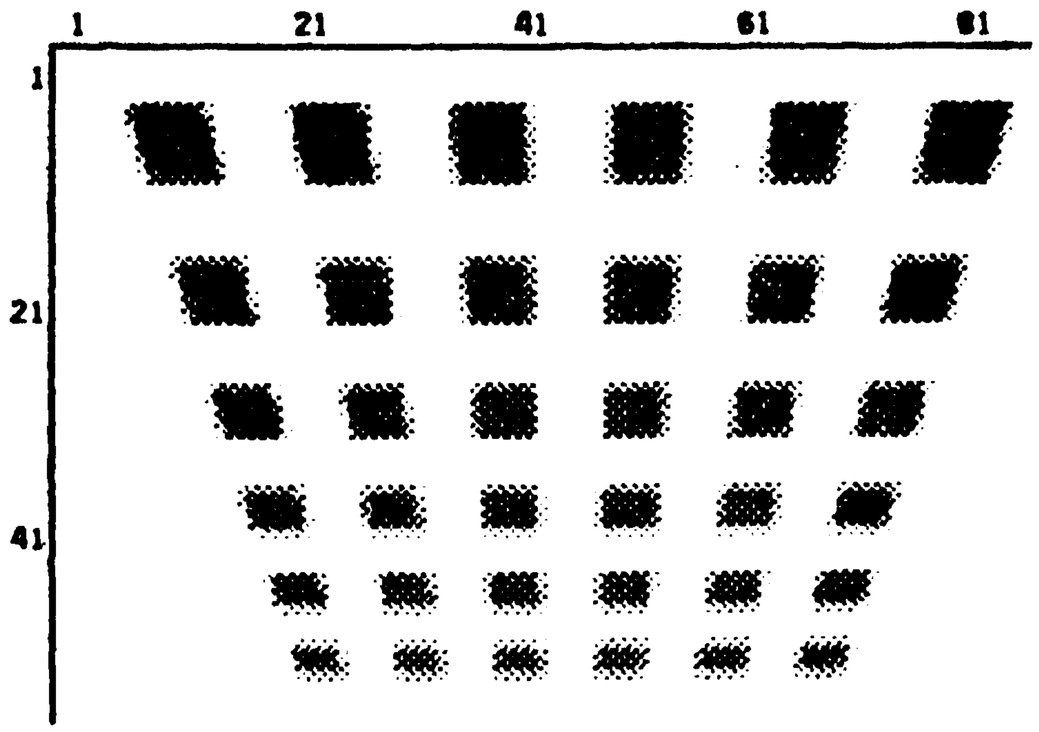


Fig. 6. A synthetic textured scene. The focal point is at (45, 45), which becomes the origin, (0, 0), of the inherited imaging coordinate system. The focal distance is 90, and the surface gradient vector is (p, q) = (0, -1).

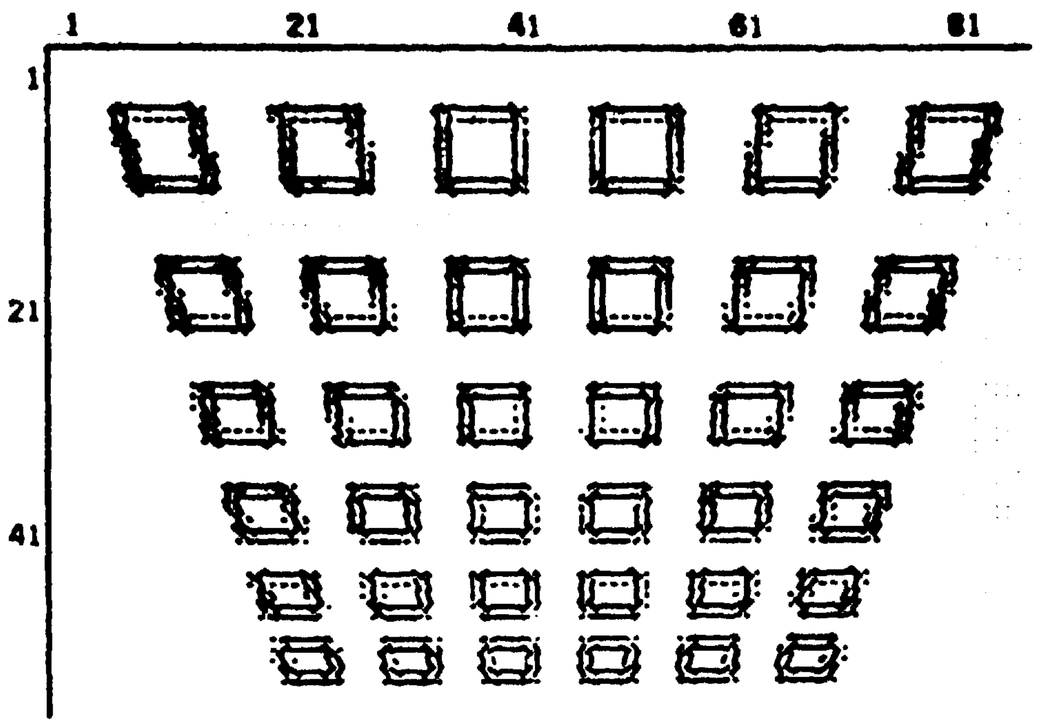


Fig. 7. Local edge elements of Fig. 6.

(0, -1), which Is both the transform of the surface's vanishing line, and the value of the surface's gradient.

Determining shape from texture has many facets; the transform reported here is only one small one. Other approaches based on different classes of texels, together with an overall computational paradigm showing their common relations, will be presented In [10).

ACKNOWLEDGEMENTS

The author gratefully acknowledges many stimulating discussions with Takeo *Kanade.*

REFERENCES

[1] S. Ullman, "The Interpretation of Visual Motion," Ph.D. Thesis, Departments of Electrical Engineering and Computer Science, M.I.T., 1977.

[2] D. Marr, and T. Poggio, "Cooperative Computation of Stereo Disparity," Science. Vol. 194, October 15, 1976.

[3] D. B. Gennery, "A Stereo Vision System for an Autonomous Vehicle," Proceedings of the Fifth International Joint Conference on Artificial Intelligence, M.I.T., 1977.

[4] B. K. P. Horn, "Understanding Image Intensities," Artificial Intelligence, Vol. 8, 1977.

[5] D. A. Huffman, "Impossible Objects as Nonsense Sentences," Machine Intelligence, Vol. 6, Edinburgh University Press, 1971.

[6] F. Tomita, M. Yachida, and S. Tsuji, "Detection of Homogeneous Regions by Structural Analysis," Proceedings of. the Third International Joint Conference on Artificial Intelligence. Stanford, 1973.

[7] R. J. Woodham, "A Cooperative Algorithm for Determining Surface Orientation from a Single View," Proceedings of the Fifth International Joint Conference on Artificial Intelligence. M.I.T., 1977.

[8] S. A. Dudani, and A. I. Luk, "Locating Straight-Line Edge Segments on Outdoor Scenes," Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing. Rensselaer Polytechnic Institute, 1977.

[9] C. L. Fennema, and W. B. Thompson, "Velocity Determination in Scenes Containing Several Moving Objects," Technical Report, Central Research Laboratory, Minnesota Mining and Manufacturing Company, St. Paul, 1977.

[10] J. R. Kendor, "Shape from Texture," Forthcoming Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, 1979.

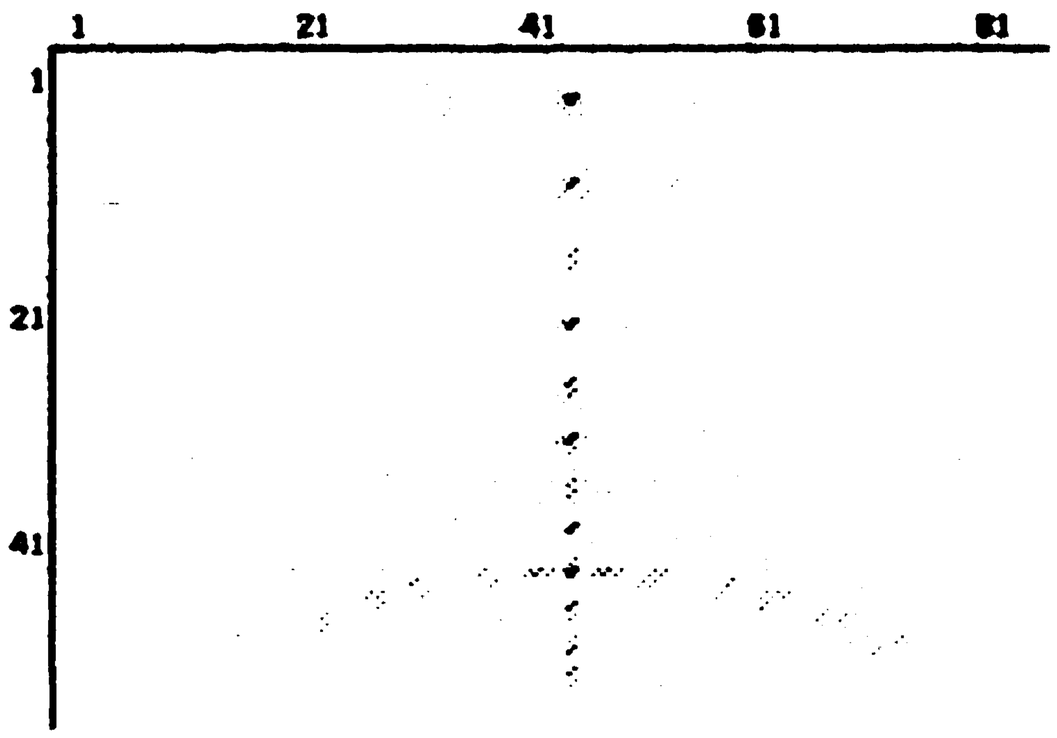


Fig. 8. The new polar form of the aggragating transform of Fig. 6. Both sets of parallel line segments are mapped into accumulation points that lie on circles through the inherited origin, which is at (45, 45).

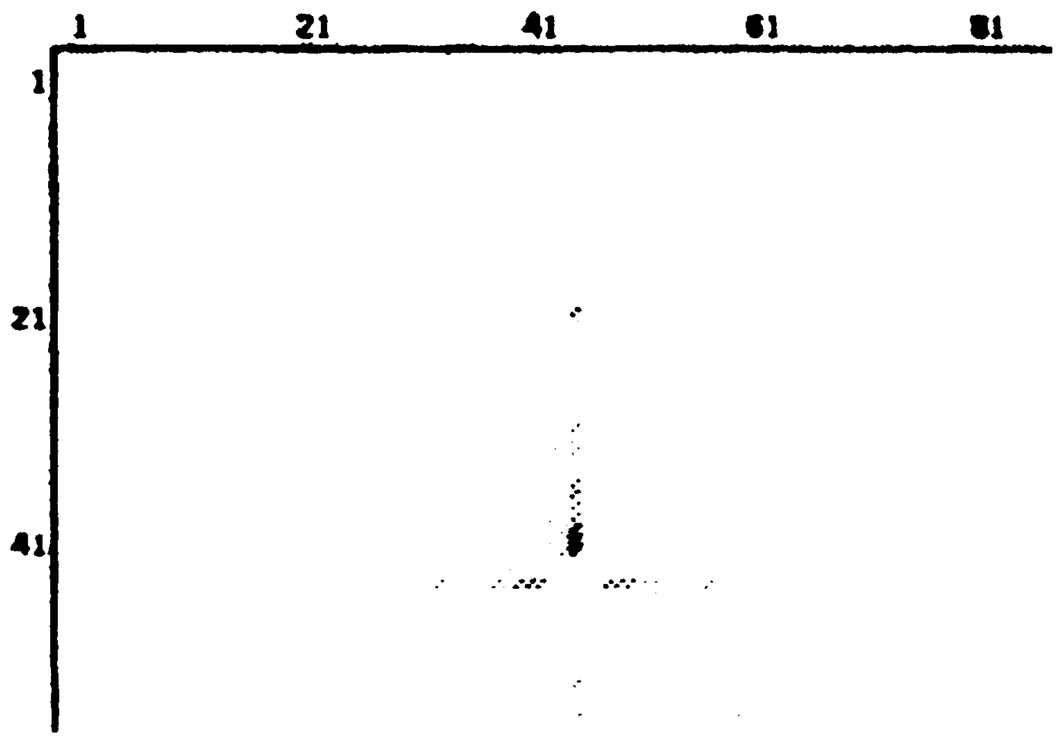


Fig. 9. The efficient, involuting form of the aggregating transform of Fig. 6. Both sets of parallel line segments are now mapped into lines. They intersect at (45, 46), which in the inherited imaging coordinate system is (0, -1). This is the value of the surface gradient vector.

CONDITIONAL ANSWERS IN QUESTION-ANSWERING SYSTEMS

Philip Klahr
The Rand Corporation
Santa Monica, California 90406

In many cases a deductive question-answering system cannot find complete proofs to answer questions requiring deductive support. In some cases information needed to complete proofs is missing from the knowledge base. In other cases processing limits may have been reached before proofs could be completed. Rather than disregarding such partial proofs as most systems do, the DADM system displays them to users and identifies subproblems that remain unresolved. Answers emanating from partial proofs include remaining subproblems as "conditions" which must be true for the answers to be valid.

1. INTRODUCTION

Most deductive question-answering systems output only complete proofs and answers in response to given queries. Partial proofs are usually ignored. Many partial proofs, however, are relevant to deducing an answer to a query but have not been completed because of missing information or because processing limits have been exceeded. We argue that such partial proofs and the resulting partial answers can be of significance to users for the following reasons:

1. Identify what deductions the system has discovered to show a user how the system interpreted his original query.

2. Allow a user to participate and aid in the construction of proofs by letting him examine proofs under development and allowing him to advise and select potentially fruitful deductive paths.

3. Identify incomplete knowledge about particular predicates either in the data base of facts or in the set of knowledge-based rules.

4. Identify what information is needed to complete existing partial proofs and to give complete answers.

In the DADM system [3, 4, 7], partial proofs are not ignored or disregarded because they represent proofs in progress. If DADM is unable to find complete proofs within specified time limits, partial proofs are displayed along with their associated "conditional answers" (answers conditional on the truth of the remaining subproblems).

2. OVERVIEW OF DADM

DADM (Deductively Augmented Data Management) is a natural-deduction system (Bledsoe [1] reviews such systems) designed to interface with existing and emerging relational data base management systems. To facilitate this design criterion, there is a distinction and separation between rules and facts. Rules (axioms, theorems, rule-based knowledge) are in the form of predicate-calculus implications. Facts are single literals containing only constants (predicates whose arguments contain no variables), to be consistent with the form of facts typical in relational data bases.

Another design criterion is that the system should be efficient in dealing with large numbers of rules and facts. To this end, DADM uses PATHFINDER [5], a planning system designed to locate relevant rules before rules are actually applied in the course of constructing proofs. PATHFINDER uses the process of middle-term chaining to locate deductive implication chains by combining forward chaining from assumptions and backward chaining from goals. This process may be envisioned as one of generating expanding wavefronts in the two directions. The purpose of middle-term

chaining is not to construct proofs but to locate potentially relevant deductive implication chains through the rules.

Middle-term chaining does not operate directly on the rules. It uses a predicate connection graph [7] which contains information about the deductive connections (unifications) among the rules and implication connections within rules. (The connection graph is similar to other theorem-proving connection graphs, e-g- [8], although in DADM it is used as a planning tool within a natural-deduction system.) This graph is compiled when rules are first entered into the system. Thus, during proof planning and proof construction, the system has knowledge about all the deductive interactions among the rules and need not compute them dynamically.

Middle-term chains form the basis of the planning process designed to selectively focus on potentially relevant rules. The planning process forms skeleton proofs whose variable substitutions remain to be shown consistent throughout the proof. This latter task is the function of the verifier. Verification is delayed until the system has planned out potential proofs. The verification process examines the variable substitutions involved in the deductions (unifications) of a proof to test that no variable takes on conflicting values.

A middle-term chain represents a deductive implication chain through a sequence of rules. Using these rules, DADM creates a partial proof and determines if subproblems exist. Remaining subproblems are set up as goals to be resolved. (STRIPS [2] is another system that makes use of partial proofs. Unresolved subproblems are used to select relevant operators that allow the proof to be continued.) DADM has three methods available to resolve remaining subproblems: by deduction through the rules, by data-base search over the file of facts, or by computation if the subgoal predicate had been defined by a computational procedure. When subproblems cannot be resolved because of missing information or time constraints, partial proofs and conditional answers are displayed to the user.

3. CONDITIONAL ANSWERS

Consider the very simple example of trying to show that A => D. DADM begins by forming assumptions and goals from the initial query. In this example there is a single assumption (A) and a single goal (D). Suppose the rules A & B => C and C => D exist in the DADM's knowledge base. Using these two rules, DADM finds a middle-term implication chain from A to D through the middle term C (i.e., A to C to D). B is set up as a subproblem. If deductive processing stops here, DADM would display a partial proof and give the answer "yes, if B." (More detailed examples are given in [6].)

When are such conditional answers given? It must be emphasized that DADM tries to find complete proofs and answers. It will find and display complete proofs before giving the user any information about partial proofs. Also, DADM does not find just one proof. It will continue its deductive processing at the user's request or until DADM exhausts its processing limits.

If processing limits have been reached, DADM will display partial proofs under construction (as in the simple example above). The user may then request DADM to increase its processing time and direct DADM to continue working on particular partial proofs. This interactive process between the user and DADM allows a user to aid in the development and expansion of proofs by advising on directions for continued processing.

Partial proofs and conditional answers are also given in those cases where remaining subproblems cannot be resolved because of missing information. Information could be missing from the fact file (the set of facts in a real-world data base is typically incomplete) or from the rule set (DADM may not be able to prove or disprove remaining subproblems). Conditional answers serve to identify information that, if true, would complete partial proofs and provide complete answers. Such knowledge may also serve to suggest the insertion of new rules and facts.

An important concern in displaying partial proofs is deciding which proofs to display to a user, particularly when DADM has generated a large number of partial proofs. It is always the case that a partial proof is verified before it is displayed. The deductions in a proof must be consistent within one another in terms of the variable substitutions required. Those partial proofs that do not successfully verify are ignored and never shown. Any partial proof that does verify may be of potential importance to a user. The system should then order the display of partial proofs on the basis of relevance and importance.

Three main methods are used for ordering partial proofs. These methods are actually used during proof construction in the planning process rather than after proofs are formed. Thus the methods apply to constructing proofs in general, partial or otherwise. These methods involve the use of advice, plausibility, and the number of remaining subproblems.

DADM allows a user to give advice on the use of rules that he feels may be particularly appropriate for deducing an answer to a query. Furthermore, a user may advise the system to focus on particular predicates that may be appropriate middle-terms for chaining. (The user may also give negative advice, in which case specified rules and/or predicates will be avoided in proof generation.) In addition to problem-specific advice, a permanent advice file also exists for storing general domain advice which is accessed for each query [7].

Advice is transformed into rule and predicate ilert lists which are used during middle-term chaining to order and prune the predicates and rules used. The system will try chaining through advised predicates and through advised rules whenever it can. Advice thus serves as a focus of attention mechanism for the chaining and planning processes. Proofs using advised rules and predicates will be generated and displayed first.

The use of plausibility measures serves a similar focusing function. Rules have plausibility measures associated with them. During middle-term chaining, rules are ordered according to their plausibilities. Resulting proofs will be generated and displayed on the basis of the plausibility of the rules used in the proof. Thus most plausible proofs will be displayed first. (Note that advice is given highest priority in the generation of proofs.)

The third method for ordering proofs concerns the number of remaining subgoals in the proof. This is particularly important for ordering the display of partial proofs. Since unresolved subgoals result in conditionals in the partial answer, the fewer the number of such conditionals, the more valuable a partial proof' and partial answer will be to a user. Partial answers are conditional on the missing information being true. The greater the number of conditionals, the less likely that all of them would be true. Thus partial proofs are ordered according to the fewest number of remaining subgoals.

DADM has been designed to be transparent and user-oriented. It tries to give the user as much information as it can or as much as is desired. DADM's ability to recognize and report partial proofs and conditional answers has enhanced the system's usability, particularly over incomplete knowledge bases (where partial proofs and answers become more significant) and over large knowledge bases (where constraints on processing time may preclude the discovery of complete proofs).

Acknowledgments

Charles Kellogg and Larry Travis have been instrumental in the design of the DADM system throughout its development. They have contributed substantially to the ideas presented here.

REFERENCES

[1] Bledsoe, W. W. "Non-resolution theorem proving." Artificial Intelligence, 9 (1977) 1-35.

[2] Fikes, R. E. and Nilsson, N. J. "STRIPS: a new approach to the application of theorem proving to problem solving." Artificial Intelligence, 2 (1971) 189-208.

[3] Kellogg, C, Klahr, P., and Travis, L. "Deductive methods for large data bases." Proc. IJCAI-77, MIT, Cambridge, Mass., 1977, 203-209.

[4] Kellogg, C, Klahr, P., and Travis, L. "Deductive planning and pathfinding for relational data bases." In Logic and Data Bases, H. Gallaire and J. Minker (Eds.), New York: Plenum, 1978, 179-200.

[5] Klahr, P. "Planning techniques for rule selection in deductive question-answering." In Pattern-Directed Inference Systems, D. A. Waterman and F. Hayes-Roth (Eds.), New York: Academic Press, 1978, 223-239.

[6] Klahr, P. "Partial proofs and partial answers." Technical paper P-6239, The Rand Corporation, Santa Monica, Calif., 1978.

[7] Klahr, P., Travis, L., and Kellogg, C. "A deductive system for natural-language question answering." To appear in Natural Language Based Computer Systems, L. Bole (Ed.).

[8] Sickel, S. "A search technique for clause interconnectivity graphs." IEEE Transactions on Computers, C-25 (1976) 823-835.

A PROCEDURAL REPRESENTATION OF LEXICAL ENTRIES IN AUGMENTED TRANSITION NETWORK GRAMMAR

Yutaka Kobayashi and Yasuhisa Niimi
Dept. of Computer Science
Kyoto Technical University
Sakyo-ku Matsugasaki
Kyoto 606, JAPAN

The authors have developed a new representation method of lexical entries and implemented it in the continuous speech recognition system. The basic structure of an entry is a transition network. Associated with each arc of the network are the expected phoneme symbol and a pair of procedures which specify the run-time treatment of phonological transformations and provide the facility of arbitrary contextual and prosodic tests.

1. INTRODUCTION

We have been working on the development of a continuous speech recognition system of a subset of 'BASIC', aiming at a voice-input programming system. The major items of task specifications are as follow: continuous speech, high S/N ratio recording, strict artificial syntax, vocabulary size of 50 words, loose semantic constraints, and multiple speaker. The first version of our system [1], [2] demonstrated a sentence recognition rate of 85 percent, comparable to any other system, but not adequate in a real situation.

The main difficulties of almost all speech recognition systems reported so far lie in the precision and accuracy of the acoustic analyzer, the representation of the word dictionary and the word matching method in the context of ambiguous phoneme and word boundaries. Although the importance of higher linguistic constraints was certified as expected, the improvement of the fundamental power of the system is now again, strongly asked for. In particular, the proper use of prosodic information is expected.

To cope with it, we have improved the acoustic analyzer and developed a new lexical matching module based on a procedural representation of lexical entries in the Augmented Transition Network Grammar (ATNG). The skelton of a lexical entry is a transition network in which a transition along an arc is conditioned by the recognition of a constituent phoneme. We also attach to each arc a pair of procedures written in LISP-like format which specify the run-time treatment of phonological transformations and provide the facility of arbitrary contextual and prosodic tests including any interaction with other levels of the system.

The comparison with relevant works is given later in Section 4.

Before entering into the details, we should better state the environment around the lexical level of our system. It receives a segment lattice from the acoustic analyzer and the possible words predicted by the parser, and returns the matched words with their score back to the parser. The parser works in the top-down, left-to-right and best-first manner.

2. REPRESENTATION OF LEXICAL ENTRIES

The ATNG [3] has been used to describe higher linguistic information such as syntax and semantics. A network is a graph made of several nodes and directed arcs connecting them. There are four types of arcs, CAT, PUSH, POP and TST, and a procedure may be associated with an arc, which operates on the global and local variables containing partial results of the analysis.

To deal with our problems we have made a few changes in notation and interpretation. An arc is a quadruple (DEST, PHON, TRUE, FALSE). PHON is a single phoneme label or a procedure which contains a precondition of the phoneme searching and a specification of one or more phonemes. The discovery of one of these phonemes causes a transition to the destination node DEST of the network. TRUE is a procedure which is carried out after the searching, especially when one of the predicted phonemes

is found in the specified region of the acoustic output. The procedure mainly consists of some tests about the matched phoneme and global features such as a syllable duration. In recognizing a short functional word, it is often the case that prosodic cues, such as the phoneme duration, play an important role. If the phoneme is not found or a test of TRUE procedure rejects the phoneme matching result as inadequate, the FALSE procedure is performed instead of the immediate truncation of that path in the word graph. Some of phonological transformations are also treated here. Some penalty increment is given to the omission of the phoneme, and the replaceable secondary phoneme is looked for if it is expected. Thus, the invention of TRUE and FALSE combination offers simple graph structures as well as appropriate interpretation of phonological transformations.

We adopted LISP-like representation to describe procedures. Atoms correspond to constants and, what we call, registers. Only numerical constants are allowed. The registers are divided In two types: local registers and global ones. The former type includes several general registers and those containing the information on the specific phonemes matched in the limited region of the acoustic output. For instance, we use a general register to hold the accumulated value of segment durations for the test of syllable duration. Registers of the latter type are chiefly those containing information on the neighboring word. As basic operations, we have comparing operators between atoms, their boolean connections, register setting and conditional execution control (COND-function). Other basic functions are penalty Incrementation, phoneme prediction, additional phoneme searching and so on. Any combination of the basic functions is possible and, moreover, arbitrary processings can be added in the manner of subroutine calls. For instance, one could ask for a finer acoustic analysis of the short interval of speech if it might be critical to distinguish the predicted words.

3. LEXICAL MATCHING CONTROL

The lexical matching control is a simulator of word acceptors. The matching is straight-forward in the sense that a transition from a node to its successor takes place when a phonemic symbol on the outgoing arc Is found in the acoustic output. Since the number of likely phonemic symbols is limited even if the number of predicted words grows, the control

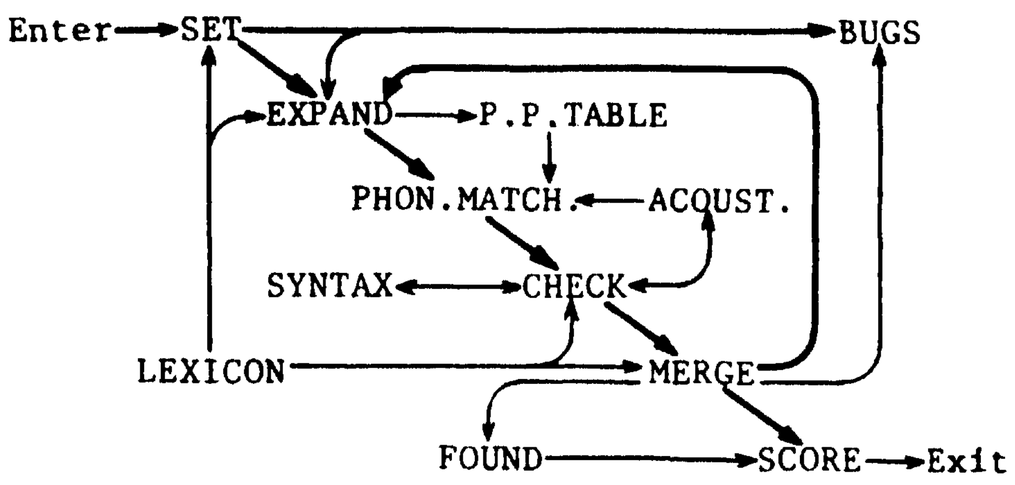


Fig.1 Flow of Lexical Matching Control

performs all the word matchings in parallel to reduce the effort. Fig. 1 illustrates the flow of the control.

To begin with, we introduce a notion of a BUG which is a quadruple (NODE, START, SEG, PEN). NODE Is a pointer to a node of the lexical entry, initially set equal to the initial node. START denotes the starting segment of the word to be verified, while SEG is a pointer to the current segment of phoneme matching, initially set equal to START. PEN contains the accumulated value of word matching penalty.

SET : The control generates as many BUGs as the number of predicted words multiplied by the number of starting segments. These BUGs are placed on the BUG list.

EXPAND : Then it picks up all the BUGs having the smallest segment number SEG, and makes a phoneme prediction table which contains all the phonemes on the arcs going out from the lexical NODEs of the selected BUGs. The contextual tests may be done in advance of the phoneme prediction on some arcs.

PHONEME MATCHING : In this step, each phonemic accordance is searched against the acoustic output. Starting from SEG, the search for the specific phoneme proceeds in the segment lattice. Short transient segments are allowed to skip. The score of phoneme matching is calculated as a duration-weighted average of component label ratings, and the phoneme matching penalty is a function of the score and the skipped gap duration. In order to obtain the optimal word matching over ambiguous boundaries, we do not select a single phoneme matching result here.

CHECK Receiving duration and scores of all

the predicted phonemes, the next step is to perform the TRUE or FALSE procedure on the relevant arc of the lexicon. The detail of this is stated in the preceding section. When passing the test, a new BUG is generated in which NODE is replaced by its successor DEST, and SEG and PEN are updated. The procedure gives an additional increment to PEN in some cases.

MERGE : Those BUGs picked up in "EXPAND" are removed from the BUG list and the newly generated ones are added to it. If several BUGs in the list have an identical NODE number of a same word and an identical SEGment number, the one having the smallest PENalty is selected among them, while the others are discarded. After the optimization is done in such a way, the BUGs which have arrived at the final node are moved to the FOUND list. Then the control returns to "EXPAND" to repeat the cycle, if the BUG list is not empty. Otherwise, it goes to the next step.

SCORE : If the FOUND list is not empty, the word matching score is calculated for each matched word from the accumulated PENalty, and the plural results of the same word having the same START segment are merged to one with a range of end segments. This greatly reduces the search space of the parser.

Thus the lexical matching is performed in such a manner that many BUGs proceed deviding, uniting and dying in the product space spanned by the segment lattice and all the predicted word graphs. In other words, this matching process is equivalent to the breadth-first search in this product space. Moreover, it is considered to be an application of the dynamic programming matching method because PEN is a non-decreasing function and the optimization is done at several points in the space.

4. EXPERIMENTAL RESULT AND DISCUSSIONS

Implementing the system in a minicomputer in Assembly language, we have carried out the recognition experiment of 72 statements of 'BASIC' language read by 2 speakers in order to estimate the facility of the new lexical representation. At first we designed very simple lexical entries from orthographic pronunciations, and then refined them observing the acoustic output of one of the two speakers. The refinement includes duration and contextual tests, creation of optional arcs and nodes, and so on. Experimental results show that this method is promising because the basic graph structures of the lexical entries are much simplified to demonstrate the comparable performance, 87 and 84 percent respectively, with that of our former system. Now the dictionary is under finer adjustment, and we are processing a large number of sentences of 10 or more speakers.

The BBN group developed a similar graphic representation of lexical entries and the dictionary expansion system. LISP functions are introduced in order to examine whether or not some phonological rules could apply in a particular context at the stage of the dictionary expansion from the orthographic description of a word. But the constructed dictionary does not contain procedural representations any more. The lexical matching of the BBN system against the output of acoustic analyzer is based or. the probability calculation.

Instead we introduced procedural representation into the lexical entries themselves so that intra- and inter-word phonological transformations are treated conditionally at the lexical matching phase. Although the descriptive power of variety of pronunciation is almost the same, one could include any tests of global context, prosodic information or finer acoustic cues dependent of individual words.

Considering that the use of prosodic information has been expected, our method is well-suited for the purpose, while those systems which adopted the uniform algorithm of lexical matching, such as the Dynamic Programming, deal the problem poorly. Ours has a disadvantage that the dictionary construction is troublesome, but we consider that more information dependent of individual words must be utilized to achieve a higher performance.

REFERENCE

[1] Y.Niimi, Y.Kobayashi, et al., The Speech Recognition System of 'SPOKEN-BASIC-1', Jour. Inform. Process. Soc. of Japan 18:5 (1977) 453-459 (In Japanese).

[2] Y.Niimi and Y.Kobayashi, A Voice-Input Programming System Using BASIC-like Language, In Proc. IEEE ICASSP, Camelot Inn, Tulsa, OK, April, 1978, pp. 425-428.

[3] W.A.Woods, Transition Network Grammars for Natural Language Analysis, Comm. of ACM 13:10 (1970) 591-606.

[4] W.A.Woods, Speech Understanding Systems: Final Report 1974-1976, BBN Report 3438 (1976).

AN INFERENCE NET COMPILER FOR THE PROSPECTOR RULE-BASED CONSULTATION SYSTEM

Kurt Konolige

ArticifiaL Intelligence Center
SRI International
Menlo Park, California 94303

A compiler for the production system used in a rule-based consultation system is described. This compiler produces machine code which performs antecedent or data-driven inference In an efficient manner. Timing results and some limitations of the compiler are discussed.

1. Introduction

Typically a rule-based consultation system (which we shall call an expert system) is used in an interactive mode as a consultant: a user volunteers Information from which conclusions are drawn, or a goal is pursued by the system and appropriate questions asked, or there is a combination of both kinds of system interaction. The end result is that the system presents a conclusion about some hypothesis to the user, based on information provided by him.

By contrast, a noninteractive mode for an expert system can also be useful. Noninteractive situations arise when information is presented to the system in Large doses, rather than parceled out piecemeal by an interactive user. Typical applications include data-base screening, in which the system must process a Large number of records; handling map data, In which the system must be run on each of a Large number of cells covering the map; and sensitivity anaLysis of the system, in which answers provided by the user are changed incrementally. These noninteractive applications are characterized by the fact that the type of input is known before the system is run, but the exact values are not. This paper describes a method for compiling a certain type of expert system into an efficient linear form under these conditions.

2. Expert Systems

Many expert systems use a production system to encode the knowledge gleaned from experts [1, 4]. Such a production system consists of a data base of assertions; a set of production ruLes of the form <antecedent> => <consequent>; a control program that decides on an inference strategy; and an inference mechanism to propagate the effects of input data. The last two functions are often integrated into a single program called the production system interpreter.

The compilation technique described here was developed for a restricted class of probabilistic production systems* with the following properties:

1. No variables are bound in the production ruLes.

2. There are no inference loops (e.g., A => B, B => C, C => A).

Given these restrictions, a set of productions can be converted into an inference net with no Loops: a portion of this net is Illustrated in Figure 1.

Productions are indicated by arrowed Lines. At the leaf nodes of the net are Input assertions to which the user (or some other input source) can assign probabilities. These probabilities are propagated from the Leaf nodes through intermediate hypotheses to the root hypotheses.

3- The Compilation Technique

A compilation technique for inference nets was developed to take advantage of the noninteractive nature of the input. The

Probabilistic production systems allow probabilistic inference by attaching probabilities rather than a binary truth value to assertions, and allowing production ruLes to have varying strengths in updating an assertion. See, e.g., [2], [6].

This research was funded in part by USGS Contract No. 14-08-001-15985 and NSF Grant No. AER77-04499-

Examples of such systems are [2, 6].

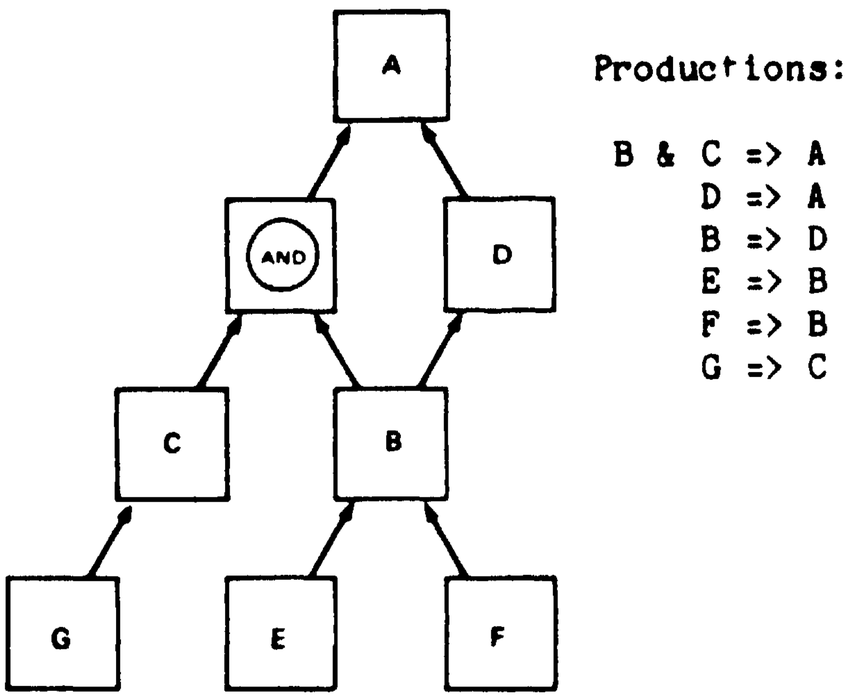


FIGURE 1 A SIMPLE INFERENCE NET

assumption of a noninteractive mode has two parts:

1. "Batch" assumption: all input to the leaf nodes of the inference net is available at the start of the run.

2. Explicit goal assumption: the hypotheses for which results are to be computed are known at the start of the run.

The consequences of these assumptions are exploited in the compilation technique as follows:

Elimination of Control Strategy Overhead — Starting from a set of root goal nodes, the control program normally searches down a path in the inference net to an input leaf node; it then asks the user for a probability for that node. The control strategy is typicaLly goal-directed; it tries to establish a query order for the leaf nodes that is both efficient for hypothesis discrimination and meaningful to the user.

Under the noninteractive assumption of an explicit goal set, it is possibLe to eliminate the goal-directed strategy entirely. Since the leaf nodes for which the probabilities are known are given in advance, there is no need to search for them in the noninteractive mode. Inferencing becomes data-directed from the input leaf nodes, rather than goal-directed from the hypothesis nodes.

Propagation — In the interactive mode, the consequences of new information on a leaf node are propagated immediately throughout the net. Obviously, the probability for a hypothesis node may be calculated many times through this procedure; for example, node A may be updated a total of four times — once each time the effects of changing the probabilities of nodes E, F, C, and D are propagated.

If ail probabilities on leaf nodes are available at the beginning of propagation, it is possible to devise a propagation scheme in which the probability of each hypothesis node is computed only once, in a single bottom-up pass. Because it has no loops, the inference net defines a partial ordering of nodes from the leaf nodes to the root node. In this ordering, for example, the probability of node B must be computed before node C, and both of them before A. The compilation technique converts the partial ordering defined by the net into a linear ordering. In the compiled net, probabilities on hypothesis nodes are computed in this linear order, and the results saved in temporary cells for later calculations.

The input to a compiler embodying this compilation technique is an inference net of the type described above. Its output is a linear (i.e., loop-free) code sequence that calculates the probability of the root hypothesis, given a data set for the leaf nodes. This sequence simulates the action of the net interpreter on the net in a noninteractive mode. The amount of time taken to execute the code can be estimated as:

$$C_L \cdot N_L + C_P \cdot N_P \qquad (1)$$

where the C's are constant for a given machine executing the code, and the N's are defined as:

N_L = number of logical productions*

N_P = the number of probabilistic productions

4. Timing Results for a Compiler Implementation

A compiler for the inference nets of the PROSPECTOR [2] expert system was implemented to test the practicality of the compilation technique. It compiles PROSPECTOR nets into a linear machine code sequence. A short initializing sequence is also included to enable linking of the compiled inference net to any set of inputs; thus the inference net need only be recompiled if the net structure has changed.

The implementation of the compiler follows in a straightforward way from the compilation techniques of the Last section. Total compilation time is proportional to the number of productions in the net; a nominal amount of time is spent processing each production.

t

* Logical productions are logical combinations of assertions, e.g., the AND node in Figure 1.

The compiler was tested using a medium-scale PROSPECTOR inference net comprising some 94 nodes and 105 productions (59 of them were logical productions). This inference net encoded expert knowledge about geological characteristics of a favorable exploration drilling site for a type of copper mineral deposit [3].

In a typical application, data for 13 of the input nodes was extracted from geological maps of an area. The area was partitioned by a 128 x 128 grid into 16,384 map cells, producing input data for 16,384 runs of the compiled model. The resulting probabilities on the root hypothesis were recombined to yield Figure 2.

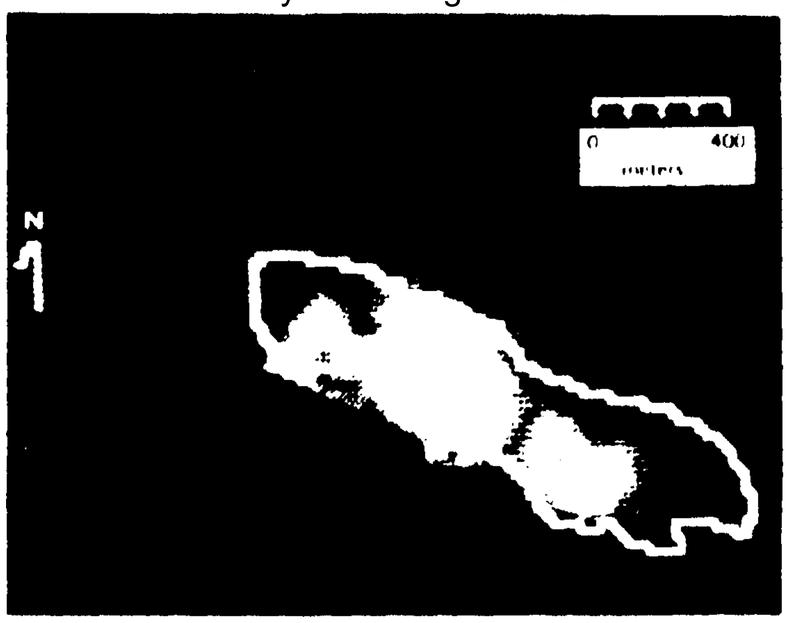


FIGURE 2 RESULTS OF 16,384 RUNS OF A COMPILED INFERENCE NET USING MAP DATA

Note: The brightness of each individual pixel indicates the favorability of the root hypothesis for that pixel. The bright line is the actual ore body outline. The area Is Island Copper in British Columbia, Canada [3].

Timings for this run are given in Table 1. The estimated running time given by equation 1 is also listed, and agrees well with the actual time. These timings indicate typical speeds that we have achieved under the noninteractive assumptions.

For comparison, the inference net interpreter was run using the data set from a single pixel. Timings for a typical interpreted run are also given in Table 1. For this specialized application the compiLed version of the net is about four orders of magnitude faster than the net interpreter in the test case.

C. and C_p were estimated to be 6.0 us and 44 us, respectively, by looking up machine instruction times in [5 3] Experience with many runs indicates that equation 1 i3 a reliable estimator of running time.

Table 1

CPU Timings	Control	Propagation	Total
Interpreted	17 s	13 s	30 s
Compiled, actual	0	3.1 ms	3.1 ms
Compiled, estimated	0	2.4 ms	2.4 ms

5. Conclusions

A compiler for PROSPECTOR-type inference nets has been described. It generates code that is approximately four orders of magnitude faster than the net interpreter. Its efficiency and practicality for noninteractive use has been demonstrated in an application using digitized map data as input.

As currently conceived and implemented, the compiler suffers from the drawback of not being able to handle production ruLes containing variables. A compiler that produces linear code is probably inadequate for dealing with arbitrary variables in production rules. However, the use of variables in the productions used by such expert systems as PROSPECTOR and MYCIN is very Limited. We are exploring restrictions on the use of variables in productions that will make compilation techniques for variables feasible.

REFERENCES

[1] Randall Davis and Jonathan King. "An Overview of Production Systems," E. W. Elcock and D. MIchie (ed.), Machine Intelligence, vol. 8, wiley, N.Y., 1976.

[2] P. E. Hart et al. PROSPECTOR - A Computer-Based Consultation System for Mineral Exploration. Mathematical Geology 10(5):589-610. 1978.

[3] Richard 0. Duda et al. System for MINERAL Exploration* Technical Report, Artificial Intelligence Center, SRI International, 1978.

[4] Frederick Hayes-Roth et. al. "Principles of Pattern-Directed Inference Systems," D. A. Waterman and Frederick Hayes-Roth (ed.), Pattern-Directed inference systems. Academic Press, 1978.

[5] Digital Equipment Corp., Maynard, Mass. DecSysten 10 System Reference Manual, 1974

[6] E. H. Shortliffe. Computer-Baaed Medical Consultations MYCIN. American Elsevier, N.Y., 1976.

ETHER - A Parallel Problem Solving System

William A. Kornfeld
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Mass. 02139
arpanet BAK@MIT-AI

ETHER is a new pattern directed invocation language for problem solving applications, ETHER programs allow arbitrarily much processing to happen in parallel. For example, when a goal generates several subgoals they are all pursued in parallel ETHER introduces the concepts of an *activity* which is a generalization of *process,* and *platform,* a generalization of *context.* Two uses for parallelism are discussed: parallel evaluation of subgoah and the use of *opponents,* concurrently running activities that attempt to show a subgoal is unattainable. If the opponent succeeds then work on the subgoal halts.

1. Introduction

ETHER is an attempt to form a synthesis of ideas about parallel programming with concepts of pattern-directed invocation. There nave been several researchers proposing parallel AI systems [1,2] is well as systems that make use of agendas and similar mechanisms [3, 4] to get parallel-like behavior. This paper is concerned with the motivation and basic concepts of parallel pattern-directed invocation using ETHER. Due to space limitations it is sketchy in many areas; a much more complete discussion can be found in *[5]*

2. Phylogeny of Planner-like Languages

In the early I970's there was great interest in developing languages for Artificial Intelligence. This began with PLRNNER 16) which was implemented as MICRO-PLRNNER *[1]* Experience with the ideas of PLRNNER and earlier theorem proving systems led to the development of several more sophisticated systems at different centers: CONNIVER [8] at MIT, QR4 [9] at SRI, and POPLER [10] at Edinburgh. The general contribution these languages have made to programming are greater flexibility in representing and accessing data and the ability to gain access to procedures, not by their name, but by an indication of what they can do (so called *pattern-directed invocation)* [6} If there was a problem to be solved, the routine charged with solving it could instantiate a goal.

```
(COm. ("My problem it ..."))
```

The implementation would then match the statement of the problem against statements of what each procedure could do, and procedures that are relevant would be invoked.

The original focus of PLRNNER was the writing of programs that manifested intelligence by the combined effect of loosely coupled procedures. In MICRO-PLRNNER it was possible to create independent units that said "If you want to accomplish the following result, try the following:..." The interpreter was charged with the task of combining this knowledge. It treated the theorems of MICRO-PLANNER as any standard language interpreter would treat subroutines. However, because nothing specified the order theorems were tried, MICRO-PIRNNER programs followed a wasteful, undirected depth-first search, using chronological backtracking.

CONNIVER tried to amend this situation by putting the control structure in the hands of the user [8). Programs had control over the generation and selection of problem solving strategies. Unfortunately, these new abilities destroyed much of the expressive ability inherent in the original MICRO-PLANNER modularity

Our view is that the failure of MICRO-PLANNER programs to control their search was because *program control had to be located at one point in the program at any given time.* This assumption is implicit in the so-called Von Neumann machine model that underlies virtually all current hardware and software systems but has come under attack recently by several researchers. We introduce the notion of an *activity* and show how it can be used to control reasoning in a highly parallel system. The context mechansm introduced in [9] is generalized to *platforms* all of which may be concurrently manipulated.

3. Pidgin ETHER

This paper describes a language called ETHER that allows the user to construct problem solving systems embodying the philosophy discussed above. The analog of the PLRNNER theorem is called in ETHER a "sprite". Sprites have two principle parts, a pattern and a body. Once created, a sprite watches for an assertion to appear that matches its pattern, and, if this occurs, its body is executed. The body may contain commands to create new sprites *or* commands to *broadcast* new assertions that other sprites may trigger on. This section develops a subset of the ETHER language known as pidgin-ETHER. The ideas in pidgin-ETHER are quite similar to those in [12). To illustrate the difference between ETHER and MICRO-PLANNER, consider the following MICRO-PLANNER-like consequent theorem:

```
(to-prove (BRCHELOR ?X>
          (CORL (UNMARRRIED ?'X))
          (GORL (MALE ?X)>)
```

This says "If you want to determine if X is a bachelor try to show that he is both male and unmarried." A MICRO-PLANNER interpreter would first try all possible ways of showing that x is unmarried, and then if it achieves success, all possible ways of showing x is male. The analog to this in pidgin-ETHER might be written:

```
(when  (CORL  (BRCHELOR -x))
       (broadcast  (GORL (UNMARRIED x))>
       (broadcast  (GORL  (RRLE x>>>
       (when&  ((UNHRRRIED  x)
                 (RRLE  x>>
                (broadcast  (BRCHELOR  x)>)>
```

This example makes use of *explicit goal assertions* [I I, 12] instead of primitive consequent rules. When this sprite receives a matching assertion (such as "(GORL FRED>") simultaneously three things occur: a goal assertion is broadcast that causes work to begin on determining whether x is unmarried, a goal assertion is broadcast that causes the system to begin work on determining

whether x is male, and a new sprite is created that waits for these assertions to appear in the database. When (and if) they finally appear the BRCHELOR assertion is broadcast satisfying the goal.

In the ETHER example both goals are worked on *simultaneously.* Each of the GORL assertions (requests to work on the goals) may each be picked up by several sprites serving as consequent theorems and in turn may themselves cause several subgoals to be pursued concurrently.

4. Activities and Stifle

"This example has one problem. When a goal has been solved, computation attempting to achieve it will not stop. For example, even if FREO has been shown to be a BACHELOR, consequent sprites working on the problem of determining if FREO is a BACHELOR will continue doing so. It is desirable to be able to stop this additional processing so that system resources can be reclaimed.

To allow this kind of behavior we will introduce the concepts of *activity* and *stifle* to the pidgin-ETHER so far described. An activity can be thought of as a *locus of control with some specific purpose.* When the work being carried out by an activity is found to be no longer necessary the activity is *stifled* This causes all computation involving sprites that are part of this activity to cease. An activity may be created to achieve some goal. If the goal has been achieved then the program should stifle this activity.

To illustrate the use of activities is redone the above example using them. Activities are initiated by giving broadcast a second argument of the name of the new activity. If a sprite receives this broadcast, newly created sprites are placed in the new activity. Suppose we wish to work on the goal of (BACHELOR FREO). The following broadcast is done with C0AL33 being the name of the activity created to work on this goal.

```
(broadcast (GOAL (BACHELOR FREO)) C0AL 33)
```

There may be a number of different ways to accomplish a goal (each by a different consequent sprite). One way, as was done above, is to demonstrate FREO is both MALE and UNHARRIEO.

Because this is one of perhaps many ways to accomplish a goal, its investigation should occur in a distinct activity.

```
(when (COAL (BACHELOR -x))
      (let (subgoal (new-activity))
           (broadcast (COAL (UNMARRIED x>) subgoal)
           (broadcast (GOAL (MALE x>) subgoal)
           (when ((UNMARRIED x)
                    (HALE x))
                  (broadcast (BACHELOR X)))
           (when (BACHELOR x)
                 (broadcast (STIFLE subqoal)>>))))
```

The code given is like the previous example with some additions to show the use of activities. The function new-activity creates a new activity. The name of this activity is returned and becomes the value of subgoal. All work begun by this sprite occurs in this activity. The last sprite occuring in its body watches for (BACHELOR FRED) to be broadcast, indicating successful completion. If this is ever broadcast, the sprite broadcasts a special assertion instructing the system to stifle the subgoal activity and any activities it might have created. A stifled activity ceases to work on the goal. It does not matter how the goal was finally achieved for the stifling to occur, only that it was achieved.

If the same goal assertion is broadcast twice with different activity markers then only one new activity rather than two will be created. This new activity will be a subactivity of both parents. Thus, if there are two different goals that create a common subgoal, the work on this subgoal will not be duplicated.

5. Platforms and Opponents

Planner-like languages frequently need to create little worlds or *contexts* inside the machine in which to reason. Contexts are usually defined so that only one is visible to the problem solver at a time. A parallel problem solving system must not contain such a restriction because different activities may be set up to concurrently reason about several incompatible world models. We have generalized the context concept to the notion of a *platform.* All platforms are accessible to all sprites at all times. We [5] have found several uses for many concurrently accessible platforms.

Gelernter, in his classic geometry theorem proving program [13], introduced the concept of a *goal filter,* His program worked by using strict backward chaining from the theorem. Before attempting to work on a subtheorem it would check a diagrammatic representation of the subtheorem to see if the subtheorem was consistent. If it was not, the goal was immediately pruned. The concept of checking a subgoal for plausibility is an important one in maintaining coherency in the problem solving process. In Gelernter's program the test was easy to make and would be guaranteed to terminate with an answer in a short span of time. In many problem solving situations goal filters can make use of the full capabilities of the problem solver and may not be guaranteed to terminate. The use of goal filters still may be justified because of the desirability of cutting off exponentially growing trees of useless backward chaining.

One common mode of reasoning that people use to filter goals is to *assume* the object of a goal is true and then seeing if there is an anomaly. For example in a geometry theorem proving program we may have a goal of demonstrating satisfiablilty of the statement "Line AB is longer than line AC in the following:

```
(lint A B)
(line  B C)
(line  A C)
(right-angleABC)
```

We create an activity and broadcast in this activity:

```
(goal (greater (line A B) (line A C)
```

As in the previous examples this will cause consequent processing to happen on this platform attempting to prove this goal. Sprites doing consequent processing will not realize that the goal is unattainable though this is easily apparent to humans. Consequent reasoning sprites will implement backward chaining from this unattainable subgoal.

A method people use for seeing this is by determining the obvious consequents of the proposition that line AB is longer than line AC and seeing that an anomaly arises.

In ETHER, data assertions are grouped in specific *platforms.* We create a new platform that inherits from the platform containing the description of the figure and in it we broadcast:

```
(greater (line A B) (line A C>>
```

Many antecedent sprites are created to allow the system to determine the implications of this assumption. We also instantiate for this platform sprites that can detect contradictions. In this particular example, the assumption has one easily seen consequent, that angle ACB is greater than angle ABC. This implies angle AC B > 90 degrees, and that triangle ABC has internal angles that sum to more than 180 degrees, an anomaly. A sprite watches for the occurrence of anomalies and then stifles the activity attempting to do consequent reasoning on this goal.

We concurrently do consequent processing attempting to prove the goal and antecedent processing to determine if the goal is not realizable. The activity attempting to achieve the goal is known as a *proponent.* The activity in which antecedent processing is being done on the goal to determine if anomalies exist is an *opponent* If the goal is achieved, the result is broadcast and both the opponent and proponent activities are stifled. If the opponent succeeds in finding an anomaly, both activities are stifled.

The inherent parallelism of ETHER is necessary to allow use of opponent platforms. In a sequential problem solving system, the problem solver must *choose* to do one or the other first. It cannot allow opponent processing that is not guaranteed to terminate in a reasonable amount of time. Goal filters can be implemented in our scheme by creating a platform that executes the task of the goal filter, stifling the goal proponent if the filter is successful.

The full power of the problem solver can be brought to bear on the task of filtering goals.

The inheritance mechanism is such that assertions appearing in the parent of a hypothesis platform *virtually* appear in the hypothesis platform itself. The program may inherit antecedent sprites from a parent to a child platform. When this is done, the only new assertions that get generated and placed in the hypothesis platform are only those that depend on the assertions broadcast in this hypothesis platform. Thus the creation of an opponent platform is relatively inexpensive.

6, Further Ideas

the ETHER language is much more extensive than the kernel described here. We briefly summarize some of the other features that are discussed in [5].

As a parallel language The semantics of the language have been carefully designed so that it is implementable on highly parallel hardware without introducing bottlenecks or the possibility of deadlock caused by the use of synchronization primitives. We believe that many of the choices made in this regard are also the right choices from the point of view of language design for modular programming; the language makes it easy to design programs devoid of the timing errors that plague most parallel systems.

Processing power. The concept of an *agenda* has been proposed for incorporating *resource-limited computation* in AI systems. We feel this is too low level a concept because the individual parts of the system must worry about their own scheduling. Imagine how much harder it would be to write programs for a time-sharing system if the individual programs had to schedule each time quantum for themselves. In ETHER the individual activities are truly parallel. To incorporate resource-limited computation we introduce the notion of *processing power.* If some activites seem to be more likely to achieve their purposes than others they are given a larger share of processing power. The user is given a set of primitives in which to build in heuristic knowledge to improve the performance of the program.

Objects and coreference. Most PLANNER-like languages allow variables to appear in goals, ETHER contains a generalization of the notion of an *anonymous object.* The program can place assertions on a platform causing the object to be coreferential with another object on that platform. We show that variables can be replaced with these objects resulting in solutions to many problems that are much more tightly controlled than a solution using variables. Objects also have important applications in planning.

Manipulative inheritance. The inheritance mechanism discussed for platforms above will inherit all assertions from a parent to a child. While this is valuable for the creation of opponents it is not generally useful in planning. Planning situations usually require the ability to make certain assertions invisible through inheritance links. For example, if a block A is taken off of a block B in a hypothetical situation we would like the assertion (ON A B) to be invisible in the hypothesis' platform even though it is present in the higher platform. We employ the use of *inheritance filters* and *justifications* in our solution to the frame problem. The user may tag assertions with markers and allow the system to automatically compute dependency information for newly created assertions. A platform inheritance link can specify a filter that indicates the class of assertions that can be "seen" through the inheritance link.

Virtual Collections of Assertions. A common complaint of pattern-directed invocation systems is that they are too inefficient. We would like to supply a facility that allows the user to interface efficient Lisp code to the ETHER system in a transparent manner, i.e. from the point of view of the ETHER program this efficient subprogram looks as if it is implemented in sprites. For each virtual collection the program must specify two items: the assertions that it would like to know about if they are asserted in the database, and the kinds of assertions for which it can decide virtual presence in the database. We envision a facility like this being used to tie together many relatively large, special purpose, programs so that they can interact in a useful fashion with one another.

7. Conclusions

ETHER is a language that allows arbitrarily much processing to occur in parallel. We believe this feature allows programs to maintain modularity and independence of individual methods while still allowing programs to carefully control their search. We have developed two kinds of generalized control structures. In ETHER it is possible to run many methods in parallel and arrange that computation that becomes unnecessary to be stifled. ETHER makes possible a generalization of the concept of a goal filter in which the filtering computation runs in parallel with the the attempted attainment of the goal, ETHER code is clear and simple and counters the commonly held belief that parallel programs are hard to design and understand.

ETHER has been implemented by the author as part of his masters thesis research. I am currently improving the implementation and modifying it for use on the MIT Lisp Machine.

Bibliography

[I] Lesser, V. R., Erman, L. D., *A Retrospective View of the Hearsay-H Architecture,* IJCAI77.

[2] Smith, R. G., Davis, R., *Distributed Problem Solving: The Contract Net Approach,* Proceedings of the Second National Conf. of the Canadian Society for Computational Studies of Intelligence, July 1978.

[3] Lenat, D.. *AM: An AI Approach to Discovery in Mathematics as Heuristic Search,* Stanford AI Lab Memo AIM-286, 1976.

[4] Bobrow, D. G., Winograd, T, *An Overview of KRL. A Knowledge Representation Language,* XEROX PARC publication CSL 76-4, 1976.

[5] Kornfeld, William, *Using Parallel Processing for Problem Solving,* S.M. thesis MIT, May 1979, forthcoming technical report.

[6] Hewitt, C, *Description and Theoretical Analysis (Using Schemata) of Planner: A Language for Proving Theorems and Manipulating Models in a Robot,* MIT AI TR-258. 1972.

[7] Sussman, G., Winograd T, Charniak, E, *Micro-Planner Reference Manual* MIT Artificial Intelligence Laboratory memo 203, 1970.

[8] McDermott, Drew, Sussman, Gerald, *The CONNIVER Reference Manual,* Artificial Intelligence Laboratory memo 259a, January 1974.

[9] Rulifson, John F., Derksen, Jan A., Waldinger, Richard J., *QA4: A Procedural Calculus for Intuitive Reasoning,* Stanford Research Institute Artificial Intelligence Center Technical Note 73.

[10] Davies, Julian, *POPLER 1.5 Reference Manual,* School of Artificial Intelligence, University of Edinburgh, TPU Report no. I.May 1973.

[II] Hewitt, C, *How to Use What You Know,* IJCAI 75.

[12] de Kleer, J., Doyle, J., Steele, G., Sussman, G., *Explicit Control of Reasoning,* MIT AI memo 427, June 1977.

[13] Gelernter, H., *Realization of a Geometry-Theorem Machine,* in Feigenbaum, Feldman, *Computers and Thought,* 1963.

A POLARIMETRIC APPROACH TO SHAPE UNDERSTANDING OF GLOSSY OBJECTS

Kazutada Koshikawa
Information Sciences Division
Electrotechnical Laboratory
Nagata-Cho 2-6-1, Chiyoda-ku
Tokyo, Japan 100

A polarimetric method is proposed to find surface normals, which will help the 3-D shape understanding of glossy objects. Attention is directed to the fact that glossy objects are characterized by conspicuous specular reflection, which has a certain polarizational property as a function of the incident angle and the refractive index of the material. The relation is formulated between the local normal and the polarizational parameters. The Stokes parameters are used to express the state of polarization. As to several materials preliminary measurements are done to examine for utility of polarizational information to finding local normals. The proposed method will be effective in extracting geometrical parameters from glossy objects whose gloss occurs on dielectric surfaces such as plastics, lacquered items, polished paper and so on.

1. INTRODUCTION

We propose a polarimetric method of obtaining surface normals, which will help the 3-D shape understanding of glossy objects.

Horn[1] has obtained shapes from shading information. Glossy surfaces, however, have little shading information, since they dominantly reflect specular rays.

The specular ray has a certain polarizational characteristic as a function of the incident angle and the refractive index of the surface material.

We examine for utility of polarizational information to finding local normals.

2. POLARIZATIONAL RELATION

In Fig.1, i and s are rays of incidence and reflection upon a plane Σ, respectively, and n is the normal. Ω is the frame to which the Stokes parameters[2] are referred, and ξ' is the reference direction. g implies the intersection of the frame and the plane of incidence.

The polarization state of the ray s contains information about the angles ψ and α, which will provide a clue for finding the normal n.

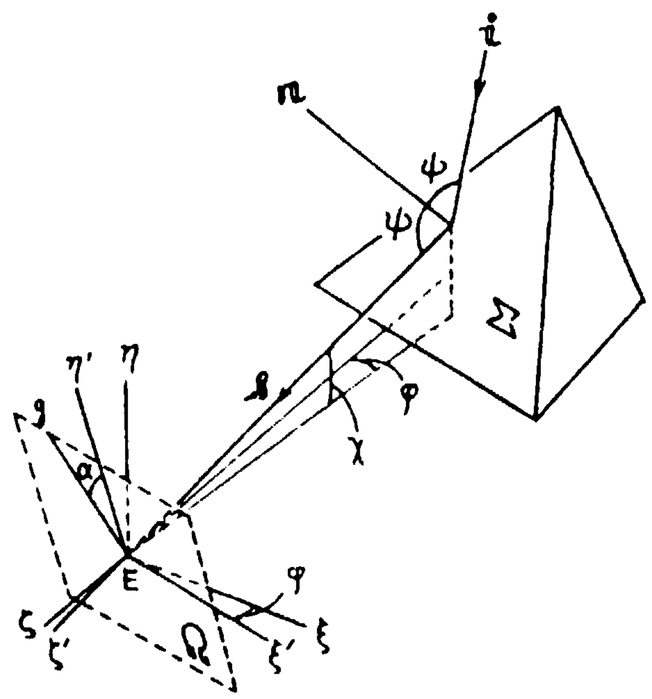


Fig.1 Observation system

We regard the specular plane as a partial polarizer with retardance, and use a right circular ray as an input. Then, the Stokes vector * of the reflected ray will be derived by the Mueller calculus as follows;

* a 4*1 column vector whose elements are the four Stokes parameters

$$\begin{pmatrix} I \\ L \\ X \\ O \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & C_{2\alpha} & -S_{2\alpha} & 0 \\ 0 & S_{2\alpha} & C_{2\alpha} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{2}(r_\perp^2+r_\parallel^2) & \frac{1}{2}(r_\perp^2-r_\parallel^2) & 0 & 0 \\ \frac{1}{2}(r_\perp^2-r_\parallel^2) & \frac{1}{2}(r_\perp^2+r_\parallel^2) & 0 & 0 \\ 0 & 0 & r_\perp r_\parallel & 0 \\ 0 & 0 & 0 & r_\perp r_\parallel \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & C_\delta & S_\delta \\ 0 & 0 & -S_\delta & C_\delta \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & C_{2\alpha} & S_{2\alpha} & 0 \\ 0 & -S_{2\alpha} & C_{2\alpha} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Ic \\ 0 \\ 0 \\ Ic \end{pmatrix} = \frac{1}{2}(r_\perp^2+r_\parallel^2)Ic \begin{pmatrix} 1 \\ PC_{2\alpha}-VS_{2\alpha} \\ PS_{2\alpha}+VC_{2\alpha} \\ U \end{pmatrix}$$

output ← partial polarizer (rotation from reference direction) ← retarder ← right circular input

where $r_\perp, r_\parallel$ the amplitude reflection coefficients of components vibrating perpendicular to and parallel to the plane of incidence

δ the phase difference between both components

C_θ, S_θ $\cos\theta$, $\sin\theta$

Ic the intensity of the light source

$P \equiv (r_\perp^2-r_\parallel^2)/(r_\perp^2+r_\parallel^2)$, $U \equiv 2r_\perp r_\parallel C_\delta/(r_\perp^2+r_\parallel^2)$, $V \equiv 2r_\perp r_\parallel S_\delta/(r_\perp^2+r_\parallel^2)$

On actual surfaces somewhat occurs depolarization. Therefore the next expression will be practical:

$$\begin{pmatrix} I \\ L \\ X \\ O \end{pmatrix} = Id \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + Ip \begin{pmatrix} 1 \\ PC_{2\alpha}- VS_{2\alpha} \\ PS_{2\alpha}+ VC_{2\alpha} \\ U \end{pmatrix} \qquad (1)$$

where Id and Ip are the intensities of depolarized and polarized components, respectively.

3. MEASUREMENT OF THE STOKES PARAMETERS

We use a simplified "rotating-compensator/fixed-analyzer ellipsometer configuration"[3], which can eliminate the polarizational effect of the detector.

The intensity transmitted by this configuration is proportional to

$$I+\cos^2 2\beta \cdot L+\cos 2\beta \cdot \sin 2\beta \cdot X-\sin 2\beta \cdot O,$$

where β is an azimuth of the quarter-wave plate, with the analyzer kept at 0°.

Using its values I_m sampled at $\beta = 22.5° \cdot m$ $(m=0,1,\cdots,7)$, we find

$$\begin{pmatrix} I \\ L \\ X \\ O \end{pmatrix} = \begin{pmatrix} & I_2 & & +I_6 \\ I_0 & -I_2 & +I_4 & -I_6 \\ I_1 & & -I_3 & +I_5 & -I_7 \\ & -I_2 & & +I_6 \end{pmatrix} \qquad (2)$$

4. SEQUENCE TO FIND THE LOCAL NORMAL

1. Calculate Ip as $\sqrt{L^2+X^2+O^2}$.
If Ip is too small, reliable values cannot be obtained at that point. Otherwise,

2. Normalize (1) as

$$\begin{pmatrix} a \\ 1 \\ x \\ \sigma \end{pmatrix} \equiv \begin{pmatrix} I/Ip \\ L/Ip \\ X/Ip \\ O/Ip \end{pmatrix} = \begin{pmatrix} Id/Ip+1 \\ P\cos 2\alpha - V\sin 2\alpha \\ P\sin 2\alpha + V\cos 2\alpha \\ U \end{pmatrix}$$

3. Find ψ to fill $\sigma=U$.
4. Find P and V corresponding to ψ.
5. Find α using 1, x, P and V.
6. Determine π using φ, χ, ψ and α.

5. PRELIMINARY EXPERIMENT

In order to examine for utility of P, U and V to finding ψ, we measured 1, x and σ as to actual materials, changing ψ at $\varphi=\chi=0°$, $\alpha=90°$ (that is, $1=-P$, $x=-V$ and $\sigma=U$).

Fig.2 shows some results of it. The curves

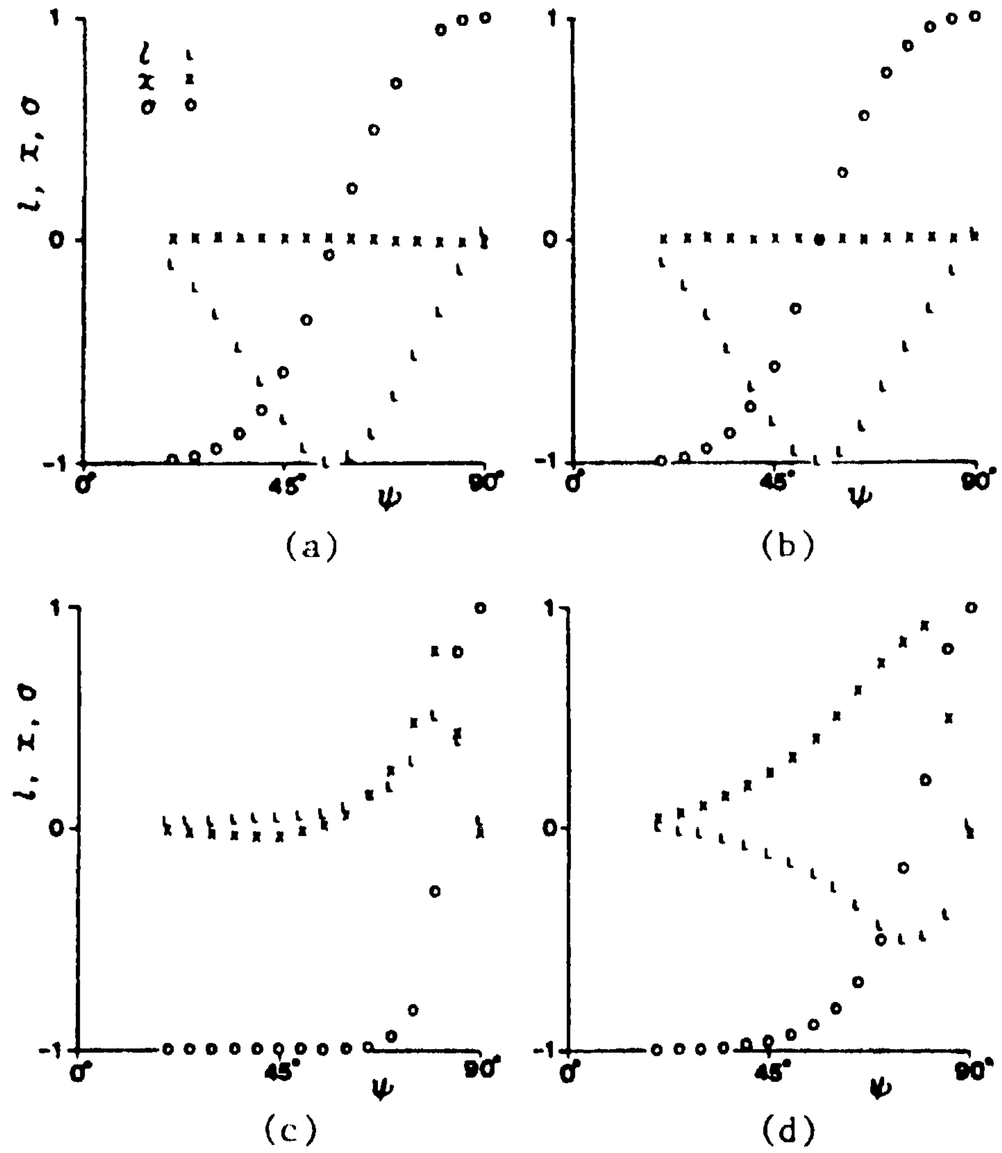


Fig.2 The polarizational characteristics measured from (a)acrylic board (b)polished paper (c)Al-coated mirror (d)chromed plate

feasible for use are obtained from the dielectrics such as the acrylic resin board and the polished paper, while not from the metals.

In this experiment we used Polaroid sheets of quarter-wave plate, linear and circular polarizers, a PIN photodiode detector and a DC-supplied incandescent lamp as a light source.

6. DISCUSSIONS

If ψ is closer to 0* or 90*, both P and V become too small to find a reliable value *of* (X . The normal can be determined for ψ =0*, since it coincides with the observed ray, while, not for ψ=90*by this method.

P, U and V can be theoretically derived from the refractive index, if known, by Fresnel equation. Otherwise it will be practical to get them by preliminary measurements as we did.

As far as a method of finding normals depends on specularly reflected lights, the light sources must be arranged so as to observe the specular ray from any investigating part of the surface. We, however, need not know their intensities, directions or positions.

The proposed method will be applicable to extracting geometrical parameters from glossy objects such as plastics, polished papers, lacquered items and so on. The objects have no need of uniform coloring, if their surface is coated with a dielectric.

It is under investigation to actually determine the normals from TV camera data.

ACKNOWLEDGMENT

The author is very grateful to Dr. Yoshiaki Shirai for suggesting to him to publish this work.

REFERENCES

[I] Horn, B. "Obtaining Shape From Shading Information, Winston,P.H.(ed.), The Psychology of Computer Vision." McGraw-hill, 1975.

[2] Clarke, D. and Grainger, J.F. "Polarized Light and Optical Measurement." Pergamon Press, 1971.

[3] Aspnes, D.E. and Hauge, P.S. "Rotating-compensator/ analyzer fixed-analyzer ellipsometer; Analysis and comparison to other automatic ellipsometers." J.Opt, Soc.ftm. 66:9 (1976) 949-954.

GRAMMATICAL INFERENCE ON THE BASIS OF LINEAR ENVIRONMENTS OF SYMBOL STRINGS

Yoshiyuki Kotani
Department of Information Science
Tokyo University of Agriculture and Technology
Koganei 2-24-16, Tokyo 184, Japan

A new method of grammatical inference is proposed. The inference procedure, receiving only positive sample sentences, produces a grammar. A notion of "bounded context" is introduced which represents linear environment of a string in sample sentences. The procedure consists of (1) calculating the bounded context of strings, (2) finding a pair of context-equivalent strings, (3) reducing the sample sentences by means of the pair, and (4) iterating (1) to (3) until there is no pair of context-equivalent strings in the reduced sample sentences. The output grammar is constructed from the reduced sentences and the pairs of strings obtained in (2). Sample sentences are grammatically correct with respect to the grammar inferred from the sentences. All of regular languages and a number of non-trivial context-free languages are inferred correctly by the method, when sufficient but a finite number of sample sentences are given.

1. INTRODUCTION

Grammatical inference is a problem of inferring a grammar which describes a language by means of a finite set of sample sentences of the language According to the development of formal linguistics, grammatical inference has recently received increasing attention as one of the most important information-processing problems. It serves as a tool for automatic acquisition of many sorts of grammatical descriptions which are dealt with in pattern recognition, graph theory, programming language design, information retrieval and other parts of artificial intelligence study.

We propose here a practical, systematic grammatical Inference procedure. It has the following properties. (1) The procedure can be applied to languages which are impossible of desciption by any regular grammar. (2) The procedure does not need any structural information of sample sentences. (3) The procedure works without feedback by teachers. (4) The procedure is efficient, i.e., enumaration of grammars is not employed.

In many effective methods of grammatical inference, production rules are often inferred on the basis of some supposed relationship between similar parts of sample sentences. We use a notion of "bounded context" which represents linear environment of a symbol string in sample sentences, and define an equivalence relation of strings by equality of the bounded context.

The next simple example explains our method. Suppose sample sentences are the strings
ab, aabb, aaabbb, ...
The two substrings 'ab' and 'aabb' appear in the same environment in the sentences, because they are put in the places denoted by [_) in the following [_], a[_]b, aa[_]bb,... . This means the two substrings are context-equivalent. Grammars are constructed from production rules which consist of pairs of context-equivalent strings and from reduced sample sentences. Here the production rule ab+aabb is inferred on the basis of the context equivalence. The reduction of sample sentences is carried into effect by replacing all 'aabb' in them by 'ab' repeatedly until no 'aabb' exists. Then all of the sample sentences are reduced to a single sentence 'ab'. We call it a reduced sample sentence. The inferred grammar is described by the triplet ({a,b},{ab+aabb}, {ab}). This description of grammar is a little different from that of ordinary formal grammars, but obviously grammars of this type can be transformed into ordinary context-sensitive grammars.

2. FUNDAMENTAL DEFINITIONS

We establish the basic concepts necessary for the expression of our procedure. First, the preliminary notations are provided. An alphabet is denoted by £, and we use a, b, c,... as symbols. Symbol strings are denoted by u, v,..., z except the empty string A. A set AB consists

of concatenation strings of elements of A and B. The notation A* is the closure for A. A language is a set of strings and denoted by L, L', etc. We call an element of a language a sentence. The length of a string x is denoted by $|x|$.

DEFINITION 2.1. The kℓ-bounded context of a string w in a language L is a set of pairs of strings

$$C_L^{k\ell}(w) = \{(last_k(x), first_\ell(y)) \mid xwy \in L\} ,$$

where the parameters k,ℓ are non-negative integers or infinity and $first_k(y)$ ($last_k(y)$) is y's leftmost(rightmost) substring of length k, if $|w| < k$, otherwise w. The context equivalence $\simeq_L^{k\ell}$ is defined as follows: $w \simeq_L^{k\ell} w'$ if and only if $C_L^{k\ell}(w) = C_L^{k\ell}(w')$.

DEFINITION 2.2. A symbol substitution grammar is a triplet G = (Σ, P, S), where Σ is an alphabet, P is a set of production rules which are of the form x→y and S is a finite set of strings called an initial set of sentences. In correspondence with ordinary formal grammars, we call it a type-1 ssg if x is not longer than y for any x, y in x→y ∈ P. The language L(G) generated by a grammar G=(Σ,P,S) is the set of strings generated from S by P, denoted by $g_P(S)$. More precisely, $g_P(S)$ is defined by

$$\{u \mid v \overset{*}{\underset{P}{\Rightarrow}} u,\ v \in S\} ,$$

where the relation of strings $v \overset{*}{\underset{P}{\Rightarrow}} u$ means that v = u or there is a sequence of strings $v_0, \cdots, v_n$ such that $v = v_0 \underset{P}{\Rightarrow} \cdots \underset{P}{\Rightarrow} v_n = u$ while $v \underset{P}{\Rightarrow} v'$ denotes that there exist x, y, w, w' which satisfy v = xwy, v'= xw'y, w → w' ∈ P.

Our procedure infers grammars of this type.

3. GRAMMATICAL INFERENCE PROCEDURE

Before presenting our procedure, we need some preliminaries. An order relation ◂ of strings is defined as follows: i) x ◂ y, if x is shorter than y, and ii) two strings of the same length follows the lexicographic order. The j-th non-empty string in the order is denoted by z(j).

Next we define an operator g_P^{-1} which is a mapping from the set of strings to itself. Intuitively, $g_P^{-1}(w)$ is the reduced sample sentence of w.

DEFINITION 3.2. Let a set of production rules P be $\{x_m \to y_m \mid m=1,\cdots,j\}$. The reduced string of w, denoted by $g_P^{-1}(w)$, is the string obtained by applying the production rules in P to w in such a manner that any y_m in w is replaced by x_m repeatedly, until no y_m remains in w. The operator can be applied to a set of strings.

Our grammatical inference procedure produces grammars by receiving as input a finite number of positive sample sentences. The system includes two arbitrary parameters: an order relation ◂ of symbols, and a pair (k,ℓ) of non-negative integers and/or infinities. The system outputs an ssg defined above. It is also type-1 because of the way of making production rules in the procedure.The procedure is described in the ALGOL-like form as follows:

```
S := L';  P :=φ ;  jj := 1 ;
for i := 1 step 1 until ∞ do  begin
  for j := jj step 1 until ∞ do begin
    if |z(j)| < max_{w∈S} |w| then go to OUT ;
    if C_S^{kℓ}(z(j)) ≠ φ  then
      for m := 1 step 1 until j-1 do
        if z(j) ≃_L^{kℓ} z(m) then begin
          P := {z(m)→z(j)} ∪ P ;
          S := g_P^{-1}(L') ;  go to NEXTSTRING
        end
  end;
  NEXTSTRING: jj := j+1
end;
out: print( (Σ, P, S) )
```

In the procedure, L' is the set of sample sentences, and S is the set of (partially) reduced sentences. The strings z(j) (j=1,2,···) are tested in order: I) If z(j) is longer than any string in S, the grammar is output, since no more rules can be found. II) Otherwise it is tested for every string z(m) shorter than z(j) whether or not z(j) is context-equivalent to z(m). If it is true, the procedure adds a production rule z(m)→z(j) to P, and recalculates S reducing the sample sentences L' by the new P.

EXAMPLE 3.1. Let sample sentences be ab, aabb, aaabbb, abc, abcc, abccc, abcccc, aabbc, aabbcc. When we choose parameters (1,1) and ◂ such that a◂b◂c, a type-1 ssg

$$G_{\simeq 1,1\blacktriangleleft}(L') = (\{a,b,c\},\{c\to cc, ab\to aabb\},\{ab,abc\})$$

is inferred. This grammar describes a language

$$L(G_{\simeq 1,1\blacktriangleleft}(L')) = \{a^i b^i c^j \mid i \geq 1,\ j \geq 0\} .$$

Table 1 shows the process of obtaining the ssg.

4. PROPERTY OF INFERRED GRAMMARS

We describe property of grammars inferred by our procedure and possibility of correct inference. There have been found the following facts:
i)Inferred grammars are consistent with the sample sentences. Precisely, sample sentences are grammatically correct with respect to the

TABLE 1. The Process of the Procedure for Ex.3.1

i	j	z(j)	$C_L^{i,j}(z(j))$	P; S
1	1	a	(λ,a),(λ,b),(a,b),(a,a)	
	2	b	(a,λ),(a,b),(b,λ),(b,b),(a,c),(b,c)	
	3	c	(b,λ),(b,c),(c,λ),(c,c)	c→cc ;
	4	aa	(λ,b),(λ,a),(a,b)	
	5	ab	(λ,λ),(a,b),(λ,c)	ab,aabb,
	:	:	:	aaabbb,
	12	cc	(b,λ),(b,c),(c,λ),(c,c)	abc,aabbc
2	13	aaa	(λ,b)	
	:	:	:	c→cc,
	46	aabb	(λ,λ),(a,b),(λ,c)	ab→aabb; ab,abc

grammar obtained from the sentences, that is, the language described by the grammar includes the set of sample sentences.

ii) The grammar inferred from sufficiently many sample sentences describes correctly the source language of the sample sentences. Precisely, if a source language L satisfies a condition, which we call $\overset{\infty\infty}{\simeq}$-inferability, then the language is inferred correctly from any complete sequence of L's sample sets by the use of any kℓ-bounded context in which k and ℓ are sufficiently large integers. A sequence of sets of sample sentences from a language L is complete iff for any w in L there exists a positive integer t such that for all t'>t $L_{t'}$, includes w.

iii) The class of $\overset{\infty\infty}{\simeq}$-inferable languages includes all of regular languages and many non-trivial context-free languages.

When the procedure in which non-bounded context is used produces a grammar, we call the input language is $\overset{\infty\infty}{\simeq}$-inferable. We also let source languages which are usually infinite sets be input, by giving an algorithm to calculate any description of bounded context.

EXAMPLE 4.1. Let us consider the following sequence of sample sets from a language $L = \{a^n b^n \mid n \geq 1\}$: $L_1=\{ab\}$, $L_2=\{ab,aabb\},\cdots$, $L_N=\{ab,\cdots,a^N b^N\},\cdots$. For example, by using 4,4-bounded context, we get the sequence of grammars: $G_1,G_2,\cdot\cdot,G_5,\cdots$, where $G_N=(\Sigma,\phi,L_N)$, (N=1,···,5) and $G_6=G_7=\cdots=(\Sigma,\{ab\rightarrow aabb\},\{ab\})$. We must notice that the grammar $G_6=G_7=\cdots$ describes the source language L correctly.

EXAMPLE 4.2. The following table makes us understand that most of inferred grammars are not so redundant and may reflect properties of their source languages. They are similar to the cfg's we may think of from the source language. These grammars can be inferred from any sample sets which includes all strings longer than a costant length.

TABLE 2. Result Grammars by $\overset{\infty\infty}{\simeq}$-Bounded Context for Various Languages

Input Languages	Inferred Grammars
{w | w consists of a,b and the number of a is odd}	({a,b},{b→aa,a→ababa, b→bb},{a,ab,ba,bab})
{w | w consists of a,b and includes aa}	({a,b},{b→bb,aa→aaa, b→bab,b→baab}, {aa,aab,baa})
{w | w consists of a,b, (the number of a)= (the number of b)}	({a,b},{ab→ba, ab→aabb},{ab})
the set of adequate sequences of parentheses	({<,>},{<→<<>,<→<><},{<>})
the set of reversed Polish formulas	({a,b,*,+},{a→b,+→*, a→aa+},{a})
{wcρ(w) | w consists of a,b} where $\rho(a_1a_2\cdots a_n)=a_n\cdots a_2a_1$.	({a,b,c}, {c→ aca,c→bcb}, {c})
the set of arithmetic expressions without redundant parentheses described by the cfg: S→S+T|T,T→T*F|F,F→(S+T)|a	({a,+,*,(,)}, {a→a*a,a→(a+a), a→a+a+}, {a,a+a})

5. DISCUSSION

We implemented the procedure, which was written in LISP, and tested real sample data (simple sentences of natural languages and feature sequences of letter patterns). The processing of the data helped us to recognize the following ideas: i) It is important to select good parameters k, ℓ in accordance with the application. We need too many sample sentences to obtain correct grammars, if k, ℓ are very large, ii) It may be interesting to change the parameters dynamically by observing the convergence of inferred grammars, iii) If noise is expected, we had better employ 'similarity' instead of equivalence, iv) Quick parser can be constructed, which may be simpler than that of cfg. It seems able to be deterministic, or not needed backtracking.

ACKNOWLEDGMENTS

The auther is grateful to Dr. S. Amari of Univ. of Tokyo for many helpful suggestions.

REFERENCES

[1] Biermann, A.W. and J.A. Feldman, "A Survey Results in Grammatical Inference", 4th Hawaii Int. Conf. System Sciences (1971) 31-53.
[2] Fu, K.S. and T.L. Booth, "Grammatical Inference: Introduction and Survey", IEEE Trans., vol.SMC-5,1,95-111 and 4,409-423 (1975).

COMMONSENSE KNOWLEDGE OF SPACE: LEARNING FROM EXPERIENCE

Benjamin Kuipers
Department of Mathematics
Tufts University
Medford, Massachusetts 02155

The TOUR model is a computational model of human commonsense Knowledge of large-scale space. It shows how observations arc assimilated into a description, from multiple perspectives, of the spatial environment. In this paper we propose a representation for sensory events at a level suited to this investigation, and an inference strategy by which these sensory events are assimilated into descriptions of spatial structure. We also discuss the states of partial Knowledge that occur during the learning process, and show that the representation exhibits graceful degradation of performance under resource limitations.

1. INTRODUCTION

Commonsense knowledge of space is a particularly interesting domain of human knowledge, because commonsense knowledge is a powerful and mysterious aspect of human thought, and because spatial knowledge is widely applied and distinctly different from linguistic knowledge. Knowledge of large-scale space (eg. the "cognitive map" of a city) is particularly accessible to study because the process of assimilating new information is constrained by the speed of physical travel, so states of partial knowledge are easily observable.

The TOUR model [1,2] is a computational model of the cognitive map, exhibiting a solution to the problem:

How can local observations acquired during travel be assimilated into a structure that permits answering route finding and relative position questions[9]

The model consists of a number of different representations for spatial relationships, and inference rules for creating descriptions in one representation from information in others. The different representations can be categorized:

1. Procedural. A description specifies a procedure for travel between two given places.
2. Network. Descriptions specify the topological connectivity between places and paths, and the local geometry of intersections.
3. Geometrical. Descriptions specify relative position vectors, and the coordinate frames in which they are defined.
4. Structural. Descriptions specify containment, the role of region boundaries, parallel relations, and rectangular grids.
5. Abstraction. Descriptions link representations of spatial features at different levels of detail.

The observational input to the TOUR model is simulated as a sequence of partially specified travel instructions containing only information that could be acquired from a local observation. All other information is supplied by the assimilation process [1,2]. While this permits subsequent acquisition of order and local geometry information (and thence the rest of the cognitive map), there remain easily observable phenomena for which no explanation can be stated in terms of this representation.

1. "I can't tell you how I get there. I just know what to do." In some circumstances, a person is able to use a stored route description for physical travel, but not to anticipate up-coming landmarks or to describe the route verbally.

2 Occasionally a person will have several different internal descriptions for the same place and not realize it. Upon noticing the identity, his cognitive map undergoes $ significant reorganization.

The problem is to find a representation for observational input that (a) refers to sensory events rather than previously acquired descriptions, (b) serves as an adequate base for further processing, and (c) can express mechanisms to explain a wider range of phenomena. The new representation is defined (Section 2), shown to interface correctly with the existing TOUR model (Section 3), and shown to exhibit graceful degradation of performance under severe resource limitations (Section 4).

2. TRIPLES

We assume two primitive descriptions of sensory events for interaction with the physical world: the view, or perceptual snapshot, and the action, causing or recording physical motion and simultaneously changing the current view. More formally:

Definition: A view (V) is the description produced by the perceptual system of the scene observed from one vantage point. While the internal structure of a view must certainly be very complex, for our purposes it is subject to only two operations:

1. Matching. Two views can be compared to see if they describe the same place. Matching is subject to errors, primarily in the direction of failing to detect an identity (e.g. due to differences in illumination, etc.).
2. Indexing. A view can be used as the key for storage and retrieval of other internal descriptions.

Definition: An action (A) is the description produced by the perceptual system of a physical motion within the environment. Such a motion always produces a change in the current view. There are two types of action description, each of which may provide the observed value of an associated magnitude:

1. (1RAVFL <distance-measure>)
2. (ROTATE <angte-measure>)

A TRAVEL action corresponds to motion along some unambiguously identifiable path within the initial view (i.e. no path-selection decisions are required during the motion). A ROTATE action corresponds to a change in the direction faced without translational motion. In addition to being observed, stored, and manipulated like any other description, an action can he executed to produce physical motion.

The observations corresponding to travel in an environment will be a sequence of alternating views and actions:

$$V_1\ A_1\ V_2\ A_2\ \ldots\ V_i\ A_i\ \ldots$$

The information provided by the views and actions and their temporal sequence is used by the assimilation processes to construct the cognitive map. We will summarize the spatial properties of the physical world by defining a function relating observed views and actions:

$$RESULT(V, A) = V'$$

if the action A, taken when V is the current perceived view, results in the view V'.

Definition: A sequence of observations is stored as an internal description whose basic element is the Triple (V A V'). As part of an internal description, a triple corresponds to:

a) the assertion that RESULT(V, A) = V'.

b) the imperative to do A, when V is the current view.

A triple can be partially specified: (V - -) and (V A -) are both meaningful triples, though they do not carry the same assertional or imperative weight as a full triple. A view or action can be inserted in an empty slot of a partially-filled triple.

Definition: A route description (RD) is a set of full or partial triples (V A V'). It is subject to two operations:

a) A new triple can be added to the set.

b) A triple in the set can be retrieved given its first element. (In LISP, (ASSOC V RD) ==> (V A V').)

The assimilation process that creates internal route descriptions from sequences of observations simply creates triples from temporally associated views and actions and collects them into a set. In humans, this process is extended over several traversals of the same route, and involves creating a set of partial triples and later filling in empty slots in those triples. The assimilation process works by using the current perceived view to retrieve a triple (if any) from the route description, then filling empty slots if possible. The state of partial knowledge represented by the route description passes through three qualitatively distinct phases of behavior as assimilation proceeds. Similar phases can be observed in human behavior.

1. Landmark Recognition. When the route desciption consists of partial triples of the form (V - -), each triple serves only to confirm that the given view does in fact occur along that route. It carries no further assertional or imperative value, and cannot be used for self-guided travel or for assimilation of information into the other perspectives in the cognitive map.

2. Self-Guided Travel. When the route description consists of partial triples of the form (V A -), each triple can be interpreted as an imperative. If sufficiently many are available to form a sequence

$$V_1\ A_1\ V_2\ A_2\ \ldots\ V_n$$

where $(V_i\ A_i\ -)$ is in RD, and $RESULT(V_i, A_i)=V_{i+1}$, and V_1 and V_n are the views at the endpoints of the route, then self-guided travel is possible along the route. However, the *assertions* $(V_i\ A_i\ V_{i+1})$ are not available in the internal description, so it is not possible to rehearse the route without physical travel, or to assimilate information to other perspectives.

3. Mental Rehearsal. When sufficiently many full triples $(V_i\ A_i\ V_{i+1})$ exist in RD to form the sequence

$$V_1\ A_1\ V_2\ A_2\ \ldots\ V_n$$

the entire route description can be reviewed from the mental description alone. Both the assertional and imperative import of the triples are available, supporting self-guided travel, rehearsal of the route without physical travel, and assimilation of information to other perspectives.

Insight into the contrast between phases 2 and 3 can come from seeing a route description as consisting of decision information and sequence information. In phase 2, the decision information is represented internally, while sequence information is represented only by the physical structure of the world. Thus physical travel is possible because information contained in the structure of the environment is accessible, but mental rehearsal is not. In phase 3, both kinds of information are represented mentally. A route description in phase 2 accounts for the first phenomenon cited in the introduction: "I can't tell you how I get there. I just Know what to do."

3. INTERFACE WITH THE TOUR MODEL

A complete set of fully specified triples can be used to create the partially specified route instructions previously assumed as input to the TOUR model [4]. Thus, the previously described assimilation processes can operate with the new form of observations. The two procedural route descriptions are nonetheless necessary, since the Triples are stated in terms of sensory events, while the instructions are stated in terms of internal descriptions of places and paths. Furthermore, the order of the steps in a route description cannot be observed trivially, but must be acquired in the process of assimilation from the associatively retrieved Triples to the sequentially ordered route instructions.

4. GRACEFUL DEGRADATION

One important property of a representation for commonsense knowledge is graceful degradation of performance under resource limitations: as conditions get bad, performance should become gradually worse, rather than failing catastrophically. It has its inverse in learning: as more information is acquired or more assimilation takes place, performance should gradually improve, rather than remaining constant until a large threshhold is passed, then improving dramatically. This property is fundamental to McCarthy's Advice Taker [7], stated in terms of learning. Graceful degradation was explicitly presented as a design goal by Marr [6], and used as an explanatory device for several psychological phenomena by Norman and Bobrow [8]. Kuipers [3] presents a specific hierarchy of catastrophes of different levels of severity, and discusses the way different representations achieve graceful degradation of performance under resource limitations.

In this section we reformulate the Triples representation as a database accessible to three associative retrieval functions, to see the level of performance resulting from various kinds of errors. We show that the level of performance degrades gracefully as errors become more serious and frequent.

Information in the route description RD is used by calling the retrieval functions. Information is assimilated by indexing it under the appropriate key and retrieval function.

$$\text{EN-ROUTE}(V, RD) = \begin{cases} T & \text{if } V \text{ is observed on the route;} \\ nil & \text{otherwise.} \end{cases}$$

$$\text{ACTION-AT}(V, RD) = \begin{cases} A & \text{if } A \text{ is known to be the action to perform when } V \text{ is observed;} \\ nil & \text{otherwise.} \end{cases}$$

$$\text{NEXT-VIEW}(V, RD) = \begin{cases} V' & \text{if } V' \text{ is known to be RESULT}(V, \text{action-at}(V, RD)); \\ nil & \text{otherwise.} \end{cases}$$

Travel along a route corresponds to a sequence of alternating views and actions:

$$V_1\ A_1\ V_2\ A_2\ \ldots\ V_n$$

such that RESULT(V_i, A_i) = V_{i+1}, for i=1,2, ... n-1. (Recall that RESULT summarizes a property of the physical world, not of a stored description of it. The definitions of the retrieval functions allow us to define precisely the different levels of performance we observed above for the Triples representation.

1. Landmark Recognition: (each landmark is familiar)
   EN-ROUTE(V_i, RD) = T, for i=1,2, ... n-1.
2. Self-Guided Travel: (can use RD to get from A to B)
   EN-ROUTE(V_i, RD) = T,
   ACTION-AT(V_i, RD) = A_i, for i=1,2, ... n-1.
3. Mental Rehearsal: (can use RD for assimilation)
   EN-ROUTE(V_i, RD) = T,
   ACTION-AT(V_i, RD) = A_i,
   NEXT-VIEW(V_i, RD) = V_{i+1}, for i=1,2, ... n-1.

The level of performance provided by a given route-descnption can be intermediate between these thresholds, with higher performance along some parts of the route than can be sustained along its length.

It is clear from the definitions given above that a failure of one of the associative retrieval functions results in a drop, for that point in the route, to the next lower level of performance. Thus, we can easily determine the level of performance resulting from any given failure. Note that an error on retrieval causes a failure of a particular call to a retrieval function, which may succeed on a later attempt. An error during storage (auses failures of all subsequent attempts to retrieve that information until it is observed again and successfully assimilated. Thus errors during storage *are* intrinsically more serious than errors during retrieval.

We can give intuitive descriptions of the result of a failure as follows. Notice that none of these errors crash the system or propagate to invalidate other stored information. A failure of NEXT-VIEW provides the mildest possible catastrophe. A failure of ACT10N-AT is more serious, and EN-ROUTE is the most critical of all.

EN-ROUTE: We're lost. We may or may not be on the desired route. The current travel goal must be abandoned and special processing invoked to find familiar territory. The remainder of the route-description is still intact, however, and subsequent observations can rebuild the description.

ACTI0N-AT: We don't know what to do. While still on the route, explicit guidance is needed to reach the next landmark, so the current travel goal *need* not necessarily be abandoned. The remainder of the route description still exists, and can be rebuilt.

NEXT-VIEW: We can't anticipate the next landmark. The current trip is unaffected, but no further deductions are possible. An attempt at mental rehearsal must be abandoned. Assimilation of the next-observed view in a physical trip will repair the failure.

We can see from the definitions of lhe levels of performance how different failures affect performance. To complete an argument for graceful degradation of performance we must 'how that the most likely errors cause the mildest catastrophes. This demonstration proceeds by considering first the levels of performance encountered in the normal learning process, second the effects of errors during storage, and finally the effects of errors during retrieval. This discussion can be found in [4]. Thus we put a precise meaning to the claim that the Triples representation exhibits graceful degradation of performance under resource limitations.

REFERENCES

[1] Kuipcrs, B. J. Representing Knowledge of Large-Scale Space. MIT Artificial Intelligence Laboratory TR-418, 1977. (Doctoral dissertation, Department of Mathematics, MIT).

[2] Kuipers, B. J. "Modelling spatial knowledge." *Cognitive Science 2:2 (1978)* 129-153.

[3] Kuipcrs, B. J. "On representing commonsense knowledge." In N. V. Findler *(16.)*, *Associative Networks — The Representation* and *Use of Knowledge by Computers.* New York: Academic Press, 1979.

[4] Kuipers, B. J. "Commonsense Knowledge of Space: Learning from Expenence." Tufts University Cognitive Science Working Paper 1, 1979. (An expanded version of this paper.)

[5] Lynch, K. *The Image of the City.* Cambridge, MA: MIT Press, 1960.

[6] Marr, D. "Early processing of visual information." Phil. Trans. Roy. Soc. Lon. B 275(942): (1976) 483-524. (Also MIT AI Memo 340.)

[7] McCarthy, J. "Programs with common sense." In M. Minsky (Ed), *Semantic Information Processing.* Cambridge, MA: MIT Press, 1968.

[8] Norman, D. A. and Bobrow, D. G. "On data-timited and resource-limited processes." *Cognitive Psychology 7* (1975) 44-64.

LISP ACTIVITIES IN JAPAN

Toshiaki Kurokawa
Information Systems Laboratry,
TOSHIBA R & D Center
Kawasaki 210 Japan

Lisp is one of the major programming languages for AI research. This paper surveys the current status of Lisp activities in Japan. First, Lisp systems developed in Japan on commercial machines are reviewed. Next, projects for Lisp machines are reported with commentary. The current problems and future prospects for the Lisp developments are discussed. We propose that the general educational activities and developments of more powerful portable systems for this language are necessary.

0. INTRODUCTION

Lisp [1] is now one of the major programming languages for AI software development. Some institutions in Japan are said to have chosen their computer systems on the criteria if they have good Lisp systems to promote the AI research. In other words many activities in AI field in Japan indicate that many Lisp systems for AI research have been developed in this country.

In this Paper, Lisp systems developed in Japan are surveyed. First, Lisp systems for commercial machines are listed. Then, the systems for special hardware (i.e. Lisp machines) are reported, some of which are now being developed

Some problems on Lisp developments are discussed with suggestions for future prospects.

1. LISP SYSTEMS ON COMMERCIAL MACHINES IN JAPAN

No reports have been available on Japanese Lisp systems, except for the reports on the Lisp contests in Japan [2], one of which was held in April 1978 by the working group on symbol manipulation under Information Processing Society of Japan (Kigoshori Kenkyuukai). In this contest about 20 Lisp systems participated including 3 overseas systems. The primary purpose of the contest was to know the current status of the art of Lisp systems developed in Japan. However, there existed some problems, as Eric Sandewall pointed out [2], that the benchmark programs for the contest were not appropriate especially from the viewpoint of AI applications, and that some systems were not included in the contest.

The reported items are as follows: system's name, authors, their institution, dates for first and current version installed, primary purposes, its resemblance, its service, the number of users, the number of Lisp system functions, its top level function, its data types available, the number of cells, stack, compiler status, program size, its hardware systems, memory size, the number of general registers, CPU cycle time, time for add (memory to register), operating systems, version-up schedule and other information. The report also contains the execution time of several test programs, which are cited at many places for performance evaluation.

Tables 1,2,3, and 4 contain the list of the existing Lisp systems in Japan. Table 1 lists the portable systems, Table 2 big and medium, machines, Table 3 for small machines and Tabled shows the micro-processor based systerns.

The Tables contain only the system's name, authors, their institutions, hardware systems, and the main features. The imported Lisp systems such as Interlisp, UCL-Lisp, Standerd-Lisp and Maclisp are not included.

The followings should be noted:

1) Most of the systems have been recently implemented (during 1975-1978.)

2) Portable Lisp systems are implemented on

Tables for Lisp Systems on Commercial Machines in Japan

System name	Author	Institution	Hardware	Features
Table 1. Portable Lisp System				
HLISP	Y.Kanada et al.	Univ. of Tokyo	HITAC-8800/8700 FACOM-230/75 IBM-370/168	Written in Fortran
LISP/P	T.Chikayama	Univ. of Tokyo	FACOM-230-38	Written in Pascal
TLICS	M.Suzuki et al.	Tohoku Univ.	ACOS-6	Written in Fortran
Table 2. Lisp Systems on Big Machines				
LISP 1.5	H.Kumura	Okayama Univ.	MELCOM COSMO-700	TSS
OLISP	H.Yasui et al.	Osaka Univ.	ACOS-77 System 800 System 900	
PETL	M.Tsukamoto	ETL	MELCOM COSMO-700II	TSS
TOSBAC-5600 LISP 1.9	T.Kurokawa	Toshiba Co.	TOSBAC-5600 ACOS-77	TSS
VS-DLISP 1.5	H.Imura	Doshisha Univ.	HITAC 8350	
Table 3. Lisp Systems on Small Machines				
FLISP	N.Abe et al.	Osaka Univ.	HP2108A	Hardware modified
HITAC-10 LISP	Y.Nakamura	Keio Univ.	HITAC-10	Hardware modified
KLISP-11	M.Nakanishi et al.	Keio Univ.	PDP-11	
LIPQ LIPX	I.Takeuchi & H.Okuno	ECL of NTT	PDP-11/55	Parallel 4-Cell
LISP 1.7	K.Hiramatsu et al.	Kyoto Univ.	FACOM U-200	Hardware modified
LISP 38	Y.Ohta	Nagoya Univ.	FACOM 230-38	
NIT LISP-U	T.Niwa et al.	NIT	FACOM U-200	
Table 4. Lisp Systems on Micro-computers				
LISPM	T.Chikayama	Univ. of Tokyo	INTEL 8080	

Fortran or on Pascal.

3) Many systems are implemented on small computers, some with hardware modifications for commercial machines.

2. LISP MACHINE PROJECTS IN JAPAN

At the time of April 1979, there are 6 Lisp machine projects reported in Japan. The implementors, machines, principal purposes and progress atatus are listed in Table 5.

There are many kinds of approaches for Lisp machines projects: FLATS by Goto[3] is a special purpose machine (Formula manipulations). Fortran and Algol can also run on that machine. ECL(Electro Communication Lab. of NTT) machine [4] is made to experiment the parallel garbage collection. Although the feasible result was not obtained, it has provided Lisp engineers with the valuable information. ALPS/1[5,6] is one of the earliest Lisp machine in the world, and the significant revision is now considered. NK3 made by Kyoto Univ. [8,9] utilizes 32-bit mini-computers and is used for large natural language processing programs. ETL(Electro-technical Laboratory) [10,16] is trying to make a Lisp machine on a chip. (It will really be on a few chips.) They use LSI microprocessors produced by a national project, PIPS. (Pattern Information Processing System)

A Kobe machine[7,11,12,13,] is rather in expensive but is very fast. Its interpreter runs as fast as the compiled Lisp programs on a big machine, although its address space is rather limited (16 bit). Keio Machine[14,15] is a Lisp system on an experimental multi-microprocessors.

3. PROBLEMS AND FUTURE PROSPECTS

The author would like to point out the following two problems: one is for education and the other is for computing facility.

i) Education As Lisp is (seemed to be) tied up with the AI programs, those who do not study AI, have a tendency to neglect Lisp for the

Table 5. Lisp machine projects in Japan

Implementor (Institution)	Machine	Purpose	Status (April 1979)
E.Goto et al. (Institute of Physical and Chemical Research)	FLATS	Formula manipulation	Design
Y.Hibino (ECL of NTT)	Tosbac-40L	Parallel GC	Working/Data Gathered
M.Ida et al. (Aoyama Gakuin Univ.)	ALPS/I using Intel 8080	General (esp. formula manipulation)	Working from 1977
M.Nagao et al. (Kyoto Univ.)	NK-3 using Interdata 8/32	AI esp. for natural language processing	Working/Data Gathered
T.Shimada & Y.Yamaguchi (ETL)	PULCE	PIPS	Design Finished
K.Taki et al. (Kobe Univ.)	using Am 2903	General	Working/Data Gathered
T.Usuki et al. (Keio Univ.)	Multi-micro computers	General	Working

programming instruction. From the author's viewpoint, however, we should promote the Lisp programming instructions not only for the wide activities of AI, but for the better understanding of the computer science. We also need a good organization for Lisp instructions.

ii) Computing facility-Interactive programming and cooperative program development are two vital elements for the present software development, especially on AI. Utilization of time-sharing services have been the easiest way to experience them. But TSS are rare in Japan even today. In fact, some of the Lisp machine projects are motivated in order to have an interactive facility for Lisp. Even if you are doing research by yourself in your own style, you should promote to have a center machine with time-sharing services and should provide a network facility to communicate with others.

The author would like to discuss some future prospects for Lisp developments in Japan.

i) Although many Lisp machines have been developed good portable Lisp systems are indispensable to meet a wide variety of users and their applications.

ii) Memory-neck problems will be solved by the recent 32-bit architectures. A portable Lisp on 32-bits hardware(eg. Tnterdata 8/32, Tosbac 7/70, and DEC VAX-11) and a special purpose Lisp machine will be good competitors.

ill) Database facilities arc necessary for large AI applications.

iv) Parallel processing facilities are needed, since the techniques for parallel processings are now accumulated. The AI applications such as many worlds hypothesis are utilizing parallelisms.

v) For the AI applications in Japan, most users need the Japanese language interfaces. Thus, Kana and Kanji characters should be handled by Lisp. The general pattern handling facilities are also preferable.

vi) Modularity of Lisp programs encourages to involve special features useful for AI, eg. Pattern matching, context switelling, etc.

vii) Recently Lisp has become more popular than before. It Is studied now in the software engineering fields as well as AI and the mathematical computer science. However, it is still a research language(at least they say), and its application and education are rather limited.

As AI applications are widely recongnized as feasible, Lisp implementors should note the practical aspects of the Lisp language. From the author's viewpoint, Lisp may be regarded as a system language for high-level application programs.

REFERENCES

[I] J.Allen: Anatomy of Lisp, McCrawHill, 1978.
[21 I.Takeuehi ed.: Report oi the 2nd Lisp contest, Working paper of Kigoshori-Kenkyuukai, IPSJ, Aug. 1978. The report of the 1st contest is included in the Kigoshori Symposium Report, IPSJ, July 1974.
The followings are on Lisp machines. Except (3,6,9,] are in Japanese.
[3] E.Goto et al.: Report of the FLATS project Vol.1, The Institute of Physical and Chemical Research, Oct. 1978.
[4] Y.Hibino: A Parallel garbage collection algorithm and its application to Lisp, IECE Working paper, Nov. 1978.
[5] M.Ida et al.: Performance evaluation of ALPS/1 Lisp machine, IPSJ Working paper, Oct. 1977.
[6]————————: Some Topics from ALPS, IPSJ Working paper, Mar. 1979.
[7] Y.Kaneda and K.Taki: An Experimental Lisp Machine, in this 6th IJCAI Proceedings.
[8] M.Nagao et al.: Design and implementation of Lisp-machine NK-3, Kyoto Univ., 1977.
[9]————————: Lisp Machine NK3 and Its Performance Evaluation, IPSJ Working paper, March 1979. also in this 6th IJCAI Proceedings.
[10] T.Shimada et al.: A Lisp Machine and its evaluation, Trans. IECE, Vol.59-D, No.6, June 1976 pp.406-413.
[II] K.Taki et al.: Implementation of Lisp Machine, IPSJ Working paper, Aug. 1978.
[12]---------- : Microprogrammed Interpreter of the Experimental LISP Machine, IECE Working paper, Nov. 1978.
[13]————————— : An Experimental Lisp Machine and its Evaluation, IPSJ Working paper, March 1979.
[14] T.Usuki et al.: Implementation of LISP machine on multi-processor, IPSJ Conference Proceedings, 1978, pp.27-28.
[15]————————: LISP Machine Implementation on Multi-microprocessor System, IECE Working paper, Nov. 1978.
|16] Y.Yamaguchi and T.Shimada: Dynamic Measurements of LISP Programs on a Virtual Machine, Trans. IECE, Vol. 61-1), No.8, Aug. 1978.

Rediscovering Physics With BACON.3*

Pat Langley
Department of Psychology
Carnegie-Mellon University
Pittsburgh, Pennsylvania 152:3

BACON.3 is a production system that discovers empirical laws. The program uses a few simple heuristics to solve a broad range of tasks. These rules detect constancies and trends in data, and lead to the formulation of hypotheses and the definition of theoretical terms. BACON.3 represents data at varying levels of description, where the lowest have been directly observed and the highest correspond to hypotheses that explain everything so far observed. The system can also run and relate multiple experiments, collapse hypotheses with identical conditions, ignore differences between similar concepts, and discover and ignore irrelevant variables. BACON.3 has shown its generality by rediscovering versions of the Ideal gas law, Kepler's third law, Coulomb's law, Ohm's law, and Galileo's laws for the pendulum and constant acceleration.

1. INTRODUCTION

Centuries ago, early physicists such as Kepler and Galileo began to discover laws that described the physical world. In this paper I describe BACON.3, a program that rediscovers a number of these laws. I begin with an example of how one might discover the ideal gas law. Next, I consider the program's representation of laws and data. Following this, I examine the heuristics used by the system. I conclude with a discussion of BACON.3's generality.

MOLES	TEMP.	PRESSURE	VOLUME	PV
1	300	300000	0.008320	2496.0
1	300	400000	0.006240	2496.0
1	300	500000	0.004992	2496.0
1	310	300000	0.008597	2579.2
1	310	400000	0.006448	2579.2
1	310	500000	0.005158	2579.2
1	320	300000	0.008875	2662.4
1	320	400000	0.006656	2662.4
1	320	500000	0.005325	2662.4

TABLE 1. DATA OBEYING THE IDEAL GAS LAW.

2. AN EXAMPLE: THE IDEAL GAS LAW

The general law for ideal gases may be stated as $pV/nT = R$, where p is the pressure, T is the temperature in degrees Kelvin, n is the quantity of gas in moles, V is the volume, and R is the constant 8.32. Table 1 shows some of the data gathered by varying p and T when n is 1. Note that as the pressure increases, the volume decreases. Since a function of the form $pV = k$ would lead to such a trend, one might calculate the values of the product pV. In fact, the values of this term are different constants for different values of n and T, as shown in Table 2.

MOLES	TEMP.	PV	PV/T
1	300	2496.0	8.32
1	310	2579.2	8.32
1	320	2662.4	8.32
2	300	4992.0	16.64
2	310	5158.4	16.64
2	320	5324.8	16.64
3	300	7488.0	24.96
3	310	7737.6	24.96
3	320	7987.2	24.96

TABLE 2. SECOND LEVEL SUMMARY OF THE GAS LAW DATA.

Upon examining the first three rows of Table 2, one might note that the values of pV and the temperature increase together. Moreover, this relationship is linear with a slope of 8.32; since the intercept is 0, the slope term can be represented as pV/T. When the number of moles is varied, other linear relations are found with different slopes and the same intercept, as shown in Table 3.

MOLES	PV/T	PV/NT
1	8.32	8.32
2	16.64	8.32
3	24.96	8.32

TABLE 3. THIRD LEVEL SUMMARY OF THE IDEAL GAS DATA.

Finally, one might notice that the values of pV/T increase along with the number of moles. When the term pV/nT is defined and found to have the constant value 8.32, one has arrived at the ideal gas law.

*This research was supported in part by Grant HES75-22021 from the National Science Foundation, in part by NIMH Grant MH07722, and in part by ARPA Grant F44620-73-C0074. I would like to thank H. A. Simon, Eric Johnson, Marshall Atlas, Marilyn Mantei, Doug Lenat, and Robert Neches for discussions leading to the ideas in this paper.

3. LEVELS OF DESCRIPTION

As the reader may have guessed, BACON.3 rediscovers the ideal gas law in a manner much like the above. BACON.3 uses strategies very similar to those used by its precursor, BACON. 1 [1] yet BACON.3 can discover the gas law while its predecessor could not. BACON. 1 made a sharp distinction between the data it had observed and the hypotheses which explained those data. BACON.3 blurs the distinction between data and hypotheses by allowing various levels of description. In the new program, regularities in one level of description lead to the creation of a higher level of description.

Like the earlier program, BACON.3 is implemented as an OPS2 [2] production system. BACON.3 shares a number of heuristics with BACON. 1, though these have been generalized to deal with any level of description. Like BACON. 1, the new system defines theoretical terms like pV, pV/T, and pV/nT to describe its data parsimoniously. These heuristics and others are discussed in the following section.

4. THE HEURISTICS OF BACON3

The BACON.3 program consists of some 86 0PS2 productions. These can be divided into seven major sets, which I discuss below. The first four sets are held in common with the BACON. 1 system; the final three *are* additions required by the new representation and the tasks BACON.3 must handle.

The first set of productions is responsible for gathering directly observable data. Seven of these are responsible for gathering information from the user about the task to be considered. The remaining 10 productions gather data through a standard factorial design, varying first one independent term, then another.

The second set of 16 productions is responsible for noting regularities in the data collected by the first set. BAC0N.3's constancy detectors can deal with either symbolic or numerical data, and lead to the creation of higher level descriptions. The basic constancy detector is a simple restatement of the traditional inductive inference rule for making generalizations. Similar rules add conditions to newly created hypotheses. The program has primitive facilities for dealing with near constancies in noisy data; this is accomplished by redefining the LISP equal function to ignore small differences.

BACON.3'* trend detectors operate only on numerical data. Some of these notice increasing and decreasing monotonic trends between variables. These heuristics work in conjunction with other trend detectors that further analyze the data. One of these applies if the elope is constant, and leads to the definition of two new theoretical terms, the slope and the intercept of the line relating the two variables. Otherwise, a new term is defined as the product or *the* ratio of the variables, depending on the numbers involved.

After a theoretical term has been defined, a third set of 3 productions calculates the values of this term. Once calculated, these values are fair game for the regularity detectors. Defined terms are not distinguished from direct observables when noting regularities; it is this recursive ability to apply the same heuristics to concepts of increasing complexity which gives BACON.3 its power.

Before calculating the values of a new theoretical term, BACON.3 must make sure that the term is not equivalent to an existing concept. Accordingly, a fourth set of 22 productions decomposes the new term into its primitive components. If the definition of the new term is identical with an existing definition, the term is thrown out and other relations are considered.

Suppose BACON.3 has defined two intercept concepts for the ideal gas data. The values of the first, $intercept_{pv,t,1}$ are 0 when the number of moles is 1, while the values of the second, $intercept_{pv,t,2}$ 0 when the number of moles is 2. One would like BACON.3 to generalize at this point, but because the two intercepts are different terms, the constancy detector cannot be applied. BACON.3 notes such similar terms, and defines an abstracted term which ignores their differences. The values of the new term are copied from the originals, and the constancy detector is applied to the new data.

BACON.3 generates different descriptions to summarize different constancies. If two descriptions are found to have identical conditions, they are combined into a single structure; only 3 productions are devoted to this process. Once this has happened to a number of descriptions, the values of the dependent terms can be compared and regularities may emerge.

In rediscovering Galileo's law for pendulums, BACON.3 begins by varying the weight of the suspended object and the initial angle of the string. These variables are irrelevant to the period of the pendulum, but this is not obvious from the outset. BACON.3 draws on a final set of 8 productions for noting irrelevant terms. These modify the data gathering scheme so the values of the irrelevant terms are no longer varied.

Ideal Gas Law	$pV/nT = k_1$
Kepler's Third Law	$d^3(s - k_2 t)^2 = k_3$
Coulomb's Law	$Fd^2/q_1q_2 = k_4$
Galileo's Laws	$dP^2/L^2 = k_5$
Ohm's Law	$Td^2/(lc - k_6c) = k_7$

Table 4. Equations discovered by BACON.3

5. THE GENERALITY OF BACON.3

BACON.3 successfully rediscovered the five laws summarized in Table 4. These equations do not entirely do justice to BAC0N.3's discoveries. Along with omitting the conditions placed on some of the laws, only one equation is shown for each task, while a number were formulated for some. However, they do suggest the diversity of the laws the program generated from its data. The ability to define ratios and products leads to terms taken to a power, as in Coulomb's and Galileo's laws. The abstraction strategy allows the use of linear combinations in new terms, as In Ohm's law. Taken together, these two strategies lead to a version of Kepler's third law, in which the square of a linear combination plays a role.

Table 5 presents statistics on the relative complexity of the laws found by BACON.3. Three measures *are* used - the number of productions fired, the average size of working memory, and the average size of the conflict set. These measures are not completely correlated, but one trend is clear; the discovery of Ohm's law required much more computation than did the other tasks.

	IDEAL GAS	KEPLER	COULOMB	GALILEO	OHM
PRODS. FIRED	1203	1119	1297	1150	4593
WORKING MEMORY	55.9	111.0	119.7	66.6	241.9
CONFLICT SET	6.2	7.3	6.8	10.9	8.3

TABLE 5 RELATIVE COMPLEXITY OF THE FIVE TASKS

Closer analysis reveals some of the reasons for the complexity of this task. Table 6 presents some characteristics of the problem spaces for the five tasks. Since more terms were related in this law, more levels of description were needed to arrive at it. Moreover, two completely independent sets of laws were discovered In this run. These required a large number of theoretical terms, and a still greater number of terms which were considered and rejected.

	IDEAL GAS	KEPLER	COULOMB	GALILEO	OHM
SIZE OF DESCRIPT.	4	5	4	4.5	5
NUMBER OF DESCRIPT.	27	18	27	23	54
NUMBER OF LEVELS	4	3	4	3	5
DEFINED TERMS	8	17	16	9	31
REDUNDANT TERMS	2	1	12	7	84

TABLE 6. PROBLEM SPACE CHARACTERISTICS FOR THE TASKS.

How general were the heuristics of BACON.3? The heuristics borrowed from BACON. 1 were used in each of the tasks. Table 7 shows those tasks in which each of BACON.3'5 new heuristics were used. Most of the heuristics were used in multiple tasks, suggesting considerable generality for the rules. The single exception is misleading, since irrelevant variables could be added to each of the tasks.

	IDEAL GAS	KEPLER	COULOMB	GALILEO	OHM	TOTAL
GNORING DIFFS.	X	X	X		X	4
RRELEVANT VARS.				X		1
:OLL. DESCRIPT.	X	X	X	X	X	5
)IFF. EXPERS.		X		X		2

TABLE 7 USE OF BACON 3'S HEURISTICS

A general discovery system should be sensitive to the order in which it observes its data, but robust enough to arrive at equivalent laws regardless of the order. In the Galilean run reported in Table 4, the irrelevant variables weight and angle were varied first and immediately found to have no influence. In a second run where these variables were varied last, identical laws were eventually reached but the computation required was greater.

Modifying the order of relevant variables also affected the behavior of the system. In a second run on the ideal gas law, the number of moles was varied first, followed by the temperature, followed by the pressure. In the initial run, three major theoretical terms were generated - pV at level 1, pV/T at level 2, and pV/nT at level 3. In the second run, a different path was taken to the same conclusion - V/n was defined at the first level, V/nT at the second level, and pV/nT at the third level.

In summary, BACON.3 is a production system that can rediscover a number of laws from the history of physics. The system draws on a small number of strategies for finding regularities, defining terms, ignoring differences, collapsing hypotheses, and determining Irrelevant variables. Like its predecessor, BACON.3 is a general discovery system. One piece of evidence for this claim is that BACON.3 solved five different tasks using the same small set of heuristics. A second point in favor of BACON.3's generality was its ability to resolve tasks when the data were gathered in different orders. In conclusion, the progress to date has been encouraging, and suggests that more interesting discoveries lie ahead.

[1] Langley, P. BACON. 1: A general discovery system. In Proc. CSCSI, 1977, 173-180.

[2] Forgy, C. and McDermott, J. 0PS2 Manual. Pittsburgh, Pa.: Carnegie-Mellon University, Department of Computer Science, 1977.

FAILURE PROCESSING IN A SYSTEM FOR DESIGNING COMPLEX ASSEMBLIES

Jean-Claude LATOMBE

IMAG, BP 53X, 38041 GRENOBLE cedex
France

We believe that the capacity to recover intelligently from intermediate failures and to learn from these failures is an essential ingrrdient of attempting to solve complex real-world problems. Intelligent failure recovery requires the problem-solver not only to diagnose the causes of its failures, but also to circumscribe carefully the propagation of recovery updating. Learning from failures requires that the problem-solver remember the conditions *of* its failures in such a way that it will not recreate these conditions. In this paper, we describe methods providing such capabilities. We have implemented and experimented with these methods in a rule-based system used for designing complex assemblies such as electromechanical equipment.

1. INTRODUCTION

A traditional difficulty in automating the problem-eolving process is that of dealing with the combinatorial explosion of the search space. Another difficulty is the difficulty of handling large amounts of domain-dependent knowledge. In this paper, we present new methods for coping with both these difficulties simultaneously.They combine a search representation based on the distinction between control data and logical data, and procedures working with this representation that facilitate selective recovery and learning from failures. The recovery procedure diagnoses the causes of the failures in order to perform selective backtracking and the representation makes it possible to circumscribe carefully the propagation of recovery updating. This combination turns out to be particularly important in reducing unnecessary operations on knowledge.

We have implemented and experimented with these methods in a rule-based system named TROPIC, a problem-solving tool for designing complex assemblies such as electromechanical equipment4]. Problems with which this system can deal consists of generating models of devices. TROPIC is not restricted to a particular discipline, and we have used it to design electric power transformers among other applications [5]. This application required us to encode large amounts of special-purpose knowledge (about 350 rules) related to such fields as electricity, electromagnetism, mechanics and thermodynamics. Only a few aspects of TROPIC are presented hero.More detailed descriptions can be found in [5] & [6].

Although v;e claim originality for the methods to be discussed in this paper, the fundamental issues underlying these methods are not new in Artificial Intelligence. For example, the distinction between control data and logical data has been an important aspect of N0AH [9]; failure diagnosis and learning has been studied in EL/ARS [10]. Relations of TROPIC to these works will be discussed in Section 8.

2. OVERVIEW OF THE PROBLEM-SOLVING MODEL USED BY TROPIC

The problem-solving model used by TROPIC is based on the concurrent execution of a problem-solving process, which we call "task-reductive synthesis" and a deduction process, which we call "fact-deductive analysis" :

1) Task-reductive synthesis consists of assuming facts about a potential device (for example the type of components, the connection between these components and the values of various parameters), while partitioning a given "initial task" into successive subtasks. For example,if the problem is to design a transformer trl, a step of this process might be to assume that parts of trl are a magnetic circuit mc1 and an electrical circuit eel, and to reduce the task "design transformer trl" into three subtasks, "design magnetic circuit mc1", "design electrical circuit eel" and "select a cooler for trl".

This work was carried out under contract IRIA/SKSORT n° 77052

One can regard the facts that are assumed when a task is reduced as a solution to this task at some level of abstraction, and task-reductive synthesis as a process of adding successive detail to a description. The system may have to choose among alternative ways of partitioning the same task, each of them corresponding to different assumptions. For example, the task "select a cooler for tr1" may be reduced to the empty set by assuming that tr1's cooler is air ; it may also be reduced to the subtask "design tank tk1" based on the assumption that tr1's cooler is oil, and that it is contained in a tank tk1. As we will see next, some combinations of choices may produce contradictions.

2) Fact-deductive analysis consists of deducing new facts from the assumed facts and of controling the consistency of the set of both assumed and deduced assertions. Deductions are made by using such kinds of knowledge as constraints and approximations to physical laws. For example, the design constraints "if air is the cooler of a transformer, then the voltage between the terminals of any coil in this transformer must be lower than 3KV ; otherwise it must be lower than 5KV" may be used to deduce the fact "the voltage between the terminals of c11 is lower than 5KV" from the facts "the cooler of tr1 is oil","c11 is a part of tr1" and "c11 is a coil". Contradictions among assumed and deduced facts produce failures, from which the system recovers by resuming task-reductive synthesis at a previous decision point. For example, the facts "the number of turns of c11 is 350","the voltage per turn of c11 is 18V","the voltage between the terminals of c11 is lower than 5KV", and "the voltage between the terminals of c11 is equal to the number of turns of c11 times the voltage per turn of c11" are contradictory facts.

A problem is specified as an initial task, a set of synthesis rules to be used for task-reductive synthesis, and a set of analysis rules to be used for fact-deductive analysis. It is solved when the initial task has been reduced to the empty set in such a way that no contradiction derives from the assumptions generated by this reduction.

3. TASK-REDUCTIVE SYNTHESIS

3.1. Form and use of synthesis rules

Knowledge for task-reductive synthesis is represented in TROPIC by means of synthesis rules. The form of these rules is the production :

t => ({f1,...,fm}, {t1,...,tn})

where t,t1,...,tn ($n \geq 0$) are expressions representing tasks and f1,...,fm ($m \geq 0$) are expressions representing facts. The right-hand side may also be a function that evaluates to the expression ({f1,...,fm}, {t1,...,tn}). The production uses to contain variables that are instantiated when it is applied.

Given an initial task and a set of synthesis rules, task-reductive synthesis proceeds according to a depth-first search by successive expansion of an AND/OR tree, which we call task-reduction tree. The AND nodes of this tree are labeled with instantiated tasks, and its OR nodes are labeled with instantiated synthesis rules. At every moment during the problem-solving process, this tree represents the partitioning of the initial task that is currently considered by the system.

Initially, the task-reduction tree consists of a single node that is labeled with the initial task. Next, each expansion of the tree is carried out by applying a synthesis rule such that the task in its left-hand side matches the task associated to a pending AND node N of the current tree. The following two operations are then executed :
(1) An OR node N' is added to the tree as a successor of N, and the instantiated synthesis rule is associated with it.
(2) Each task in the right-hand side of the instantiated rule is associated with an AND node that is added to the tree as a successor of N'; if the set of tasks in the rule's right-hand side is empty, then N' is a terminal node.

Simultaneously, the facts contained in the rule's instantiated right-hand side are entered into a factual data base as new assumptions about the device being designed.

The task-reduction tree is terminal when all its pending nodes are terminal nodes. If there is no contradiction in the factual data base, the problem is solved then. TROPIC applies various general-purpose and input domain-specific heuristics to guide task-reductive synthesis [6].

3.2. Examples

These synthesis rules correspond to the informal examples of section 2 :

```
SY1:(design-transformer &tr)
    => (progn
         (setq &mc (! mc))
         (setq &ec (! ec))
         (i&({(part-of &tr &mc),
              (=(is-a &mc) magnetic-circuit),
              (part-of &tr &ec),
              (=(is-a &ec) electrical-circuit)},
             {(design-magnetic-circuit &mc),
              (design-electrical-circuit &ec),
              (select-cooler-for &tr)})))
SY2:(select-cooler-for &tr)
    => (progn
         (setq &tk (! tk))
         (i&({(=(cooler-of &tr)oil),
              (part-of &tr &tk),
              (=(is-a &tk)tank)},
             {(design-tank &tk)})))
```

```
SY3:(select-cooler-for &tr)
    => ({(=(cooler-of &tr) air)},nil)
```

[Comment : When rule SY1 is applied, the variable "&tr" (all atoms prefixed by "&" are variables) is first bound to a constant (for example: "tr1") by unifying the left-hand side of the rule with an instantiated task ; next, the right-hand side of the rule is evaluated. This evaluation binds the variables "&mc" and "&ec" to constants (for example "mc1" and "ec1") generated by evaluating (!mc) and (!ec) ("!" generates a new atom each time it is evaluated by concatenating its quoted argument and an integer). The evaluation of the TROPIC function "i&" then returns the instantiated expression ({(part-of tr1 mc1),...}, {(design-magnetic-circuit mc1),...}). When rule SY3 is applied, its right-hand side is merely instantiated).

About 100 synthesis rules were necessary to run the transformer example.

3.3. Fundamental aspects of task-reductive synthesis

Task-reductive synthesis explicitly represents the problem-solver's control state (the task-reduction tree) and logical state (the factual data base). At any moment, the logical state contains the facts that are currently believed to be true by the system, while the control state records both the choices on which these beliefs are based and the control dependencies among these choices. This twofold representation may be regarded as a combination of the "state-space" and "problem-reduction" representation[8]. The failure recovery procedure used by TROPIC draws much of its power from the following two possibilities :

1 - When a choice is rejected, the explicit knowledge of the current control state permits the system to circumscribe the extent of the updating necessary. Only that part of the task-reduction tree which is rooted at the rejected choice,and the subsequent assumptions, need to be discarded. Other choices can be kept effective even if they are more recent choices. This possibility may save a lot of effort reintroducing large parts of the solution that were needlessly discarded. It is not provided by the state-space representation, at least in an easy way.

2 -The logical state permits the system to consider the combined effects of the individual choices recorded in the control state. Choices, which may not be linked among them by control dependencies, may concur to generate contradictory assertions, so that the maintenance of the logical state's consistency permits the system to take into account logical dependencies among such choices. This possibility to consider transversal interactions among choices is critical because the hypothesis according to which design tasks are reducible to independent subtasks would be unrealistic. Its is not provided by the problem-reduction representation.

4. FACT-DEDUCTIVE ANALYSIS

4.1. Form and use of analysis rules

Knowledge for fact-deductive analysis is represented in TROPIC by means of analysis rules.The form of these rules is the production

$\{f1,...,fm\} \Rightarrow \{f1',...,fn'\}$

where $f1,...,fm$, $f1',...,fn'$ ($m \geq 0$, $n > 0$) are expressions representing facts. The right-hand side may also be a function that evaluates to the expression $\{f1',...,fn'\}$ ($n > 0$).

Each analysis rule is subject to pattern-directed invocation. When the facts in its left-hand side match facts in the factual data base, its right-hand side is instantiated or evaluated according to the matching substitution and the resulting facts are entered into the data base. These facts may in turn contribute to the invocation of other analysis rules. Rules with empty left-hand sides are invoked as soon as problem solving starts so that facts in their right-hand sides are premises to the solution. A deduction graph records all the deductions that have resulted from the invocation of analysis rules. When a contradiction is noted, it is used in reverse to determine the set of assumptions that are antecedents of the contradictory facts. During failure recovery, it is used in the forward direction to update the factual data base.

4.2. Examples

These are sample analysis rules :

```
- {(=(is-a &tr) transformer),(=(cooler of &tr)&cf),
   (part-of &tr &cl), (=(is-a &cl) coil)}
    =>(progn
        (if(eq &cf air)
            then (i&{(<(voltage &cl) 3e3)})
            else (i&{(<(voltage &cl) 5e3)})))
```

This rule implements: "if air is the cooler of a transformer, then the voltage between the terminals of any coil in this transformer must be lower than 3KV ; otherwise, it must be lower than 5KV".

```
- {(=(is-a &tr) transformer),
   (part-of &tr &tk), (=(is-a &tk) tank)}
  =>{(=(temperature-rise-in-oil-of &tr)
       (temp-oil (losses-in &tr)
       (radiating-surface-of &tk)))}
```

This rule encodes a physical law in its right-hand side. Atom "temp-oil" is the name of an input function that computes the temperature rise in the oil based on the losses in the transformer and the radiating surface of the tank.

4.3. Detection of contradictions

A contradiction occurs when some facts in the factual data base cannot be true simultaneously.

Two programs of TROPIC are responsible for detecting contradictions. One is based on heuristic numerical search techniques. We refer the reader to [5] for a description of these programs. Contradictions produce failures from which TROPIC immediately attempts to recover.

5. RECOVERY FROM FAILURES

5.1. Principle

When a failure is noted, the failure recovery procedure used by TROPIC determines the logical support of the contradiction, i.e. the subset of those assumptions from which the contradictory facts were derived. Then, it rejects the choice of a synthesis rule the application of which has produced at least one of these assumptions. Thus, it never tries to reject a choice that is irrelevant to the failure. In addition, only those choices which are subordinated to the rejected choice by control dependencies are immediately discarded However, there are usually several relevant choices that may be rejected alternatively in order to recover from a failure. The recovery procedure must be able to consider them in a systematic way in order to avoid perpetual iteration through successivefailures or omission of possible solutions. Indeed, the choice that is rejected will be replaced later by another choice and it may happen that this new choice leads to another failure from which it is possible to recover by reintroducing the previously rejected choice. But, since this choice already led to failure, its return, if it is not accompanied with other actions, will drive TROPIC into an infinite loop of successive failures. On the other hand, ruling out the possibility of reactivating this choice may result in omitting potential solutions, if there was more than one way to recover from the earlier failure. This difficulty requires that the system remembers the conditions of its failures so that, if a once rejected choice is to be reintroduced, the recovery procedure can previously correct the failure responsible for this choice being rejected in a different way.

The recovery procedure of TROPIC avoids infinite loops through failures and omission of possible solution as follows :

- The conditions of the failures that are encountered are remembered in expressions called failure indicators, which are attached to the choice nodes (AND nodes) corresponding to the rejected choices. If recovering from a new failure is possible by reactivating a previously rejected choice, then the recovery procedure can use the corresponding failure indicator to recover differently from the failure that was responsible for this choice being rejected. It is only after making this change that the once rejected choice is reintroduced. In this way, the system never tries the same combination of choices more than once.

- The recovery procedure avoids omitting potential solutions by operating chronologically. It never rejects a choice such that rejecting a more recent choice would enable TROPIC to recover from the current failure without recreating the conditions of an earlier failure.

5.2. A simple expository example

We now describe a possible behavior of TROPTC on the oversimplified problem defined by Table 1, "A" being the initial task. For the sake of simplicity, we represent tasks by capital letters and facts by lower-case letters. Each fact has the form "a" or "-a" (negation of "a"), so that a contradiction can only occur between a fact and its negation. In addition, we omit synthesis rules providing alternative ways of reducing tasks A,B,C,..., because they will not be used in the particular solution considered below ; similarly, the analysis rules that will never be invoked are now shown. Three failures will be encountered before a solution is ultimately produced.

Suppose that TROPIC applies synthesis rules <1>, <2>, <5>, <3>, <8>, <9>,<15>, <16>, <4> , *<11>,* <13>, <17>, <18> and <6> successively. Then, the task-reduction tree is that of figure 1. For the sake of readibility, the assumed facts have been associated with the corresponding OR nodes of the tree. The oriented dashed line that is superposed on the tree connects the expanded AND nodes in the chronological order of their expansion. We call it the decision path and it is generated during task-reductive synthesis.

At this moment, the system notes the first failure, since fact -e, which is contradictory with assumption e, is deduced (invocation of analysis rules <$1^{!}$> and<2^{f}>). An expression, called the diagnostic expression, is then initialized to ({e,h,c\,f},nil), where {e,h,cj,f} is the logical support of the contradiction (cf.§ 5.1.). The general form of this expression is (A,T), where A is a set of assumed facts and T is a set of tasks. It is always such that the conjunction of the conditions "each element of A is an assumption in the factual data base" and "each element of T is a task labeling an AND node of the task-reduction tree" is a sufficient condition for having encountered a failure.

Then, the system traces back the decision path until it meets an AND node N that satisfies the following two criteria :
CI : An assumption or a task contained in the set A or T of the diagnostic expression (A,T) has been generated when N was expanded.
C2 : There exists an alternative synthesis rule for expanding N, the application of which would not result in generating this assumption or this task.

N is called the recovery node. In the present case it is the node labeled by task F, since the reduction of F by rule <7> would not generate assumption f. The expression (({e,h,c_1},nil),({f},nil)) is then drawn from the diagnostic expression and it is associated with N as a failure indicator Φ_1. The general form of this expression is ((A_1,T_1), (A_2,T_2)), where A_1 and A_2 are sets of assumed or assumable facts, and T_1 and T_2 are sets of tasks. It is a conditional statement saying that expanding N with a synthesis rule that generates each assumption in A_2 and each task in T_2 always leads to failure if each element of A_1 is an assumption in the current factual data base and if each element of T_1 is a task labeling an AND node of the current task-reduction tree. Expressions (A_1,T_1) and (A_2,T_2) are respectively called the antecedent and the consequent of the failure indicator. When the conditions "each element of A_1 is an assumption in the data factual data base" and "each element of T_1 is a task labeling an AND node of the task-reduction tree" are true because of the expansion of AND nodes that are all predecessors of N in the decision path, the failure indicator is ON ; otherwise, it is OFF. All failure indicators recorded by TROPIC are ON at the time of their creation ; it is in particular the case of Φ_1.

The choice of synthesis rule <6> is now rejected, so that fact f is no longer assumed and analysis rules <1'> and <2'> are desinvoked. Then, task-reductive synthesis is resumed by applying synthesis rule <7> . At this moment, the task-reduction tree is that of figure 2 but the system notes the second failure since fact-b is deduced (invocation of analysis rules <3'>, <4'>and <5'>). It sets the diagnostic expression to ({b,g,f'_2, c_2,f'_1},nil) and it traces back the decision path:

- The node labeled by F satisfies both criteria C1 and C2 stated above since the application of rule <6> would note generate assumptions f'_1 and f'_2. But failure indicator Φ_1, which is currently ON, shows that reselecting rule <6> would inevitably recreate the conditions of an earlier failure. Therefore, this node is not chosen for recovery node. Instead, the system executes the following two operations :

. it draws the expression (({b,g,c_2},nil),({f'_1, f'_2},nil)) from the current diagnostic expression and it associates that expression with the node labeled by F as a failure indicator Φ_2.

. it sets the diagnostic expression to ({b,g,c_2, e,h,c_1},{F}), by combining the antecedents of Φ_1 and Φ_2. Indeed, in order to recover from the current failure, TROPIC may now either reject a choice responsible for one of the assumptions b, g and c_2 (so that a cause of the current failure will be removed), or reject a choice responsible for one of the assumptions e, h and c_1 (so that it will be possible to reselect rule <6>), or reject the choice responsible for the generation of task F (so that the need for assuming either f, or f'_1 and f'_2, will be removed).

- Nodes labeled by Q,P,J,I,D,N and M do not verify criterion C1. None of them is chosen for recovery node. Thus, the system passes over seven backtrackable nodes that would have been successively considered by a blind chronological procedure.

- The node labeled by H satisfies both criteria C1 and C2, since applying rule <10> would not generate assumption h. No ON failure indicator being associated with this node, TROPIC chooses it for recovery node and associates the failure indicator Φ_3 = (({b,g,c_2,e,c_1},{F}),({h},nil)) with it. The recovery node and the node labeled by F,the expansion of which made the current failure effective, belong to two different AND paths. This means that the corresponding two choices are not logically independent.

The system now updates the current solutions.first its rejects the choice of rule <9>, so that fact h is no longer assumed. Because they are subordinated to this choice by control dependencies, rules <15> and <16> are also discarded. Next, TROPIC notes that failure indicator Φ_1 has become OFF (because of the removal of assumption h), while failure indicator Φ_2 remains ON. The consequent of Φ_2 shows that the choice of rule <7> cannot be maintained which means that the current failure has not been remedied yet. However, since reselecting rule <6> has just been made possible (Φ_1 being OFF), TROPIC can now reject the choice of rule <7>, which was a choice responsible for this failure. After rejecting this choice, the task-reduction tree is that of figure 3 (note the updated decision path). Several parts of the current solution remain unchanged although they were produced after the recovery node was expanded.

Task-reductive synthesis is resumed again: rules <10> and <6> (because of Φ_2) are successively applied (cf. figure 4). This leads the system to note the third failure because fact -g is deduced (invocation of analysis rules <6'>, <7'> and<8'>). The diagnostic expression is set to ({g,d,i,f,h'_1}, nil) and the decision path is traced back :

- The node labeled by F satisfies both criteria C1 ans C2, but because of Φ2, which is currently ON, it is not chosen for recovery node. The failure indicator Φ_4 = (({g,d,i,h'_1},nil),({f},nil)) is associated with this node and the diagnostic expression is set to ({g,d,i,h'_1,b,c_2},{F}).
- The node labeled by H satisfies both criteria

C1 and C2, but because of Φ_3, which is currently ON, it is not chosen for recovery node. The failure indicator Φ_5 = (({g,d,i,b,c_2},{F}),({h'}, nil)) is associated with this node and the diagnostic expression is set to ({g,d,i,b,c_2,e,c_1}, {F,H}). Note that this expression combines conditions of the first two failures and conditions of the current failure.

- Nodes labled by Q,P and J do not satisfy criterion C1. None of them is chosen for recovery node.
- The node labeled by I satisfies both criteria C1 and C2. No ON failure indicator being associated with this node,it is chosen for recovery node and the failure indicator Φ_6 = (({g,d,b,c_2,e,c_1}, {F H}),({i},nil)) is associated with it.

TROPIC rejects the choice of rule <11>, so that fact i is no longer assumed. Then, it examines the failure indicators that have been associated with successors of the node labeled by I in the decision path. Because fact i is no longer assumed, failure indicators Φ_4 and Φ_5 are now OFF, so that the system recognizes that it has recovered from the current failure. Thus, no other choice is rejected.

Task-reductive synthesis is resumed by applying rule <12> and rule <14>. At this moment, the task-reduction tree is terminal (cf.figure 5). Since no contradiction is detected, the problem is solved.

This example shows that., each time a failure is recovered from, an ON failure indicator Φ is associated with the recovery node N. This indicator prevents the system from reselecting the rejected synthesis rule as long as it remains ON. Since Φ may be turned OFF only if an AND node, which preceded N in the decision path when N was chosen for recovery node, is chosen for recovery node in turn, the same combination of choices cannot be tried more than once, and therefore perpetuel iteration is impossible. On the other hand, the diagnostic expression always determines a sufficient condition for encountering a failure. Thus, by tracing back the decision path, the recovery procedure always chooses for recovery node the most recently expanded AND node that may enable the system to resume task-reductive synthesis without recreating the conditions of an earlier failure.In this way, omission of potential solutions is impossible.

6. LEARNING FROM FAILURES

Failure indicators arc efficient means of recording failure conditions because they are connected to the choice nodes of the task-reduction tree. However they do not completely preventthe system from re-achieving the conditions of earlier failures. Indeed,due to intervoning failures, an ON failure indicator may be turned to OFF,so that the conditions determined by its consequent may then be achieved by the application of a synthesis rule; if it happens later that the conditions in its antecedent are also achieved, then the occurrence of a failure is inevitable.

OFF failure indicators are not needed to avoid infinite loops and omission of possible solutions and they also have little chance to become ON again. Thus, TROPIC discards them, so that updating the ON/OFF status of failure indicators merely consists of erasing the failure indicators that are no longer ON. However, the information they contain is not systematically forgotten : when a failure indicator ((A. ,T) , (A_2,T_2)) is discarded,the system may decide to generate the <u>failure expression</u> (A_1. u A_2.T_1. u T_1) that determines a context in which failure is unavoidable. Failures expressions are stored separately from the task-reduction tree. When expanding an AND node N, TROPIC eliminates from consideration every rule the application of which would reestablish the conditions recorded either in an ON failure indicator associated with N, or in a failure expression.

In order to keep the list of failure expressions a reasonable size, TROPIC converts an OFF failure indicator into a failure expression only if it combines the conditions of several failures. In addition, it reconverts failure expressions into ON failure indicator whenever possible : let(A,T) be a failure expression, and let call A_1 and T_1 the subsets of A and T determining conditions that are already achieved before the expansion of an AND node N ; if there exists a synthesis rule applicable to N, the application of which would achieve all the conditions in A_2 - A- A. and in T_2 - T - T_1 then not only does TROPIC not apply this rule, it also reconverts the failure expression into an ON failure indicator ((A_1.,T_1),(A_2,T_2)) associated with N.

Failure indicators and failure expressions provide means of a good compromise for recording failure conditions. ON failure indicators avoid infinite loops and omission of potential solutions and they can be used efficiently. Failure expressions enable the system to convertinto a usable form information contained in OFF failure indicators. They are less efficiently used, but their number may be arbitrarily limited since they have no effect on the completeness of the problem-solver.

7. RELATION TO OTHER WORKS

Other systems use representations that distinguish between control and logical data, but they do not exploit this distinction in the same way nor for the same purpose as TROPIC. One of them is NOAH [9], a program for generating plans of actions at varying levels of detail. NOAH's control state is a hierarchical net of actions that combines the

decomposition of an action into successively more detailed subactions and the partial orderings of the pjans at each level of detail. All actions bring their effects into a model of the world and logical interdependences among actions are considered in this model. When a conflicting interdependence is noted in a plan, correction is attempted only by enforcing ordering of the partially ordered actions. Research on NO All was deliberately restricted to non-search aspects of problem solving and consequently the system includes no provision for backtracking.

Another related system is NASL [7], a program for designing electronic circuits. NASL's control state is a net of design tasks similar to NOAH's net of actions. As in TROPIC, the goal of the system is to produce the model of a device. NASL includes no backtracking procedure and Mc Dermott proposes an alternative scheme according to which correcting a mistake is treated as a task like another, drawn from the description of the mistake. However, the recovery procedure based on this scheme was not implemented.

Hayes [3] describes a representation for robot plans that consists of two interlinked data structures representing the subgoal structure of a plan and the logical relationships of the decisions taken in constructing the plan. Hayes states that "the representation enables a decision to be remade without remaking chronologically subsequent but logically independent decisions". Nevertheless, the improved backtracking procedure was not implemented .

Selective backtracking has been investigated in EL/ARS [10] a system for analyzing electrical circuits. The recovery procedure implemented in this system has been generalized by Doyle [2j. It has no explicit representation of the problem-solver's control state in the sense of Section 3.3. Subsequent]y the conditions of the encountered failures are recorded into NOGOOD lists that are very similar to the TROPIC's failure expressions. In order to avoid perpetual looping, these lists nu.st all be examined before making a choice. A possible improvement is de-emphasizing some conditions in the NOGOOD lists in order to produce conditional expressions which are similar to TROPIC*s failure indicators, except they are not connected to a control structure. De-emphasis reduces the number of NOGOODs to be considered at any moment, but is difficult to implement because it has no sound domain-independent basis.

SYNTHESIS RULES

<1>	A	=>	({a},{B,C,D})
<2>	B	=>	({b},{E,F})
<3>	C	=>	({c_1,c_2},{G,H})
<4>	D	=>	({d},{I,J})
<5>	E	=>	({e},nil)
<6>	F	=>	({f},{K})
<7>	F	=>	({f'_1,f'_2},{L})
<8>	G	=>	({g},nil)
<9>	H	=>	({h},{M,N})
<10>	H	=>	({h'_1,h'_2},nil)
<11>	I	=>	({i},nil)
<12>	I	=>	({i'},nil)
<13>	J	=>	({j},{P,Q})
<14>	K	=>	({k},nil)
<15>	M	=>	({m},nil)
<16>	N	=>	({n},nil)
<17>	P	=>	({p},nil)
<18>	Q	=>	({q},nil)

ANALYSIS RULES

<1'>	{f,c_1}	=>	{-r}
<2'>	{-r,h}	=>	{-e}
<3'>	{f'_1,c_2}	=>	{s}
<4'>	{f'_2,g}	=>	{t}
<5'>	{s,t}	=>	{-b}
<6'>	{h'_1,f}	=>	{u,-v}
<7'>	{u,i}	=>	{w}
<8'>	{-v,w,d}	=>	{-g}

Table 1

REFERENCES

[1] R.DAVTS, B.G.BUCHANAN0977) : "Metal-1 eve 1 knowledge : Overview and applications", IJCAI 77, Cambridge, 1977.

[2] J.D0YLE(1977) : "Truth-maintenance systems for problem solving", Master thesis, MIT, Cambridge, 1977.

[3] P.HAYES(1975):"A representation for robot plans", IJCAI 75, Tbilisi, 1975.

[4] J.C.LATOMBE(1976) -."Artificial Intel! igence in computer-aided design: the TROPIC system" SRI AIC, Technical note 125, MeAlo Park, January 1976.

[5] J.C.LAT0MBE(1977):"Unc application de I n - telligence Artificielle a la conception assistee par ordinateur (TROPIC)", These d'Etat, Grenoble 1977 (in French).

[6] J.C.LAT0MBEO979) :"Problem-solving methods in a system for designing complex assemblies" Rapport de recherche IMAG, Grenbble, 1979 (in preparation).

[7] D.V.McDERMOTT(1977):"Flexibility and efffi- ciency in a computer program for designing circuits", AI Lab. TR 402, MIT, Cambridge, June 77.

[8] N.J.NILSS0N(1971):"Problem-solving methods in Artificial Intelligence", McGraw-Hill, 1971.

[9] E.D.SACERD0T1(1977):"A structure for plans and behavior", American Elsevier Publishing Company, 1977.

[10] R.STALLMAN, G.J.SUSSMAN(1976):"Forward Reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis", MIT AI Lab, Memo 380, Cambridge, September 1976.

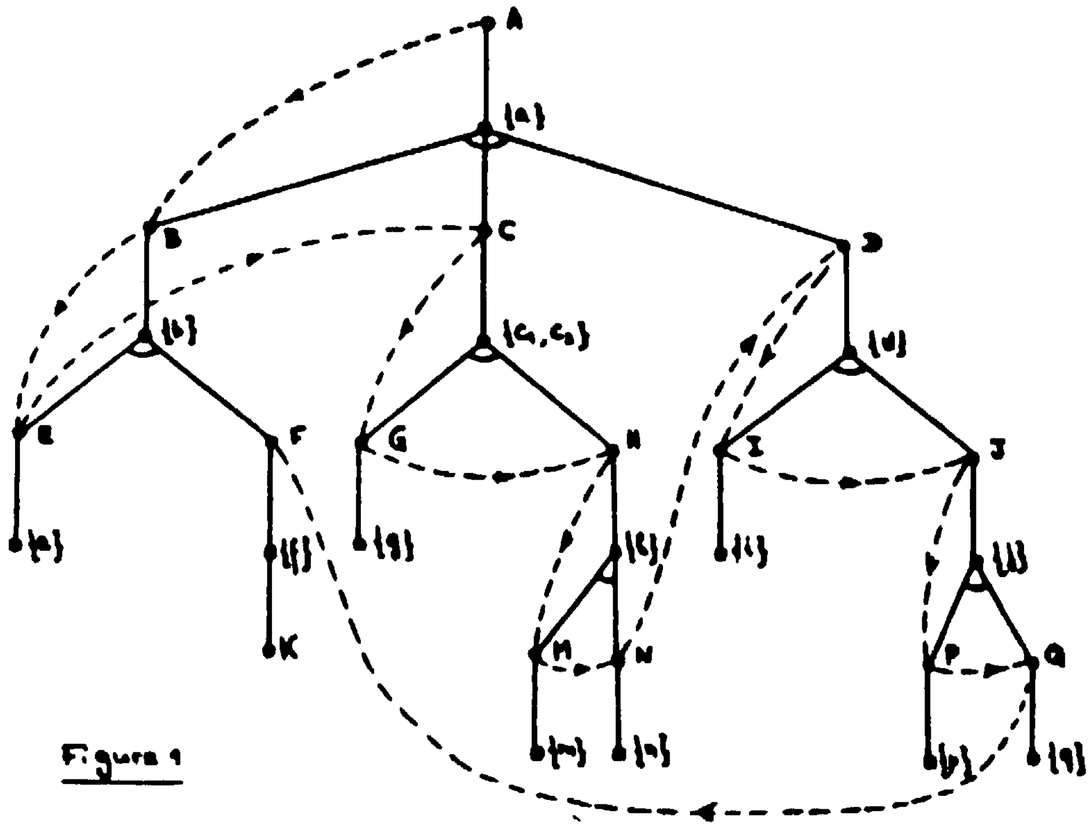


Figure 1

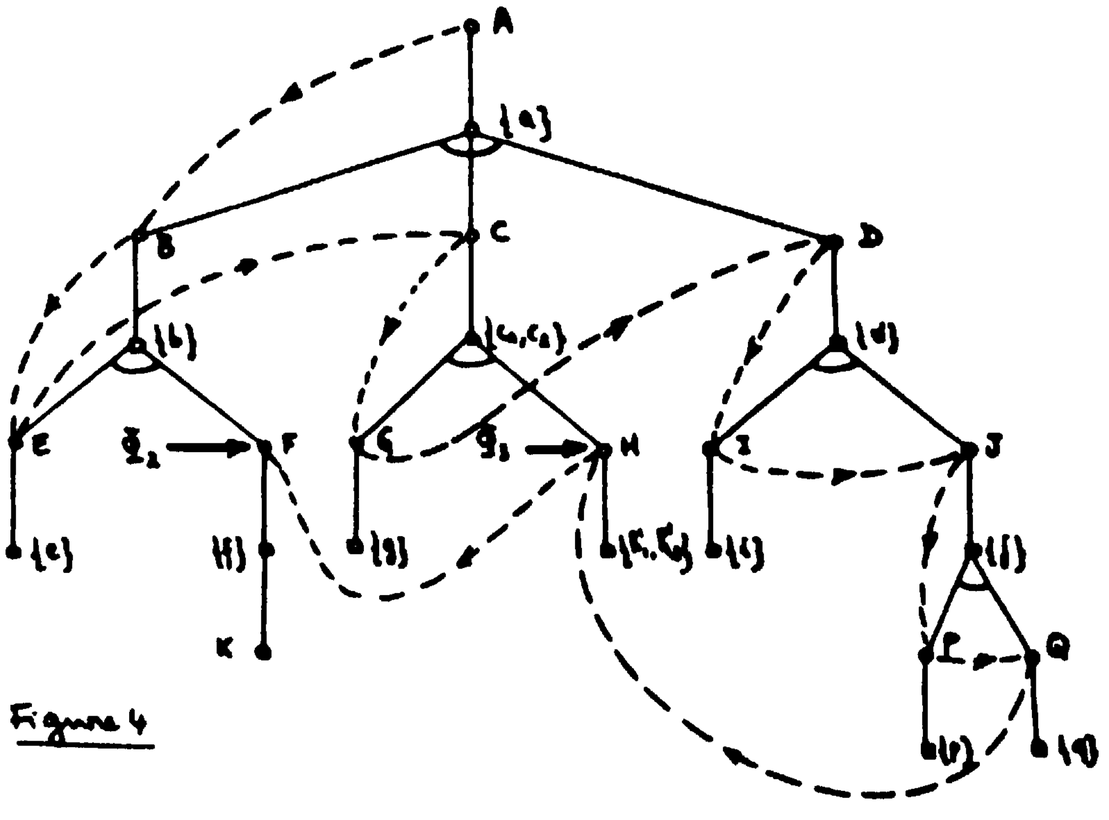


Figure 4

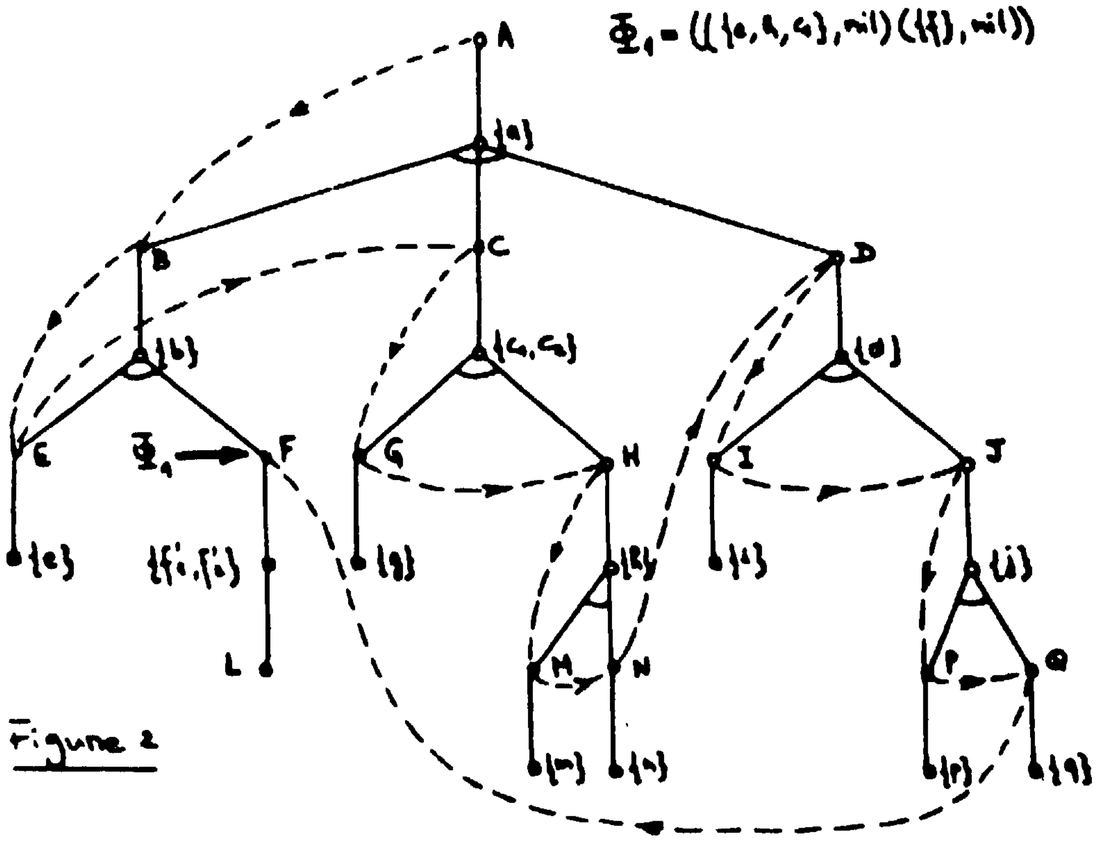


Figure 2

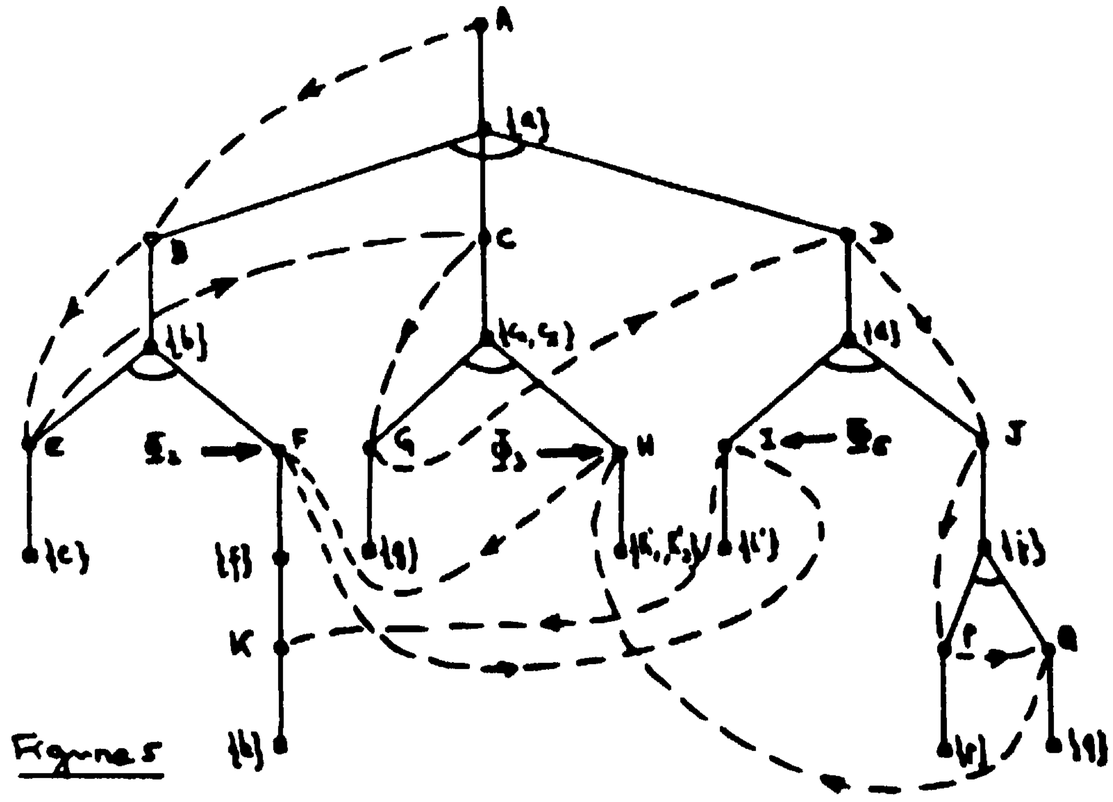


Figure 5

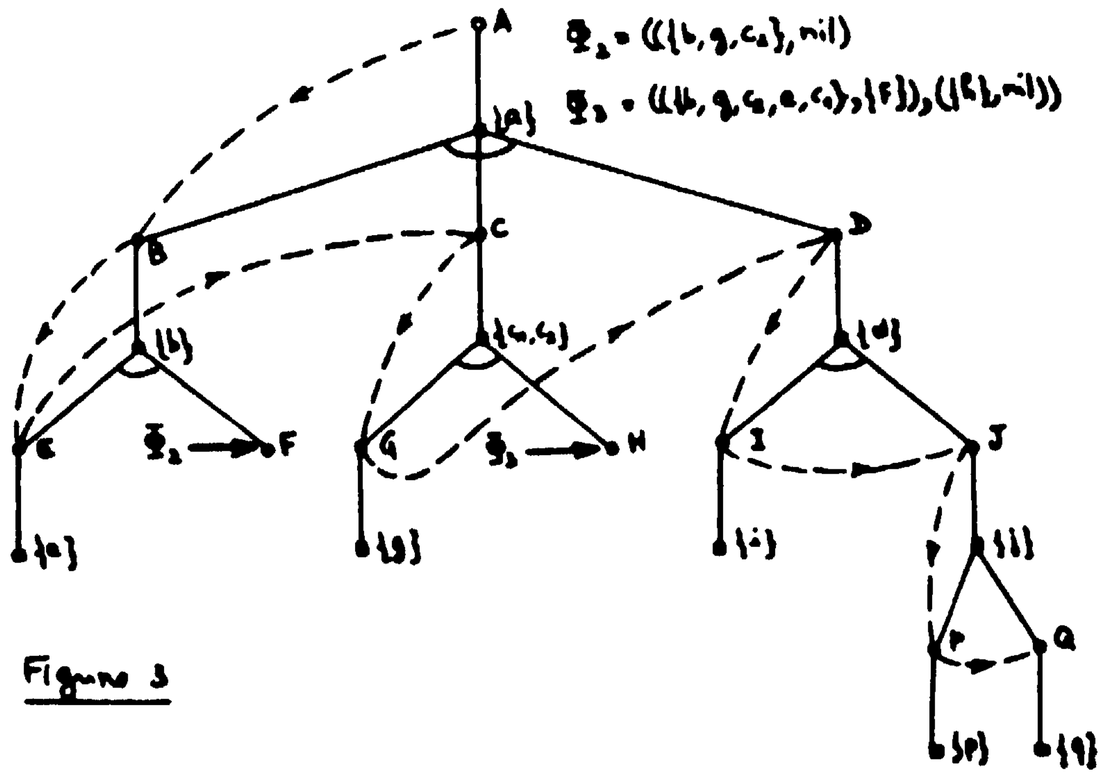


Figure 3

INTERFACING A SEMANTIC NET WITH AN AUGMENTED TRANSITION NETWORK

Joachim H. Laubsch
Institut fuer Informatik
Universitaet Stuttgart
West-Germany

The main advantages of semantic ATN-grammars are efficient execution and easy implementation in small discourse domains. *The* drawback lies in the growth of the semantic grammar if the domain of discourse is to be extended. The thesis of this paper is that a higher level of modularity for grammar rules can be attained by making semantic ATNs sharable through schemata (called "concepts") of the knowledge base. A workspace of activatable subATNs guides the parser in choosing among competing PUSH arcs. Expected subATNs are entered into this workspace either indirectly from the schemata through a filtering mechanism, or directly as a side effect of actions on arcs which instantiate a concept.

1. Introduction

Question-anwering systems such as LUNAR [13], SOPHIE [4], PLANES [12], and various applications of LIFER [7] contributed valuable methodological tools for the development of a natural language user interface. Of these, semantic ATN-grammars and semantic nets have been successfully used in our group in several small projects [1,11]. The drawback of semantic grammars lies in the growth of the ATN if the discourse domain is enlarged. We discuss an approach to reduce the task-specifity of semantic ATNs by interfacing them with an inheritance network which performs many of the tasks usually embedded in tests and actions of a semantic grammar.

An object-oriented representation language - called OBJTALK [9] - will be used to illustrate the interface between ATN and semantic net. OBJTALK was inspired by SMALLTALK [6], FRL [10], KRL [3], KLONE [2], and DIRECTOR [8].

The basic ideas of object-oriented knowledge representation are:

(1) objects are active data structures with behavioral traits, such as
- slot & filler - pairs (where the filler may be defaulted, restricted etc) and
- methods consisting of a pattern to receive a message and a body describing what is to be done.

(2) behavioral traits are inherited in a **generalisation hierarchy** (or network). An object is either a **class** or an **instance**. Each object has one or several superclasses from which it inherits behavioral traits recursively up to a **root** node.

(3) An object will exhibit behavior if it is sent a message.

2. Procedural attachment of ATNs to concepts in an inheritance net

In order to achieve semantic-driven parsing, a semantic SubATN should be activated whenever a particular **word** or **concept** is found. The basic mechanism is to attach **trigger-keys** to a schema that state which words or concepts put some named subATNs in a preferred activatable state. Once activated, a subATN may access the slots of the schema in tests or actions, and/or pop a structure used to fill slots of that schema. The inheritance structure between concepts is defined such that semantic subATNs are sharable among many concepts.

A concept definition in OBJTALK has the form:

```
(Concept <concept-name>
   (Individuates <name of a super-concept> ...)
   (Generic-properties
      <Slot>: <Filler description>
      ...)
   (Trigger-atn <Trigger-keys>  ; attached ATN
      [<subATN-node> <production>] ...))
```

The Trigger-atn form is a special case of pattern-directed message receipt by an object. The Trigger-keys serve as activation keys for a **bottom up** instantiation of the current concept through words or other concepts. The form of **Trigger-keys** is:

```
(triggered-by: {(Words <lexical entry> ...)}
               {(Concepts <concept name> ...)})
```

If

(1) a concept has been **partially** instantiated (an

instance of this concept or one lower in the hierarchy was made, but not all obligatory slots are given values yet), or
(2) one of the **concepts** mentioned in trigger-keys has been instantiated, or
(3) one of the **words** mentioned in trigger keys was entered into the chart during a lexical prepass,

then the concept's Trigger-atn form is evaluated. This has the effect that all subATN-nodes named in it are given a preferred activation status (leading the parser to try PUSHes to those subATNs first). The slots in the concept can be accessed from the ATN analogous to registers. The register **self** contains the name of the activating concept.

The <production> describes what will be done with the result of the activated subATN in case it was left by a POP arc. Its form is:

[(<Filter> ...) <action> ...]

The convention is that the value returned by a POP is a nested a-list. The filters match such an a-list, thereby binding filter variables. A (sequence of) path-indicator(s) [$!<symbol> ...] is used to find the embedded value in an a-list.

The form of the <Filter> is
($!<symbol> {$!<symbol> ...} .
<list of constants, variables and slot-values>)

All filters have to match successfully. A fail action in <action> will try alternative matches if possible or finally fail back to the ATN. A success deactivates the current <subATN node>.

3. Concept-driven control structure

We are using an ATN compiler developed by BURTON [5], modified to our MacLISP. The depth-first control structure of this parser leads to a large number of back-ups because, from a single node, many PUSH arcs are possible. We want to reduce the number of failing PUSHes and POPs by entering those subATNs occuring in evaluated Trigger-atn forms into a **workspace** of **activatable subnets** (**WAS**).

A **morphological prepass** extracts wordclass, features etc and puts them into the **chart**. At this point, trigger-atn procedures associated as the words-part of trigger-keys may enter WAS. If the ATN machine reaches a node, containing PUSH arcs, a check is made to see whether any of them is among the expected subATNs in WAS. In case of a word-triggered subATN an additional test ensures that the triggering word has almost been reached by the scanner. This is psychologically motivated, since we think of parsing as basically a serial process with only a small look-ahead (of 2 - 4 words).

New trigger-atn procedures may be added to WAS in basically two ways:
(1) **indirectly** by processes in the inheritance network: A triggered subATN returns as the form of its POP arc an a-list which (partially) instantiatiates a concept. If instantiation is partial, the other subATNs stay activatable, else all are deactivated. Those attached subATNs where a concepts-part of trigger-keys contains an instantiated concept enter WAS. Also actions of productions may instantiate new concepts whose attached subATNs will become activatable.
(2) **directly** by the ATN as a side-effect of evaluating an action on an arc, if the action instantiates a new concept.

Indirect instantiation of concepts via a-lists allows a high generalizability of subnets - in the extreme to purely syntactic ATNs. On the other hand, **direct instantiation** presupposes that at the level of the ATN enough task-specific knowledge was coded to instantiate a concept, which in the extreme would result in a purely semantic grammar. The grammar programmer may chose between both alternatives. The ATN-programming style may change towards greater modularity, since PUSHes can be used more extensively.

4. Examples of an ATN <-> semantic net interface

Example 1: It shows **indirect instantiation** of concepts via a trigger-atn procedure attached to a particular attribute. The concept 'consumption' is defined as follows:

```
(Concept consumption
   (Individuates Quant/attr/prod) ; an attribute
   (Generic-properties
      of/product: (class PRODUKT)
      consumed/quantity: (class QUANTITY) ...)
   (Trigger-atn
    (triggered-by: (Words verbrauch- hoch Hoehe ...))
    [Q/PROP/OBJ ; subATN for query of object's property
        (($!OBJ1 (PROP: $?PROP))          |
         ($!OBJ2 (PROD/CODE: $?PROD)))    | filter
        |(Ask PRODUKT make: P with: $=,OBJ1)
        | ; make an exemplar of a prod/code product
action  |(Ask self set: of/product: $=P)
        | ; retain it in my of/product slot
        |(Ask P prop: $?PROP)
        | ; ask it how much it consumes on that
        | ; attribute ;]))
```

Q/PRP/OBJ can be shared by different attributes and products.

Example 2: This example illustrates how a subATN **directly instantiates** a concept which in turn makes further subATNs activatable. Consider sentences like:

>> Klaus faehrt mit dem Zug um 17:35
(Klaus goes by train at 17:35)

```
>> Klaus fliegt mit Gerhard von Stuttgart nach Berlin
  (Klaus flies with Gerhard from Stuttgart to Berlin)

For concepts like fly, drive etc a super-concept move
is defined:
(Concept fly
   (Individuates move)
   (Generic-properties
      instr:    (default FLIGHT)
      origin:   (constraint HAS-AIRPORT)
      destin:   (constaint HAS-AIRPORT)))

(Concept move
   (Individuates act/people)
   (Generic-properties
      agent:     (class PERSON)
      coagent:   (class PERSON)
      instr:     (constraint TRANSP/MED?)
      origin:    (class LOC)
      destin:    (class LOC)
      dep/time:  (constraint (BEFORE * $=arr/time))
      arr/time:  (constraint (BEFORE $=dep/time *))
      ...)
   Trigger-atn)  ; explained below
```

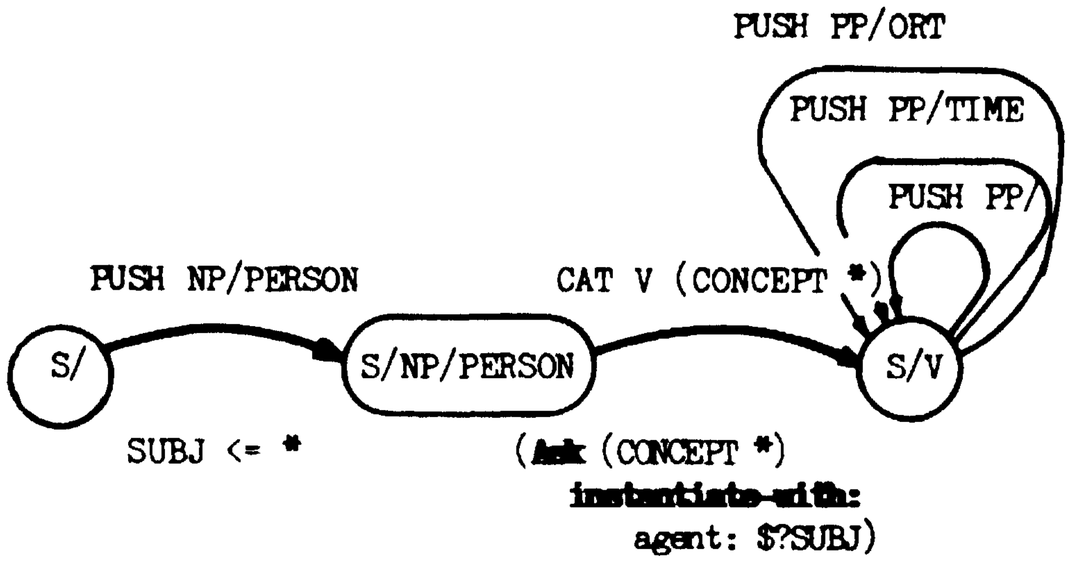


Fig.1: Triggering subATNs by instantiating a concept through an arc action

Parsing leads to instantiation of a concept (like **fly** or **drive**) as a side-effect of an arc action. This concept is a key in **move** and activates the subATNs contained in it:

```
(Trigger-atn
   (triggered-by (concepts fly drive walk ...))
   (PP/ORT [(($!HEAD $?ORT) ($!PREP nach))
            (Ask self set: destin: $?ORT)]
           [(($!HEAD $?x) ($!PREP von))
            (Ask self set: origin $?ORT)])
   (PP/    [(($!HEAD $?x) ($!PREP mit))
            (cond ((Ask $?x TRANSP/MED: ?)
                     (Ask self set: instr: $?x))
                  ((Ask $?x PERSON: ?)
                     (Ask self set: coagent: $?x))
                  ((FAIL)))])
   (PP/TIME ...) ...)
```

Among the many HJSH arcs from S/V (see Fig. 1) the subset |PP/ORT PP/TIME PP/} is preferred, expressing the expectation to associate only a restricted subset of cases, each signalled by particular syntactic constructions.

5- Conclusions

The proposed ATN-semantic net interface allows higher generality and extensibility of the semantic grammar because we may use PUSH arcs more freely. There is increased work in the filtering phase (reducable by pattern-compilation). *The* interface has been tested in a semantic grammar for the micro-world of soccer. In our current research we concentrate on observing the grammar programming process in order to establish guidelines for semantic ATN design.

REFERENCES

[I] Berth, K. "Zur Implementierung eines Lehrsystems flier LISP." Diplomarb. 62, IFI-UNI Stuttgart, 1977.

[2] Brachman, H.J. "A Structural Paradign for Representing Knowlede." HBN Rep. No. 3605, Cambridge, 1978.

[3] Bobrow, D.G. & Winograd, T. "An overview of KRL, a knowledge representation language." Congritive Science. 1:1 (1977) 3-46.

[4] Brown, J.S., Burton, R.R. & Bell, A.G. "SOPHIE - a Sophisticated Instructional Environment for teaching electronic trouble-shooting." EBN Rep. 2790, Cambridge, 1974.

[5] Burton, R.R. "Semantic Grammar: An engineering technique for constructing natural language understanding systems." HBN Rep. 3453, Cambridge, 1976.

[6] Ooldberg, A. & Kay, A. (Bis.) "anAILTAIK-72 Instructional Manual." XEROX, PARC, Palo Alto, 1976.

[7] Hendrix, G., Sacerdoti, E.D., Sagalowitz, D. & Slocim, J. "Developing a natural language interface to complex data." AXN . Trans on data base. 3:2 (1978) 105-147.

[8] Kahn, K.M. "DIRECTOR Guide." MIT-AI Memo 482, Cambridge, 19/78.

[9] laubsch, J.H. "OBJTALK - eine auf semantischen Netzen basierende Sprache zur Darstellung von Wissen". MMK Memo 12, IFI-UNI Stuttgart, 1979-

[10] Roberts, R.B. & Goldstein, I.P. "The FRL Manual." MIT-AI Memo 409, Cambridge, 1977.

[11] Schairer, M. "Darstellung des zum Verstehen einfacher LISP Programme benoetigten Wissens in FRL." Diplcmarbeit # 76, IFI-UNI Stuttgart, 1978.

[12] Waltz, D.L. "An English language Question Answering System for a large Relational Database." GAOL 21:7 (1978; 526-539.

[13] Woods, W.A., Kaplan, R.M. & Nash-Webber, B. ,, "The LUNAR sciences natural language information system." EBN Rep. No. 2378, Cambridge, 19/72.

TOWARD EFFICIENCY TROUGH GENERALITY

Jean Louis LAURIERE
Groupe Structures de l'Information
Institut de Programmation
4 Place Jussieu
75230 Paris Cedex 05 FRANCE

ABSTRACT: In the following lines we shall adduce new elements to a controversy which lies at the heart of current discussions about programming activities. The question is : must we write specific programs or general systems ? Wo shall contrast the two approaches to this methodological problem by means of a case in point. Their respective interest will be compared and a surprising conclusion will eventually be reached which is crucial to the core of AI and justifies many researches in the field : efficiency can only be attained trough generality.

I - INTRODUCTION.

Work in computer aided programming has two closely related aims: one is program verifying the other is program generating. Since the pioneering results [3] , [9] before the seventies, the two basic domains have been extensively studied. Computer aided verifications of programs, assertion-based proofs of programs wherein assertions are both produced and proved automaticaly, syntactic and semantic transformations of programs, program syntheses from examples, high level languages are the present common topics [2],[8],[10].
In this context, our purpose is on the contrary to show, however strange it might seem, that a specific algorithm, tailored by hand or by program for a particular problem, can be less effective than a general system, the essential nature of which is to be non-deterministic and which is able to deal with quite a large family of different problems.
Thus there is no need for provers or syntheti-sers but rather for general problem solvers.
This paradoxical proposition will be studied through the enlargment of an example belonging to the field of discrete mathematics. We shall argue over an a priori NP-complete [4] problem:

"Build all the permutations over the first n integers which verify the following constraints, where m and a are two boolean 1×(n-1) given vectors and σ is whatever solution looked for:

$\forall i \in [1,n-1] : m(i)=1 \Leftrightarrow \sigma(i) < \sigma(i+1)$,
$a(i)=0 \Rightarrow \forall j \in [i+1,n] \ \sigma(j) \neq \sigma(i)+1$,
$a(i)=1 \Rightarrow \forall j \in [1,i-1] \ ((\sigma(j) \neq \sigma(i)+1) \wedge \sigma(i) \neq n)$".

For example if n=4 , m=(101) , a=(110) , the permutations σ_1=(1324) and σ_2=(2314) are the only two to meet the above requirements.

II - FIRST APPROACH : "the natural one".

The possible solutions must be gradually generated and tested; the classical backtracking scheme is therefore called up and the following "script" is immediately set up in our brain :

- Point α: place the elements one at a time.
- Point β: test the feasibility of the thus obtained partial σ .
- Point γ: carry on if it works, backtrack if part failure or if solution.

This general script must now be made more precise and refined in order to take into account the particulars of our problem. In all cases point α will correspond to something like i=i+1 together with σ(i)= σ(i)+1 and with the convenient initializations. As often, point β will now prove a delicate one. Three tests - say t1, t2 and t3 - must be applied here :

- t1: σ is a permutation, i.e. $i \neq j \Rightarrow \sigma(i) \neq \sigma(j)$.
- t2: m imposes some monotony constraints.
- t3: a imposes less handy constraints.

As in any backtrack approach, the manual optimization task will now consist in setting up these tests early enough and in the right order so as to avoid the generation of useless part solutions. Here, the first test is easy to handle and is certainly the most binding; it will therefore reasonably be undergone first. The monotony test is binding too, and it is much easier to deal with this test than with the conditions derived from a. Indeed this last conditions must be translated by a DO-loop instead of a single IF-statement as with the former ones. Eventually we keep the order t1, t2,t3 to obtain our program wich is thus nearly

```
completed  :  i=0 ; σ ≡ 0 ;
● α : i=i+1 ; IF i>n THEN GOTO γ ;
[By hypothesis the first (i-1) elements of σ are
known; a convenient value for σ(i) is looked for]
  * : σ(i)=σ(i)+1 ; IF σ(i) > n THEN GOTO γ ;
                    IF    i≠1   THEN GOTO α ;
● β : t1 [test on σ]: DO j=1, i-1 ;
          IF σ(i)=σ(j) THEN GOTO * ; END;
      t2  test on m : IF m(i-1)=1 THEN
          IF σ(i)<σ(i-1) THEN GOTO * ELSE GOTO t3;
          IF m(i-1)=0 THEN
          IF σ(i)>σ(i-1) THEN GOTO γ ELSE GOTO t3;
      t3 [test on a]: ie=σ(i)+1;
          IF a(i)=1 THEN DO; IF ie > n THEN GOTO γ;
          DO j=1,i-1; IF σ(j)=ie THEN GOTO * ;END;
          GOTO α ; END;
          IF a(i)=0 THEN DO; IF ie > n THEN GOTO γ;
          DO j=1,i-1; IF σ(j)=ie THEN GOTO α ;END;
          GOTO * ; END ;
● γ : [Backtrack]   σ(i)=0 ; i=i-1 ;
                    IF i≠0 THEN GOTO α ELSE STOP;
```

This program has two important virtues: first it was very rapidly written and secondly it works correctly and can easily be proved. However this program can of course be strongly improved. The first way is to find out some good lemmas for the formal problem, the second way is to undertake direct syntactic transformations of the program itself. The article by KNUTH in 1974 [5] originated many researches in such program manipulations. STANDISH et al [11] made up a general catalogue of such accepted transformations. ARSAC [1] programmed a nicely interactive system as an aid to program optimization. We give in [7] a more sophisticated version of the program which takes into account other mathematical properties, which avoids the use of any GOTO statement, and which keeps in memory all the useful information on vector a. The backtrack work will therefore be largely reduced. But we are concerned with a space the size of which is n! and it is likely that any such specialized program cannot work effectively for each couple of m and a. However it seems impossible to achieve much better along this line!

III - SECOND APPROACH : "the general one"

In this section we want to introduce and discuss a quite different way of solving this kind of combinatorial problems. It is in fact clear that *all the previous work will prove useless if one should change if only a little the constraints attached to our problem:* we can therefore have a mind to write a much more powerful system which will solve this whole family of problems involving discrete finite sets. A great deal of mathematical knowledge and a language to state the different problems are required for achieving this goal.

ALICE, already described in [6], is such a code. The permutation generation problem is given to ALICE in a non procedural manner as follows:

```
GIVEN CONSTANT      n
      SETS          E1=[1,n] ; E2=E1-n.
      COEFFICIENTS  m  ON E2 ; a ON E2.
FIND  BIJECTION     σ  ON E1 .
WITH  ∀i ∈ E2  m(i)=1 ⇔ σ(i+1) > σ(i) AND
a(i)=0 ⇒ ∀j ∈ [i+1,n] σ(j)≠σ(i)+1 AND
a(i)=1 ⇒ ∀j ∈ [1,i-1] σ(j)≠σ(i)+1 AND σ(i)≠n.
```

Then the system asks for the missing delta : value of n, vectors of coefficients m and a . *Alice solves the problem quite in the same manner as we solve it by hand.* So the main difference with the previous program is the ability of formal manipulations of mathematical expressions. This allows the system to derive implications, to stick to the numerical characteristics of each problem, therefore to work differently as soon as the data change, to set up good hypotheses and finally *to use backtracking in intelligent fashion and non predetermined order.* Let us see how ALICE works through some examples of this permutation problem. So if n=9,m=(I 10101-01) and a=(1 I 1 I 0 1 1 0), the system interprets first the constraints deriving from vector m: (A):σ(2) >σ(I) and (B):σ(3) >σ(2). Now a zero value, namely m(3) is encountered; so m(i)=I is false, the program produces :

[o(i+I) > σ(i)J id est σ(i+1) < σ(i).

But as σ is by hypothesis a bijection it obtains finally o(i+I) < σ(i); so here with i=3 : σ(A) < σ(3) and likewise o(5) >σ(4) , σ(5)>o(6), σ(7)>σ(6), σ(7)>σ(8), σ(9)>σ(8).

The constraints derived from vector a give birth to 22 new expressions from o(I)/σ(2)+I to σ(9)σ(8)+I and besides the system knows that σ(1) ,σ(2),σ(3),σ(4),σ(6) and σ(7) cannot be equal to 9. By using an ordinal function the system examines now the constraints all together; 0(1) σ(1) and σ(2) >2 are for instance obtained from (A) and (B). Other values are eliminated in thismanner until no other information can be obtained. Now comes the time to choose; the choice will bear on the element which has the fewest possibilities; in this case, our system notices that only two elements can take, value 9, these are o(5) and 0(9); the tie is broken by considering the element which appears most often in the 'V constraints. So setting *σ(5)=9* allows new implications and in particular 0(7)8,0(6)^7, o(8)7. The next choice is 0(3)=8 and the first solution the system gets to is (678193425). In this case 42 solutions altogether are exhibited. Taking another case with m-(00001111) and a=(1 0000111) the program obtains 48 constraints,but here it is made obvious from vector m that : σ(I)>σ(2)iσ(3)σ(4)>σ(5) and therefore as the lowest value of σ(5) is one: σ(I)"5, σ(2)4, σ(3) > 3, σ(4) : 2; in the same manner the program

shows that $\sigma(9)\geq 5$, $\sigma(8)\geq 4$, $\sigma(7)\geq 3$, and $\sigma(6)\geq 2$. Now only one element for 5 and one for 9 are convenient: $\sigma(5)=1$ and $\sigma(9)=9$ are made necessary. As a(5) equals 0, $\sigma(6)$, $\sigma(7)$ and $\sigma(8)$ are not equal to 2 and therefore the system comes to $\sigma(4)=2$; in the same way the system deduces $\sigma(7)\neq 3$ and the only convenient antecedent for 3 is 3 : $\sigma(3)=3$; then $\sigma(8)\neq 4$ implies $\sigma(2)=4$ which produces $\sigma(6)\neq 5$. Substitutions give: $\sigma(1)=5$, $\sigma(6)=6$, $\sigma(7)=7$, $\sigma(8)=8$ in that order the unique solution has been reached directly. We notice that the problem solver uses, by decovering it implicity, the fact that $\sigma(j)=n$ can only occur for a j such that m(j-1)=1, m(j)=0 and a(j)=0; such a lemma is very useful as an aid for fruitful choices. *Many other results are in fact found by the system as it proceeds through formal manipulation of constraints*. (This permutation problem can be viewed too as a EI onto itself matching problem and some convenient graph procedures allow the system to derive other implications through graphical arguments [6]).

Let us take a last example to see an obviously polynomial with n behaviour of ALICE together with a catastrophic execution of the specific program first written.

Suppose for n=50 or greater we set: m(1)=a(1)=1 and for any i > 1 : m(i)=a(i)=0. In ALICE the constraints are interpreted and produce $\sigma(1)<\sigma(2)$ with $\sigma(2)>\sigma(3)>\sigma(4)>\ldots>\sigma(n)$; thus $\sigma(2)=n$ and (n-1) values remain for $\sigma(1)$; since none contradicts the hypotheses on vector a, (n-1) solutions are immediately printed : (n-1,n,n-2,n-3,...,1),...,(1,n,n-1,n-2,...,2). On the contrary, the execution of the first written program is not good at all; it will generate the partial permutations 12, 132,...,8 9 4 1,...,8 9 4 3 2 1,... with many and many useless attemps. Furthermore, in all of the previous cases, additional constraints like: $\sigma(1)=n/2$, $\sigma(n/2)=1$ or others, will but make the task of our general problem solver easier, as proved on machine, while in the first approach, one is forced to build new and new lemmas again and to design in each case a new specialized program.

IV - CONCLUSION.

My aim was not to claim that it is impossible to write a specific program which can work correctly on specific examples; but, there are always new particular cases to take into account and therefore some new adequate statements to add into this program. So, little by little, one is induced to write a quite large code; at this point, if we try a bit harder, we can deal with many other combinatorial problems, say for instance the eight queens, cryptarithmetic, integer mathematical programming, optimization problems and so on: *such is indeed the scope of a general program*. It has been actually shown by ALICE, in all these cases, that generality does not contradict efficiency: numerical and graphical analysis, supported by good heuristics adjusted to each particular case, allows the system to foresee quickly constraint violation even with large size problems.

- *Why can a specific program work badly?* The first reason is that it lacks one indispensable tool : formal manipulation; although we use this tool very commonly by hand, too few specialized programs are able to do so. The second, perhaps more important, reason has to do with the trivial statement i=i+1 just at the begining of our specific backtrack programs; this means that we *construct the objects we are looking for in a predetermined order. But this is unfortunately not the right way to proceed:* one element may have an imposed value or another may have significantly fewer possibilities than the others. *In one word when we write a specific program we make implicit choices : we have to decide without correct elements of decision because the missing information lies in the data. In that way, to each specific algorithm will correspond a worst case. On the opposite this can hardly be said of a general system with good mathematical knowledge together with heuristics based on rich and varied representations of objects in the universe dealt with : it discovers facts about the problem at the problem statement level not at the code level.*

- REFERENCES

[1] Arsac J.:"Syntactic source to source transforms and program manipulation". CACM 22,1(1979) 43-54.

[2] Burstall R.and Darlington J."A system which automatically improves programs" Acta Informatica 6, (1976), 41-60.

[3] Hoare C.:"An axiomatic basis for computer programming" CACM 12,10(1969),576-583.

[4] Karp R.: "On the computational complexity of combinatorial problems" Networks 5,(1975),45-68.

[5] Knuth D.:"Structured programming with GOTO statements" ACM.Comp.sur. 6,4(1974), 261-301.

[6] Laurière J.L.:"A language and a program for stating and solving combinatorial problems" Art. Int. 10, 1 (1978), 29-127.

[7] Laurière J.L.:"Specific algorithms versus general programs", GR 22, A.I. (1979).

[8] Lee et al :"An improved program synthesis and its correctness" CACM 17,4(1974),211-217.

[9] Manna Z.:"The correctness of programs". J. Comp. Sys. Sciences 3,2(1969),142-156.

[10] Manna Z,Waldinger R :"A deductive approach to program synthesis" SRI Tech. R.177 (1978).

[11] Standish et al: "Improving and refining programs by program manipulation". ACM Nat. C. (1976), 509-516.

THE ROLE OF OBJECT PRIMITIVES IN NATURAL LANGUAGE PROCESSING •

Wendy G. Lehnert and Mark H. Bursteln
Computer Science Department
Yale University
Box 2158 Yale Station
New Haven, CT 06520

A computer program, OPUS, is described which uses a set of Object Primitives to represent knowledge of physical objects and provide an organizing structure for associative memory. OPUS applies this representational system to the problem of analyzing natural language sentences dealing with objects.

1. INTRODUCTION

It is widely recognized that the process of understanding natural language texts cannot be accomplished without accessing mundane knowledge about the world [1,2,4,7]. In this paper, we are concerned with the way functional knowledge of objects, and associations between objects can be exploited in an understanding system.

Consider the sentence

(1) John opened the bottle so he could pour the wine.

Anyone reading this sentence makes assumptions about what happened which go far beyond what is stated. We assume without hesitation that the wine was poured from inside the bottle. However, there are many other interpretations which are equally valid. For example, John could have been filling the bottle rather than emptying wine out of it. Yet, some causal inference mechanism, making use of functional knowledge of objects, causes us (as human understanders) to find the common Interpretation in the process of connecting these two events causally.

In interpreting this sentence, we also rely on our knowledge of bottles and what it means for a bottle to be "open'', when interpreting the sentence. Only by drawing on knowledge of what is possible when a bottle is open are able we understand why John had to open the bottle to pour the wine out of it.

When reading the sentence

(2) John turned on the faucet and filled his glass.

we immediately assume that John filled his glass with water from the faucet. Yet, water is never mentioned in the sentence. The glass could conceivably be filled with milk from a carton. However, in the absence of some greater context which forces a different interpretation on us, we immediately assume that the glass is being filled with water from the faucet.

This paper describes a computer program, OPUS (Object Primitive Understanding System) which constructs representations of the meanings of sentences such as those above, including assumptions that a human understander would normally make. To do this requires the use of stereotypic knowledge of physical objects, captured in OPUS using Object Primitives [3] which were designed to act in conjunction with the primitives of Schank's conceptual dependency representational system [7].

A complete description of the seven Object Primitives. can be found in [3]. The two Object Primitives which we will exploit in our examples here are CONNECTOR, and SOURCE. A CONNECTOR is an object which enables some action between spatial regions, and a SOURCE is an object which "produces" another object. In addition, four associative links have been used, each pointing to a particular type of OP description. An example is the OUTPUTFROM link which connects objects such as wine with their normal SOURCE objects, in this case, wine bottles.

2. THE PROGRAM

2.1 The *Intial* Analyasis

In the processing of the sentence (1) above, the phrase "John opened the bottle" results roughly in a representation of the fact that

•This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored under the Office of Naval Research under contract N00014-75-C-1111.

John did something which caused the bottle to assume a state where the CONNECTOR description shown below applied.

```
*bottle* IS a CONNECTOR
                which ENABLES
?HUM0 <=> PTRANS ?0BJ FROM (INSIDE SELF)
     (or)
?HUM0 <=> PTRANS ?OBJ TO (INSIDE SELF)
     (or)
7HUM0 <r> ATTEND ?SENSE TO ?0BJ
                              (loc INSIDE SELF)
```

Where SELF refers to the object being described (the bottle) and ? indicates an unfilled slot.

The CONNECTOR description Indicates that something can be removed from the bottle, put into the bottle, or Its contents can be smelled, looked at, or examined by some other sense modality. This CONNECTOR description is not part of the definition of the word 'open'. It is specific knowledge that people have about what it means to say that a bottle is open. To arrive at this representation, the program retrieves from memory the OP description of what it means for a bottle to be open, stored beneath its prototype for bottles. Presumably, there is also script-like information about the different methods for opening bottles, the different types of caps (corks, twist-off, ...), etc. However, for the purpose of understanding a text which does not refer to a specific type of bottle, cap, or opening procedure, what is important is the information about how the bottle can then be used once it is opened. This is the kind of knowledge that Object Primitives were designed to capture.

2.2 Concept Driven inferences

When the phrase "so he could pour the wine." is analyzed by OPUS, the fact that the wine was poured from the bottle is not represented. This Inference is made in the program by a slot-filling demon called the CONTAINER-FINDER, attached to the primitive act PTRANS. This demon looks on the list of active tokens (a part of short term memory) for objects that might normally contain the substance poured, In this case wine. The object found must be either a container (RELATIONAL = INSIDE) with the default object contained being wine, or be listed as a SOURCE of wine via an associative OUTPUTFROM link from the memory token for wine. If either test succeeds for some object in STM, then that object is inferred to be the container poured from. In the case of (1) above, the test suceeds since wine is OUTPUTFROM wine bottles.

2.3 Causal Verification

Once the slot filling inferences have been considered, the process which attempts to make causal connections between conceptualizations is activated. This process first looks for a match between the conceptual representation for the enabled action (pouring the wine), and one of the potentially enabled acts derived from the OP description of the open bottle. For sentence (1), a match Is immediately found between the action of pouring from the bottle and the expected action generated from the CONNECTOR description of the open bottle (PTRANS FROM (INSIDE PART SELF)). The match causes a merging of the expected concept with the concept produced by the analyzer, linking the events described in the sentence and filling slots which were filled in one but not both of the original concepts.

2.4 Causal chain Construction

In processing the sentence

(3) John turned on the faucet so he could drink.

OPUS builds a token for a faucet as an active SOURCE of water. The principle expectation for SOURCE objects is that the person who "turned on" the SOURCE object wants to take control of (ATRANS) the object output from that SOURCE. This expectation has the important side effect, here, of creating a token for some water, which later is inferred as the substance John drank by a slot filling inference.

The phrase "he could drink" is represented in OPUS by an INGEST with water inferred as the liquid in the OBJECT slot. Since the expectation generated from turning on the faucet is for an ATRANS, the causal chain completion is clearly going to be more complicated here than in the previous example. When the chain connector fails to find a match between the ATRANS and either the INGEST or its instrumental PTRANS, additional inference procedures are called to generate any intermediate states that might connect these two acts. Using a combination of forward (resultatIve) and backward (causative) inference rules described by Rieger [5] in a restricted intersection search, OPUS infers that the intermediate state (John possess water) can be used to complete the causal chain.

3. CONCLUSIONS

It is important to understand how OPUS differs from previous inference strategies in natural language processing, such as that of RIeger [51. A cursory comparison of OPUS and Rieger[1]s MEMORY system reveals a number of similarities. The causative and resultative inferences used to complete the causal chain in our last example came directly from that work, and many of the demons used by OPUS are similar in flavor to the forward inferences and specification (slot-filling) inferences described by Rieger.

There are, however, two ways in which OPUS departs from the inference strategies of MEMORY in significant ways. (1] On one the level of computer implementation there is a reorganization of process control in OPUS, and [2] on a theoretical level OPUS exploits an additional representational system which allows inference generation to be more strongly directed and controlled.

In terms of implementation, OPUS integrates the processes of conceptual analysis and memory-based inference processing. By using demons, inferences can be made during conceptual analysis, as the conceptual memory representations are generated, thus eliminating much of the artificial modularity used in MEMORY more for pragmatic than theoretical reasons.

On a more theoretical level, the inference processes used for causal chain completion in OPUS are more highly constrained than was possible in Rieger*s system. In MEMORY, all possible inferences were made for each new conceptualization which was input to the program. Causal chains were connected when matches were found between inferred concepts and concepts already stored in its memory. However, the Inference mechanisms were not directed specifically to the task of making connections between concepts found in its input text. This led to a combinatorial explosion in the number of inferences made from each new input. In OPUS, forward expectations are based on specific associations from the objects mentioned, and only when the objects in the text are described in a manner that indicates they are being used functionally. During causal chain completion, at most two levels of additional forward or backward inferences are made before the procedure is exhausted, and the system stops once a match is made. Thus, there is little chance for the kinds of combinatorial explosion Rieger experienced.

OPUS makes use of a well structured set of memory associations for objects, the Object Primitives, to encode Information which can be used in a variety of Rieger's general inference classes. Because this Information is directly associated with memory representations for the objects, rather than being embodied in disconnected inference rules elsewhere, appropriate inferences for the objects mentioned can be found directly. By using this extended representational system, we can begin to examine the kinds of associative memory required to produce what appeared from Rieger's model to be the "tremendous amount of 'hidden' computation" necessary for the processing of any natural language text.

REFERENCES

[1] Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS, a frame driven dialog system. Artificial intelligence, vol. 8, No. 1.

[2] Charnlak, E. (1972). Toward a model of childrens story comprehension. AITR-266, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

[3] Lehnert, W. G. (1978). Representing physical objects in memory. Technical Report #131. Dept. of Computer Science, Yale University, New Haven, CT.

[4] MInsky, M. (1975). A framework for representing knowledge. In Winston, p. H., ed., The Psychology of Computer vision, McGraw-Hill, New York, NY.

[5] Rieger, C. (1975). Conceptual memory. In R. c. schank, ed., conceptual Information Processing North Holland, Amsterdam.

[6] Riesbeck, C. K. and Schank, R. C. (1976) Comprehension by computer: Expectation-based analysis of sentences in context. Technical Report #78, Department of Computer Science, Yale University.

[7] Schank, R. C. and Abelson, R. P. (1977) Scripts, plans *Goals AND* Understanding. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

STRUCTURED INHERITANCE NETWORKS AND NATURAL LANGUAGE UNDERSTANDING

Henry H. leitner*
Ctnttr for Research in Computing Technology
Aiken Computation Laboratory
Harvard University
Cambridge, HA 02138

Michael U. Freeman
Advanced Development Organization
Ftdtral and Sptcial Systems Group
Burroughs Corporation, P.O. Box 517
Paoli, PA 19301

One of the most salient difficulties facing the designer of a Natural Language Interface is the appropriate characterization of the semantics and pragmatics of new domains. Uhereas various syntactic parsers have been developed which are largely domain-independent (at least in principle), the possibility of accomplishing something similar in the realm of semantics and pragmatics has never been convincingly demonstrated. Focusing on the various epistemological bases for associating attributes with entities, we examine some of the requirements that this suggests need be met by a system which claims to provide a domain-independent conceptual schema core, from which domain-specific extensions can be interactively elaborated. In addition to showing how our approach differs from one which tries to arrive at canonical representations in terms of invariant semantic primitives, we sketch out ways in which it can be integrated into a general parsing mechanism.

0. INTRODUCTION

An ultimate goal of researchers in natural language (NL) processing is to have a system that interacts appropriately with human users in their own natural language. However,

- it is unfeasible at present to construct a general system which can handle all NL inputs in every domain and situational context.

Thus a revised goal might be a system that interacts intelligently with certain types of users in their own NL in particular universes of discourse (e.g., PARRY, LUNAR, SOPHIE, PLANES, SAM, ...). However,

- users may find it difficult (if not impossible) to make themselves understood in various instances;
- the effort involved in converting such systems to new domain and context coverage is CONSIDERABLE

For these reasons, a revised goal might be a system that can be easily extended or customized by users, at run time (e.g., LIFER or RED. However,

- to be generally effective, this may demand that a user be more linguistically and computationally sophisticated than can be reasonably expected.

This work was supportedin part by the Rome Ai Development Center under contrac F-30602-77-C-0197.

Thus a revised goal might be a system that can be easily shifted to new domains by its designers or maintainers (e.g., ROBOT). However,

- the real expertise concerning a new domain is generally not in the hands of the NLI (NL, interface) designers or maintainers.

Thus a revised goal might be a system that can be conveniently shifted to new domains by "knowledge base experts" who need not also be linguistic or computational specialists (e.g., KRL or KLONE). However,

- there is generally the need to start over each time from scratch in order to avoid introducing inconsistencies and incomprehensible conceptualizations into the pre-existing knowledge network.

Thus a fifth goal revision is to have a system which can be easily adapted to new domains by knowledge base experts who work within a conceptual frank work that builds upon a core of very abstract concepts which can be further generalized or specialized to represent domain-specific concepts, viewpoints, etc. This raises a number of questions that we will address in the course of this paper:

- Is it possible to develop a "core" knowledge nftwork (KNET) that can fulfill the function proposed for it above?
- Assuming that we can develop such a KNET, what additional proficiency would a domain-expert or an interactive system need in order to expand it appropriately for a particular domain or application?

Assuming that we can develop such a KNET, what additional proficiency would a domain-expert or an interactive system need in order to expand it appropriately for a particular domain or application?

What type of general parsing mechanism would be needed for automatic adaptation to any newly expanded network? Specifically, how can standard syntactic analysis mechanism be used to create conceptual "Meaning" structures, rather than purely formal syntactic objects?

What would the role of the lexicon be in such a parser, and what sorts of expertise would be needed for creating, expanding, or modifying any particular instance of such a lexicon?

Assuming that we have this NL parser which naps English (for example) into conceptual representations of the user's intent, what relation do such representations bear to formal queries that would actually realize this intent?

1. OVERVIEW OF A SIRUCIURED INHERIIANCE NEIWORK FORMALISM

Brachman developed a knowledge representation notation (SI-Nets) which has a very clear and explicit epistemology. According to Brachman, "Structured conceptual objects are organized in lattice-like networks, with inheritance between descriptions carried by structured 'cables'."* An example conceptual structure is illustrated in Figure 1:

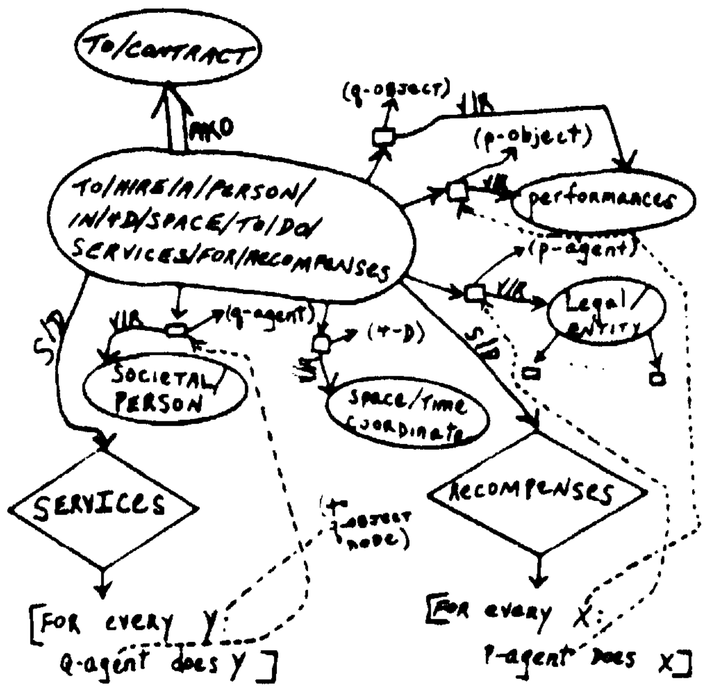


FIGURE 1: SIMPLE SI-NET

The oval shapes denote conceptual entities, while the role nodes <squares) are intended to represent the intensional description of potential role fillers in context. The third component part of the formalism is the structural description (diamond shape) which expresses interrelationships between the conceptual subparts. As described by Brachman, "these relationships [i.e., the role descriptions plus structural descriptions} give the Concept its gestalt'."

Whenever two concepts are in a generalization/specialization relationship to one another, the features they have in common can be factored out in such a way that only the differences need be represented. This is illustrated in Figure 2, where the AKO (a-kind-Qf) link represents a channel along which common properties can be inherited either intact, restricted or differentiated from a more general to a more specialized concept. Whenever a concept is specialized to the extent that all of its role-description nodes have values associated with them, it takes on a special status as an individual concept. This is denoted by a shaded oval (cf. "HARVARD" in Figure 2), and is connected to its parent concept via an INSTS link.

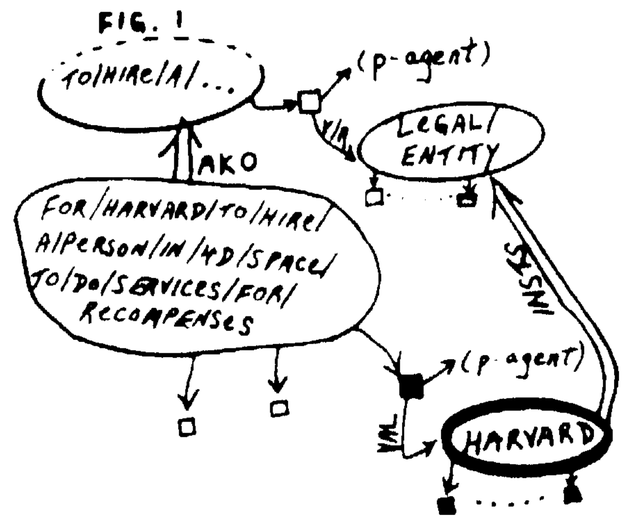


FIGURE 2: THE AKO LINK

It should be stressed that Brachnan's primary goal was to make explicit the semantics of a representation system, thereby overcoming the expressive inadequacies of the many "senantic network" node-arc schemes in recent years. He makes no claims whatsoever about the semantics of what is being represents — any concept can be represented in the SI-Net formalism any way the user chooses. Thus a particular concept might be defined in functional terms or in terms of its major parts and subparts, etc. The semantics of the world one thinks Cslhe is Modelling are not implicitly embedded in Brachnan's systen -- but the semantics of the representation system itself ar$ clear and precise.

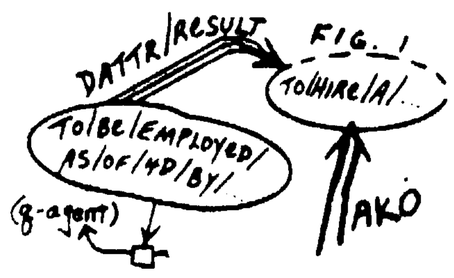
* Brachman, Rons "Theoretical Studies in Natural Language Understanding." BBN Report M3888, Cambridge, Mass., September, 1978, p. 25.

2. EXTENSIONS TO THE BASIC NETWORK FORMALISM

The preceding remarks raise some rather basic issues, such as: what does it mean for a concept to have an attribute or to have a role; how do the latter become associated with concepts originally, etc. One reason we need to concern ourselves with such issues is that if a taxonomy of the ways in which different types of attributes can be associated with concepts is developed, then there is hope -- at least in this area — of being able to give a knowledge base administrator (KBA) automated assistance in creating or extending a core semantic network. This natter is discussed further in Section 4.

Our point of departure here is the observation that the initial association of attributes with entities results from the creation of something which did not exist as such before. This does not necessarily entail the creation of a new physical object. In the context of human society, for instance, one is usually concerned with the creation of new social, political or legal entities. Thus, being born into a particular society transforms a were (human) being into a societal entity having attributes such as NAME, RESIDENCE, MARITAL-STATUS, and so on. Note that we are not speaking of the association of a particular value with an attribute, but rather of the attribute itself with a new entry, within the framework of a highly developed social or legal order.

In this framework, the relationship of state,?, resulting from specific events takes on fundamental prominence. For example, as the result of an instance of the "HIRE" event concept represented in Figure 1, a state of "EMPLOYMENT" comes into being. The semantics of the DATTR/RESULT arrow in Figure 3 is meant to capture this relationship.



FI6URE 3: THE DATTR-RESULT LIMK

In turn, this allows us to conceptualize an EMPLOYEE as a societal person put into a state of employment as a result of playing the rglf of the "second party" (or q-agent) in a particular type of contractual event, namely that represented by the -HIRE" concept in Figure 1.* The QUA link in Figure 4 is the means by which we represent this interdependence of relationships.

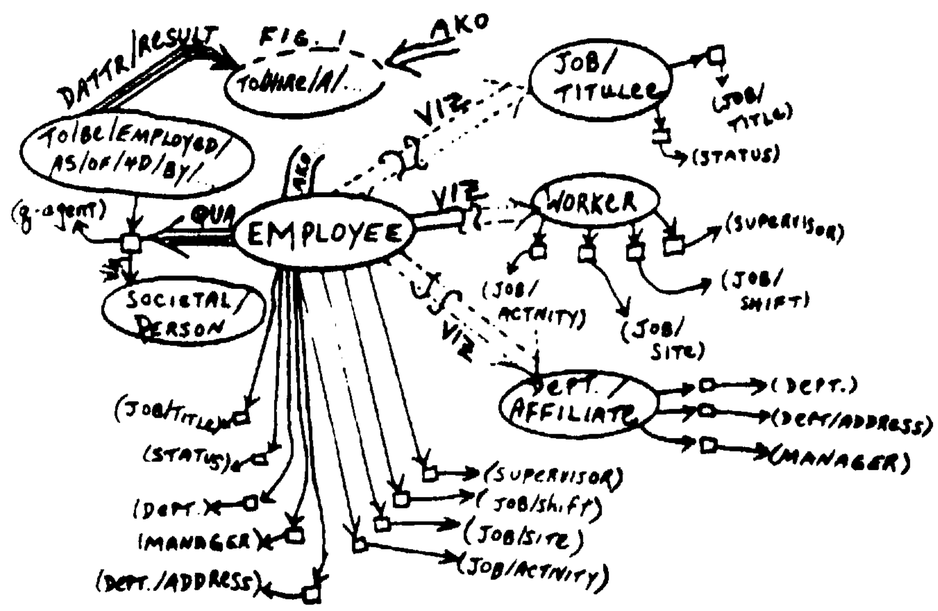


FIGURE 4: THE QUA AND VIZ LINKS

The QUA link is the primary channel over which new attributes come to be attached to already existing entity concepts. Note that this is not simply a matter of viewing a given entity from some particular perspective, although it does give rise to such possibilities, as illustrated by the VIZ links in Figure 4. In this latter case, however, the perspective is taken as already having been intrinsically established, whereas in the former case what is being represented is how such perspectives are actually to be defined. This distinction between Viz and QUA links becomes quite clear when one addresses the problem of making explicit the semantics of inheritance associated with each of the two link types.

The VIZ link has precisely the same inheritance properties as the AKO link. The only difference is that it serves to partition the multi-dimensional attribute space of a given entity type into proper subsets. This allows one to distinguish higher level entity concepts in the ''gestalt'' of a lower level entity concept having multiple inheritance paths. The reason for doing so, however, is contained within the semantics of the QUA link.

Since QUA links can derive from role description nodes within verbal concepts, the relationships between the parent role node and other role nodes within the verbal concept open up new paths for attributive inheritance that are not present in just the AKO (or VIZ) links. In the case of EMPLOYEE, for example, as illustrated in Figure 4, attributes such as ENPLOYER, DATE/PLACE-OF-NIRE, etc., result not from its AKO relationship to

* The importance of modelling ROLE-ENTITY associations has also been stressed by Bachman and Daya : "The Role Concept in Data Models." VLDB-J Tokyo, Japan, October 6-8, 1977.

SOCIETAL-PERSON, but rather to the other role description nodes in the TO/BE/EHPLOYED concept, which themselves derive fom the TO/HIRE concept via the DATTR/RESULT link illustrated in Figure 3. Now, by descending into the structural-description (S/D) nodes of these verbal concepts, one arrives at conceptual representations of how the various roles subsumed under the verbal concept are themselves related to one another, Uherever an S/D component co-references the (q-agent) node in TO/HIRE to which the EMPLOYEE concept is ultimately rooted, one has a potential source for generating a VIZ link. If one regards a TO/HIRE event, for example, as consisting of a job-offer and a job-acceptance, in which the applicant <q-agent) commits himself or herself to perform certain services in return for certain recompenses over some sort of time-span, then for each service to be performed, the employee can be viewed as its agent (e.g., WORKER), and for each recompense to be received, the employee can be viewed as its benefactive (e.g., PAYEE, GROUP-INSUREE, etc.). These are precisely the perspectives that we have represented via the VIZ links in Figure 4. In this way, an attribute such as SALARY comes down the VIZ link from PAYEE to EMPLOYEE, just as J08-SITE does from WORKER, whereas RESIDENCE comes down the AKO link from SOCIETAL-PERSON, and ENPLOYER over the QUA link from TO/BE/EMPLOYED via the DATTR/RESULT link from TO/HIRE.

The discussion up to this point has focused on how attributes come to be associated with entities through performance events which result in certain socially or legally recognized states. Ue have said relatively little about the specification of domains from which these attributes can take on meaningful values. We have simply placed a value/restriction (V/R) link on role-description nodes (pointing to such concepts as SOCIETAL/PERSON, LEBAL/ENTITY, PERFORMANCE), and have relied on the indulgence of our audience to accept this as obvious. This has led us to gloss over the determination of the AKO-link between a QUA-derived concept and the V/R concept associated with the role-description node to which the QUA points. Yet it is not necessarily obvious in what sense an essentially functionally specified entity can be regarded as "e-klnd-of'' object pure and simple. Put another way, we need to look more closely at the implications of saying that the event of casting an entity into a particular role results in the "creation'' of a new entity through its association with attributes (not just attribute-values) which it did not previously possess. In what follows, we will briefly consider how this approach leads one to view the domain specification (V/R) of attributes of this type as an event-driven state-transition network, from which secondary attributes can be derived, as dictated by whatever the pragmatic needs of a given user community might be. This also provides an extremely straightforward means for specifying certain types of data base integrity constraints, update histories, identity conditions on transformable objects, etc.

In Figure 3 we represented the concept TO/BE/EMPLOYED as connected to the event concept TO/HIRE via a DATTR/RESULT link. We're all aware, however, that one remains employed by a particular legal entity only so long as certain other events do not occur, e.g., being fired, laid off, released, retired or dying. Whenever any of these other events occurs, one enters a corresponding new state. In fact, the history of this progression from one state to another via events of the types just mentioned can itself be viewed as the value of some higher level attribute, say, EMPLOYMENT/STATUS. Just as a SOCIETAL/PERSON acquires an EMPLOYER attribute through becoming an EMPLOYEE (cf. Figure 4), so a new-born human acquires an EMPLOYMENT/STATUS through becoming a SOCIETAL/PERSON as a result of being born into a particular type of society. Part of the "definition" of such a society is precisely the types of attributes that are associated with individuals upon their birth into it. And the domain-specifications of most of these attributes are most naturally represented as constellations of event-mediated state-transition networks, such as that illustrated in Figure 5 for EMPLOYMENT-STATUS.

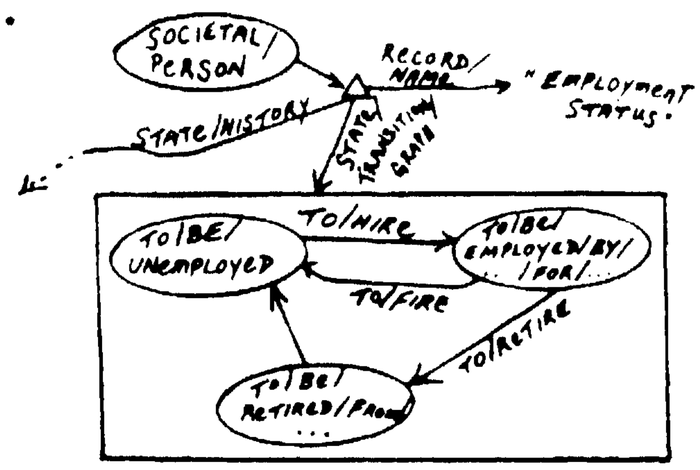


FIGURE 5: STATE AND EVENT MONITORS

For any instance of a SOCIETAL/PERSON, the actual value of its EMPLOYMENT/STATUS is the current state it happens to be in. Since the latter results from a specific event-instance, one could just as well regard the event itself as the primary value, from which a current state value can always be computed on demand. The converse, of course, does not hold, thus underscoring the derived or secondary nature of state attributes such as EMPLOYMENT-STATUS. Let us therefore restrict our attention simply to an EMPL0YMENT/DETERMININ6/EVENT/HIST0RY. In order to

capture the same sort of knowledge as that represented in the state-transition network of Figure 5, we can conceive of the attribute itself as an active object, much along the lines of an abstract data type. In addition to recording a series of events of certain types (constituting the "history'' of the participating object in this respect), such an abstract attribute would also behave as a monitor for events yet to occur. Since the set of possible events at any given moment is constrained by the previous event-history of the object, the analog of the state-transition network in Figure 5 would be an event-monitoring transition network. From this one can not only derive values for the secondary state-attributes of Figure 5, but one can also set triggers for automatic updates to preserve data base integrity in a natural way.

Consider, for instance, the case of two employees of a company who are Married to one another. Since the SPQUSE/DETERHINING/EVENT attribute of each would be monitoring, among other things, for the death of the other spouse, the value for the derived HARITAL/STATUS attribute would be subject to automatic update for both employees. This could automatically propagate updates in tax-status, group-insurance status, etc.

3. A PARSING MECHANISM BASED UPON SI-NETWORK

The usefulness of the KNET distinctions we have made so far become apparent in discussing how a conventional syntactic analyzer could guide a direct mapping from NL input into a corresponding conceptual representation, as well as for interactive construction of domain-specialized KNETs. We will consider the interpretation of the noun phrase, "MY employee". Our NL parsers lexicon, in addition to providing the syntactic processor with various grammatical/syntactic information, also contains the possible conceptual bases for the concepts (denoted by lexicon entries) which may already be present in the KNET. The lexical item NY, for instance, May be regarded as a city/state ENTITY, as a LOCATION/SITE, or as an ACTION/SITE.* The linkage between lexicon entries and the actual KNET is illustrated below for NY and ENPLOYEE:

* Note the different senses of NY in: "NY located on the East coast," "He is a NY employ and "No NY shipments were sent out."

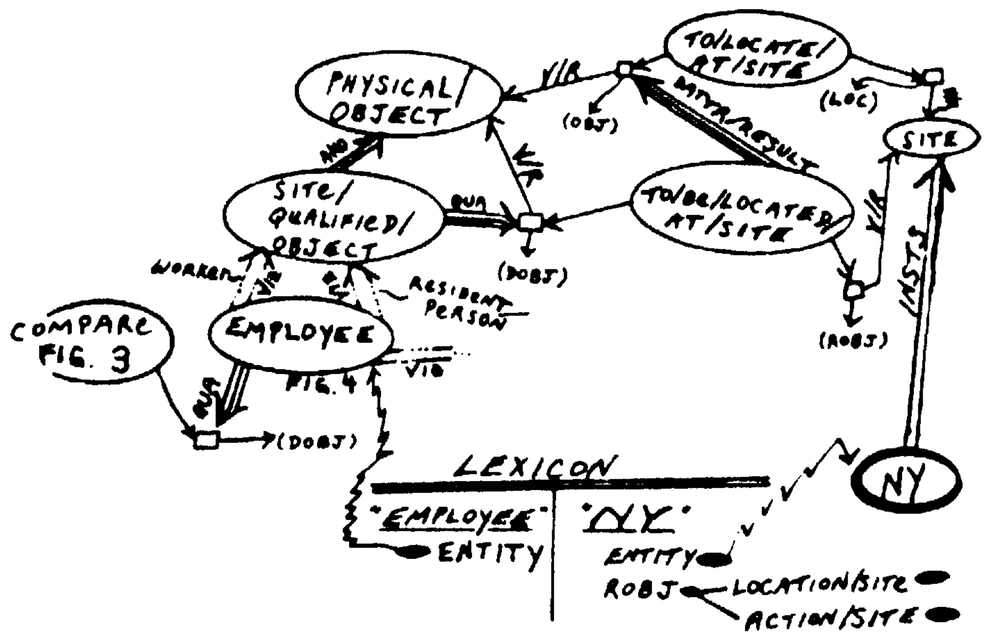


FIGURE 6: LEXICON HOOKS INTO KNET

In processing an input containing the noun phrase "MY employee", the syntactic analysis routine first identifies employes as head noun and NY as-potential modifier; the conceptual processor is sent this information in the form of a template which "says" that we have identified the potential concept NY/EMPLOYEE — for which a representation does not yet exist. But certain facts are known. The syntactic analyzer will ask the conceptual unit to try and link the NY and ENPLOYEE concepts in such a way that NY can be considered an ATTR/VALUE/REFERENCE of EMPLOYEE via the schematic template illustrated in Figure 7:

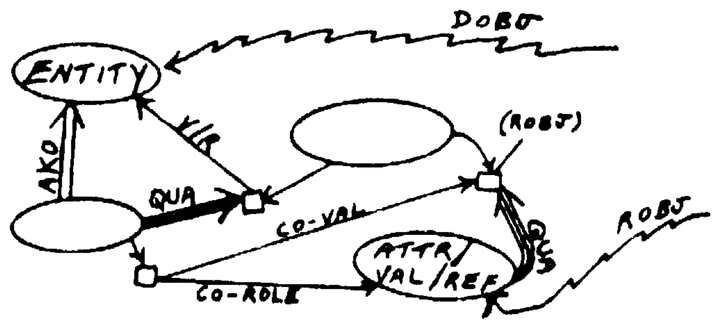


FIGURE 7: CONCEPTUAL TEMPLATE

In examining the possible ways in which a path between NY and ENPLOYEE can be established to fit the template, we find that NY could be viewed as a LOCATION/SITE. Ue therefore try the interpretation of NY as modifying a LOCATION/SITE. Procedurally we will have to build specializations of the TO/BE/tOCATED/AT/SITE concept, as well as additional specializations of other concepts in order to create the proper connections. The pattern we are trying to instantiate is one in which the domain-object (DOJB) has EMPLOYEE as its value/restriction. However, at the start of our matching we discover that the TO/LOCATE/AT/SITE concept has a v/R of PHYSICAL/OBJECT. The strategy at this point is to see if there is an AKO path between ENPLOYEE and PHYSICAL/OBJECT. Note the multiple visits we have from EMPLOYEE to PHYSICAL/OBJECT (e.g., employee as a

"site/qualified" worker or resident-person). For each possible path the parser fills in the appropriate structure, so we actually produce two interpretations of NY employees (a) an employee who lives in NY; and (b) an employee who works in NY. <We could even produce a third interpretation: a person who is born in NY -- if the KNET contained that view).

One should note that in the structural description part of concepts, there exists not only information about how the various roles are related to one another, but also implementatio-specfic information on how to actually 90 about retrieving information from a data base in a given computer environment. The same sort of building operations which generate the concept nodes leading to the representation of NY/EHPLOYEE will tailor the information retrieval routines in the SB's — so that for each conceptual representation of NY/EHPLOYEE we also have an implementation-bound procedure. The implementation-bound procedure is NOT the "meaning" of NY employee — as "proceduralists" often seen to claim. Our approach makes it clear that two very different things really occurs One is the conceptual representation building process, and the other is the process which constructs an implementation-dependent representation which can be executed to check whether any NY employee exists, who they are, how many there are, etc., depending upon the intent of the original input. The latter representation, however, really is irrelevant so far as building conceptual representations are concerned. Below is illusrated the KNET after the virtual nodes have been constructed for the parsing of NY/EMPLOYEE.

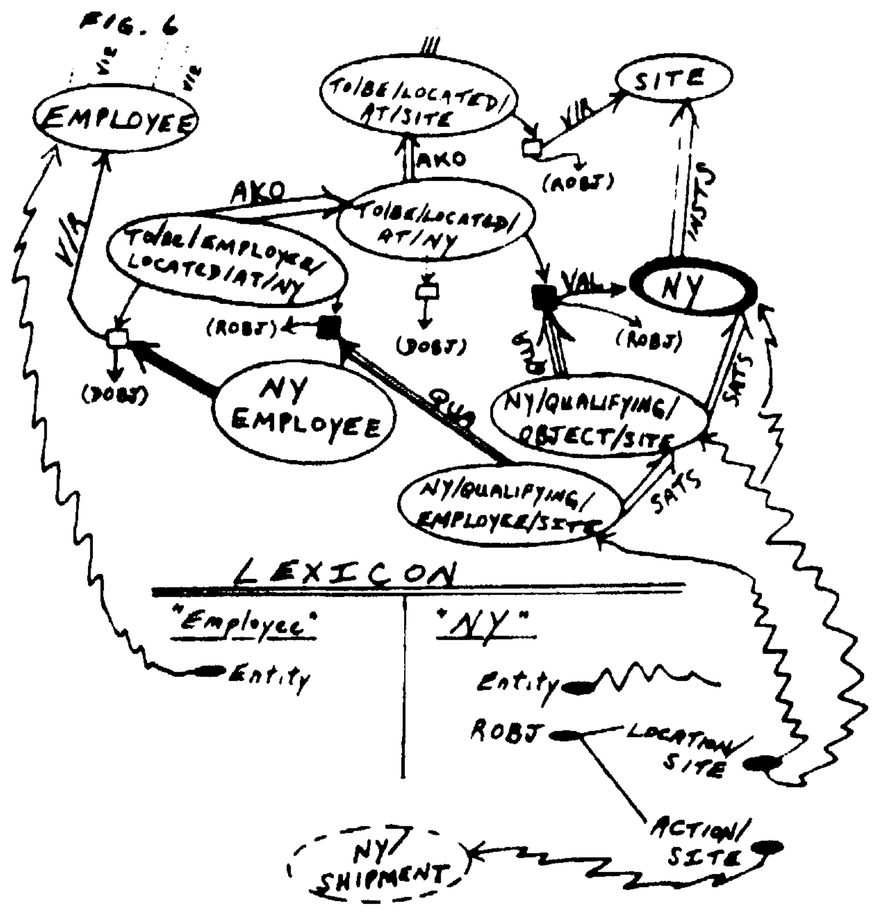


FIGURE 8: INTERPRETATION OF "NY/EMPLOYEE"

4. ADVANTAGES OF THE SI-NET FORMALIN

Nodularity has been achieved in the separation of domain independent from dependent knowledge, and the separation of syntactic, semantic (and pragmatic) expertise. No longer is the lexicon a "wastebesket" for idiosyncratic information which can't be factored out and formalized. Secondly, we have a perspicuous, comprehensible and communicable formalism which lends itself readily to conceptual/linguistic extension by a KBA who builds upon a core SI-Net for a given universe of discourse. Note that we have not gone the route of Schank and his co-workers -- who argue in favor of "semantic primitives" and the belief that all concepts can be algorithmically decomposed into their primitives (via the encoding of semantic definitions of words in terns of a small set of ACTs) for all NLs. Schank's assumptions run counter to a fairly old and well established philosophical tradition that warns against trying to arrive at a canonical decomposition of NL utterances in terms of semantic primitives. More seriously, Woods and others have pointed out that such (CD) decomposition is often undesirable (if it were possible), since necessary information which is present in the original surface encoding of the message usually gets lost. To make matters worse, the lexicons of Schank's implemented systems are enormously complex, and have never been formally characterized. The hope, then, of adapting a lexicon from one application to another seems fraught with difficulties.

A third important contribution is the delegation of syntax to what appears to be its appropriate role. Just as human parsers apparently do not build, maintain and manipulate syntactic structures as formal objects so too do we believe that our parsing technique need not construct intricate, intermediate syntactic structures. Instead, we believe that the "appropriate role" of syntax dwells more within such areas asx identifying the boundaries of phrasal constituents and establishing various "case" relationships among them; focusing in on user's Intent (i.e., helping to resolve certain forms of ellipsis and anaphora); and handling other discourse-sensitive features such as negation-quantification interaction and scoping.

Finally, we believe that the only hope for realizing truly flexible Nil's lies in the elaboration of an epistemologlcal framework firmly grounded in the social/contractual aspects of human interaction. It should then be possible to shift to new domain applications simply through the interactive specialization of a core KNET created within this framework. we hereby dedicate all such systems to Jean Jacques ROUSSEAU.

COGNITIVE ECONOMY

in Artificial Intelligence Systems

Douglas B. Lenat, Computer Science Dept., Stanford University, Stanford, Ca.

Frederick Hayes-Roth, Information Sciences Dept., The Rand Corporation, Santa Monica, Ca.

Philip Klahr, Information Science! Dept., The Rand Corporation, Santa Monica, Ca.

Intelligent system* can explore only tiny subsets of their potential external and conceptual worlds. To increase their cfTcctive capacities, they must develop efficient forms of representation, access, and operation. In this paper we develop several techniques which do not sacrifice exprcssibility, yet enable programs to (semi-)automatically improve themselves and thus increase their productivity. The basic source of power is the ability to predict the way that the program will be used in the future, and to tailor the program to expedite such UBEB. Caching, abstraction, and expectation-simplified processing are principal examples of such techniques. We discuss the use of these and other economic principles for modern AI systems. Our analysis leads to some counterintuitive ideas (e.g., favoring redundancy over minimal storage in inheritance hierarchies).

1. INTRODUCTION

When we build an AI program, we often find ourselves caught in a tradeoff between expansibility and efficiency. Many AI researchers cum language designers have focused on *exprcssibility,* the problem of rapidly constructing a working experimental vehicle. Four fundamental techniques utilized in highly *expressive* programs arc; (t) reliance upon a very-high-level language, (it) planning and reasoning at multiple levels of abstraction, (iii) inference by pattern-directed knowledge sources, and (iV) minimal, nonredundant representation, as in a canonical generalization/specialization hierarchy.

This paper addresses the second goal of AI programming, *efficiency,* getting programs to use as few resources (time, space) as possible. We present techniques which do not sacrifice cxprcssibility, yet enable programs to (scmi)-automatically improve themselves and thus increase their productivity. The basic source of power is the ability to predict the way that the program will be used in the future, and to tailor the program to expedite such uses.

The traditional approach to program optimization has assumed that the programmer characterizes the predicted program behavior (e.g., by explicitly providing assertions) or that static analysis can identify significant optimization opportunities. Three types of methods for analyzing program descriptions in this way include: (i) analyzing program flow and structure *[a la* Knuth and Dijkstra], (ii) designing data structures to be appropriate, and (iii) compiling (as in the FORTRAN H compiler) and optimizing transformations of the program (as in [1,2]).

We propose the use of methods more *dynamic* than these. Rather than improving the static description of a program, we advocate a *plastic* program structure which adapts to its operational environment. We believe that a program's "intelligence" can be increased in this way; that is, by increasing its ability to acquire appropriate knowledge, to organize that knowledge, and to refine the conditions under which that knowledge is recognized to be applicable. For any fixed quantum of manpower resources we expend, there is a limit to the size/complexity of programs which can be successfully implemented. This barrier might be overcome by programs which are self-extending, which actively reason — and reprogram themselves — to enlarge their input domain, their capabilities.

Abilities which make a program *efficient* include:

• Dynamic self-monitoring (the ability to sense, record,

[1]This research wot supported in part by the National Science Foundation under grant. MCS77-04440 and MCS77-03373.

and analyze dynamic usage) and self-modification (the ability to use that knowledge to redesign/recompile itself with more appropriate representations, algorithms, data structures; i.e., *intelligent learning.)*

• Caching of computed results (storing the results of frequently-requested searches, so they needn't be repeated over and over again; i.e., *intelligent* redundancy.)

• Expectation-filtering (using predictions to filter away expected, unsurprising data, thereby freeing up processing time for more productive subtasks; i.e., *intelligent focus of attention.)*

"Cognitive economy" is the term by which wc describe such heightened productivity. Computer programs, no less than biological creatures, must perform in an environment: an externally imposed set of demands, pressures, opportunities, regularities. Extending this analogy we find that *cognitive economy is the degree to which a program is adapted to its environment,* the extent to which its internal capabilities (structures and processes) accurately and efficiently reflect its environmental niche.

Notice that representing a corpus of knowledge as a minimal (canonical) generalization/specialization hierarchy, with intcrprctativcly-computed inheritance, is *not* cognitively economical: this technique favors cxpressionbility, compaction of representation, at the expense of performance It is true that a dog is a mammal, and a mammal is an animal, and from those two we could compute that a dog is an animal, but it is more cognitivcly economical to store one redundant arc than to recompute it frequently. Psychological studies [3] indicate just such redundancies being created in human memory.

Obviously, the economy of specific representations and related inference methods depends heavily on underlying machine architectures and costs. We assume that intelligent systems aim chiefly to produce useful results (e.g., novel hypotheses) as rapidly as possible. In short, they should search cleverly and fast. But we are advocating efficiency in *addition* to, not in *lieu* of, expres8ibility (iiiterpretability, intcrruptibility, and modiflability). In typical situations, both efficiency of compiled forms and accessibility to declarative forms are intermittently needed. These different needs point to the economic benefits of simultaneously maintaining alternative and redundant representations.

2. THE ASSUMPTIONS

Every scientific theory is constructed in a rich context of surrounding theories, methods, and standards determining which experiments arc reasonable ones to perform and which point of view to take when interpreting the results — in short, a *paradigm.* We feel it useful to articulate the "core" of our paradigm (the assumptions our theory rests upon) before delving into more detail about cognitive economy:

(i) Wc continually face searches in more or less immense spaces; intelligence is the ability to bring ap*pvopriatc* knowledge to bear, to speed up such searching. Increasing intelligence then comprises increasing knowledge, improving its organization, and refining the conditions for its applicability

(ii) Since wc want intelligent systems to access, reason about, and expand their own knowledge bases, it is useful to represent such knowledge in a clean, modular form (e.g., employing any one of the current schematized representations) Its *applicability* can be made explicit by encoding it as condition-action (if/then) rules

(iii) Current computing technology presents us with limited cycles, cheap storage, and expensive knowledge acquisition They arc the symbol manipulators we use, but were not designed for that purpose.

3. SELF-IMPROVEMENT

NOTE: Due to IJCAI length limitations, the authors have been forced to eliminate many sections of this paper; usually, at least their summaries remain. The complete version can be obtained as (5).

3.1. DYNAMICALLY MODIFYING ITSELF

Summary: Wc *illlustrate* various *ways in which* a *program might use to advantage knowledge gleaned dynamically: selecting (6] (or perhaps even discovering)*

new data structures, representations, algorithms, etc. which seem better suited to the current runtime environment, the current user, the current problem, etc.

3.1.1. Extending a Schematized Representation.

For a schematized representation,[1] "extension" could occur by defining new types of slots. The Eurisko program (succeesor to the AM program [4]) has this capability, and we briefly describe how this mechanism was developed. First, we isolated four ways in which new slots are defined from old ones:
STAR (e.g., Ancestor $=_{df}$ Parent* which means a Parent of a Parent of... of a Parent of me; the general case should be aparent);
UNION (e.g., Parent $=_{df}$ Father $\bigcup$ Mother);
COMPOSITION (e.g., Cousin $=_{df}$ ChildoSiblingoParent);
DIFFERENCE (e.g., RemoteAncestor $=_{df}$ Ancestor − Parent).

These four operators which define new types of slots (*, $\bigcup$, o, −) are called *slot-combiners*.

Next, we added to AM's knowledge base a concept for each type of slot, and a concept for each type of slot-combiner. Figure 1 shows part of the Generalizations and Star concepts. Note in particular the way in which Generalizations is defined as Genl* — i.e., immediate generalizations, *their* immediate generalizations, and so on.

Finally, we modified the way in which slot entries are accessed. To illustrate this, we choose a simple task in mathematics, whose paraphrase is, "What arc all the generalizations of the concept 'primes'?" The traditional way in which (GET PRIMES GENERALIZATIONS) would work is to examine the property list of PRIMES, looking for the attribute GENERALIZATIONS; if found, the entries listed there would be returned. Assume that there is no such property recorded for PRIMES. If we looked, we would find a property called GENL *(immediate* generalization), whose value would be NUMBERS. Similarly, the NUMBERS concept has only a GENL slot, containing the entry OBJECTS. Anyway, since there is no GENERALIZATIONS attribute, the call upon GET will return NIL (the empty list).

In Eurisko, we modified the way in which any retrieval request of the form (GET C F) operates. In case the F attribute of C has no entries (or doesn't exist), we examine the definition of F and — if one exists — try to use it to compute the entries that could legally be 6torcd on the F attribute of C More precisely, before quitting we try to (GET F DEFN), and if that succeeds we apply it to O. Let's continue the example of (GET PRIMES GENERALIZATIONS). As we stated in the last paragraph, there are none recorded. So GET now calls itself recursively; our original call is replaced by ((GET GENERALIZATIONS DEFN) PRIMES). But as Fig. 1 shows, there is no slot labelled DEFN on the concept for GENERALIZATIONS. So wc recur one more time. By now our call has become
(((GET DEFN DEFN) GENERALIZATIONS) PRIMES).

GENERALIZATIONS
ISA: Slot
SPEC: Extreme-Generalizations, Genl, Proper-Generalizations
WORTH: 875 (out of 1000)
AVG-SIZE: 7 entries
INVERSE: Specializations
ANALOGS: Supersets, Relaxations,
SLOT-COMBINER: Star
BUILT-FROM: Genl

STAR
ISA: Slot-combiner
WORTH: 488
ANALOGS: Repeat, Closure
COMBINER-FOR: Generalizations, Specializations, Ancestor, ...
DEFN: λ (s) (SUBST s for SLOT in: [λ (*c*) (CONS *c* (*self* (GET *c* SLOT)))])

DEFN
ISA: Slot
SPEC: Neccessary-Defn, Sufficient-Defn, Operational-Defn, Recursive-Defn
WORTH: 975
AVG-SIZE: 1.3 entries
ANALOGS: Statement, Algorithm
DEFN: λ (*x*) ([GET (GET *x* Slot-Combiner) Defn]
(GET *x* Built-From))

FIGURE It Gentralisations & Star & Defn concepts

'Knowledge broken in pieces called schemata, frames, concepts, beings, association lists, units, scripts,... which in turn arc merely collections of smaller pieces called properties, slots, facets, aspects.

Luckily, the DEFN concept does have a DEFN slot (see Fig. 1), so we end the recursion. Applying the entry stored there to the argument "GENERALIZATIONS," we see our original call becoming transformed into

```
|((GET (GET GENERALIZATIONS SLOT-COMBINER) DEFN)
   (GET GENERALIZATIONS BUILT-FROM))
 PRIMES]
```

We see from Fig. 1 that the slot-combiner of Generalizations is "Star/* and the argument (old slot) which it is built from is "Genl." So the entire call shrinks into (((GET STAR DEFN) GENL) PRIMES). The Star concept has an entry for its Dcfn slot; it turns the preceding call into ((X (c) (CONS c (self (GET c GENL)))) PRIMES). This function first calls for (GET PRIMES GENL), which is NUMBERS, then calls itself on NUMBERS; that in turn calls for (GET NUMBERS GENL), which is OBJECTS, and calls itself recursively on OBJECTS; that calls for (GET OBJECTS GENL), which is ANYTHING, and the next recursive call terminates when it is discovered that ANYTHING has no GENL (no immediate generalization.) The list CONStructed and returned is thus (PRIMES NUMBERS OBJECTS ANYTHING). These four items are the legal entries for the GENERALIZATIONS slot of PRIMES, according to the definition of GENERALIZATIONS.

Notationally there is no distinction between slots which are "primitive" (values actually stored as attributes on a property list) and slots which are "virtual" (values must be computed using the slot's definition). A heuristic might refer to the Generalizations of Primes without knowing, or caring, whether that initiated a single access or a dizzy chase.

To define a new kind of slot, then, one need merely specify one of the slot-combiners and list the old pre-existing slots from which it is built. Thus we might define a new slot, by creating a new concept (calling it, say, ''DG"), filling its Slot-combiner slot with the entry "Difference", filling its "Built-from" 6lot with the arguments "Generalizations Genl." This would be a new kind of slot, one which returned all generalizations of a concept except its immediate generalizations, the call (GET PRIMES DG) would return (PRIMES OBJECTS ANYTHING).

It IS only a small extension to see how new kinds of slot-combiners can be defined. For instance, one which took two old slot names as arguments, f and g, and defined a new slot which was f og o f*, would be extremely useful (e.g., in database searches). In particular the crucial 6lot "Examples" is defined as Spec* oImmed-ExsoSpec*.

We have discussed how a new slot *can* be defined; consider now how a program is to know *uhen/hew* to define a new one. The new type of slot might be defined for purely exploratory reasons (e.g., it's aesthetic to define "first cousins": the specializations of the generalizations of a concept). The slot's definition might be based soundly upon need — or the absence of need. For example, by monitoring usage, we might notice that many concepts have a large number of entries for their F slot, and infer that the F slot should be specialized into several new slots.

3.2. DYNAMICALLY MONITORING THE TASK ENVIRONMENT

Summary) *The previous* section *illustrated how* a *program might profit from knowledge it gathered dynamically. This suggests that rather than work ing on the given problem exclusively, a program may be better off to expend some time learning (about the specific problem, its broader domain, or problem-solving in general). Note this suggestion would encompass traditional education (being told), empirical research (observing), and theoretical exploration (predicting). While such "very high-level" problem-solving strategies typify human problem-sol vers (especially those of us who have rationalized spending twenty or more years "in* schoo/"), very *few programs to date have employed any of these forms of learning to improve their operation.*

4. CACHING

Summary! "Caching" *the results of computations can dramatically improve the performance of many programs. Reasoning can be brought to bear to decide whether to cache, if so what to remember, and (later) whether or not to ignore the cached value and recompute it. We often refer differently to "caching" depending upon what it is that's being retained: open-ended, inductive searches can be condensed in hindsight (i.e. cached) into heuristics, deductive searches can be cached into much less branchy algorithms, subroutines can be cached into tables of frequently called arguments and resultant values, and variable quantities have their value cached simply by the process of binding.*

Hardware designers have long recognized the tremendous gain in efficiency affordable by *caching,* recording a result, at least temporarily, in case it is called for again soon. Here we consider an analogous technique: *software cocking.* The simplest casc occurs just after computing F(x), for some function F called with argument x: simply store the value returned. Whenever F is later called, check first to sec if a cached value is stored.

A more sophisticated kind of caching would involve saving information about the *process* of computing F(x), the path that was followed, the traps, the blind alleys, the overall type of successful approach, etc. The analogue in Search is to have the subtrees intercommunicate.

To illustrate the process of caching values, consider again our old access, (GET PRIMES GENERALIZATIONS). After computing (PRIMES NUMBERS OBJECTS ANYTHING) as the answer, Eurisko simply stores that list as the value of the GENERALIZATIONS attribute of PRIMES. If the same call to GET is reissued, it tries to access this very spot. Though it had failed previously, it now would succeed, and it would return the cached list almost instantly. Thus, with but a small one-time cost, our program might run as quickly as if all slots were primitive.

Note that in originally computing Generalizations of Primes, it was necessary to call for (GET GENERALIZATIONS DEFN), and the value of this virtual slot was also computed. The Eurisko policy is to cache this value also. It is useful because, when a request such as (GET DUCK GENERALIZATIONS) is received, it would otherwise have to be recomputed all over again. The definitions of slots arc very slowly — if ever — changing, hence the rccomputation of Defn of Generalitations is quite a rare event. Caching that value must be cost-effective in the long run.

In general, we see that caching a value for slot F of concept C is most applicable when the value can be expected to vary quite infrequently. In Eurisko, this gives us the flexibility to redefine a slot's definition if we wish, but (since this change of representation will be rare) the program will run just about as efficiently as if that capability were absent. This is analogous to compiling: so long as the definition of a function doesn't change too often, it's undisputcdly worthwhile to compile it.

What happens if the value *docs* change? Eurisko handles this in a passive manner: it doesn't attempt to maintain a completely consistent, up-to-date knowledge base. Rather, when a call on GET is executed, and a cached value is encountered, a formula then determines whether to accept that cache, or to ignore it and recompute a fresher value. Each call on GET is supplied with a few extra parameters which specify how much cpu time and space are budgeted for this particular function call, whether the user may be queried about this, how recent a cache must be to be acceptable, and a minimum amount of resources which must have been expended at the time the cache was written.

So (GET PRIMES GENERALIZATIONS 200 40 NIL 3 0) would allow any cached value to be accepted if it appeared that recomputing would take more than 200 milliseconds, or would use up more than 40 list cells, or if the value had been put there less than three Tasks ago. Otherwise, the cache would be ignored, a newer value would be computed, and it would be stored away. With it would also be recorded the following information: (i) the fact that it was a cached value, not a primitive one, (ii) how long it took to compute, (iii) how many list cells it used up in computing this value, (iv) the current Task number. The above call on GET might result in the following value being stored on the GENERALIZATIONS slot of PRIMES: (Vache* (PRIMES NUMBERS OBJECTS ANYTHING) 54 8 0).

[At thif point In |5] comes a discussion contrasting our ideas with psychological Ideas of economy. Similarly omitted here is a lengthy section containing heuristics for Storing and Updating cached values: when (not) to, how to, and what to.)

5. Expectation-Filtering

Summary: *For efficiency's sake,* an *intelligent system should be willing and able to add new facts, but should be eager to add surprising new facts. Surprises can only be noticed by contrast with expectations, so* an *intelligent system should maintain a context of expectations and Filter incoming observations against that. Furthermore, expectations and surprises can aid an intelligent system in comparing its model and processing of the domain to the real world. Through such monitoring,* discrepancies may *be found and diagnosed, leading to changes in the model making it more* consistent with observed behavior. Such expectations can be used *to focus and filter intelligent processing.*

6. CONCLUSIONS

Summary! We identified three characteristics that serve the needs of intelligent adaptation. First, we showed that most inferential systems can benefit from learning about their task environment and their own behavior. For example, they can exploit new schemata or different slot-names to simplify and restructure their knowledge. Second, the need to explore large search spaces with some repetitive regularity motivates the use of caching to save partial results. Wc described a variety of techniques to implement caching, and explained how caching specific results is one of a spectrum of methods that can make trade-offs among precision, speed, certainty, and generality. The third dynamic capability wc identified was expectation-filtering. In general, intelligent systems need to exploit their knowledge about typicality to reduce their cognitive load and increase their attention to important data. In many situations, we believe that expectations can both reduce processing requirements for handling ordinary events as well as simplify the identification of surprising events.

In the years to conic, AI programs will employ greatly expanded knowledge bases and, as a consequence, they will explore increasingly open-ended problem 6paccs. Already, a few existing systems show signs of having more potentially interesting things to do than they have resources to pursue (eg., AM, Eurisko). In the past decades of intelligent systems R& D, several design concepts have emerged in response to contemporary needs for creating ever larger knowledge bases. For example, many researchers proposed multiple levels of abstraction and automatic property inheritance as keystones of efficiency or "cognitive economy." Wc believe that the value of such mechanisms derives largely from their usefulness in describing initial knowledgebases. Once an intelligent system begins to explore the consequences of its knowledge and to solve novel problems in a dynamic environment, it needs to adapt its knowledge to achieve faster and more profitable retrievals.

A theory of cognitive economy should explain why knowledge needs to be adapted and should prescribe how to do it. In this paper, we have tried to lay the groundwork for such a theory.

In conclusion, wc have tried to 6how what cognitive economy is and is not. It docs not consist of a set of static knowledge-base design principles, such as those proposing taxonomic concept structures with automatic property inheritance. Rather, cognitive economy is a feature of those intelligent systems that learn to solve problems efficiently and consequently realize more of their lifetime potential. Toward that end, wc have proposed an initial set of three basic design characteristics. We anticipate these characteristics will find widespread application in many future AI systems. As knowledge bases expand and basic software obstacles are overcome, AI systems will increasingly address the same question facing intelligent humans: "What would I most like to accomplish next, and how can I do that economically?"

Acknowledgements

We wish to thank Dave Barstow, John Seely Drown, Bruce Buchanan, Ed Feigenbaum, Cordell Green, Greg Harris, Barbara Hayes-Roth, Allen Newell, Herbert Simon, and Don Waterman for several discussions which have left their mark in this paper.

References

[1] Balzer, R., Goldman, N., and Wile, D. "Informality in program specifications,'" 5IJCA1, Cambridge, 1977, 389-397.

[2] Darlington, J. and Burstall, R. M. "A system which automatically improves programs," 3IJCAI, Stanford, 1973, 479-485.

[3] Hayes-Roth, B. and Hayes-Roth, F. "Plasticity in memorial networks," Journal of Verbal Learning and Verbal Behavior, 14, 1975, 506-522.

[4] Lenat, D. B "AM: an artificial intelligence approach to discovery in mathematics as heuristic search," Memo A1M-286, Stanford AI Lab, 1976.

[5] Lenat, D.B., Hayes-Roth, F., and Klahr, P., "Cognitive Economy," Memo HPP-79-15, Heuristic Programming Project, Computer Science Dept., Stanford University, 1979; also issued as Rand Report N-1185, Rand Corporation, Santa Monica, Ca., 1979.

[6] Low, J. "Automatic coding: choice of data structures/' Memo AIM-242, Stanford AI Lab, 1974.

THE APPLICATION OF ARTIFICIAL INTELLIGENCE TECHNIQUES TO COOPERATIVE DISTRIBUTED PROCESSING

Victor R. Lesser and Daniel D. Corkill

Computer and Information Science
University of Massachusetts
Amherst, Massachusetts, 01003

Knowledge-based Artificial Intelligence (AI) systems have incorporated mechanisms which resolve uncertainty. This uncertainty stems from incompleteness and noise in input data and from errorful processing.

Resolution of uncertainty is also an important issue in the design of distributed processing systems. Uncertainty is introduced in these systems from the use of incomplete and inconsistent local data bases and from errorful communication channels.

The mechanisms used in knowledge-based AI systems provide a model for the design of distributed algorithms which can resolve uncertainty as an integral part of their problem solving activity. Use of such algorithms in a distributed processing system makes possible a reduction in the amount of inter-node communication required to resolve uncertainty. This reduction in communication requirements allows effective distribution of applications that are impractical using current approaches to the design of distributed algorithms.

1. INTRODUCTION

Conventional approaches to distributed-system design can be characterized by their emphasis on the maintenance of correctness in all aspects of the distributed computation. The distributed processing system is organized so that a processing node's local data bases contain exact copies of appropriate portions of the overall problem solving data base needed by the node's algorithms [1, 2].

In these systems, a node rarely needs the assistance of another node in carrying out its problem solving function. We call this type of distributed processing decomposition completely-accurate, nearly-autonomous (CA/NA), because each node's algorithms operate on complete and correct information ("completely-accurate") and because each node usually has in its local data base the information it requires to complete processing ("nearly-autonomous"). When this information is not locally available, a node must request another node to calculate the information, which is returned as a complete and correct result.

The CA/NA approach, however, is not suitable for applications in which algorithms and control structures cannot be replicated or partitioned based on the distribution of data in the network. In this situation, a CA/NA system is very expensive to implement due to the high communication and synchronization costs required to guarantee completeness and consistency of the local data bases. We feel that the almost exclusive use of the CA/NA approach has severely limited the application areas to which distributed processing has been effectively applied. This is especially true for distributed interpretation and monitoring applications. Although apparently well-suited to distributed processing implementations, these applications are difficult to implement effectively using CA/NA techniques. This difficulty arises because a non-local view of the sensory data is required to perform the processing.

An alternative and new approach to structuring distributed problem solving systems is to allow Incorrect and inconsistent intermediate results and a range of acceptable answers. In this approach, inter-node communication can be reduced, at the risk of inconsistency and incompleteness of local views and the possibility of unnecessary, redundant, or incorrect processing. We call a system with this problem solving structure functionally-aceurate (FA) because it exhibits correct (within tolerance) input/output behavior but is distinct from completely-aceurate problem solving structures, Tn which all intermediate aspects of the computation are required to be correct and consistent.

2. A MODEL FOR FUNCTIONALLY-ACCURATE, COOPERATIVE *DISTRIBUTED* SYSTEMS

FA distributed systems are more complex than CA/NA distributed systems because the algorithms and control structures operate on

This research was supported by National Science Foundation Grant MCS78-042112.

local data bases which are incomplete, inconsistent, and possibly errorful. In order to resolve the uncertainties in these local data bases and still keep communication bandwidth low, nodes must exchange partial results (at various levels of abstraction) and share common goals. Since new information may be based on processing which used incomplete or incorrect data, an Iterative, coroutine type of node interaction Ts requireD Eo resolve uncertainty This type of interaction leads us to view such a distributed system as a "cooperative network of interrelated tasks." Therefore, we call such FA systems functionally-accurate, cooperative (FA/C).

In FA/C systems, it is often appropriate to have the control of cooperation among the nodes decentralized and implicit. Each node uses its local estimate of the state of network problem solving to control its processing (i.e., what new information to generate) and its transmissions to other nodes [3]- This allows node activity to be self-directed. For instance, if a node does not receive an appropriate partial result in a given amount of time, it is able to continue processing, utilizing whatever data are available at that time.

When organizing a FA/C distributed system, it is appropriate to think of the system as being synthesized from local systems operating at each node. This is in contrast with the way a CA/NA distributed system is thought of: a centralized system distributed over a network, with each piece (node) in the decomposition viewed as a part of the whole system.

In a FA/C system, it is natural to think of dealing with uncertainty and errors in control, data, and algorithms caused by the distribution as an integral part of the network problem solving process. In fact, additional mechanisms required to handle hardware, communication, and processing errors may be unnecessary given that uncertainty resolving mechanisms are already an integral part of the distributed system architecture.

3. KNOWLEDGE-BASED AI AND COOPERATIVE DISTRIBUTED SYSTEMS

We feel the key to the design of distributed systems is to incorporate mechanisms which can deal with uncertainty and error as an integral part of their problem solving approach. Knowledge-based interpretation algorithms such as Hearsay-II [4] and MSYS [5] are examples of algorithmic structures that can resolve errors in this way. Problem solving in these systems involves the examination of many alternative partial solutions in order to construct a consistent overall solution. This style of problem solving is required because of uncertainty (error) in data and the incomplete, approximate, and inconsistent nature of knowledge used in these systems.

The exploration of alternative partial solutions takes the form of a search process in which a solution is constructed through the incremental piecing together of mutually constraining or reinforcing partial solutions. These partial solutions arise both from application of diverse knowledge to the same aspects of the problem and from application of the same knowledge to diverse aspects of the problem. If sufficient constraints are available during this search process, incorrect data and decisions based on those data will naturally die out. In this way, uncertainty is resolved as an integral part of the problem solving process.

In many knowledge-based systems, the number of possible partial solutions is very large. In general, the more uncertainty that exists, the larger the number of alternatives that must be explored. If there exists a large amount of uncertainty in the data and (in the approximate nature of) knowledge, a significant amount of search is required. In order to limit the search, it is important to focus quickly on information which constrains the search space. Therefore, problem solving in these systems is often asynchronous and opportunistic: there is no a priori order for decision making, and decisions, if they look promising, are tentatively made with incomplete information and later re-evaluated in the light of new information.

Due to the asynchronous nature of processing and the existence of diverse and overlapping knowledge in these systems, a solution may be derivable in many different ways (i.e., different ordering sequences of incrementally constructed solution components and possibly different solution components).

Another focusing technique used in some knowledge-based systems is to structure the search space into a loose hierarchy of increasingly more abstract representations of the problem. Using this structure, a high-level partial solution developed in an opportunistic way can be used to constrain the search.

We feel this approach to problem solving provides a basis for the development of design methodologies for FA/C distributed systems. The mechanisms used in these problem solving systems to resolve error from incorrect and Incomplete data and knowledge can also be used to structure distributed algorithms so that they will work effectively with incomplete and inconsistent local data bases. Let us examine these mechanisms from this viewpoint.

Asynchronous Nature of Information Gathering / Reduced Need for Synchronization --Problem solving is viewed as an incremental, opportunistic, and asynchronous process. Because of this style of problem solving, a node does not have an a priori order for processing information and can exploit incomplete local information. Thus, the processing order within nodes, and the transmission of information among nodes, does not need to be synchronized.

Use of Abstract Information / Reduced Inter-Node Communication Bandwidth Requirements -- the ability to use abstract information permits nodes to cooperate using messages having high information content. This reduces the inter-node communication bandwidth needed for effective cooperation.

Resolution of Uncertainty through Incremental Aggregation / Automatic Error 'Resolution -- Errors and uncertainty are implicitly resolved when partial results are aggregated and compared with alternative partial solutions. This incremental method of problem solving allows a distributed system to detect and reduce the impact of incorrect decisions caused by incomplete and inconsistent local data bases and by hardware malfunction.

Problem Solving as *a* Search Process / Inter-node Parallelism -- Because many alternative partial solutions need to be examined, parallel search by different nodes is possible. Furthermore, the additional uncertainty caused by incomplete and inconsistent local data bases can be traded off against more search. To the degree that this extra search can be performed in parallel without proportionally more inter-node interaction, communication bandwidth can be lowered without significant degradation in network processing time.

Multiple Paths to Solution / Self-Correcting Behavior -- Because there are many paths to a solution, it is possible to correct for what would be considered fatal *errors* in a normal distributed system. In addition, system reliability can be varied (at the cost of additional processing and inter-node communication) without modifying the basic problem solving structure. This variability is achieved through appropriate selection and focusing of local node activity.

Knowledge-based systems use a number of additional .mechanisms to implement uncertainty resolution. These mechanisms are also important in the effective decomposition of data and function in a distributed system, and include: (1) integrated representation of alternative partial solutions, (2) data-directed control structures, (3) focus of attention strategies for dynamic allocation of resources, and (4) generator control structures which incrementally generate credibility-ordered alternative hypotheses. (A more detailed discussion of these points and of the appropriateness of knowledge-based AI as a basis for distributed problem solving systems is contained in a longer version of this paper [6].)

4. EXPERIMENTS IN FUNCTIONALLY-ACCURATE, COOPERATIVE DISTRIBUTED PROCESSING

In order to test the relevance of knowledge-based AI techniques to distributed problem solving, a number of distributed applications based on these techniques have

One such application is distributed signal interpretation. In this application the Hearsay-11 architecture was used as the basic model for interpretation. The distributed interpretation architecture based on this model was structured as a network of Hearsay-II systems. Each Hearsay-II system (node) in the network had a limited view of the complete problem solving data base, a limited set of knowledge sources, and transmitted only a limited subset of its (partial) results to a limited set of the nodes.

An experiment using this distributed architecture indicates that the basic uncertainty resolving mechanism of the Hearsay-II architecture can also correct for error introduced by the use of incomplete and inconsistent local data bases. In addition, this mechanism can also handle errors resulting from communication loss [3].

A second application studied is distributed network traffic-light control. In this application, the distributed algorithm was based on a centralized, sequential relaxation (iterative refinement) algorithm called SIGOP-II [7]. SIGOP-II computes optimal traffic-light settings for a network of lighted intersections.

Experiments with a distributed version of this algorithm show that good, but not optimal, solutions can be generated. We have found it difficult to reproduce the performance of the sequential version without significant additional inter-node communication and synchronization. We attribute this difficulty to the strong non-local interaction among traffic-light settings [8]. In these experiments, the uncertainties introduced in the distributed version of SIGOP-II cannot be resolved completely by the distributed relaxation process.

The experiments with these two applications indicate that knowledge-based AI techniques are potentially useful in FA/C distributed systems. However, much work needs to be done to understand the appropriateness of a given knowledge-based AI technique to a particular distributed application.

5. CONCLUSION

We feel methodologies can be developed for functionally-accurate, cooperative (FA/C) distributed systems in which the distributed algorithms and control structures function with both inconsistent and incomplete data. These methodologies are necessary in order to extend the range of applications that can be effectively implemented in distributed environments.

FA/C problem solving structures are also important to the implementation of complex applications in centralized environments. These applications are often organized in the form of a collection of independent modules. In such a structure it can be conceptually difficult to develop and expensive to maintain

a complete and consistent centralized problem solving data base with which the modules interact. Techniques which permit relaxation of completeness and consistency requirements would be a significant aid in the development and maintenance of these (logically distributed) systems.

We feel there are two concepts that form the basis of FA/C distributed methodologies:

1. To view a FA/C distributed system as a network of cooperating systems which share common goals. Each system is able to perform significant local processing on incomplete and inconsistent non-local information.

2. To handle the uncertainty in control, data, and algorithms introduced by distribution as an integral part of the problem solving process.

Techniques developed in the context of knowledge-based AI systems provide a basis for implementing both concepts.

ACKNOWLEDGMENTS

The authors would like to acknowledge the research efforts of Lee Erman on distributed Hearsay-II and Richard Brooks on network traffic control which *are* discussed in this paper. We would also like to acknowledge the very useful criticisms received from Elliot Soloway.

REFERENCES

[1] Peebles, R., and E. Manning. "System Architecture for Distributed Data Management." Computer. 11:1 (1978) 40-47.

[2] Bernstein, P.A., et al. "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case)." IEEE Transactions on Software EngIneerIng SE-4:3 (1978) 154-168.

[3] Lesser, V.R., and L.D. Erman. "An Experiment in Distributed Interpretation." Technical Report CMU-CS-79-120, Computer Science Department, Carnegie-Mellon University, May 1979.

[4] Erman, L.D., and V.R. Lesser. "The Hearsay-II System: A Tutorial." Chapter 16 in W.A. Lee (Ed.), Trends in Speech Recognition. Englewood Cliffs, N.J.: Prentice-Hall, 1979.

[5] Barrow, H.G., and J.M. Tenenbaum. "MSYS: A System for Reasoning About Scenes." AI Center Technical Note 121, Stanford Research Institute, 1976.

[6] Lesser, V.R., and D.D. Corkill. "The Application of Artificial Intelligence Techniques to Cooperative Distributed Processing." Technical report, COINS, University of Massachusetts, Amherst, February 1979.

[7] Lieberman, E.B., and Woo. "SICOP-II: A New Computer Program for Calculating Optimal Signal Timing Patterns." Transportation Research Record. Report 596, 1976.

[8] Brooks, R.S., and V.R. Lesser. "Distributed Problem Solving Using Iterative Refinement." Technical report, COINS, University of Massachusetts, Amherst, May 1979.

TIME IN ROBOTS AND DIALOG SYSTEMS

L. Litvintseva and D.A. Pospelov

Academy of Sciences, USSR
Computer Center
Vavilova Str.
117333 Moscow
USSR

ABSTRACT

An approach is outlined to the problem of understanding of temporal relations in natural language texts by a computer. Discussed are the principles of constructing of a time logic, appropriate for the application in the artificial intelligence systems. Means for constructing a time logic for text analysis in robots and dialog systems are proposed. TIMER, the processor implementing "time understanding" in DILOS (a system of natural language man-machine communication) is briefly described.

References

Briabrin, V.M., Pospelov, D.A. "DILOS: a dialog system for information retrieval, logical inference and computation," AI and OA-Systems, IIASA Publ., 1976.

*Paper not received in time to appear in full in the Proceedings.

A DEDUCTIVE APPROACH TO PROGRAM SYNTHESIS

Zohar Manna
Artificial Intelligence Laboratory
Stanford University

Richard Waldlnger
Artificial Intelligence Center
SRI International

ABSTRACT

Program synthesis Is the systematic derivation of a program from a given specification. A deductive approach to program synthesis Is presented for the construction of recursive programs. This approach regards program synthesis as a theorem-proving task and relies on a theorem-proving method that combines the features of transformation rules, unification, and mathematical induction within a single framework.

MOTIVATION

The early work In program synthesis relied strongly on mechanical theorem-proving techniques. The work of Green [1069] and Waldlnger and Lee [1969], for example, depended on resolution-based theorem proving; however, the difficulty of representing the principle of mathematical Induction in a resolution framework hampered these systems in the formation of programs with iterative or recursive loops. More recently, program synthesis and theorem proving have tended to go their separate ways. Newer theorem-proving systems are able to perform proofs by mathematical Induction *(e.g.,* Boyer and Moore [1976]), but are useless for program synthesis because they have sacrificed the ability to prove theorems involving existential quantifiers. Recent work in program synthesis *(e.g.,* Burstall and Darlington [1977] and Manna and Waldlnger [1979]), on the other hand, has abandoned the theorem-proving approach, and has relied Instead on the direct application of transformation or rewriting rules to the program's specifications; In choosing this path, these systems have renounced the use of such theorem-proving techniques as unification or induction.

In this paper, we describe a framework for program synthesis that again relies on a theorem-proving approach. This approach combines techniques of unification, mathematical induction, and transformation rules within a single deductive system. We will outline the logical structure of this system without considering the strategic aspects of how deductions are directed. Although no implementation exists, the approach is machine oriented and ultimately Intended for Implementation in automatic synthesis systems.

In the next section, we will give examples of specifications accepted by the system, in the succeeding sections, we explain the relation between theorem proving and our approach to program synthesis. This paper is an abbreviated version of a Stanford University and SRI International technical report, which includes more detailed discussion and some complete examples to illustrate the application of the method.

SPECIFICATION

The specification of a program allows us to express the purpose of the desired program, without indicating an algorithm by which that purpose is to be achieved. Specifications may contain high-level constructs that are not computable, but are close to our way of thinking. Typically, specifications involve such constructs as the quantifiers *for all* ... and *for some*..., the set constructor {*x*: ...}, and the descriptor *find z such that*. . . .

For example, to specify a program to compute the remainder of dividing a nonnegative integer *i* by a positive integer *j*, we would write

$$rem(i, j) \Leftarrow \text{find } z \text{ such that } (\text{for some } y,\ i = y \cdot j + z \text{ and } 0 \le z \text{ and } z < j) \text{ where } 0 \le i \text{ and } 0 < j$$

Here, the *input condition*

$$0 \le i \text{ and } 0 < j.$$

expresses the class of legal inputs to which the program is expected to apply. The *output condition*

$$\text{for some } y,\ i = y \cdot j + z \text{ and } 0 \le z \text{ and } z \le j$$

describes the relation the output *z* is intended to satisfy. (Note that, for simplicity, we omit type requirements such as *integer*(*i*).)

In general, we are considering the synthesis of programs whose specifications have the form

$$f(a) \Leftarrow \text{find } z \text{ such that } R(a, z)$$

where $P(a)$.

Thus, in this paper we limit our discussion to the synthesis of applicative programs, which yield an output but produce no side effects. To derive a program from such a specification, we attempt to prove a theorem of the form

for all a,
if $P(a)$
then for some z, $R(a, z)$.

The proof of this theorem must be constructive, in the sense that it must tell us how to find an output z satisfying the desired output condition. From such a proof, a program to compute z can be extracted.

BASIC STRUCTURE

The basic structure employed in our approach is the *sequent*, which consists of two lists of sentences, the *assertions* $A_1, A_2, \ldots, A_m$, and the *goals* $G_1, G_2, \ldots, G_n$. With each assertion or goal there may be associated an entry called the *output expression*. This output entry has no bearing on the proof itself, but records the program segment that has been constructed at each stage of the derivation (cf. the "answer literal" in Green [1969]). We will denote a sequent by a table with three columns: assertions, goals, and output. Each row in the sequent has the form

assertions	*goals*	*output*
$A_i(a, x)$		$t_i(a, x)$

or

	$G_j(a, x)$	$t_j(a, x)$

The meaning of a sequent is that if all instances of each of the assertions are true, then some instance of at least one of the goals is true; more precisely, the sequent has the same meaning as its *associated sentence*

if for all x, $A(a, x)$
then for some x, $G(a, x)$

where a denotes all the constants of the sequent, x denotes all the free variables, $A(a, x)$ denotes the conjunction of all the assertions $A_i(a, x)$, and $G(x, a)$ denotes the disjunction of all the goals $G_j(a, x)$. (In general, we will denote constants or tuples of constants by $a, b, c, \ldots, n$ and variables or tuples of variables by $u, v, w, \ldots, z$.) If some instance of a goal is true [or some instance of an assertion is false], the corresponding instance of its output expression satisfies the given specification. In other words, if some instance $G_j(a, e)$ is true [or some instance $A_i(a, e)$ is false], then the corresponding instance $t_j(a, e)$ [or $t_i(a, e)$] is an acceptable output.

Note that: (1) an assertion or goal is not required to have an output entry; (2) an assertion and a goal never occupy the same row of the sequent; (3) the variables in each row are "dummys," which we can systematically rename without changing the meaning of the sequent.

The distinction between assertions and goals is artificial, and does not increase the logical power of the deductive system. In fact, if we delete a goal from a sequent, and add its negation as a new assertion, we obtain an equivalent sequent; similarly, we can delete an assertion from a sequent, and add its negation as a new goal, without changing the meaning of the sequent. This property is known as *duality*. Nevertheless, the distinction between assertions and goals makes our deductions easier to understand.

If initially we are given the specification

$f(a)$ <= *find* z *such that* $R(a, z)$
where $P(a)$,

we construct the initial sequent

Assertions	*Goals*	*Output*
P(a)	*R(a, z)*	*z*

In other words, we assume that the input condition *P(a)* is true, and we want to prove that for some z, the goal *R(a, z)* is true; If so, *x* represents the desired output. Quantifiers have been removed by the usual skolemizatlon prooedure (see, *e.g.,* Nllsaon [1971]). The output x Is a variable, for which we oan make substitutional the input *a* Is a constant.

The Input condition *P(a)* Is not the only assertion In the sequent; typically, simple, basic axioms, such as u = u, are represented as assertions that are tacitly present in all sequents. Many properties of the subject domain, however, *are* represented by other means, as we shall see.

The deductive system we describe operates by causing new assertions and goals, and corresponding new output expressions, to be added to the sequent without changing Its meaning. The process terminates if the goal *true* (or the assertion *false)* Is produced, whose corresponding output expression consists entirely of primitives from the target programming language; this expression is the desired program.

Note that this deductive procedure never requires us to establish new sequents or (except for strategic purposes) to delete an existing assertion or goal. In this sense, the approach more resembles resolution than "natural deduction."

In the remainder of this paper we outline the deductive rules of our system, and we present two complete examples illustrating the application of the system to program synthesis.

SPLITTING RULES

The splitting rules allow us to decompose an assertion or goal into its logical components. For example, if our sequent contains *an* assertion of form *F and G,* we can introduce the two assertions *F* and *C* into the sequent without changing its meaning. We will call this the *andsplit rule* and express it in the following notation:

the *andsplit rule*

assertions	*goals*	*output*
F and G		*t*
F		*t*
G		*t*

Similarly, we have the *orsplit rule* If our sequent contains a goal of form *F or G*, we can Introduce the two goals *F* and *G* Into our sequent; and the *if split rule:* If our sequent contains a goal of form *if F then G*, we can Introduce the new assertion *F* and the new goal G.

There is no orsplit rule for assertions or andspllt rule for goals. Note that the output entries for the consequents of the splitting rules are exactly the same as the entries for their antecedents.

TRANSFORMATION RULES

Transformation rules allow one assertion or goal to be derived from another. Typically, transformations are expressed as conditional rewriting rules

$$r \Rightarrow s \quad \text{if } P$$

meaning that In any assertion, goal, or output expression, a subexpression of form r can be replaced by the corresponding expression of form J, provided that the condition P holds. We never write such a rule unless r and *s* are equal terms or equivalent sentences, whenever condition *P* holds. For example, the transformation rule

$$u \in v \;\Rightarrow\; u = head(v) \text{ or } u \in tail(v) \quad \text{if } islist(v) \text{ and } v \neq []$$

expresses that an element belongs to a nonempty list if it equals the head of the list or belongs to its tail. (Here, $head(v)$ denotes the first element of the list v, and $tail(v)$ denotes the list of all but the first element.) The rule

$$u|0 \;\Rightarrow\; true \quad \text{if } integer(u) \text{ and } u \neq 0$$

expresses that every nonzero integer divides zero.

If a rule has the vacuous condition *true*, we write it with no condition; for example, the logical rule

$$Q \text{ and } true \;\Rightarrow\; Q$$

may be applied to any subexpression that matches its left-hand side.

Assertions and goals are affected differently by transformation rules. Suppose

$$r \Rightarrow s \quad \text{if } P$$

is a transformation rule and $F(r')$ is an assertion such that its subexpression r' is not within the scope of any quantifier. Suppose also that there exists a *unifier* for r and r', *i.e.*, a substitution θ such that $r\theta$ and $r'\theta$ are identical. Here, $r\theta$ denotes the result of applying the substitution θ to the expression r. We can assume that θ is a "most general" unifier (in the sense of Robinson [1965]) of r and r'. (We rename the variables of $F(r')$, if necessary, to insure that it has no variables in common with the transformation rule.) By the rule, we can conclude that if $P\theta$ holds, then $r\theta$ and $s\theta$ are equal terms or equivalent sentences. Therefore, we can add the assertion

$$\text{if } P\theta \text{ then } F(s)\theta$$

to our sequent.

For example, suppose we have the assertion

$$a \in l \text{ and } a \neq 0$$

and we apply the transformation rule

$$u \in v \;\Rightarrow\; u = head(v) \text{ or } u \in tail(v) \quad \text{if } islist(v) \text{ and } v \neq [],$$

taking r' to be $a \in l$ and θ to be the substitution $[\, u \leftarrow a;\ v \leftarrow l \,]$; then we obtain the new assertion

$$\text{if } islist(l) \text{ and } l \neq []$$
$$\text{then } (a = head(l) \text{ or } a \in tail(l)) \text{ and } a \neq 0.$$

Note that a and l are constants, while u and v are variables, and indeed, the substitution was made for the variables of the rule but not for the constants of the assertion.

In general, if the given assertion $F(r')$ has an associated output entry t, the new output entry is formed by applying the substitution θ to t.

The corresponding dual deduction rule for goals, in our deduction-rule notation, is

assertions	*goals*	*output*
	$F(r')$	t
	$P\theta$ *and* $F(s)\theta$	$t\theta$

(Transformation rules can also be applied to output entries in an analogous manner.)

For example, suppose we have the goal

	$a\|z$ *and* $b\|z$	$z+1$

and we apply the transformation rule

$$u|0 \;\Rightarrow\; true \quad \text{if } integer(u) \text{ and } u \neq 0,$$

taking r' to be $a|z$ and θ to be the substitution $[\, z \leftarrow 0;\ u \leftarrow a \,]$. Then we obtain the goal

	(integer(a) and a ≠ 0) and *(true and b|0)*	0+1

which can be further transformed to

	integer(a) and a ≠ 0 and b|0	1

Note that applying the transformation rule caused a substitution to be made for the occurrences of the variable z in the goal and the output entry.

Transformation rules need not be simple rewriting rules; they may represent arbitrary procedures. For example, r could be an equation $f(x) = a$, s could be its solution $x = e$, and P could be the condition under which that solution applies. In general, efficient procedures for particular subtheories may be represented as transformation rules (see, *e.g.*, Bledsoe [1977] or Nelson and Oppen [1978]). Some transformation rules effect the removal of quantifiers. Other rules play the role of the "antecedent theorems" and "consequent theorems" of PLANNER (Hewitt [1971]).

RESOLUTION

The original resolution principle (Robinson [1965]) applied only to sentences in conjunctive normal form. The version we employ does not make this requirement.

Assume our sequent contains two assertions of form $F(P_1)$ and $G(P_2)$, where P_1 and P_2 are subsentences of these assertions not within the scope of any quantifier. For the time being, let us ignore the output expressions corresponding to these assertions. Suppose there exists a unifier for P_1 and P_2, *i.e.*, a substitution θ such that $P_1\theta$ and $P_2\theta$ are identical. We can take θ to be the most general unifier. The *AA–resolution rule* allows us to deduce the new assertion

$$F(true)\theta \text{ or } G(false)\theta,$$

and add it to the sequent. (Here, $F(true)$ denotes the result of replacing P_1 by true in $F(P_1)$. Of course, we may need to do the usual renaming to ensure that $F(P_1)$ and $G(P_2)$ have no variables in common.) We will call θ the *unifying substitution* and $P_1\theta$ $(=P_2\theta)$ the *eliminated subexpression*; the deduced assertion is called the *resolvent*. Note that the rule is symmetric, so the roles of $F(P_1)$ and $G(P_2)$ may be reversed.

For example, suppose our sequent contains the assertions

$$if\ (P(x)\ and\ Q(b))\ then\ R(x)$$

and

$$P(a)\ and\ Q(y).$$

The two subsentences "$P(x)$ *and* $Q(b)$" and "$P(a)$ *and* $Q(y)$" can be unified by the substitution

$$\theta = [\ x \leftarrow a;\ y \leftarrow b\ ].$$

Therefore, the AA-resolution rule allows us to eliminate the subexpression "$P(a)$ *and* $Q(b)$" and derive the conclusion

$$(if\ true\ then\ R(a))\ or\ false,$$

which reduces to

$$R(a)$$

by application of the appropriate transformation rules.

A "nonclausal" resolution rule similar to ours has been developed by Murray [1978]. Other such rules have been proposed by Wilkins [1973] and Nilsson [1979].

THE RESOLUTION RULES

We have defined the AA-resolution rule to derive conclusions from assertions:

the *AA–resolution rule*

assertions	*goals*
$F(P_1)$ $G(P_2)$	
$F(true)\theta$ *or* $G(false)\theta$	

where $P_1\theta = P_2\theta$, and θ is most general.

By duality, we can regard goals as negated assertions; consequently, we obtain a "GG-resolution rule", a "GA-resolution rule", and an "AG-resolution rule" as corollaries of the AA-resolution rule.

We have ignored the output expressions of the assertions and goals. However, if at least one of the sentences to which a resolution rule is applied has a corresponding output expression, the resolvent will also have an output expression. If only one of the sentences has an output expression, say t, then the resolvent will have the output expression $t\theta$. On the other hand, if the two sentences $F(P_1)$ and $G(P_2)$ have output expressions t_1 and t_2, respectively, the resolvent will have the output expression

if $P_1\theta$ *then* $t_1\theta$ *else* $t_2\theta$.

For example, suppose we have derived the two goals

	$max(tail(l)) \geq head(l)$ *and* $tail(l) \neq []$	$max(tail(l))$
	$not(\ max(tail(l)) \geq head(l)\ )$ *and* $tail(l) \neq []$	$head(l)$

Then by GG-resolution, eliminating the subsentence $max(tail(l)) \geq head(l)$, we can derive the new goal

	$(true$ *and* $tail(l) \neq [])$ *and* $(not(false)$ *and* $tail(l) \neq [])$	*if* $max(tail(l)) \geq head(l)$ *then* $max(tail(l))$ *else* $head(l)$

which can be reduced to

	$tail(l) \neq []$	*if* $max(tail(l)) \geq head(l)$ *then* $max(tail(l))$ *else* $head(l)$

A *polarity strategy* adapted from Murray [1978] restricts the application of the resolution rules. The deductive system we have presented so far, Including the splitting rules, the resolution rules, and an appropriate set of logical transformation rules, has been proved by Murray to constitute a complete system for first-order logic, In the sense that a derivation exists for *every* valid sentence. (Actually, only the resolution rules and some of the logical transformation rules are strictly necessary.)

MATHEMATICAL INDUCTION AND THE FORMATION OF RECURSIVE CALLS

Mathematical Induction is of special importance for deductive systems intended for program synthesis, because It la only by the application of some form of the Induction principle that recursive calls or iterative loops are introduced Into the program being constructed. The Induction rule we employ Is a version of the principle of mathematical Induction over a well-founded set.

We *may* describe this principle as follows: In attempting to prove that a sentence of form *F(a)* holds for every element *a* of some well-founded set. we may assume inductively that the sentence holds for all u that are strictly less than *a* In the well-founded ordering <. Thus, in trying to prove *F(a),* the well-founded Induction principle allows us to assume the Induction hypothesis

for all u, if u < a then F(u).

In the case that the well-founded set is the nonnegative integers under the usual < ordering, well-founded Induction reduces to the familiar complete induction principle: to prove that *F(n)* holds for every nonnegative integer *n,* we may assume Inductively that the sentence *F(u)* holds for all nonnegative Integers *u* such that *u* < *n.*

In our inference system, the principle of well-founded induction is represented as a deduction rule (rather than, say, an axiom schema). We present only a special case of this rule here.

f(a) <= find z such that
for some y, R(a, y, z)
where P(a).

Sup whose specification is of form

Then o sequent Is

assertions	goals	output
P(a)	*R(a, y, z)*	*z*

Then we can always add to our sequent a new assertion, the induction hypothesis

if u < a		
then if P(u)		
then R(u, g(u), f(u))		

Here f denotes the program we are trying to construct, and g Is a new skolem function corresponding to the variable *y.* The well-founded set and the particular well-founded ordering < to be employed In the proof have not yet been determined.

Let us paraphrase: We are attempting to construct a program/ such that, for an arbitrary Input *a* satisfying the Input condition *P(a),* the output *f(a)* will satisfy the output condition *R(a, y, f(a)),* for some *y;* or, equivalently, ***R(a, g(a), f(a))***. By the well-founded induction principle, we can assume inductively that for every *u* less than *a* In some well-founded ordering such that the input condition *P(u)* holds, the output *f(u)* will satisfy the same output condition ***R(u, g(u), f(u))***.

In general, we could Introduce an Induction hypothesis corresponding to any subset of the assertions or goals in our sequent, not Just the initial assertion and goal; most of these induction hypotheses would not be relevant to the final proof, and the proliferation of new assertions would obstruct our efforts to find a proof. Therefore, we employ the following *recurrence strategy* for determining when to introduce an induction hypothesis.

Let us restrict our attention to the case where the induction hypothesis is derived from the initial assertion and goal. Suppose that ***Q(a, y, z)*** Is some subsentence of the initial goal; then that goal may be written

R(Q(a, y, z)).

Suppose further that at some point In the derivation an assertion or goal of form

S(Q(t, y', z'))

is developed, where t is an arbitrary term and y' and z' are distinct variables. In other words, the newly developed assertion or goal has a subsentence $Q(t, y', z')$ that is a precise instance of a subsentence $Q(a, y, z)$ of the initial goal. This recurrence motivates us to add the induction hypothesis

if $u < a$
then if $P(u)$
then $R(Q(u, g(u), f(u)))$.

The rationale for introducing the induction hypothesis at this point is that now we can perform resolution between the induction hypothesis and the newly developed assertion or goal $S(Q(t, y', z'))$, eliminating the subexpression $Q(t, g(t), f(t))$.

Let us look at an example. Suppose we are constructing a program $rem(i, j)$ to compute the remainder of dividing a nonnegative integer i by a positive integer j; the specification may be expressed as

$rem(i, j)$ <= *find* z *such that*
for some y,
$i = y \cdot j + z$ *and* $0 \le z$ and $z < j$
where $0 \le i$ *and* $0 < j$.

Our initial sequent is then

assertions	*goals*	*outputs*
$0 \le i$ *and* $0 < j$		
	$i = y \cdot j + z$ *and* $0 \le z$ *and* $z < j$	z

Here, the inputs i and j are constants, for which we can make no substitution; y and the output z are variables.

Assume that during the course of the derivation we develop the goal

	$i-j = y_1 \cdot j + z$ *and* $0 \le z$ *and* $z < j$	z

This goal is a precise instance of the initial goal

$i = y \cdot j + z$ *and* $0 \le z$ *and* $z < j$

obtained by replacing i by $i-j$. Therefore, taking $Q(i, j, y, z)$ to be the initial goal itself, we add as a new assertion the induction hypothesis

if $(u_1, u_2) < (i, j)$ *then if* $0 \le u_1$ *and* $0 < u_2$ *then* $u_1 = g(u_1, u_2) \cdot u_2 + rem(u_1, u_2)$ *and* $0 \le rem(u_1, u_2)$ *and* $rem(u_1, u_2) < u_2$		

Here, g is a new skolem function corresponding to the variable y, and $<$ is an arbitrary well-founded ordering. Note that $<$ is to be defined on pairs because the desired program f has a pair of inputs.

We can now apply GA-resolution between the goal

	$i-j = y_1 \cdot j + z$ *and* $0 \le z$ *and* $z < j$	z

and the induction hypothesis; the unifying substitution θ is

$[\, u_1 \leftarrow i-j;\ u_2 \leftarrow j;\ y_1 \leftarrow g(i-j, j);\ z \leftarrow rem(i-j, j) \,]$.

The new goal is

	true and *not (if* $(i-j, j) < (i, j)$ *then if* $0 \le i-j$ *and* $0 < j$ *then false)*	$rem(i-j, j)$

which reduces to

	$(i-j, j) < (i, j)$ *and* $0 \le i-j$ *and* $0 < j$	$rem(i-j, j)$

Note that the recursive call $rem(i-j, j)$ has been introduced into the output entry.

The particular well-founded ordering < to be employed in the proof has not yet been determined. To choose the ordering requires special transformation rules, which describe known well-founded orderings and ways of combining them. In this case, the ordering < is chosen to be the < ordering on the first component of the pairs. Ultimately, the following goal is obtained:

	$j \le i$	$rem(i-j, j)$

In other words, In the case that $j < i$, the output re$m(i-), j)$ satisfies the desired program's specification.

We will not discuss here the more general case, where a newly developed assertion or goal has a subsentence that is an Instance of a subsentence not of the initial goal, but of some intermediate goal or assertion; this situation accounts for the Introduction of "auxiliary procedures" to be called by the program under construction. We will also not discuss the case where the new subsentence is not a precise instance of the earlier subsentence, but where both are Instances of a somewhat more general sentence.

Some early efforts toward Incorporating mathematical induction In a resolution framework were made by J. L. Darlington [1968]. His system treated the induction principle as a second-order axiom schema rather than as a deduction rule; it had a limited ability to perform second-order unifications.

CONCLUSION

Theorem proving was abandoned as an approach to program synthesis when the development of sufficiently powerful automatic theorem provers appeared to flounder. However, theorem provers have been exhibiting a steady Increase in their effectiveness, and program synthesis Is one of the most natural applications of these systems.

ACKNOWLEDGMENTS: We would like to thank John Darlington, Chris Goad, Jim King, Nell Murray, Nils Nilsson, and Earl Saoerdotl for valuable discussions and comments. Thanks are due also to Patte Wood for aid In the preparation of this manuscript.

This research was supported in part by the National Science Foundation under Grants MCS 76-83655 and MCS 78-02601, by the office of Naval Research under Contracts N00014-76-C-0687 and N00014-76-C-0816, by the Defense Advanced Research Projects Agency of the Department of Defense under Contract MDA903-76-C-0206, and by the United States-Israel Blnatlona! Science Foundation.

REFERENCES)

Bledsoe, W. W. [1977], *Non-resolution theorem proving,* Artificial Intelligence Journal, Vol. 9, *pp.* 1-35.

Boyer, R. 8. and J S. Moore [Jen, 1976], *Proving theorems about LISP functions,* JACM, Vol. 22, *pp.* 129-144.

Burstall, R. M. end J, Darlington [Jan. 1977], *A transformation system for developing recursive programs,* JACM, Vol. 24, No. 1, pp. 44-67.

Darlington, J. L. [1068], Automatic theorem proving with equality substitutions and mathematical induction, Machine intelligence 3, Edinburgh, Scotland, pp. 113-127.

Green, C. C. [May 1060], Application of theorem proving to problem solving, Proceedings of the International Joint Conference on Artificial Intelligence, Washington DC, pp. 210-230.

Hewitt, C. [Apr. 1071], Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot, Ph.D. thesis, MIT, Cambridge, MA.

Manna, Z. and R. Waldinger [July 1979], Synthesis: dreams =» programs, IEEE Transactions on Software Engineering, Vol. SE-6, No. 4, (to appear).

Murray, N. [1978], A proof procedure for non-clausal first-order logic. Technical Report, Syracuse University, Syracuse, NY.

Nelson, G. and D. C. Oppen [Jan. 1978], A simplifier based on efficient decision algorithms, Proceedings of the Fifth ACM Symposium on Principles of Programming Languages, Tuscon, AZ, pp. 141-160.

Nilason, N. J. [1971], Problem-solving methods in artificial intelligence, McGraw-Hill Book Co., New York, NY [pp. 166-168].

Nilsson, N. J. [June 1070], A production system for automatic deduction, Machine Intelligence 0, Chichester, England (to appear).

Robinson, J. A. [Jan. 1965], A machine-oriented logic based on the resolution principle, JACM, Vol. 12, No. 1, pp. 23-41.

WaldInger, R. J. and R. C. T. Lee [May 1969], PROW, a step toward automatic program writing, Proceedings of the International Joint Conference on Artificial Intelligence, Washington, DC, pp. 241-252.

WilkIns, D. [1973], QUEST—a non-clausal theorem proving system, M.Sc. thesis, University of Essex, England.

IMAGE PROCESSING BY EXPERIMENTAL ARRAYED PROCESSOR

Hitoshi Matsushima Takeshi Uno Masakazu Ejiri

Central Research Lahoratorv
Hitachi Ltd.
Higashi-koigakubo 1-280
Kokuhunji, Tokyo 185
Japan

Due to late receipt, this paper appears in the Supplement section, hack of Vol. II.

THE EXECUTION OF PLANS
IN AN INDEPENDENT DYNAMIC MICROWORLD

Gordon McCalla
Department of Computational Science
University of Saskatchewan
Saskatoon, CANADA S7N 0W0

Peter F. Schneider
Department of Computer Science
University of Toronto
Toronto, CANADA M5S 1A7

The problem of executing plans in an independent dynamic microworld is discussed. The microworld is a simulated city that not only contains permanent features (e.g. streets) but also transient features (e.g. other cars). A simulated taxi driver (ELMER) must plan and execute routes through this city. Among the interesting features of ELMER are the use of hierarchical rather than linear plans; the representation of knowledge by routes (similar to and possibly built from plans); and planning by splicing together these routes.

1 INTRODUCTION

One of the more productive areas of AI research has been the study of producing plans to solve problems in a microworld. Usually such worlds (e.g. the blocks world) are static. We are interested in planning for microworlds which can change continuously and independently of the planning agent. Such microworlds have received some attention [1] [2].

The microworld here is an abstraction of the environment a taxi driver might encounter. The microworld and the taxi driver (ELMER) are simulated as asynchronous processes where ELMER informs the microworld of speed and direction changes and the microworld provides ELMER with a "window" of what he can "see" at any given instant. The three central components of ELMER are the Planner that creates a travel plan from a customer's request; the Executor that carries out the plan; and the Map that maintains ELMER's developing knowledge of the city.

ELMER exists in Simon City (Figure 1). The microworld component of the system must simulate this city, including not only the permanent street arrangement, but also transient features such as traffic lights or other cars. It must also maintain the window and keep track of ELMER. The microworld is not yet implemented, but an implementation strategy based on something like Hendrix's scenarios [3] may be appropriate.

2 THE EXECUTOR

The Executor undertakes plans produced by the Planner. Plans are represented as hierarchies of ever more detailed sub-plans similar to the plans of Miller et al. [4], but in contrast to the usual linear sequences of actions (e.g. HACKER [5] or STRIPS [6]). Figure 2 shows a plan for going from Lenat Lane at Kuipers Crescent to Reiter Road at Schubert Street in Simon City. The numbered "boxes" are the core of the plan; the other boxes are demon-like secondary plans controlling ELMER'S speed and direction and looking out for obstructions such

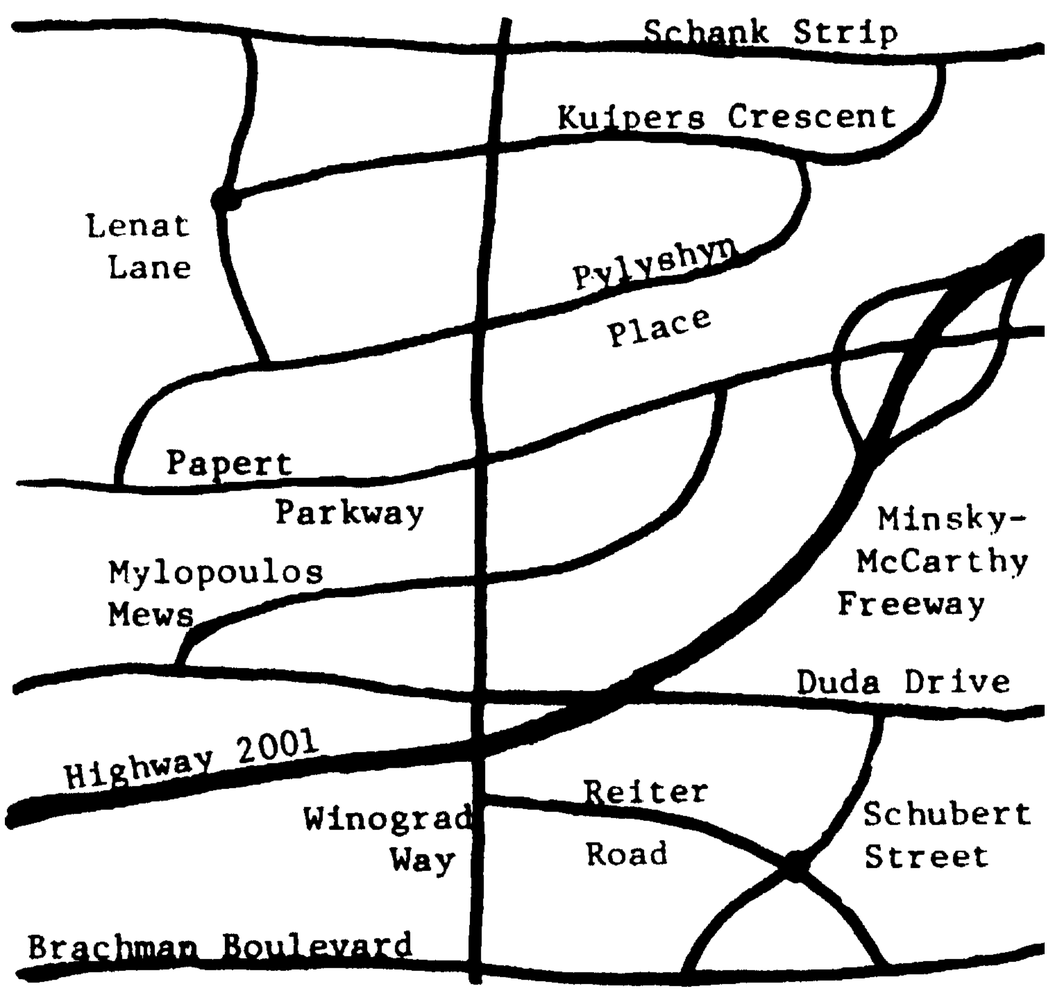


Figure 1 - Simon City

aa red lights. In the core each box represents a plan connected to more detailed sub-plans below and more general super-plans above, and connected by arrows to subsequent brother plans at the same level of detail. Plans have a type., Indicated by the first word in the plan box (e.g. 16 has type "right-turn"). Angle brackets contain commands to be sent directly to the microworld (e.g. 15).

When the Executor executes a plan, it executes the top plan (e.g. 1) and all its left-most descendants (e.g. 2, 5, 15). Such plans are then active and remain so until they are either suspended or de-activated by an absolute transition into another plan along an arrow leaving the plan. Such a transition occurs when information labelling the arrow corresponds to information in the window. For example, when the window indicates that "on Kuipers" is true, a transition is made from 5 to 6 (if 5 is active). We don't currently see a need to allow labels that would require complex inferences before matching the window information, thus ensuring fairly rapid computations when deciding on transitions. Once a transition is made, the old plan and all its active descendants are de-activated, and the new plan and all its left-most descendants are activated. Note the similarity of a plan to a TOTE (Test-Operate-Test-Exit) [4] and to a DOWHEN, DOWHAT, DOUNTIL triple [3]. Of course, the labels and actions here are more limited than the arbitrary procedural capabilities of these other approaches.

Attached to the Planner-generated core at a level of detail appropriate to their action are Executor-generated secondary plans. These fall into two categories: (i) plans (indicated by ♦ in Fig. 2) to handle normal driving perils such as stop signs and (ii) plans (indicated by •) to handle Planner suggestions on speed limits, etc. A secondary plan is activated in the same way as a core plan, except the transition is parallel in that the old plan continues to execute. Parallel transitions allow ELMER to do several things at once (e.g. slow for a stop sign and continue undertaking the core plan). During parallel execution, one plan can suggest an action which contradicts the action of another plan (e.g. both maintaining speed and stopping at a stop sign). To overcome such conflicts, a priority and an exclusion list are associated with each plan. The exclusion list indicates other plan types which cannot be executed in parallel with the plan; the priority determines whether another plan has

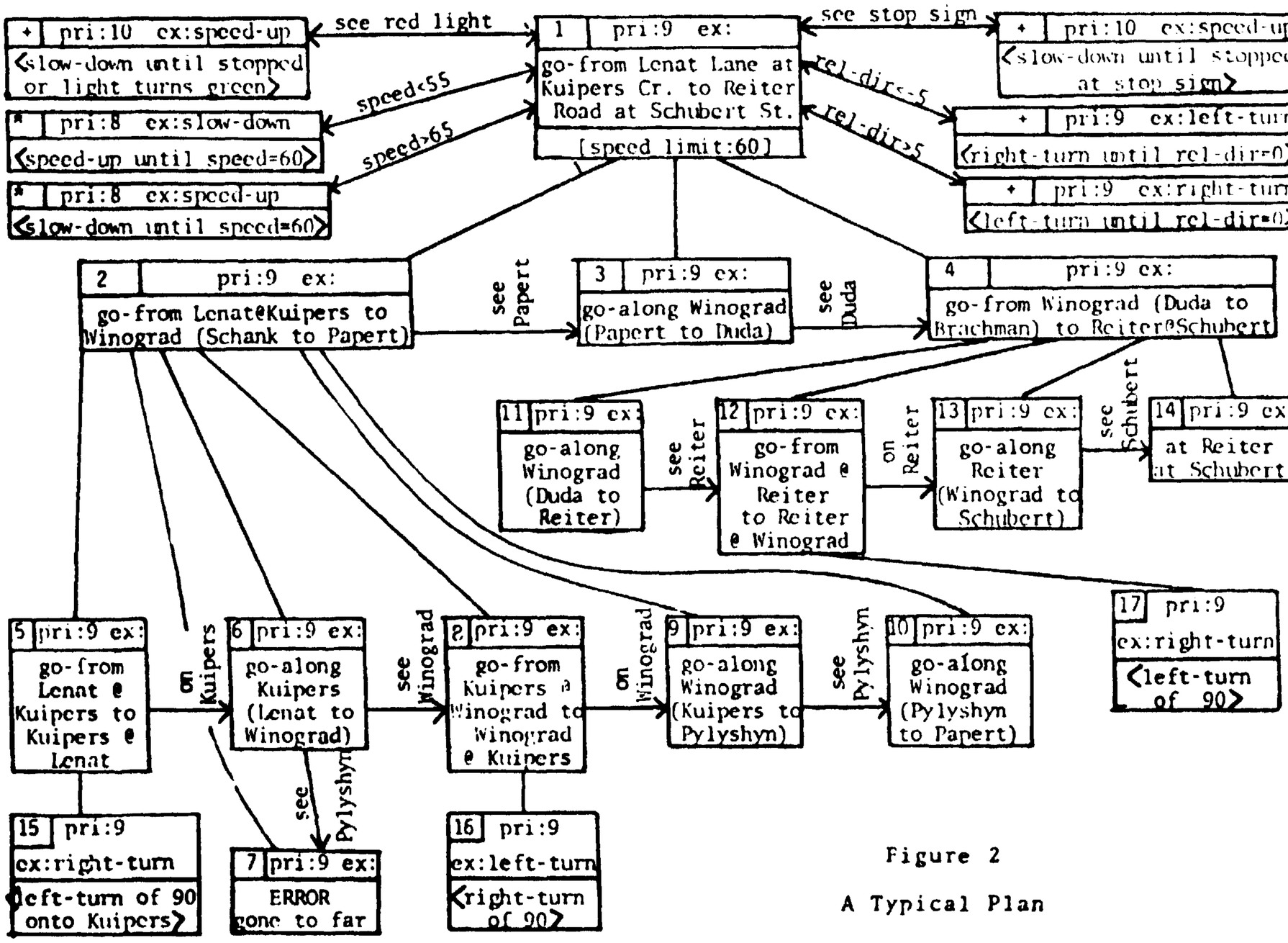


Figure 2

A Typical Plan

the authority to exclude the plan (it does if it has higher priority). In the example, the stop sign would exclude speed maintenance. When a plan is excluded by another plan, it and all of its active descendants are suspended until the excluding plan is de-activated or is itself suspended. A suspended plan can send no commands to the microworld, but transitions from it will still be made (allowing a "you-are-here" pointer (a la Kuipers [7]) to be kept even if some parallel action has suspended active processing of the core).

Suspension can be tricky since the suspension of a plan which is itself suspending other plana may allow the re-activation of some of these plans. A suspension rule is given in McCalla et. al [8] that guarantees a unique set of suspensions providing a plan has priority no greater than the priorities of its ancestors. Since most plans having ancestors are core plans which usually don't exclude one another, this isn't a serious restriction.

4 THE MAP AND THE PLANNER

The Map contains ELMER's knowledge of the city represented mostly as routes (unlike the multiple representations of [7]). Such routes are similar in structure to plans, differing mainly in that associated with each route is the region ELMER would travel if he traversed the route. The containment relations of such regions are integrated into a region hierarchy. We feel that "instantiated" old plans will provide the main source of new Map routes. This would correspond nicely with Miller et al's intuition that "probably the major source of new plans is old plans" [4, p. 177].

The Planner uses Map routes to produce a plan to solve a particular problem. If the problem is to go from region b to region d, the Planner essentially reasons as follows: if there is a Map route which connects super-region B ? b (in the region hierarchy) to super-region D ? d, use this route as the skeleton for the desired plan; else look for a route connecting B to some region C and another route connecting C to D and splice them together to form the skeleton; else give up. Converting a skeletal plan into an actual plan may involve the further elaboration of parts of the skeleton (as well as adding recommendations for Executor •-type plans). Such elaboration is done on the earliest portions of the plan first so that ELMER can start executing the plan as quickly as possible (contrast this to Sacerdoti's [9] planner which produces the whole plan before any execution is considered). More details of the Map and Planner can be found in [8].

5 CONCLUSION

The two main contributions of this research are the ability to handle a dynamic world (via secondary plans attached to the core plan) and the uniformity of representation of plans and routes which not only allows a route splicing planning methodology but may also allow ELMER to add new routes built from old plans. This latter aspect is currently under investigation as are problems of error recovery and the need to discover the limits of this approach.

ACKNOWLEDGEMENTS

We would like to thank Robin Cohen and Hector Levesque for their contributions to this research. The support of the Natural Sciences and Engineering Research Council of Canada through grant A3323 is also acknowledged.

[1] Hayes, P. J. "A Representation for Robot Plans", Proc. IJCAI-75. Tbilisi, U.S.S.R., 1975, pp. 181-188.

[2] Chien, R. T. and S. Weissman. "Planning and Execution in Incompletely Specified Environments", Proc. IJCAI-75. Tbilisi, U.S.S.R., August, 1975, pp. 169-174.

[3] Hendrix, G. G. "Modeling Simultaneous Actions and Continuous Processes." ArtIf. Intell. 4 (1973) 145-180.

[4] Miller, G. A., E. Galanter, and K. H. Pribaum. Plana and the Structure *of* Behaviour- New York, N.Y.: Holt, Rinehart & Winston, 1960.

[5] Sussman, G. J. "A Computational Model of Skill Acquisition." AI Lab. AI-TR-297, MIT, Cambridge, Mass., 1973.

[6] Fikes, R. E. and N. J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving." ArtIf. Intell. 2 (1971) 189-208.

[7] Kaipers, B. J. "Representing Knowledge of Large Scale Space." AI Lab. AI-TR-IIb, MIT, Cambridge, Mass., 1977.

[8] McCalla, G. I., P. Schneider, R. Cohen, and H. Levesque. "Investigations into Planning and Executing in an Independent and Continuously Changing Microworld." Dept. of Comp. Sci. AI Memo 78-2, U. of Toronto, Toronto, Ont., 1978.

[9] Sacerdoti, E. A Structure for Plans and Behaviour. New York, N. Y.: Elsevier, 1977.

REPRESENTATION AND EFFICIENCY IN A PRODUCTION SYSTEM FOR SPEECH UNDERSTANDING

Donald L McCracken
Computer Scionco Department
Carnegie-Mellon University
Pittsburgh, PA 15213

ABSTRACT. The research question studied is whether it is practical to use a production system (PS) architecture for a version of the Hearsay-II (IISIII speech understanding system. The study focuses on adequacy of representation, space efficiency and time efficiency A running PS architecture called HSP was designed and implemented and twelve HSH Knowledge sources were translated to HSP productions, with two of the twelve actually run under MSP. Detailed comparisons of HSH and HSP produced the following results: (i) HSP is adequate to represent the HSII speech Knowledge, even though HSP is a comparatively simple PS architecture; (2) HSP suffers a moderate space penalty for representing declarative HSII Knowledge, which is serious since HSII contains a high proportion of such Knowledge; (3) representation of HSII declarative Knowledge as HSP productions causes problems with multiple use of that Knowledge; (4) lacK of a local worKing memory in the HSP architecture has serious consequences both for space and time efficiency; *and* (5) HSP has a time efficiency handicap of two to three-and-a-half *orders of* magnitude, in spite of efficiency mechanisms which make HSP comparable in time efficiency to other PSs. The paper ends with a set of questions that remain to be answered for a thorough evaluation of a PS version of HSII.

1. INTRODUCTION

A prime candidate organization for large knowledge-rich systems is that of a production system (PS) [10,2,15]. PSs are rule-based architectures that have been used successfully for tasks ranging from models of human behavior [11], to large application systems in chemistry [3] and medicine [13], to general artificial intelligence programming tasks [12]. The uniform rule structure of knowledge in PSs makes them particularly useful for tasks requiring iterative upgrading of knowledge content.

The research question reported on In this paper* is whether a PS architecture (PSA) is adequate to represent the rich variety of speech knowledge contained in Hearsay-H (HSII), a large artificial intelligence system for understanding speech, developed at Carnegie-Mellon University [1]. We consider explicitly the questions of representational adequacy, space efficiency and time efficiency.**

HSII is a system organization for the speech understanding task designed to coordinate the contributions of diverse, essentially independent sources of knowledge which operate at the different hierarchical levels of speech knowledge. The essential device for communication among knowledge sources (KSs) is a large shared working memory called the *Blackboard,* which all KSs inspect and modify. The Blackboard is a medium for growing linked networks of hypotheses about what elements exist at the various levels and for various time Intervals of the utterance (e.g., words, syllables, etc.). The KSs are structured to contribute their knowledge *b)* creating and modifying Blackboard hypotheses and links in response to relevant changes by other KSs (i.e., in a data-directed fashion).

To carry out the research reported here, an actual production system (called HSP, for "HearSay-Production system") was implemented on C.mmp, the CMU multi-miniprocessor [16], with a portion of the HSII speech knowledge translated into productions. Two knowledge-source (KS) programs from a full HSII configuration were completely translated and run in HSP, and these provide a basis for some detailed comparisons between HSII and HSP. These KSs were (1) POM [14], which recognizes syllables from speech segments, using a complex, multi-phase process involving three intermediate levels of representation; and (2) RPOL [1], which propagates validity ratings for the Blackboard hypotheses along the links 'hat interconnect them. POM and RPOL together required 945 HSP productions. Ten other KSs were translated, but with some simplifications and omissions, and were *never* completely debugged and run. However, their static structure does provide evidence for representation issues. The total number of HSP productions for all twelve KSs was 2211. If this count is adjusted to discount highly similar productions that simply enumerate cases for a single situation, the result is 293 substantially different productions out of the total of 2211.

2. THE PROMISE OF PRODUCTION SYSTEMS

A production system architecture (PSA), in the sense used here, consists of a set of rules, called productions, stored in a production memory (PM), plus a working memory (WM) which holds symbolic structures operated on by the productions. A production consists of a condition part which tests the current state of WM, and an action part which specifies additions, modifications and deletions to WM to take place in case the condition part is true. The basic operating cycle of a PS is a "recognize-act" cycle: I.e., recognize which production

*This research was sponsored in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

**The Ph.D. thesis that this paper is based on [6] considers these same questions in greater detail, plus the questions of exploiting parallelism on multiprocessors and of decomposing a system to run on processors with a small address space.

conditions are true of the current WM, then execute the corresponding production actions. Since each cycle makes some change to the WM state (which changes the set of productions that *are* true), iteration of the cycle produces (potentially, at least) an interesting stream of behavior.

PSAs at large show a great deal of diversity. The strain to which HSP belongs has a lineage beginning with studies of human problem solving [11] and a PSA called PSG [10], and extending to the more recent architectures Of PSNLST [12] and OPS [4]. A variant strain with roots in performance-oriented, knowledge-based expert systems (DENDRAL [3] and MYCIN [13]) embodies a major architectural difference In the use of rules. These systems treat rules as declarative knowledge structures to be interpreted; i.e., the production rules are only a piece of the total system (though the most important piece). On the other hand, PSG-like systems have no declarative long-term knowledge; all knowledge is encoded as (procedural) productions, which combine knowledge with how it is to be used (control Information). The tradeoff between the two views is essentially this: PSG-like systems have more flexibility in representing control information (i.e., knowledge Interactions), but at the cost of greater difficulties than MYCIN-like systems in augmentation and permitting multiple use of knowledge.

PSAs show substantial promise for large artificial Intelligence systems (specifically, speech understanding systems). In [6] this promise is enumerated for the following set of concerns: *adequacy of representation, space efficiency, time efficiency, directionality, augmentation, performance analysis, testing, debugging,* and *handling of error.* Unfortunately, adequacy, space efficiency, and time efficiency (the three aspects reported on here) are probably the least promising of those listed. The more promising aspects were not within the scope of the research, though some of them are mentioned in Section 4 on "Remaining Questions".

3. THE RESULTS

For sake of clarity, the results of the research are presented in the form of assertions. The assertions are stated in a strong form. A discussion following each assertion explains It and summarizes some of the supporting evidence. In cases where the evidence is weak, the assertion is qualified appropriately. Figure 1 gives a preview.

/. HSP productions are adequate for representation of all HSH speech knowledge.

The basic evidence for adequacy* is that a large number (12) of HSII KSs were translated to HSP productions. There are nearly 20 other KSs which have been used at some time throughout HSII's development, but a moderate acquaintance with them has turned up no reasons to doubt HSP's adequacy. Efficiency is another matter (e.g., see assertion 11).

---

*By adequacy we do not mean theoretical adequacy (which is trivially present), but rather something we might call "representational practicality". We also do not mean to include time efficiency, since that is considered separately.

Representation and Architecture

1. HSP productions are adequate for representation of all HSII speech knowledge'.
2. The adequate architecture of HSP is simple.
3. Translating HSII declarative knowledge to HSP creates problems of multiple use.
4. Explicit conditionality on changes is a strong feature of HSP's architecture.

Space Efficiency

5. Procedural HSII knowledge decreases slightly in size when translated to HSP.
6. Declarative HSII knowledge increases In size by up to half *an* order of magnitude.
7. Total HSII knowledge increases in size by up to half an order of magnitude.
8. HSP requires a much larger global working memory than HSII.

Time Efficiency

9. The implemented time efficiency mechanisms in HSP are critical for its viability.
10. The time cost of HSP relative to HSII is at least two orders of magnitude.
11. Local working memory and control are the prime sources of HSII time efficiency.

Figure 1: Results in Assertion Form

There were several minor difficulties encountered in the KS translation process, for example: iteration over WM element list fields, controlling duplicate actions, tallying events, and redundant arithmetic expression evaluation. Yet these difficulties *are* not serious enough to constitute a refutation of HSP's adequacy. Most of them could be solved through design iteration of the HSP architecture, without deviating from basic PS philosophy.

HSII KSs do a significant arrount of simple table lookup from local arrays containing long-term knowledge. This might at first seem tc cause a representation problem for HSP, but actually an array translates cleanly into a mutually exclusive set of productions, one for each entry. Direct indexing to select an array entry in HSII is essentially the same as selecting the correct production from the set in HSP. In other words, a PS is really a large, complex table lookup, while an array access is just an optimization of the lookup process that is possible with a highly uniform set of productions. However, a PSA is overly general for simple table lookup; the resulting space and time costs are discussed below under assertions 6 and 10.

The adequacy of HSP for speech understanding provides

an additional data point for the broader question of adequacy of PSAs for artificial intelligence applications. More extensive evidence for adequacy has already been reported by Rychener [12], in his study of six classical artificial intelligence programs. The speech understanding task of HSP has some characteristics that set It apart from Rychener's tasks: (1) rich set of relatively independent knowledge sources, (2) each of which has a sound theoretical basis, providing lexicons of basic entities and rules relat ng entities, and (3) the large direct recognition component, i.e., no need for extensive serial reasoning.

2. *The adequate architecture of HSP is simple.*

This assertion is of interest only in the context of assertion 1. Simplicity without assurance of adequacy is trivial.

Contrary to standard architectural practice for PSs, HSP has no *conflict resolution;* i.e., on each cycle it fires *every* true production rather than using a conflict resolution mechanism to select a single production to fire. This permits in HSP a natural expression of the multiple, independent KSs that come from HSII and the speech task. Perhaps more importantly, it allows much greater parallelism: while PSG, PSNLST and OPS respond to only a few changes and fire only a single production each cycle, HSP can respond to hundreds of changes and fire a hundred productions or more. However, such a feature could seriously impair a PS's ability to switch focus rapidly in response to a novel situation or new external stimulus [7]. But this does not seem worrisome for HSP since most of the multiple firings are operating in parallel on separate levels or time regions of the representation of the speech utterance.

In the absence of conflict resolution, HSP must occasionally resort to a somewhat inflexible mechanism called an *n-cycle delay.* This Involves a chain of productions that waits for other activity to finish by marking time for a certain fixed number of cycles.

HSP permits neither disjunctions nor negated conjunctions within production conditions. (Negations of single condition elements are essential and of course permitted). These restrictions simplify the production interpreter without seriously affecting adequacy of representation. Disjunctions or negated conjunctions can be eliminated from a production by splitting it into several productions, at the cost of some decrease ir, space and time efficiency.

HSP has no mechanism for *special case inhibition;* i.e., preventing a true production from firing when another production representing a special case of the first one is also true. Such a mechanism would have complicated the production interpreter and probably caused a serious reduction in parallelism. Doing without special case inhibition is *an* inconvenience. Either the more general production must be augmented to make it complementary to the related special cases *or* special productions must be added to detect when both general and special case productions fire and favor the special case result.

3. *Translating HSII declarative knowledge to HSP creates problems of multiple use.*

A basic feature of the HSP architecture is the absence of any form of declarative long-term memory. Thus all long-term knowledge must be encoded as productions. Since each production specifies under what conditions its piece of knowledge is to apply, there is a problem with using that knowledge under different circumstances. In some cases the problem can be solved by merely duplicating the knowledge, with a different production for each different use. This was done frequently in the HSP KSs, but will not extend to systems of growing complexity where multiple use can be expected to increase. Subroutines of productions provide another solution to the problem of multiple use. But subroutines require a high degree of similarity of the uses, plus rigid conventions for communication. A third, somewhat novel, solution employed in HSP is to deposit knowledge temporarily into WM whenever there is a reasonable expectation that it may be useful. In WM it is then available in declarative form to whatever production wants to make use of it. It is not known whether these solutions to the problem of multiple use would be adequate for much larger and more complex systems than HSII, but the suspicion is that they would not.

4. Explicit cond it tonality on changes is a strong feature of HSP's architecture.

The first condition element of every HSP production explicitly tests the nature of a WM change, and must match a change made in the previous cycle. Thus a production cannot fire any time its condition (excluding the first element) is true, but only when a particular type of change (tested by the first condition element) occurs in conjunction with a true condition. (Normally, the change *causes* the condition to become true). This explicit condition on a change has two important uses in HSP: (1) It provides the basis for a time efficiency mechanism called the P*M index* (see assertion 9). (2) It solves the *excitatory instability* problem; i.e., a production cannot continue to fire cycle after cycle once become true, because it is also conditional on the occurrence of the change that made it true.

5. *Procedural HSII knowledge decreases slightly in sire when translated to HSP.*

Detailed space analysis of the POM KS shows that about 160 Kbits of long-term HSII procedural knowledge translated to only .7 times that much in HSP. There are two possible explanations for this decrease: (1) Since HSP productions are interpreted, a more compact representation is possible; and (2) HSP can represent condition testing and searching of the global working memory more concisely. However, the explanation is probably more complex than this, as suggested by high variation of the HSP/HSII space ratio over eleven subparts into which POM was partitioned for analysis.

6. *Declarative HSII knowledge increases in size by up to half an order of magnitude.*

HSII declarative structures are mostly either simple arrays, dictionaries of spellings, or linked networks, although all are encoded internally as arrays for efficiency. Arrays translate to HSP as one production *per* entry; the networks are typically represented by one production for each transition, or one production for all transitions out of each state. In any case, the number of productions required is large.

Seven instances of large declarative knowledge structures in the translated KSs were compared for size. Simple arrays require almost a factor of 4 more space in HSP, but sparseness can be exploited to reduce that factor by the fraction of non-default array entries. Tables of spellings (actually arrays of strings) have HSP/HSH space ratios of about 1. Two examples of network structures have the following ratios: 1.2 (grammar) and 3.2 (state transition network). The overall HSP/HSII ratio for the seven instances compared, excluding one exceptional case (a huge bit matrix), is 1.6.

HSP facilities for *condition and action procedures* allow a sequence of similar condition or action elements to be packaged as a parameterized procedure and then referenced in many different productions with appropriate parameters supplied.* These procedures are crucial for representation of HSII declarative structures because of the large number of similar productions Involved. In the HSP POM KS, condition and action procedures give a savings of a factor of 10 in the number of condition and action elements that must be represented explicitly.

7. *Total HSH knou/ledge increases in size by up to half an order of magnitude.*

In HSII POM, with an overall HSP/HSII space ratio of 1.1, the declarative/procedural split for long-term knowledge Is .3/.7. But this is not a typical split. Many of the large KSs most recently added to HSII are estimated to be split about .9/.1. Assuming a .9/.1 split for a full HSII configuration would give an overall HSP/HSII ratio of 1.5, assuming the declarative HSP/HSII ratio to hold at the value of 1.6 obtained above. If instead we assume a declarative ratio of 3.8 (the worst observed), we get an overall ratio of 3.5. We cannot be more precise than this because the declarative structures of the new KSs have not been analyzed. Their ratios may well be larger than 3.8.

8. HSP requires a much larger global working memory than HSII.

Many HSII KSs use large local working memories in addition to the global Blackboard, and these local memories can be highly specialized for efficiency. HSP has only its global WM, and specialization of it is strongly limited by requirements of generality and uniform accessibility. This difference In specialization costs HSP less than half an order of magnitude in space for simple data Items. For more complex data structures such as network nodes the cost can be as much as a full order of magnitude. Furthermore, such complex structures are so abundant that they dominate the overall cost.

9. *The implemented time efficiency mechanisms in HSP* are *critical for its viability**

Three existing time efficiency mechanisms in HSP give a combined speedup of 3 or 4 orders of magnitude over a naive Implementation (i.e., one that evaluates every production as a result of every WM change, and that searches WM for *every* condition element evaluated). All three mechanisms *are* suggested by analogs in the HSII architecture.

The first, called the *PM index,* reduces a linear dependence of execution time on PM size to sublinear by associating subsets of relevant productions (i.e., exactly those to evaluate) from classes of WM changes. In the HSP POM + RPOL run this resulted in a speedup of about 200.

The second mechanism, use of explicit pointers between WM elements, reduces the amount of WM searching during condition element evaluation. This mechanism does not reduce the degree of execution time dependence on WM size, but does give a constant factor of roughly 10 to 50 overall speedup.

The third mechanism, an index into WM according to the first two fields of a WM element, serves to reduce WM searching costs. Since it operates in the shadow of the second mechanism which greatly reduces the necessity for WM searching, the WM index makes only a small contribution.

These mechanisms bring HSP to roughly the same level of time efficiency as PSG with filters [8], PSNLST, and OPS.

10. The time cost of HSP relative to HSII is at least two orders of magnitude.

In equivalent uniprocess runs of the POM + RPOL KS configuration, HSP took 255 times as long as HSII (917 sec as opposed to 3.6). This factor of 255 reduces to a range of 6 to 36 when corrected for eight underlying system differences: execution rate of machine, instruction set of machine, address space size, operating system, implementation system, degree of system kernel optimization, speech knowledge content, and complications of parallelism.

There are problems with the generality of this comparison since it is based on a single run of a small configuration (i.e., containing only the two KSs: POM and RPOL), and on only a single syllable of input utterance. Correcting for syllable length and atypicality of the KSs gives a rough overall time efficiency loss of two to three-and-a-half orders of magnitude for a full configuration of KSs in HSP.

Related data by Rychener [12] shows a factor of 6 to 10 loss in time efficiency for PS translations vs. original versions of six classic artificial intelligence systems.

11. Local control and working memory are the prime sources of HSII time efficiency.

Since HSP has such limited local control (its actions are simple sequences and cannot call other productions), it must rely on data-directed invocation operating from the global WM. Much of the data-directed invocation might be avoided if productions could be much larger, compressing multiple PS cycles into single ones. But this is made difficult by an accompanying blowup in the number of productions.

The cost of data-directed control in HSP has several sources: (1) the creation/deletion of change elements, (2) creation/deletion of control signals in WM, (3) PM

---

*These procedure facilities are purely for space efficiency. A production using them behaves exactly as if all its elements were explicitly written out, except for a small time efficiency loss in assigning parameters to variables.

Indexing (finding productions to evaluate based on the changes), and (4) an initial portion of condition evaluation for each production that is necessary to reobtain the context of the preceeding production. The total cost of these is estimated to be 30-457- of execution time. In HSH the cost of data-directed control is *St* of execution time, and would be much smaller except that the cost is dominated by the monitoring of Blackboard changes rather than by the number of actual invocations.

The other side of the coin from local control is local working memory use. HSII makes heavy use of local working *memory* for KS efficiency. Since a PSA contains no analogous facility, HSP is forced to use its global WM for such functions. This puts HSP at a serious disadvantage. The HSP run of POM + RPOL made over 5 times as many global working memory reads as a corresponding HSII run, and more than twice as *many* creations. Further, as mentioned above, there are other KSs that make much heavier use of local data than POM and RPOL. We expect HSP versions of these others would make hundreds of times as many global working memory reads and creations as the HSII versions.

Two other sources of HSP inefficiency are identified: (1) the absence of a declarative long-term memory facility, •nd (2) searching of WM. The former consumes 152 of execution time in the HSP run (but some of this overlaps with the WM access costs discussed above). The latter is Insignificant (37) because of the explicit WM element references, but is projected to increase to around 307 of total time with a full input utterance.

Taking all the sources of inefficiency together, we can account for roughly a factor of 7 in execution time. This takes us well on the way toward explaining the normalized HSH-HSP difference of 6 to 36 obtained for the HSP run. But it also suggests there may be other sources. One such possibility is the limited power of the HSP production language, as exemplified by the inability of a single production to deal with a data list of arbitrary length.

4. REMAINING QUESTIONS

From the vantage point of the research done, several questions emerge as the important ones yet to be answered for a thorough evaluation of an HSP-like version of HSII. Six such questions are discussed briefly below. Given the current state of the HSP-HSH comparison (I.e., considering only adequacy and efficiency), with HSP faring rather poorly, it is natural to ask about factors that might swing the balance back toward HSP. Questions (1) through (3) below are of this nature. Questions (4) through (6), on the other hand, ask •bout possible weaknesses of HSP that need to be probed further.

i. *Do there exist situations that require the inhomogeneity of representation in HSP?*

The HSP architecture is better suited than HSII to representation of inhomogeneous knowledge, e.g., knowledge containing many special cases. HSH's strong point Is homogeneous encodings; when it is forced out of these it must fall back on more expensive and ad hoc representations. This could turn into a strong advantage for HSP if the drive for improved performance pushed in the direction of inhomogeneity, as we suspect it may.

2. Can a *significant improvement in ease of augmentation be realized in a PSA?*

Ease of augmentation Is one of the most promising aspects of a PSA, so we need to show a significant advantage here to balance negative aspects such as efficiency. HSP's current architecture, lacking conflict resolution, may not *proper]y* support augmentation. If necessary, conflict resolution can be added to HSP, at the risk of sharply reduced parallelism.

3. *Is the level of individual productions right for performance analysis?*

Performance analysis at the level of productions can be accomodated easily in HSP. The only issues are whether that level is a useful one (it is perhaps too low), and whether performance analysis at higher levels (always necessary) is made more difficult in a PSA. These issues can be resolved only by experience with larger KS configurations in HSP; the current two-KS configuration is too small to provide any significant insight.

4. Do there or can there exist HSII KSs which cannot be represented adequately in HSP?

Since there a*re* about 20 HSII KSs that have not been translated to HSP, plus a virtually open-ended set of others which might conceivably be added to HSII, our evidence for adequacy is incomplete. Particularly suspicious are some of the most recently added HSH KSs that use highly specialized local data structures. The current belief is that HSP is adequate for these, though inefficient. A more thorough HSI1-HSP evaluation requires evidence for this belief.

5. *Can HSII KSs that heavily rely on local efficiency ultimately be made tractable in HSP?*

We have seen that the bulk of HSP's efficiency handicap relative to HSII comes from HSP's lack of local working memory and control. It is important to make some inroads on this handicap if HSP is to be useful, and several possibilities present themselves: (1) optimization in the HSP kernel of data-directed invocation and WM access mechanisms, *(2)* permitting productions to have complex actions with their own conditionality and local working memory, and (3) compiling many PS cycles into one by combining production sequences into many complex productions that act within a single cycle.

6. *Is the distributed control of directionality as dictated by a PSA feasible?*

Claims have been made that PSAs are unsuitable for representation of directionality control [9]. Thus it is Incumbent upon us to prove them wrong if HSP is to survive its comparison with HSII. HSII has a specialized focussing mechanism which operates with global knowledge of the state of the computation [5]. In HSP we must show (if possible) that the requisite global state information can be represented in WM, and continually updated by a set of special added productions; and that task productions can be *made* to schedule themselves with additional condition elements sensitive to this representation of global state.

5. CONCLUSION

We have shown that the HSP architecture is almost

surely adequate for representing the HSU speech knowledge. This includes HSH declarative knowledge which must translate to procedural form in HSP. Adequacy of MSP was not a foregone conclusion because the simplicity of the HSP architecture compared to HS1I gave grounds for some doubt about adequacy. Space and time efficiency are another story. The moderate space penalty for representing declarative HSH knowledge in HSP is cause for concern since HSII does contain many large declarative knowledge structures. *Even* more serious concern arises over space inefficiency of the global HSP working memory, since it must be used in place of large, highly optimized local working memories that are typical in HSII KSs. HSP's lack of local working memory also causes a large loss of time efficiency because of greater creation/read/write costs and heavier use of relatively inefficient data-directed control HSP's large time efficiency handicap (two to three-and-a-half orders of magnitude) exists in spite of efficiency mechanisms which make HSP comparable in time efficiency to several other PSAs.

The partial evaluation of a PSA for HSII reported here is unfavorable for the PSA, but there are still other aspects (such as augmentation or performance analysis) that hold promise for PSAs. At least we have quantified the efficiency aspects so that a complete picture can be fit together as additional questions are answered.

A side benefit of this research is in providing enhanced understanding of HSII, a system which is a highly visible and Important contribution to artificial intelligence. By investigation through HSP of marked alternatives to HSHs philosophy, we shed light on it. For example, we have contributed a better understanding of HSHs use of local memory and control for efficiency.

Another result of this research is the existence of another data point for the applicability of PSAs to artificial intelligence systems. The significant aspect of speech understanding that distinguishes it from the domains studied by Rychener [12] is the heavy use of declarative knowledge structures.

HSP demonstrates several novel features that are of possible interest to designers of PSAs. We have shown that permitting multiple firings per cycle of the PS has appeal (at least for some tasks), and that eliminating conflict resolution does not necessarily cripple; that writing productions to contain explicit conditions on changes is a simple way to avoid repeated firing, and has efficiency benefits as well; that an attribute-value structure for WM and condition elements adds a useful bit of flexibility; and finally, that allowing WM elements to contain explicit references to each other dramatically cuts WM searching during evaluation.

Finally, we have shown that it is possible to obtain meaningful comparisons of related but different complex systems. The field of artificial intelligence could benefit from more comparisons of this sort.

REFERENCES

[1] CMU Speech Group (1977), "Speech understanding systems: Summary of results of the five-year research effort", (second version), Technical Report, Computer Science Department, Carnegie-Mellon University.

[2] Davis, R. and J. King (1975), "An overview of production systems", Report STAN-CS-75-524, Computer Science Department, Stanford University.

[3] Feigenbaum, E. A., B. G. Buchanan and J. Lederberg (1971), "On generality and problem solving: a case study using the DENDRAL program", Machine Intelligence 6, Edinburgh University Press.

[4] Forgy C. and J. McDermott (1977), "OPS: a domain-independent production system language", Proc. 5th IXA1. pp.933-939.

[5] Hayes-Roth, F. and V. Lesser (1977), "Focus of attention in the Hearsay-II speech understanding system", Proc. 5th 1XAI. pp.27-35.

[6] McCracken, D. (1978), "A production system version of the Mearsay-II speech understanding system", Ph.D. Thesis, Computer Science Department, Carnegie-Mellon University.

[7] McDermott, J. and C. Forgy (1978), "Production system conflict resolution strategies", in [15}

[8] McDermott, J., A. Newell and J. Moore (1978), "The efficiency of certain production system implementations", in [15].

[9] Mostow, DJ. and F. Hayes-Roth (1978), "A production system for speech understanding", in [15].

[10] Newell, A. (1973), "Production systems* models of control structures", in W. Chase (ed.), Visual Information Processing. Academic Press.

[11] Newell, A. and H. A. Simon (1972), Human Problem Solving. Prentice-Hall.

[12] Rychener, M. R. (1976), "Production systems as a programming language for artificial intelligence applications , Ph.D. Thesis, Computer Science Department, Carnegie-Mellon University.

[13] Shortliffe, EH (1974), "MYCIN: A rule-based computer program for advising physicians regarding anti-microbial therapy selection", Ph.D. Thesis, Computer Science Department, Stanford University.

[14] Smith, A.R. (1976), "Word hypothecation in the Hearsay-II speech understanding system", IEEE Int. Conf, on Acoustics. Speech and Signal Processing, pp.549-552.

[15] Waterman, D. and F. Hayes-Roth (eds.) (1978), Pattern-Directed Inference Systems. Academic Press.

[16] Wulf, W.A. and C. G. Bell (1972), "C.mmp - A multi-mini-processor", Proceedings of. the Fall Joint Computer Conference, pp.765-777.

AN INTRODUCTION TO NONMONOTONIC LOGIC

Drew McDermott
Department of Computer Science
Yale University
New Haven. Connecticut 06520

Jon Doyle
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

"Non-monotonic" logical systems are logics in which the introduction of new axioms can invalidate old theorems. Such logics are very important in modeling the beliefs of active processes which, acting in the presence of incomplete information, must make and subsequently revise predictions in light of new observations. We present the motivation and history of such logics, develop a model theory for one important non-monotonic logic, and prove the completeness of the first-order non-monotonic predicate calculus.

1. THE PROBLEM OF INCOMPLETE KNOWLEDGE

The relation between formal logic and the operation of the mind has always been unclear. Some of the more striking differences between properties of formal logics and mental phenomenology occur in situations dealing with perception, ambiguity, common-sense, causality and prediction. One common feature of these problems is that they seem to involve working with incomplete knowledge. Perception must account for the noticing of overlooked features, common-sense ignores myriad special exceptions, assigners of blame can be misled, and plans for the future must consider never-to-be-realized contingencies. It is this apparently unavoidable making of mistakes in these cases that leads to some of the deepest problems of the formal analysis of mind.

4c

Drew McDermott acknowledges the support of a Josiah Willard Cibbs instructorship. Jon Doyle thanks the Fannie and John Herti Foundation for supporting his research with a graduate fellowship. This research was conducted in part at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-7S-C-0643, and in part by NSF Grant MCS77-04828.

* *

This paper condenses the first part of "Non-Monotonic Logic I" (appearing as MIT Al Memo 486, and to appear in *Artificial Intelligence),* which discusses proof theory for non-monotonic logic, the evolution of logical theories, the relationship of non-monotonic logic to stronger logics, logics of incomplete information, and how non-monotonic logic models the logic of Doyle's Truth Maintenance System.

Some studies of these problems occur in the philosophical literature, the most relevant here being Rescher's [15] analysis of counterfactual conditionals and belief-contravening hypotheses. In artificial intelligence, studies of perception, ambiguity and common-sense have led to knowledge representations which explicitly and implicitly embody much information about typical cases, defaults, and methods for handling mistakes. [11,143 Studies of problem-solving and acting have attempted representing predictive and causal knowledge so that decisions to act require only limited contemplation, and that actions, their variations, and their effects can be conveniently described and computed. [1,2] Indeed, one of the original names applied to these efforts, "heuristic programming", stems from efficiency requirements forcing the use of methods which occasionally are wrong or which fail. The possibility of failure means that formalizations of reasoning in these areas must capture the process of revisions of perceptions, predictions, deductions and other beliefs.

Classical symbolic logic lacks tools for describing how to revise a formal theory to deal with inconsistencies caused by new information. This lack is due to a recognition that the general problem of finding and selecting among alternate revisions is very hard. (For an attack on this problem, see [15]. [12] surveys the complexities.) Although logicians have been able to ignore this problem, philosophers and researchers in artificial intelligence have been forced to face it because humans and computational models are subject to a continuous flow of new information. One important insight gained through computational experience is that there are at least two different problems involved, what might be called "routine revision" and "world -model reorganization".

World-model reorganization is the very hard problem of revising a complex model of a situation when it turns out to be wrong. Routine revision, on the other hand, is the problem of maintaining a set of facts which, although expressed as universally true, have exceptions. For example, a program may have the belief that all animals with beaks are birds. Telling this program about a platypus will cause a contradiction, but intuitively not as serious a contradiction as those requiring total revision. Classical logics, by lumping all contradictions together, have overlooked the possibility of handling the easy ones by expanding the notation in which rules are stated. That is, we could have avoided this problem by stating the belief as "If something is an animal with a beak, then *unless proven otherwise,* it is a bird." If we allow statements of this kind, the problem becomes how to coordinate sets of such rules, Each such statement may be seen as providing a piece of advice about belief revision; for our approach to make sense, all the little pieces of advice must determine a unique revision. This is the subject of this paper. Of course, even if we are successful, the world-model reorganization problem will still be unsolved. But we hope factoring out the routine revision problem will make the more difficult problem clearer.

2. APPROACHES TO NONMONOTONIC LOGIC AND THE SEMANTICAL DIFFICULTIES

Traditional logics are called *monotonic* because the theorems of a theory are always a subset of the theorems of any extension of the theory. ([11] introduced this name for this property of classical logics. [2] terms it the "extension" property.) In this paper, by *theory* we will mean a set of axioms. A more precise statement of monotonicity is this: If A and B are two theories, and A C B, then Th(A) C Th(B), where Th(S) = (p: Shp} is the set of theorems of S. We will be even more precise about the definition of -I- later.

Monotonic logics lack the phenomenon of new information leading to a revision of old conclusions. We obtain *non-monotonic logics* from classical logics by extending them with a modality ("consistent") well-known in artificial intelligence circles, and show that the resulting logics have well-founded, if unusual, model and proof theories. We introduce the proposition-forming modality M (read "consistent"). Informally, Mp is to mean that p is consistent with everything believed. (See [81) One small theory employing this modality is

(1) noon A M[sun-shining] D sun-shining
(2) noon
(3) eclipse => -win-shining,

in which we can prove

(4) sun-shining.

If we add the axiom

(5) eclipse

then (4) is inconsistent, so (4) is not a theorem of the extended theory.

The use of non-monotonic techniques has some history, but until recently the intuitions underlying these techniques were inadequate and led to difficulties involving the semantics of non-monotonic inference rules in certain cases. We mention some of the guises in which non-monotonic reasoning methods and belief revising processes have appeared.

In PLANNER [3], a programming language based on a negationless calculus, the THNOT primitive formed the basis of such reasoning. THNOT, as a goat, succeeded only if its argument failed, and failed otherwise. Thus If the argument to THNOT was a formula to be proved, the THNOT would succeed only if the attempt to prove the embedded formula failed.

Two related forms of non-monotonic deductive systems are those described in [8] and [161 McCarthy and Hayes give some indications of how actions might be described using modal operators like "normally" and "consistent", but present no detailed guidelines on how such operators might be carefully defined. Sandewall, in a deductive system applied to the frame problem used a deductive representation of non-monotonic rules based on a primitive called UNLESS. This was used to deduce conditions of situations resulting from actions except in those cases where properties of the action changed the extant conditions. Thus one might say that things retain their color unless painted.

Sandewall's interpretation of UNLESS was in accord with then current intuitions: UNLESS(p) is true if p is not deducible from the axioms using the classical first-order inference rules. Unfortunately, this definition has several problems, as pointed out by Sandewall. One problem Is that it can happen that both p and UNLESS(p) are deducible, since from a rule like "from UNLESS(C) infer D" D can be inferred, but at the same time UNLESS(D) is also deducible since D is not deducible by classical rules. These

problems are partly due to the dependence of the notion of "deducible" on the intention of deduction rules based on "not deducible". This question-begging definition leads to perplexing questions of beliefs when complicated relations between UNLESS statements are present For example, given the axioms

$$A$$
$$A \wedge \mathrm{Unless}(B) \supset C$$
$$A \wedge \mathrm{Unless}(C) \supset B,$$

we are faced with the somewhat paradoxical situation that either B or C can be deduced, but not both simultaneously. On the other hand, in the axiom system

$$A$$
$$A \wedge \mathrm{Unless}(B) \supset C$$
$$A \wedge \mathrm{Unless}(C) \supset D$$
$$A \wedge \mathrm{Unless}(D) \supset E,$$

one would expect to see A, C and E believed, and B and D not believed. One might be tempted to dismiss these anomalous cases as uninteresting. In fact, such cases are not perverse; rather, they occur naturally and are very important in many applications.

Spurred by Sandewall's presentation of the problems arising through such non-monotonic inference rules, Kramosil [5] considered sets of inference rules of the form

"From $\vdash p$, $\not\vdash q$, infer $\vdash r$",

where I- and H are tokens of the meta-language and the number of antecedents can be arbitrary. ([19] presents a working system organized around a theory-metal heory distinction in which such rules can be written.) Kramosil defined the set of theorems in such a system as the intersection of all subsets of the language closed under the inference rules. He noted that this set may not itself be closed under the inference rules, and showed that in the special case in which the inference rules preserve truth values (that is, are effectively monotonic) that if the set of theorems of the monotonic inference rules alone is also closed with respect to the non-monotonic inference rules, then this set is the set of non-monotonic theorems. Kramosil's conclusion was that a set of inference rules defines a formalized theory (one in which all formulas have a well-defined truth value) if and only if this same theory is that of the monotonic inference rules alone, which he interprets to mean that the non-monotonic rules are either useless or meaningless.

As we will show in this paper, Kramosil's interpretation was too pessimistic with regard to the possibility of formalizing such rules and their unusual properties. As we have argued above, the purpose of non-monotonic inference rules is not to add certain knowledge where there is none, but rather to guide the selection of tentatively held beliefs in the hope that fruitful investigations and good guesses will result This means that one should not *a priori* expect non-monotonic rules to derive valid conclusions independent of the monotonic rules. Rather one should expect to be led to a set of beliefs which while perhaps eventually shown incorrect will meanwhile coherently guide investigations.

One class of non-monotonic inferences consist of what might be called "minimal" inferences, in which a minimal model for some set of beliefs is assumed by assuming the set of beliefs to be a complete description of a state of affairs. Such techniques are discussed in [4,9,133, but none of these discuss the problem of revising beliefs in the face of new information or contradictions.

The semantical problems and routine belief revisions problems were in large part resolved by Doyle in his Truth Maintenance System (TMS) [I], and in related systems [6,7,17,18] In the TMS, each statement has an associated set of justifications, each of which represents a reason for holding the statements as a belief. These justifications are used to determine the set of current beliefs by examining the recorded justifications to find well-founded support (non-circular proofs) whenever possible for each belief. When hypotheses change, these justifications are again examined to update the set of current beliefs. This scheme provides a more accurate version of antecedent and erasing procedures of PLANNER without the need to explicitly check for circular proofs. The non-monotonic capability appears as so-called non-monotonic justifications, which support belief just in case some specified statements are believed, and other specified statements are not believed. This allows, for example, belief in a statement to be justified whenever no proof of the negation of the statement is known. This representation of non-monotonic justifications, in combination with the belief revision algorithms, produced the first system capable of performing the routine revision of apparently inconsistent theories into consistent theories.

3. L1NCUISTIC PRELIMINARIES

We settle on a language *L* which will be the language of all theories mentioned in the following. *L* is the usual language of predicate calculus augmented by the unary modality M. In this paper, the letters C, D, E and F will be used as syntactic variables ranging over propositional constant

letters. The letters p, q and r will be used for formulas.

The inferential system used defines a first-order theory to be a set of axioms including the following infinite class of axioms: For all formulas p, q and r:

(6) (i) $p \supset [q \supset p]$
(ii) $[p \supset [q \supset r]] \supset [[p \supset q] \supset [q \supset r]]$
(iii) $[\neg q \supset \neg p] \supset [[\neg q \supset p] \supset q]$
(iv) $\forall x p(x) \supset p(t)$

where $p(x)$ is a formula and t is a constant or a variable free for x in $p(x)$ and $p(t)$ denotes the result of substituting t for every free occurrence of x in $p(x)$, and

(v) $\forall x [p \supset q] \supset [p \supset \forall x q]$

if p is a formula containing no free occurrence of x. (These axioms are from [10].) These are the *logical* axioms. All other axioms are called *proper*, or non-logical axioms. The theory with no proper axioms is called the *predicate calculus* (PC). (Note that this theory also contains strings containing the letter M, so it is actually not strict PC.) The *sentential calculus* (SC) consists of axioms which are instances of (i), (ii) and (iii) only. A theory consisting only of the sentential calculus plus a finite number of statements is called a *statement theory*. In this paper, the letters A and B will be used to stand for theories.

4. PROOF-THEORETIC OPERATORS

The monotonic rules of inference we will use (also from [10]) are

(7) Modus Ponens: from p and $p \supset q$, infer q
Generalization: from p, infer $\forall x p$.

If S is a set of formulas, and p follows from S and the axioms of A by the rules (7), we say $S \vdash_A p$. We abbreviate $\vdash_{PC}$ by $\vdash$ alone. We define $Th(S) = \{p: S \vdash p\}$.

The particular inference rules (7) are not very important. Later in the paper, when we concentrate on statement theories, the rule of generalization will be dropped without much fanfare. All that is important is that the operator Th have the following properties, which together are called *monotonicity*:

(8) (i) $A \subseteq Th(A)$
(ii) If $A \subseteq B$, then $Th(A) \subseteq Th(B)$,

and the property (9) of *idempotence*

(9) $Th(Th(A)) = Th(A)$.

Clearly, any classical inference system satisfies these conditions. Condition (9) can also be viewed as a fixed point equation, stating that the set of theorems monotonically derivable from a theory is a fixed point of the operator which computes the closure of a set of formulas under the monotonic inference rules. A well-known property of the monotonic inference rules is that $Th(A)$ is the smallest fixed point of this closing process; in fact, that $Th(A)$ is the intersection of all S such that $A \subseteq S$ and $Th(S) = S$.

In order to deal with non-monotonic logic, we need a new inference rule like this one (which we will take back immediately):

(10) "If $\not\vdash_A \neg p$, then $\vdash_A Mp$."

That is, if a formula's negation is not derivable, it may be inferred to be consistent. As it stands, however, this rule is of no value because it is circular. "Derivable" means "derivable from axioms by inference rules", so we cannot define an inference rule in terms of derivability so casually.

Instead, we retain the definition of $\vdash$ as meaning monotonic derivability, and define the operator NM as follows: for any first-order theory A and any set of formulas $S \subseteq L$ (L, recall, is the entire language), let

(11) $NM_A(S) = Th(A \cup As_A(S))$,

where $As_A(S)$, the set of *assumptions* from S, is given by

(12) $As_A(S) = \{Mq: q \in L \text{ and } \neg q \notin S\} - Th(A)$.

Notice that theorems of A of the form Mq are never counted as assumptions. NM_A takes a set S and produces a new set which includes $Th(A)$ but also includes much more: everything provable from the enlarged set of axioms and assumptions which is the original theory together with all assumptions not ruled out by S. We would like to define $TH(A)$, the set of theorems non-monotonically derivable from A, by analogy with the monotonic case as

(13) "$TH(A)$ = the smallest fixed point of NM_A."

This "definition" tries to capture the idea of adding the non-monotonic inference rule (10) to a first-order theory A. This is plausible, since it demands a set such that all of its elements may be proven from axioms and assumptions not wiped out by the proofs. Unfortunately, there is in general

no appropriate fixed point of NM_A. It can happen that a theory has no fixed point under the operator NM_A. Even if there are fixed points, there need not be a smallest fixed point.

For example, consider the theory T1 obtained as

(14) $$T1 = PC \cup \{ MC \supset \neg D, MD \supset \neg C \},$$

where C and D are propositional constants. NM_{T1} has two fixed points, which can be called F1 and F2. F1 contains $\neg C$ but not $\neg D$, and F2 contains $\neg D$ but not $\neg C$. Since $\neg D$ is not in F1, MD is in F1, and so $\neg C$ is in F1. Similarly, the presence of $\neg D$ in F2 keeps $\neg C$ out and MC in F2. The problem is that neither $F1 \cap F2$ nor $F1 \cup F2$ is a fixed point of NM_{T1}. Since neither $\neg C$ nor $\neg D$ is in $F1 \cap F2$, MC and MD are both in $NM_{T1}(F1 \cap F2)$, so $\neg C$ and $\neg D$ are in $NM_{T1}(F1 \cap F2)$, so $F1 \cap F2 \neq NM_{T1}(F1 \cap F2)$. Similarly, both $\neg C$ and $\neg D$ are in $NM_{T1}(F1 \cup F2)$, so applying NM_{T1} to the union results in a smaller set. So in this case there is no natural status for $\neg C$ and $\neg D$.

An example of a theory with no fixed point of the corresponding operator is the theory T2 obtained as

(15) $$T2 = PC \cup \{ MC \supset \neg C \}.$$

In this case, NM_{T2} has no fixed point, since alternate applications of the operator to any set produce new sets in which either both MC and $\neg C$ exist or neither exist.

Therefore, we must accept a somewhat less elegant definition of TH. Let us define TH as follows:

(16) $$TH(A) = \cap(\{L\} \cup \{S: NM_A(S) = S\}).$$

That is, the set of provable formulas is the intersection of all fixed points of NM_A, or the entire language if there are no fixed points. We will use the abbreviation $A \vdash p$ to indicate that $p \in TH(A)$. With this definition, neither MC nor MD is a theorem of T1 in (14), but $MC \vee MD$ is. In the following, we will abbreviate $\{S: NM_A(S) = S\}$ as $FP(A)$, and (somewhat abusing the terms) call the elements of this set *fixed points* of the theory A.

5. MODEL THEORY

An *interpretation* V of formulas over a language *L* is a pair $\langle X, U \rangle$, where X is a nonempty set, and U is a function which associates relations and values over the domain X with each predicate, variable, constant and propositional constant letter in the usual fashion. That is, for each n-ary predicate letter P, $U(P) \subseteq X^n$; for each variable or constant x, $U(x) \in X$; and for each propositional constant letter C, $U(C) \in \{0, 1\}$. Using this mapping function U we define the value $V(p)$ of a formula p in the interpretation V to be an element of $\{0, 1\}$ satisfying the following conditions: For an atomic formula $p(x_1, \ldots, x_n)$, the value is 1 if $\langle U(x_1), \ldots, U(x_n) \rangle \in U(p)$, and is 0 otherwise. $V(\neg p) = 1$ if $V(p) = 0$, and is 0 otherwise. $V(p \supset q) = 1$ if either $V(p) = 0$ or $V(q) = 1$, and is 0 otherwise. $V(\forall x p) = 1$ if for all $y \in X$, $V'(p) = 1$, where $V' = \langle X\ [y / x]U \rangle$, where $[y / x]U$ is the mapping derived from U by changing its value at the point x to the value y. $V(\forall x p) = 0$ otherwise. If $V(p) = 1$, we say that V *satisfies* p, and write $V \models p$.

A *monotonic model* of a set of formulas $S \subseteq L$ is an interpretation V which satisfies each formula in S, that is, $V(p) = 1$ for each formula $p \in S$. A *non-monotonic model* of a theory A is a pair $\langle V, S \rangle$, where V is a monotonic model of S, and $S \in FP(A)$. When the context makes the intended meaning clear, we will use the term *model* of A to mean either a non-monotonic model, a monotonic model, or an element of $FP(A)$ for the theory A.

Although unorthodox, this definition provides a meaning for formulas Mp which reflects the proof-theoretic property that "p is consistent with what is believed". This notion is made precise by including in the model a set of "current assumptions" (namely, $As_A(S)$). A model for a theory must assign 1 to all of these assumptions, so the effect is that Mp is assigned 1 in a model if $\neg p$ is not derivable and $\neg Mp$ is not derivable from the current assumptions and the original theory, that is, if p is consistent with what is "believed" in the model. Unfortunately, Mp may be assigned 1 in some model even when $\neg p$ is derivable (for example, when no axiom mentions Mp at all). This indicates that the logic is too weak. We discuss this problem in the full paper. We present a stronger logic, with a more elegant model theory, in a forthcoming paper.

With this definition of model, we can justify the definition of provability.

Theorem 1. (Soundness) If $A \vdash p$, then $V \models p$ for all models $\langle V, S \rangle$ of A.

Proof: Assume $A \vdash p$. If there are no models of A, the theorem follows trivially. Otherwise, p is a member of every fixed point of A. But since every model of A is a monotonic model of a fixed point of A, every model assigns 1 to p. ∎

Theorem 2. (Completeness) If $V \models p$ for all models $\langle V, S \rangle$ of

A, then A⊢p.

Proof: Assume that it is not true that A⊢p. Thus there is a fixed point S of NM_A which does not contain p. Now Th(S) = S by idempotence, so S⊬p. But the predicate calculus is complete, so some monotonic model V of S has V(p) = 0. ■

It is not surprising that we have completeness, since the definition of truth makes reference to provability. The proof was for first-order theories, but it can easily be generalized to any complete formal logic. For example, if we take care not to confuse M with the SS operator "possibly", we can easily get a complete non-monotonic extension of SS.

ACKNOWLEDGEMENTS

We wish to thank Gerald Jay Sussman, Rohit Parikh, David Harel, Mitch Marcus, Lucia Vaina, and Vaughan Pratt for helpful criticisms and comments,

REFERENCES

[I] Doyle, J., "Truth Maintenance Systems for Problem Solving," MIT MS Thesis, 1977. See "A Truth Maintenance System", MIT AI Memo S21, 1979.

[2] Hayes, P. J., "The Frame Problem and Related Problems in Artificial Intelligence," in A. Elithorn and D. Jones, editors, *Artificial and Human Thinking,* San Francisco: Josey-Bass, 1973.

[3] Hewitt, C. E., "Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot," MIT AI Laboratory TR-258, 1972.

[4] Josht, A. K., and S. J. Rosenschein, "A formalism for relating lexical and pragmatic information: its relevance to recognition and generation," *Proc. Workshop on Theoretical Issues in Natural Language Processing,* Cambridge, Massachusetts, 197S, pp. 79-83.

[5] Kramosil, I., "A Note on Deduction Rules with Negative Premises," *Proc. Fourth IJCAI,* pp. S3-S6, 197$.

C6] London, P. E., "Dependency Networks as a Representation for Modelling in Ceneral Problem Solvers," Department of Computer Science TR-698, University of Maryland, 1978.

C73 McAllester, D. A., "A Three-Valued Truth Maintenance System," MIT AI Lab, Memo 473,1978.

[8] McCarthy, J. and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in B. Meltzer and D. Michie, *Machine Intelligence 4,* New York: American Elsevier 1969, pp. 463-SOI

[93 McCarthy, J., "Epistemological Problems of Artificial Intelligence," *Proc. Fifth IJCAI,* pp. 1038-1044, 1977.

[103 Mendelson, E., *Introduction to Mathematical Logic,* New York: Van Nostrand Reinhold, 1964.

[11] Minsky, M., "A Framework for Representing Knowledge," MIT AI Lab, AI Memo 306,1974.

[12] Quine, W. V., and J. S. Ullian, *The Web of Belief* second edition, New York: Random House, 1978,

[13] Reiter, R., "On Closed World Data Bases," Department of Computer Science, University of British Columbia, TR 77-14, 1977.

[14] Reiter, R., "On Reasoning by Default," *Proc. Second Symp. on TINLAP,* Urbana, Illinois, 1978.

[I5] Rescher, N., *Hypothetical Reasoning,* Amsterdam: North Holland 1964.

[16] Sandewall, E., "An Approach to the Frame Problem, and its Implementation," in B. Meltzer and D. Michie, editors, *Machine Intelligence 7,* New York: John Wiley and Sons, 1972, pp. 19S-204.

[17] Stallman, R. M., and C. J. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence,* Vol. 9, No. 2, (1977), pp. 13S-196,

[18] Thompson, A., "Network Truth Maintenance for Deduction and Modelling," Jet Propulsion Lab, 1979.

[19] Weyhrauch, R. "Prolegomena to a Theory of Formal Reasoning," Stanford AI Lab, Memo AIM-31S, 1978.

LEARNING TO USE ANALOGIES

John McDermott
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract. In order for a system to learn how to do new tasks, it must be capable of assimilation and accommodation. The assimilation capability enables a system to relate unfamiliar situations to situations that it knows about. Through assimilation, an unfamiliar situation is transformed, for a time, into a familiar situation. The accommodation capability enables a system to make such transformations permanent. ANA, the system described in this paper, is a production system that is capable of both assimilation and accommodation. Initially, ANA has a few methods for accomplishing a variety of simple tasks. When it is given a not too unfamiliar task, it performs that task by analogy with one of the tasks it has a method for. When it accomplishes a new task (and typically this happens only after the method has been extended to handle problems that it was not designed to cope with), it stores the knowledge of how it did the task. If ANA is subsequently faced with the same task, it recognizes the task and performs it using the knowledge previously gained.

1. INTRODUCTION

Any AI system that hopes to amount to much must have two capabilities. First, it must be capable of assimilation; it must be able to bring to bear whatever knowledge it has that is relevant to an unfamiliar task — even though that knowledge was acquired in a variety of unrelated contexts. Second, it must be capable of accommodation; It must be able to augment and modify its knowledge so that unfamiliar tasks become familiar. The work described in this paper shows one way in which these two capabilities can, in a modest way, be realized.

The system described, ANA, has limited knowledge about how to function in a simple environment. ANA is a production system; its productions are organized as a set of methods. The particular (and only) assimilation and accommodation strategy that ANA employs is to use these methods analogically. When given an unfamiliar task, it maps the description of a task for which it has a method into the description of that unfamiliar task. As it uses its method, instead of executing the actions prescribed by the method, it executes the actions dictated by the mapping. Whenever ANA uses a method successfully on an unfamiliar task, it builds a production that associates that method with the description of the task (thereby making the unfamiliar task familiar). If in the course of doing a task the analogy breaks down, ANA attempts to patch the method; if it finds a patch, it builds a production that associates the patch with whatever caused the breakdown.

Thia raaaarch waa aponaorad by tha Dafanaa Advanced Raaaarch Projacts Agancy (DOD), ARPA Ordar No 3597, tod monitored by tha Air Forca Avionica Laboratory *under* Contract F33615-7S-C-1151 Tha viawa a*nd* conclusion! containad in this documant ara those of the author and should not ba intarpratad aa rapraaantinf tha official policies, either expressed—*6* or implied, of tha Oafanaa Advancad Rasearch Projacta Agancy or tha US Government.

[2]For a general diacussion of tha iaauaa that faca anyona dasigning a system with essmition a accommodation capabilities, — *[6]*

2 SOME CONTEXT

The production system architecture used to implement ANA is called 0PS2 [2, 4, *7).* An 0PS2 production system consists of a collection of productions held in production memory and a collection of data elements held in working memory. A production is a conditional statement composed of condition elements and action elements. Condition elements are templates; when each can be matched by an element in working memory, the production containing them is said to be instantiated. The production system interpreter operates within a control framework called the recognize-act cycle. In recognition, it finds all instantiations, and In act, executes the action elements of one of them. The recognize-act cycle is repeated until either no production can be instantiated or an action element explicitly stops the processing. Recognition can be divided into match and conflict resolution. In match, the interpreter finds the conflict set, the set of all instantiations of productions that are satisfied on the current cycle; 0PS2 is implemented in such a way that the time needed to compute the conflict set is essentially independent of the size of production memory (see [1]). In conflict resolution, the interpreter selects (on the basis of a few simple rules) one instantiation to execute (see [5]). The actions that can be performed include adding elements to and deleting elements from working memory and building new productions composed of elements in working memory.

To provide ANA with the capability of performing a variety of tasks, its knowledge is represented as a set of methods. Each method contains the knowledge that ANA needs in order to achieve some goal. Since part of this knowledge is the knowledge of subgoals that have to be achieved, ANA's knowledge is organized, though only implicitly, as a hierarchy of methods. The productions that comprise each method have condition sides with a quite simple form. This form is perhaps most easily described in terms of the type of data element that each condition can match. A data element contains information about a goal or about some feature of ANA's environment. In addition, each data element contains a subelement, which I will refer to as the marker, that specifies, among other things, the element's type (see [8]). There are four types of data elements: goals, constraints, percepts, and concepts.

A data element marked "goal" contains two subelements (plus a marker). One of these is a pointer to the set of constraints on the action to be performed; the other is a pointer to the set of constraints on the object to be acted on. A data element marked "constraint" contains three subelements (plus a marker); constraints are attribute-name-value triples. A goal element and the two sets of constraints it points to are collectively a goal description. When ANA attempts a task, it finds an object that fits the description stipulated by the object constraints and performs the type of action stipulated by the constraint containing the type of the action. The other action constraints specify the expected effect of performing that type of action on the class of objects described. Figure 2-1 displays a goal description. The stipulated action type is paint. The object to be painted is a yellow table in a location whose label is L32. The expected effect is a change to the surface of the objectj specifically, the color of the object should be red once the goal is achieved.

```
(act*1 object*2 (goal not-achieved 3))

(type act*1 (constraint given 4) paint)
(effect act*1 (constraint given 4) effect*5 surface)
(color effect*5 (constraint given 4) red)
(type object*2 (constraint given 4) table)
(color object*2 (constraint given 4) yellow)
(location object*2 (constraint given 4) location*6)
(label location*6 (constraint given 4) L32)
```

Figure 1: A goal and a set of constraints

A data element marked "percept" contains information about one of the objects in ANA's environment. In ANA's world, an object is just a bundle of percepts (a set of attribute-value pairs). ANA "sees" an object whenever a production containing the operator ©scan fires. ©scan takes a (partial) description (a set of attribute-value pairs) as its argument and searches the environment for an object matching that description. If it finds such an object, it deposits a sot of elements, each of which is marked "percept", in working memory) each of these data elements corresponds to one of the attribute-value pairs of the object found. Clearly, In order for ANA to be able to distinguisn among objects, these percepts must somehow be linked Thus, ©scan puts a "token-name", as well as an attribute and a value, in each percept that it deposits in working memory.

A data element marked "concept" can be thought of as a processed percept. When ANA looks at something, it does so for a reason — ie, to find out something about the object. For each percept that contains information that it cares about, it asserts an identical element marked "concept". In addition, it asserts an element, also marked "concept", that relates the set of constraints on the object with the object's token-name. The concept marker has several uses: (1) It enables ANA to focus its attention on particular features of en object. (2) ANA can pretend that an object has a value that it in fact does not have by asserting a concept containing that value. (3) ANA can use concepts to store non-perceptual knowledge about an object. Figure 2-2 shows a collection of percepts (the attribute-value pairs that comprise the object that ANA thinks of as tablet* 1); the figure also displays two concepts. When ANA is given a task, the description of the object to be operated on may, but need not, uniquely specify *an* object. In order to do the task, ANA must find an object that matches the description given. ANA's productions distinguish between information that constrains the selection of an object and information about a particular object. Elements containing the first sort of information are marked "constraint", while elements containing the second sort are marked "concept". If ANA were given the task shown in Figure 2-1, and if it looked for a chair whose color was blue in order to determine its location, then ANA would assert the concepts shown in Figure 2-2. The concepts would indicate to ANA that the table It refers to as tablet *1 is the one to be operated on.

```
(type table*1 (percept true 7) table)
(color table*1 (percept true 7) yellow)
(weight table*1 (percept true 7) light)
(location table*1 (percept true 7) L*32)
(position table*1 (percept true 7) (1))

(location table*1 (concept true 8) L*32)
(token object*2 (concept true 8) table*1)
```

Figure 2: Some percepts and concepts

As is evident from Figures 2-1 and 2-2, an element's marker specifies more than just the type of the element. It contains information about the status of the element. A constraint, for example, may be "given", "not-given", or "possible". The marker also contains information indicating the element's recency.

3. ANA'S TASK ENVIRONMENT

ANA's task is to manage a paint shop. ANA operates a machine that sits in the middle of the shop. The only things in the shop other than the machine are a variety of paintable objects. The shop and the things in it ere all abstractions of real objects; they are represented as collections of attribute-value pairs. The shop is a 3 by 6 matrix; each square in the matrix is a stack with type location, one of 18 positions ((1 1) - (3 6)), a label (L11 - L36) and a composition (the stack of objects that occupy it). Each of the objects in the shop have a type (eg, chair, box, desk), a location, a position in the location, a color, a weight, and a state (clean or dirty). The shop and its current contents are shown in Figure 3-1.

ANA has five operations that it can perform on the objects in the shop. One of its operators, ©spray, starts the machine. Three operators, @carry, ©push, and ©cart, enable it to move objects from one location to another. ANA's other operator, @scan, as I mentioned, enables it to see what is in the shop. In keeping with the somewhat artificial nature of the shop and its contents, there are some artificial constraints on these operators. Which of the move operators is applicable is determined by the weight of the object to be moved, ©carry can be used only with light objects, ©push only with heavy objects, and ©cart only with really heavy objects; ©carry cannot be applied to an object unless it is the top object in a stack, and neither ©push nor ©cart can be applied unless the object is the only object in a location. If *an* object is carried to a location that already contains four objects, it will fall onto en adjacent location; if an object is pushed or carted to a location that already contains one or more objects, it will end up in an adjacent location. In order to paint an object, ANA must move it to location L23 and must put a paint can on top of the machine (in location L24): the operator, ©spray, will modify the color of the object in L23 and cause it to be output in location L34. If there is not exactly one object in L23, ©spray does nothing; if there is no sprayable substance on top of the machine, ♦he object in L23 will be output to L34 unchanged. The move operators all take the "token-name" of the object to be moved and the "token-name" of the location to which it is to be moved as their arguments. Thus these operators all presuppose a prior ©scan.

ANA's initial task-specific knowledge consists of six methods. ANA has a method for painting tables whose location is L32 red. ANA also has a method for clearing off the top of desks that are in L32. ANA's other methods are all methods for transporting objects. One of these methods enables it to move boxes that are in L25 to some unspecified location. A second method enables ANA to carry tables in L32 to L23. A third method enables ANA to carry cabinets in L11 to L16; this method assumes that the position of the cabinet In the stack is known (and is something other than top, and so calls the cleartop method as a submethod. The fourth method enables ANA to cart cabinets In L31 to L23.

Two of the tasks that ANA has been given will be used as examples in the following sections. The two task* *are:* (1) "paint the blue chair in L21 red and then move it to L35", and (2) "wash the thing in L12". ANA's In»tel methods do not enable it to do either task directly; n both cases ANA must perform the tasks by analogy.

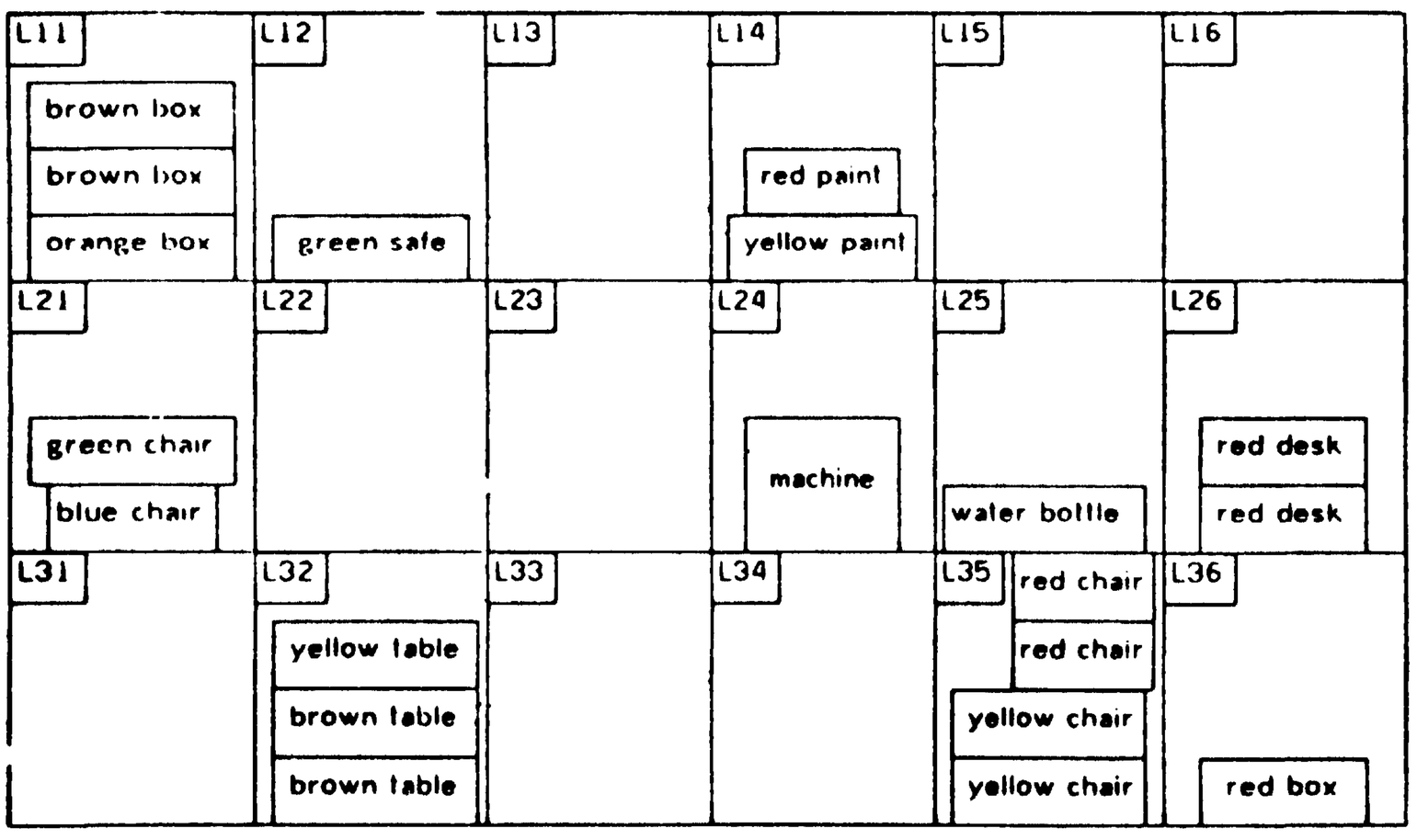


Figure 3-1: ANA's paint shop

4. ANA'S LEARNING STRATEGY

To do the tasks described in the previous section, ANA has to be able to assimilate new knowledge. Since the few methods that it has for getting things done in the shop are not (for the most part) the methods needed to do these tasks, it must find a way to apply the knowledge it has to the unfamiliar tasks. ANA's strategy is to find a method that is adequate for a related task and assume it will be adequate for the new task. Since this assumption will often lead it to do the wrong thing, it must have ways of correcting its errorful actions. ANA's solution is to watch for signs that it is not accomplishing its task. Three kinds of problems can arise. If the method it is using generates a subgoal, ANA tries to determine whether that goal is appropriate within the context of its current task; if ANA decides it is not, it substitutes an appropriate goal. If ANA sees that it has achieved a result other than the one it intended to achieve, it assumes that the method it is using does not take into account one or more constraints imposed by the particular task at hand; in this case it tries to determine what the additional constraints are and then attempts the task again keeping those constraints in mind. If ANA sees that It is unable to achieve a result because of some constraint imposed by the method it is using, it checks with its task master to see if that constraint is necessary; If the task master indicates that it is not necessary, ANA ignores the constraint.

The first time ANA does an unfamiliar task it spends a great deal of time selecting methods to use, establishing and extending mappings, and recovering from errors that it falls into. On subsequent occasions, ANA can do the tasks more directly. Roughly speaking, all that ANA does is store worked-out analogies. It associates each newly familiar goal description with a method that can achieve that goal analogically. ANA does not try to generalize; thus the only goal descriptions that it recognizes (ie, does not have to assimilate) are goal descriptions that it has previously encountered. Though this limitation is significant, it is not as severe as It perhaps sounds. In the first place, as ANA engages in some task, it typically generates a number of subtasks for itself. The knowledge that it stores about how to perform each of these subtasks is accessible in any context — ie, not just in the context of the particular task in which the knowledge was gained. Thus if ANA is given an unfamiliar task, though it will have to use its analogy mechanism in order to do the task, it may turn out that some of the subtasks that it generates in the course of doing that task are identical to subtasks that it has already learned how to do. Whenever this happens, ANA can use the knowledge that It previously acquired. Secondly, although ANA learns how to do a specific task, what this means is that it learns what to do given a particular set of constraints on the action and the object to be acted on. If ANA is given a task that includes those constraints, but others besides, It can use its acquired Knowledge. Of course the fact that there are additional constraints may mean that ANA's knowledge will be inadequate for the task; but if to, ANA can make use of its error recovery methods to further refine its knowledge.

In this section I will give a general description of ANA*s learning strategy. I will first describe how ANA sets up analogies *and* will then describe how it recovers when an analogy breaks down.

4.1 Setting Up Analogies

When ANA is given a task that it knows how to do (ie, a task for which it has a method), it needs no special mechanism to make contact with that method) the productions comprising the method are satisfied by elements in working memory and fire. When ANA is given an unfamiliar task, however, it must select an appropriate method and then assert elements that will satisfy that method's productions. To enable ANA to make contact with an appropriate method, each of its methods has associated with it a method description production. Each method description production contains two condition elements that specify the type of action and object with which its method is equipped to deal. It may contain any number of additional condition elements specifying constraints on the action and object. When given a unfamiliar task, ANA first generates action types and object types that are related to the action and object types associated with its current task. The knowledge it uses is represented as a tree of types) each node of the tree is defined by a set of productions. One of the productions has as one of its action elements an element indicating the supertype of that type; each of the other productions has as one of its action elements an element indicating an instance of that type. After some number of these isa net productions fire, one of the method description productions will be satisfied and will fire. The action part of the production contains the set of constraints that the associated method is familiar with. These constraints are marked "constraint possible" rather than "constraint given" to indicate their provisional nature. A method description production simply proposes its method) before ANA uses the method It evaluates the method's adequacy.

[3] Anyone familiar with HACKER [9] will have noticed that HACKER and ANA have similar aspirations. Both systems start out with a limited amount of knowledge and try to build on that knowledge in such a way that they become increasingly better able to accomplish unfamiliar tasks. But they have rather different building strategies. When HACKER encounters an inadequacy in one of its procedures, it examines the code; when it finds the bug, it rewrites the procedure. ANA is more of a hacker. When ANA encounters an inadequacy in one of its methods it determines what behavior is appropriate and associates that behavior with a description of its current context. Each of ANA's fixes is pre-eminently a patch.

[4] ANA's capabilities are described in considerably more detail in [3].

The first step in this evaluation involves setting up a mapping between the description of its actual goal and the pseudo-goal-description generated by the method description production. ANA finds all of the attributes in the description of the actual task ("constraint given"s) for which there are corresponding attributes ("constraint possible"s) asserted by the method description production. For each such pair of constraints, ANA asserts an element that indicates that the method's value (hereafter called the "pseudo-value") is to be mapped into the stipulated value.

After these mappings are established, the evaluation of the proposed method begins in earnest. ANA first checks to determine if there is any reason to think that the method will not work for the task at hand. Initially, it has no reason to think badly of any method. But as we will see in more detail later, when it encounters a problem as it tries to use a method, it builds a production that associates with the type of action that It is attempting an indication of the cause of the problem. If, for example, ANA attempts to carry an object that is buried down in the middle of some stack, its initial attempt will fail. But ANA will recognize that the cause of the problem is the position of the object in the stack, and will associate a pre-condition with the action type "carry", ie, that the object to be carried must not have anything on top of it. The problems that arise can be distinguished on the basis of how likely it is that they can be overcome. ANA divides problems into two types: (1) problems that arise because of some mutable attribute of an object, and (2) problems that arise because of *an* immutable attribute of an object. When a pre-condition is violated by an immutable attribute of an object, ANA concludes that the method it is considering is unlikely to work. When there are no immutable violators, but a pre-condition is violated by a mutable attribute, ANA concludes that the method will probably work, but only if a subgoal that can satisfy the pre-condition is achieved.

After ANA has evaluated a method, it typically returns to its isa nets and generates additional types with the hope that a better method will turn up. How long it spends searching depends on how good its current best candidate is; it spends more time if the current candidate falls into the "unlikely to work" category than if it falls into the "will probably work" category.

Once ANA has selected a method to use, it must somehow change its conceptual world in a way that makes it possible for it to use the method. The problem is that all of the productions comprising each of its methods are sensitive only to those constraints that are *marked* "constraint given". Since the evaluation stage results only in a set of constraints marked "constraint possible", ANA still cannot do anything. But the solution is straight-forward. ANA asserts, for each constraint marked "constraint possible", an element that is identical to the "constraint possible" element except that it is marked "constraint given". Now in ANA's working memory in place of the single set of constraints stipulated in the description of the actual task, there are three sets of constraints: (1) the original set (each marked "constraint given"); (2) the set generated during the evaluation stage (each marked "constraint possible" and containing the pseudo-values expected by the method, rather than the stipulated values); and (3) a set that will match the productions in the method selected (each marked "constraint given", but containing the same pseudo-values as the set marked "constraint possible"). As simple as this device is, it is sufficient to enable ANA to make effective use of the selected method. Whenever ANA runs into difficulties, it has all the information that it needs in order to distinguish between what it is actually trying to do and what it is pretending to do. Since ANA is always implicitly aware of the difference between what it is trying to do (Its actual goal) and what it is pretending to do (its pseudo-goal), whenever its pretending results In an action that is inconsistent with its actual goal, it can use this difference to figure out how to get back on the right path.

Next ANA prepares for one type of difficulty that might arise as it uses the method analogically -- the "unexpected result". ANA's method description productions typically include one or more condition elements that indicate the expected effect of the method. As ANA prepares to use the method, it builds a production that watches for the violation of this expectation for the particular task at hand. The method description production for ANA's "paint table" method, for example, indicates that the expected effect is that the color of the table will become red. If ANA is given the task of painting some object yellow, it wants to be sure that the effect actually achieved is that the table becomes yellow. So ANA builds a production that keeps its eye, so to speak, on the object; if after the object Is painted it is not yellow, this production can fire.

ANA is now ready to use the method it has selected. However, if it is to avoid having to set up the same analogy on subsequent occasions, it must store the knowledge it has gained in a readily accessible form. Thus, just before it actually attempts an unfamiliar task, ANA builds a production that will "set the stage" if It is ever again asked to do the task it is about to try. In order for this production to fire at the appropriate times, it must be sensitive to the set of stipulated constraints that comprise ANA's current goal. Thus, ANA generates a list containing the element in working memory that points to the constraints on this goal and each associated element, marked "constraint given" for which there is no corresponding element marked "constraint possible". The pointer in each of these elements is replaced by a variable, and the list of elements becomes the conditional part of the production that ANA will build. The action side of the production must make contact with the method that has been selected for the task; consequently, ANA generates a list containing each "constraint given"/"constraint possible" pair linked to the current goal, plus each of the map

elements associated with these constraints. After the pointer in each of these elements is replaced by a variable, the list of elements becomes the action part of the production that ANA will build.

At this point, ANA could simply build the production, but there is a problem. Since it has not yet tried to use the method it has selected, it has no idea whether the method will work. If ANA were to build the production before trying the method, and the method turned out not to work for the current task, ANA would have a piece of faulty knowledge. ANA's solution is to build a production whose condition side contains just one element ~ an element that will match the current goal element when that element is marked "goal achieved". If this production fires, the action side builds the production described above.

4.2 Using Analogies

As I indicated above, there are three ways in which ANA can get into trouble as it tries to use a method analogically: (1) The method may generate a subgoal that is inappropriate in the context of its current task. (2) The method may be under-specified (ie, insufficiently constrained), and so ANA ends up operating on (or trying to operate on) some object in an inappropriate way. (3) The method may be over-specified (overly constrained), and so ANA achieves a goal, but does not know it.

Whenever a subgoal is generated by a method, a new (distinct) description of the action and its expected effect is asserted. But the description of the object to be acted on may or may not be new. If it is not new, ANA simply checks the description of the action to see if it contains any new constraints. ANA has a production whose purpose is to watch for any constraint whose value is the pseudo-value of some map element. Whenever such an element is asserted, this production fires; the result is two additional assertions. One is identical to the assertion that triggered the production, except it is marked "constraint possible"; the other is an assertion, marked "constraint given", whose value is the stipulated value of the map element. If the object to be acted on is new, ANA tries to determine whether it is an appropriate object for the task at hand. If ANA decides that it is not appropriate, it replaces the object description asserted by the method with the description of an appropriate object. For example, ANA has a method for painting tables that generates the subgoal of putting a paint can on top of its "paint machine". If ANA is given the task of washing an object, it decides that it should substitute water for paint.

ANA has two ways of determining that it is using an under-specified method. One is to wait until one of its operations on an object in its environment fails. When an operator (eg, @carry, @cart) fails, an element marked "action failed" is deposited in working memory; this element identifies the cause of of the failure. If, for example, @carry fails because the object to be carried is not the top object in a location, the "action failed" element will Indicate that the offending attribute is "position" and that carry will work only if the object Is at the top of the stack. ANA has a production that watches for an "action failed" element to be deposited in working memory. When such an element is asserted, this production fires and generates the goal of recovering from the error. The first thing that the error recovery method does is check to determine If the offending attribute is mutable or immutable. Since ANA responds differently depending on whether the offending attribute is mutable or immutable, we will consider each case separately.

We have already seen what happens in the mutable case. If an operator fails because of a mutable attribute of some object, ANA builds a production associating the current action type with that attribute and the value it must have. If ANA knows how to change the attribute, it generates a subgoal to do so; if it does not know, it asks the task master. This fix is sufficient to enable ANA to deal effectively with the problem. In the future, if ANA considers a method whose action type Is associated with the necessary, but mutable, value of some attribute, and if the object does not have that value, ANA knows that to use the method It is considering, it must generate a subgoal to satisfy this pre-condition.

The immutable case is more complex. If the problem is due to some feature of the environment that cannot be changed, ANA must find a new way of dealing with the situation. What this means, in ANA's simple world, is that ANA must use a different operator to effect whatever change it desires. When ANA finds itself in such a situation, it builds a production whose condition side is the same as the condition side of the production that actually operates on the environment; this production generates the subgoal of finding the value of that attribute of the object that is relevant to the operator. It builds a second production whose condition side is identical to the first except that it includes a condition element that is satisfied if the value of that attribute is different from the value required by the operator; the action side of this production, executes a different operator. The reason for making the condition side of this production a special case of the original production is that ANA wants the original production to fire unless the information that is obtained about the object to be operated on indicates that the original production's operator is inappropriate.

ANA's other way of determining that it is using an under-specified method has already been discussed. When ANA decides to use a method analogously, it builds a set of productions to watch for unexpected results. Whenever ANA operates on an object and the result is different from that expected by one of these productions, it will fire. When it does, ANA first asks how to undo the unwanted result and then asks for an indication of what went wrong. ANA expects to be told that the value of one of the attributes of one of the

objects in its environment is wrong and expects to be told what the value should be. ANA then treats this information in exactly the same way that It treats the information contained in an element marked "action failed".

ANA's method for determining whether the method it is using is over-specified is to ask. It has a production that waits until the goal of finding some object is generated, and then looks to see if the values of any of the expected effect constraints on that object differ from the values of the corresponding percepts. If there is a discrepancy, the production fires. ANA asks the task master whether the value of the attribute has to be the stipulated value. If the task master answers "yes", then ANA keeps looking until it finds an object with the stipulated value. If the task master answers "no", ANA assumes that the method it is using is over-specified, and asserts an element that indicates that the object that it has found has the stipulated value (even though it does not). In order to produce a permanent fix, ANA builds a production that will assert that element whenever it is given *an* object to find In the context of the current action type.

3. TWO SAMPLE TASKS

In this section, ANA's behavior on the two sample tasks is described. Taken together, the two tasks provide examples of most of ANA's capabilities.

5.1 The Chair Painting Task

When ANA is given the task of painting the blue chair In L21 *red,* since it has no method that is directly applicable, it must make contact with an appropriate method. After a few of its isa net productions fire, the method description production for painting tables is satisfied and fires. The action type "paint" is mapped Into itself, the description of the expected effect of painting is mapped into itself, the object type "table" is mapped into "chair", and the description of the location of the table (that it is in L32) is mapped into the description of the location of the chair (that it is in L21). ANA then looks for other candidate methods. Since it finds none, it selects the "paint table" method as the method to use. For each element asserted by tne method description production marked "constraint possible", it asserts a corresponding element marked "constraint given". It then builds a production that will fire if the chair, after being painted, is not *red,* and It builds a production which will build a production that associates the task of painting a chair In L21 with the "paint table" method if that method is used successfully. The first action of the "paint table" method Is to assert the subgoal of finding the table (chair) to be painted. Once the chair is found (by means of @scan), the subgoal of carrying the table (chair) to L23 is asserted. Since ANA has a method for carrying tables from L32 to L23, and since the pseudo-goal generated by the paint method is to carry a table from L32 to L23, ANA does not need to search for a method to use to accomplish the carrying goal.

The first action of the "carry table" method is to assert the subgoal of finding the table to be carried; again the blue chair in L21 is found. Then the method asserts the subgoal of finding the location to which the table (chair) is to be carried. The goal description stipulates that this location is to be L23 and the expectation is that this location is empty. Since L23 is a new object (ie, is not mentioned in the initial description of the task to be performed), ANA has to decide whether for its current task a different location should be substituted for L23. It has no knowledge that leads it to think that a substitution should be made, so it accepts L23 as the location to which it should carry the table (chair). The next action is to use the @carry operator to move the table (chair) to L23. This operator is applied, but falls because the chair it tries to carry (the blue chair) has another chair on top of it. ANA knows that the position of an object in a stack is a mutable attribute. Thus it builds a production that associates with the action "carry" the information that in order for *an* object to be carried it must be at the top of a stack. Whenever ANA subsequently considers carrying an object, this production will fire. ANA knows that if an object must be at the top of a stack and is not, that it can remedy this by asserting a cleartop goal. Thus, after ANA builds the production it asserts the goal of clearing off the top of the blue chair.

When it asserts the cleartop goal, it finds that it has no immediately applicable method, so it uses its isa nets to make contact with an appropriate method. It selects its method of clearing the top of a desk in L32. This method generates the goal of moving the top object In the stack. Again, ANA has no method for moving a chair from L21, but it does have a method for moving a box from L25. It selects this method to use. One of the constraints on the "move box" method is that the expected effect of moving the box is that it end up In L33. When ANA uses the method, it sees that it could move the green chair to L22 and so asks the task master if it is necessary that the chair end up In L33. Since there is no reason in this case for the chair to be moved to L33 (ie, since the method is over-specified) the task master indicates that it is not necessary and ANA moves the chair to L22. Having achieved its move

[5] ANA's method for finding objects, unlike its other methods for operating on objects in the paint shop, is quite general. The "find" productions are aware of the "constraint given"/"constraint possible" distinction; if this were not the case, when "find" was invoked, ANA would be unable to determine whether to escan for the actual object desired or the pseudo-object specified by the method being used.

goal, the production that ANA built to build a production that associates the task of moving a chair in L21 with its "move box" method fires. Since the cleartop goal is also now satisfied, the production which builds a production that associates the task of clearing off the top of the chair in L21 with its "cleartop desk" method also fires. Once the blue chair is at the top of the stack, the carry method again comes into play. This time, when the @carry operator is executed, it succeeds, and so the chair ends up in L23. Since the carry goal is satisfied, the production which builds a production that associates the task of carrying the chair with the "cleartop desk" and "carry table" methods fires.

When the chair has been moved to L23, the "paint table" method asserts the goal of carrying the red paint can to L24. Since the red paint can is a new object, ANA has to decide whether it should substitute some other object for it. ANA knows that the purpose of a paint can is to contain paint. It also knows that the purpose of paint is for painting. Since painting is the stipulated action type for the task at hand, ANA assumes that the type of the object that it wants to carry is paint can. After deciding that the stipulated object type is in fact appropriate, it checks the other constraints on the object. Since it knows that the label on a paint can indicates the color of the paint inside the can, it checks whether the description of the expected effects of doing the painting make any mention of color. Since the expected effect of painting the box is that it will become red, ANA concludes that the red paint can is appropriate. Since ANA has no method for carrying a paint can, it selects its "carry table" method again and uses it successfully. When both the object to be painted and the paint can are in the appropriate locations, the "paint table" method executes the ©spray operator, and the newly painted chair ends up in L34. Then the subgoal of carrying the paint can back to its original location is generated, and "carry table" is used successfully.

Now ANA must move the newly painted chair to L35. The "carry table" method is selected. After selecting the method, ANA (as usual) builds productions to watch for unexpected results. One of the productions built In this case watches to make sure that after the chair it carried, it is in location L35. In this particular case, since there are already four chairs in L35, when the chair Is added to the top of the stack, the stack teeters and the chair ends up in location L34. When the production that watches for this unexpected result fires, the task master is asked how to undo the unexpected result. Since the chair fell back to the spot where It was, and since everything else is unchanged, the task master indicates that nothing need be done. Then the task master is asked to indicate what went wrong. All he need say is that the there are too many objects In L35; there have to be fewer than four. ANA then builds a production containing the Information that in order to be able to carry something somewhere, the location that the object Is being carried to must contain fewer than four objects. After building the production, it reconsiders what method to use for the task. This time, when it considers carrying the chair, it realizes that in order to achieve its goal it must first achieve the subgoal of moving one of the objects in L35 somewhere else. After the chair at the top in L35 is moved out of the way, the "carry table" method is again applicable and succeeds in getting the chair to L35 for keeps.

5 2 The Safe Washing Task

When ANA is given the task of washing the thing in LI2, it again selects its method for painting tables. This time, the action type "paint" is mapped into "wash", the description of the expected effect of painting (that the object painted will be red) is mapped into the expected effect of washing (that the object washed will be clean), the object type "table" is mapped into "thing", and the description of the location of the table (that it is in L32) is mapped into the description of the location of the thing (that it is in LI2). When the "paint table" method generates the subgoal of carrying the table (thing) to L23, the "carry table" method discovers that the only thing in LI2 is a safe and using one of its isa nets verifies that a safe is a thing. When it attempts to *carry* the safe to L23, @carry fails because the safe Is not light. An "action failed" element Is asserted) the production watching for such an element fires, generating the goal of recovering from the error. Since ANA knows that weight is immutable, it knows that it cannot fix the problem by modifying the state of the world. So it tries a different operator — ©push. It builds a production that temporarily masks (ie, that is selected in preference to) the production whose action side contains ©carry. The function of the new production is to look to see whether the object to be carried is light. ANA then builds another production whose condition side is identical to the condition side of the production just built except that it contains an additional condition element that will match a percept whose attribute is "weight" and whose value is anything other than "light"; the action side of this second production contains the operator ©push. Then, since ANA sees that the weight of the thing (ie, the safe) Is "heavy" (ie, not "light"), the second production will fire, this time, ANA's attempt to move the safe is successful.

The "paint table" method next asserts the subgoal of carrying the red paint can to L24. After recalling that the purpose of a paint can is to contain paint and that paint is for painting, ANA notices that what It actually wants to do is wash. So it tries to recall an object that has the same relationship to washing as a paint can haa to painting. Since it knows that water is what Is used for washing and that water bottles contain water, It decides that it should substitute something of type water bottle for the paint can. It looks around the shop for a water bottle and when it finds one, assumes that It has alt of the appropriate attributes and substitutes the perceived values of the water bottle for the corresponding values in the description of the paint can.

ANA carries the water bottle to L24, the "paint table" method executes @spray, and the task is done.

6. CONCLUSIONS

It should be evident by now that ANA has a somewhat cavalier attitude toward learning. Whenever it has to learn, it learns (at most) just enough to get by. When it is given *an* unfamiliar task, it tries to do the task by analogy; it has a few rather weak rules that enable it to select a method to use. It maps the description of the goal that the method can achieve into its actual goal; it does not consider at this point whether the mappings are plausible. It then attempts to use the method; it does not construct a plan (since the method serves as Its plan). When the method succeeds in accomplishing a task, ANA builds a production that enables it to subsequently recognize that task and to remember the analogy; it does not create a new method. When an analogy breaks down, ANA patches the method it Is using by building a production that will watch for signs of trouble and take steps to avoid it; ANA does not modify any of the productions comprising the method. Should ANA be doing some of these things that it does not do? Which of them would improve its performance and which would merely slow it down?

Given ANA's performance on the two tasks, one conclusion that seems warranted is that if a system is provided with a set of highly specific methods for performing a few tasks, and if at least one of these methods is almost adequate for each unfamiliar task the system will face, then an assimilation and accommodation strategy like the one ANA employs enables the system to learn to perform unfamiliar tasks without requiring it to know much about learning. If the methods are almost adequate, then the types of problems that the system can encounter are quite limited: (1) Subgoals that are generated by the method may not be appropriate for the unfamiliar task; so the system will have to learn what the analogous subgoals are. (2) The method being used can be under-specified; so the system will have to learn what the additional pre-conditions on the method are. (3) The method being used can be over-specified; so the system will have to learn what constraints to ignore. The mechanisms required to solve all three problems are quite simple.

ANA's strength, then, is that its learning mechanisms are simple, but effective — at least for simple tasks. And since ANA recovers from method inadequacy by creating patches locally as particular problems arise, task complexity, of itself, presents no special difficulties. If ANA is given a complex task that can be decomposed into a set of subtasks for which it has almost adequate methods, ANA's learning mechanisms will enable it to patch those methods appropriately. The fact that ANA is so dependent on a store of almost adequate methods may appear to be a significant limitation. But these methods are highly specific and thus easily acquired. The knowledge embedded in each of ANA's task methods is just that knowledge which would be acquired If ANA were to be led, step by step, through a particular task. Thus with some somewhat laborious training, ANA could acquire a store of methods sufficient to enable it to perform a wide variety of unfamiliar tasks. ANA does, however, have a serious weakness: its knowledge of how to select an appropriate (almost adequate) method is extremely limited. If ANA had a large number of methods from which to select, it would need more knowledge of the interrelationships among actions and among objects and more knowledge of how to determine the dimensions along which to compare tasks. If ANA had such knowledge, and if it had a large store of methods, its learning strategy could be effective In many non-toy domains.

ACKNOWLEDGEMENTS

The development of many of the ideas discussed above owes much to the members of a production system group at Carnegie-Mellon University. The members of this group, in addition to myself, are C. Forgy, J. Lalrd, P. Langley, A. Newell, and M. Rychener. I also want to acknowledge the helpful comments on the first draft of this paper from J. Bentley and D. Kosy.

REFERENCES

[1] Forgy, C. A production system monitor for parallel computers. Technical Report. Department of Computer Science, Carnegie-Mellon University, 1977.

[2] Forgy, C. and J. McDermott. OPS, a domain independent production system language. IJCAI, 5, 1977.

[3] McDermott, J. ANA: an assimilating and accommodating production system. Technical Report. Department of Computer Science, Carnegie-Mellon University, 1978.

[4] McDermott, J. Some strengths of production system architectures. NATO ASI on Structural/Process Theories of Complex Human Behavior. Sijthoff, 1978.

[5] McDermott, J. and C. Forgy. Production system conflict resolution strategies. In D. Waterman and F. Hayes-Roth (eds), Pattern-Directed Inference Systems. Academic Press, 1978.

[6] Moore, J. and A. Newell. How Can Merlin Understand? In L. Gregg (ed), Knowledge and Cognition. Lawrence Erlbaum Associates, 1973.

[7] Newell, A. Knowledge representation aspects of production systems. IJCAI, 5, 1977.

[8] Rychener, M. Control requirements for the design of production system architectures. Symposium on AI and Programming (SIGART/SIGPLAN), 1977.

[9] Sussman, G. J. A Computer Model of Skill Acquisition. American Elsevier, 1975.

AN ANALYSIS OF GENERALIZATION AS A SEARCH PROBLEM

Tom M. Mitchell

Computer Science Department
Rutgers University
New Brunswick, NJ 08903

The problem of concept learning, or forming a general description of a class of objects given a set of examples and non-examples, is viewed here as a search problem. Existing programs that generalize from examples are characterized in terms of the classes of search strategies that they employ. Several classes of search strategies are then analyzed and compared in terms of their relative capabilities and computational complexities.

1 Introduction

"Learning" is a broad term covering a wide range of processes. One process central to many kinds of learning is the process of generalization or learning.; that is, characterizing a class of specific observations by abstracting the important features common to members of that class. *

This problem of generalizing from a set of training instances has been studied by many researchers over the last two decades (e.g., [1], [10], [13], [4], [2], [12], [3], [6]). The results so far have been tantalizing: partially successful programs have been written for problems ranging from learning fragments of spoken English to learning rules of chemical spectroscopy. But comparing ana understanding alternate strategies has been difficult because of differences in data representations, terminology, and problem characteristics.

The goal of this paper is to compare alternate approaches to generalization in terms of a single framework. Toward this end, generalization is cast as a search problem, and alternate methods for generalization are characterized in terms of the search strategies that they employ. This characterization uncovers similarities among approaches, and leads to a comparison of relative capabilities and computational complexity of alternate approaches.

• An extended version of this paper is available as Rutgers Department of Computer Science technical report DCS-TR-78. This work has been supported in part by the National Institutes of Health under grant RR 00612-07 and by the Advanced Research Projects Agency under contract DAHC 15-73-C-0435.

2 The problem

The class of concept learning, or generalization problems considered here may be characterized as follows:

Given:
1. A language in which to describe concepts.
2. A set of positive and negative training instances of some "target concept" (concept to be learned).
3. A matching predicate that matches concept descriptions to instances.

Concept descriptions within the provided language that are consistent with the presented training instances.

Here a concept description is c6nsidered to be consistent with a set of training instances if and only if it matches every positive instance and no. negative instance in the set. This strict requirement of consistency is imposed in order to obtain concrete results in comparing alternate approaches. Although several of the approaches to be considered have been extended to deal with inconsistent training data, an analysis of performance in such cases is beyond the scope of this paper.

Generalization as defined above can conveniently be viewed as a search problem, in which the language of allowed concept descriptions defines

the domain of concepts that the program might learn, or the space of possible "solutions" to the concept learning problem. The program must examine this solution space, subject to constraints imposed by the training instances, to determine valid descriptions of the target concept.

2.1 The. Partial Ordering

A key characteristic of the above generalization problem is that there is a useful structure to the solution space of possible concept descriptions. This structure is a partial ordering based upon the "more-specific-than" relation defined as follows*:

> More-specific-than relation: Given two concept descriptions, CD1 and CD2, we say that CD1 is "more-specific-than" CD2 if and only if CD1 matches a proper subset of the instances that CD2 matches.

As an example, consider the following three concept descriptions which characterize classes of block structures:

> CD1: "A structure including a green slab supported by two posts."
>
> CD2: "A structure including a slab."
>
> CD3: "A structure including a red slab."

Here, CD1 is more-specific-than CD2: any Instance which matches CD1 must also match CD2. Similarly, CD3 is more-specific-than CD2. In contrast, CD3 and CD1 are not comparable descriptions in the partial ordering - neither is more specific than the other.

This more-specific-than relation imposes a general-to-specific partial ordering over the concept descriptions in the solution space of any generalization problem as defined above. It is important because it allows organizing the examination of the solution space in an efficient, complete manner. At the same time, it provides this paper with a basis for comparing alternate generalization strategies in a way that is independent of the particular concept description languages used.

• This ordering has been described previously for individual concept description languages [10], [5], [4], [12].

3 Model-driven and Data-driven strategies.

If concept learning is viewed as a search problem, then concept learning methods can be characterized in terms of the search strategies that they employ. Two broad classes of search strategies that have been employed for concept learning problems may be called model-driven and data-driven search strategies. These terms have been used to refer to procedures for examining spaces of possible hypotheses for a variety of problems outside concept learning [11], [9].

In a model-driven search (e.g., [2], [3]), hypotheses (concept descriptions) in the search are generated according to a predetermined model based upon prior knowledge of the problem. The generated hypotheses are then tested against the set of available data (training instances), to prune the search. In contrast, data-driven search uses the data prospectively to generate new hypotheses. Discrepancies between the current hypothesis and available data drive the generation of new hypotheses. In practice, model-driven strategies tend to consider all available training Instances at each step to test the methodically generated hypotheses, whereas existing data-driven strategies for concept learning consider the training instances one at a time to generate new hypotheses.

The model-driven search performed by the RULEGEN portion of the Meta-DENDRAL program [2] evaluates the model-generated hypotheses in terms of their performance over the entire set of training instances. This strategy can accommodate quite severe errors in the training data because it judges the generated hypotheses by their performance over a large set of instances rather than making decisions based upon single training instances. This observed robustness of model-driven search with respect to data errors is consistent with the observations of other researchers considering hypothesis formation tasks outside concept learning [9], [11].

In contrast, existing data-driven search procedures (such as [13], [10], [5], [4], [12], [6]) base the generation of new hypotheses on differences between the current hypotheses and a single new training instance.

Because of this incremental use of the data, such strategies tend to be more sensitive to errors in individual training instances, while being better suited to adapting current results when new data becomes available.

An interesting combination of the advantages of model-driven and data-driven search procedures is found in the Meta-DENDRAL program. The RULEGEN portion of the program, mentioned above, conducts a coarse, model-driven search to form approximate rules of mass spectroscopy from highly unreliable training instances. These approximate rules are then refined by a data-driven strategy which conducts a more detailed search, and takes advantage of additional data [6]. Thus, the advantages of model-driven search for dealing with inconsistent data are blended with the advantages of data-driven search for incremental use of the data.

4 Three Data-Driven strategies

The majority of concept learning programs reported in the literature employ data-driven search strategies. Although no two of these programs employ exactly the same strategy, it is informative to group them into classes whose members employ similar strategies and therefore possess similar performance characteristics. The aim of this section is not to compare alternate concept learning programs, but rather alternate classes of data-driven strategies that existing programs implement in various ways, for various concept description languages. We describe three such classes of search strategies here. The capabilities and efficiency of the classes are then compared.

4.1 Depth-flrat Search

One common data-driven strategy for examining the solution space of possible concept descriptions is depth-first search. Programs that can be characterized in this way include [13] and parts of the RULEMOD portion of the Meta-DENDRAL program [2]. In this strategy, a single concept description is chosen as the Current best *hypothesis* for describing the identity of the target concept. This current hypothesis is then tested against each newly presented training instance, and altered as needed so that the resulting concept description is consistent with the new instance. Each such alteration yields a a new current hypothesis, and corresponds to one step in a depth-first search through the solution space.

There are two awkward characteristics of this depth-first search strategy. First, each alteration to the current hypothesis must be tested to determine whether it is consistent with past training instances. Some systems (e.g., [13]) sacrifice assured consistency with past instances by not reexamining them when the current hypothesis is altered. Others (e.g., [2]) test past instances, and therefore require progressively larger computation time for each successive training instance. Second, once the program has determined a set of acceptable alterations to the current concept description, it must choose one of these as the new current hypothesis, and be prepared to backtrack if an incorrect choice has been made.

4.2 Specific-to-General Breadth-first Search

In contrast to depth-first search programs, programs which employ a breadth-first strategy maintain a set of several alternate hypotheses. Systems which fall into this class include those reported in [10], [4], and [12]. Each of these programs takes advantage of the general-to-specific partial ordering of concept descriptions to efficiently organize the breadth-first search. Starting with the most specific concept descriptions consistent with the first positive instance, the search is organized to follow the branches of the partial ordering so that progressively more general concept descriptions are considered each time the current set must be modified. The set of alternate plausible hypotheses computed by this specific-to-general breadth-first search is the set (which we shall call S) of maximally specific concept descriptions consistent with the observed training instances; that is

> S = (s ! s is a concept description that is consistent with the observed instances, and there is no concept description which is both more specific than s, and consistent with the observed instances}

One advantage of this strategy over depth-first search stems from the fact that the set S represents a threshold in the solution space. Concept descriptions that are more specific than this threshold are not consistent with all the observed positive instances, whereas those more general than this threshold are. Thus, when a concept description in S must be revised, it must be made more general, and the revision need not be tested for consistency with past positive instances. It must still, however, be tested against previous negative instances.

4.3 Version Space Strategy

The version space strategy for examining the solution space entails representing and revising the set of all hypotheses consistent with the observed training instances. This set of concept descriptions forms a convex set with respect to the partial ordering, and is referred to as the version space of the target concept (since it contains all plausible versions of the emerging concept). This strategy begins by representing the version space of concept descriptions consistent with the first positive training instance. Concept descriptions found inconsistent with subsequent instances are then eliminated from consideration. Programs that implement this strategy for two different concept description languages are described in [6].

The version space approach is feasible because the general-to-specific ordering of concept descriptions allows a compact representation for version spaces. In particular, a version space can be represented* by two sets of concept descriptions: the set S as defined above, and a complementary set G, of maximally general Concept descriptions consistent with the observed instances:

> G = {g I g is consistent with the observed instances, and there is no concept description which is both more general than g, and consistent with the instances}

The set G can be computed by conducting a breadth-first search analogous to that for computing S, but proceeding instead from general to specific concept descriptions. Since the sets S and G together delimit the version space, the version space strategy can be viewed as an extension of the above breadth-first search strategy into a bi-directional search.

The power of the version space strategy lies in the fact that the set G summarizes the information from the negative training instances that bounds how general the "correct" concept description might be, while the set S summarizes the information from positive instances that limits how specific it might be. Thus, testing whether a given concept description is more specific than some element of G and more general than some element in S, is logically equivalent to testing whether it is consistent with all observed training instances.

Since the sets S and G summarize the information in the training instances, the version space strategy does not require reexamining or saving previously processed training instances. This method of delimiting the version space leads to increased capabilities as well as efficiency.

* The version space is "represented" in the sense that it is possible to generate and recognize concept descriptions in the version space by examining the representation.

4.4 Capabilities

In comparing alternate strategies for concept learning, the central issue is relative capabilities. What are the limits of each approach in terms of questions to which it can provide answers? The major differences in capabilities among the above three data-driven strategies derive from the number of plausible concept descriptions carried along at each step, and from the use of the partial ordering in guiding the search. Below we consider two desirable capabilities: (1) the ability to determine when a concept has been fully learned, and to use incompletely learned concepts in a reasonable manner, and (2) the ability of the learning program to direct the presentation of training instances in an intelligent manner.

4.4.1 Using Imprecisely Learned Concepts

Recognizing the degree to which a concept has been learned, and using incompletely learned concepts in a reasonable manner are essential capabilities for practical learning systems. It is rare that the available training instances precisely describe the target concept.

The version space strategy provides a straightforward method for detecting the point at which a concept is precisely described by a set of training instances, with respect to the given concept description language. The concept is completely learned (assuming reliable training instances, and a sufficient concept description language) if and only if the version space contains exactly one concept description; that is, if S and G contain one and the same concept description. In contrast, it is difficult to recognize this condition when maintaining only a single current concept description, as with the depth-first search strategy.

In the case where a partially learned concept must be used, the sets S and G again provide

useful information. Consider the case in which the version space derived from the provided training instances contains a range of plausible concept descriptions, no additional training instances are available, and a new instance is presented to be classified.

In such a case, although the exact identity of the target concept is not fully determined by the training instances, it is. known that the correct description of the target concept lies somewhere within the version space delimited by S and G. Therefore, if the new instance matches every concept description in the version space (equivalently, if it matches each element of S), then it can be classified as a positive instance with the same certainty as if the concept had been uniquely described by the training instances. Alternately, if the instance matches no concept description in the version space (i.e., it matches no element of G), then it is certain that the instance does not match any description of the target concept that would be chosen by examining additional training instances. Thus, for such instances it is possible to use the partially learned concept to obtain classifications that are as reliable as if the learned concept had been completely described by the training instances.

In contrast, instances that match some, but not all concept descriptions in the version space cannot be reliably classified until further training instances are available. Of course, by considering outside knowledge and examining the proportion of concept descriptions in the version space which match the instance, one might still estimate the classification of such instances.

Because the specific-to-general breadth-first strategy also computes the set S, this strategy allows reliably classifying the same positive instances as the version space strategy. Since it does not compute the set G, however, it cannot distinguish between instances which the version space strategy would reliably classify as negative Instances, and those which cannot be reliably classified.

4.4.2 selecting Future Training Instances

A further capability afforded by computing the sets S and G is the selection of informative new training instances. Consider the following problem: After processing some sequence of training instances, a program is provided a set of new instances, without their classifications as positive or negative instances, and is allowed to request the correct classification of any one of them. *The* instance whose classification will be most informative (on the average) is the one that comes closest to matching one half of the concept descriptions in the version space. Regardless of its classification, such an instance would allow rejecting one half of the currently plausible concept descriptions. Thus, by testing each instance to determine what proportion of the concept descriptions in the version space it matches, the most informative training instance can be selected (a more complete discussion of this issue may be found in [6] and [8]).

The breadth-first strategy also provides some information for selecting new training instances. The strategy of selecting instances which match half the concept descriptions in the computed set S is reasonable, although less complete than the strategy which takes into account the entire version space.

4.5 Complexity and. Efficiency

The overall efficiency of each approach is determined by a complex of factors, including the order of presentation of training instances, the chosen concept description language and the branching of the associated partial ordering, the cost of matching concept descriptions to training instances, and the amount of space needed to store concept descriptions and observed instances.

A complete analysis is beyond the scope of this paper, but it is possible to characterize the rates of growth of time and space requirements as a function of the number of training instances, under reasonable assumptions. Under the assumption that positive and negative instances are distributed uniformly throughout the sequence of training instances, bounds on the time and space complexity of the data-driven strategies described earlier are summarized in table 1. Here p indicates the number of positive training instances, n indicates the number of negative training instances, :S: indicates the largest size obtained by the set S, and :G: represents the largest size obtained by the set G.

Strategy	Processing Time	Storage Space
Depth-first search	O(pn)	O(p+n)
Specific-to-general, Breadth-first	O(|S|pn)	O(n+|S|)
Version space strategy	O(|S||G|(p+n))	O(|S|+|G|)

Table 1: Bounds on processing time and maximum storage costs.

The differences in Table 1 reflect differences in the way in which revised hypotheses are tested for consistency with past instances. In the depth-first strategy this must be done by going back to the Instances themselves. In the specific-to-general breadth first strategy, only past negative instances need be reexamined, since S summarizes the constraints imposed by past positive Instances. The tradeoff is one of calculating and using S and G_f versus returning to the original data. A justification of these results is given in [7].

In interpreting the above results it is important to know how the sizes of the sets S and G vary over the training sequence. For the two concept description languages for which the version space strategy has been implemented, these sets were observed to first grow in size, then level off, and finally decrease in size as the version space collapsed toward the correct description of the target concept. An interesting problem is to characterize the behavior of these sets for various forms of concept description languages, and study ways in which careful presentation of training Instances can reduce their size.

5 Summary and. Conclusions

The problem of concept learning, or generalization, can be viewed as a search problem involving a large solution space of possible concept descriptions. Concept learning then involves examining this space under constraints imposed by the training instances (data constraints), as well as prior knowledge and expectations (model-based constraints). In this light, it is informative to characterize alternate approaches to concept learning In terms of the strategy that each employs in examining this solution space.

Two classes of strategies, data-driven and model-driven, differ in terms of which of these two kinds of constraints is used to drive the search. General characteristics of these two classes that appear in concept learning problems correlate with characteristics noted by others in hypothesis formation problems in signal analysis and molecular genetics.

A general-to-specific partial ordering gives structure to the search space for the class of generalization problems considered here. Several data-directed strategies for concept learning are described and compared in terms of the way in which they organize the search relative to this partial ordering. This examination leads to a comparison of relative capabilities and computational complexity.

6 Acknowledgments

Thanks to the following people who commented carefully on various drafts of these ideas: Bruce Buchanan, John Burge, Ed Feigenbaum, Cordell Green, Rick Hayes-Roth, Abe Lockman, and Richard Pantell.

References

1. J.S. Bruner, et al., A Study of Thinking. Wiley, 1956.

2. B.G. Buchanan and T.M. Mitchell, Model-directed learning of production rules, In Pattern-Directed Inference Systems (Waterman and Hayes-Roth, Eds.), Academic Press, 1978.

3. T.G. Dietterich and R.S. Michalski, Learning and generalization of characteristic descriptions. IJCAI6, Tokyo, 1979.

4. F. Hayes-Roth, Schematic classification problems and their solution. Pattern Recognition, 6, pp. 105-113 (1974).

5. R.S. Michalski, AQVAL/1 - Computer implementation of a variable valued logic system VL1 and examples of its application to pattern recognition. IJCPR1, 1973, pp. 3-17.

6. T.M. Mitchell, Version Spaces: An approach to concept learning. Ph.D. thesis, Stanford University, December, 1978.

7. T.M. Mitchell, An Analysis of Generalization as a Search Problem. Rutgers Computer Science Technical Report DCS-TR-78. January, 1979.

8. T.M. Mitchell, Generalization Using Version Spaces. In preparation.

9. H.P. Nii and E.A. Feigenbaum, Rule-based understanding of signals. In Pattern-Directed Inference Systems (Waterman and Hayes-Roth, Eds.), Academic Press, 1978.

10. G.D. Plotkin, A note on inductive generalization, Machine Intelligence 5 (Meltzer and Michie, Eds.), Edinburgh University Press, 1970, pp. 153-163.

11. M.J. Stefik, Inferring DNA structures from segmentation data. Artificial Intelligence, 11, August, 1978.

12. S.A. Vere, Inductive learning of relational productions. Pattern-Directed Inference Systems (Waterman and Hayes-Roth, Eds.), Academic Press, 1978.

13. P.H. Winston, (Ed.), The Psychology of Computer Vision, McGraw-Hill, 1975.

A Case Study of EXPERT Formalism
——An approach to a Design of Medical Consultation System through EXPERT Formalism——

Fumio MIZOGUCHI*, Kunihisa MARUYAMA*, Takashi YAMADA*, Katsuaki KITAZAWA**,
Masao SAITO***, Casimir A. KULIKOWSKI****

*Department of Industrial Administration
Tokyo University of Science
Noda, Chiba 278, Japan

**, ***Department of Ophthalmology, Institute for Medical Electronics, The Tokyo University
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

****Department of Computer Science
Rutgers University
New Brunswick, New Jersey 08903

Abstract

The present paper is based upon case studies of EXPERT formalism. The studies aim to accomplish two goals; 1) Acquisition of knowledge from specific domain expert through a design of a small scale medical consultation system. 2) Evaluation of the system's performance through various case data. The prototype models on Glaucoma medical consultation system is running at DEC-20 under TOPS-20 operating system. There is a discussion which is based upon these case studies through EXPERT formalism.

1. Introduction

Recently, a programming system for medical consultation system called EXPERT (Kulikowski. C.A.,1979) is developed along the current context of knowledge engineering or applied artificial Intelligence. That is, the features of knowledge representation and reasoning strategies in EXPERT have converged from the different specific programs developed in the past (Shortliffe,1976,Pople,1975).

Thus, the EXPERT system facilitates a user to developed a medical consultation system without further consideration of AI techniques. But rather, the user concentrates his efforts on how to describe the medical knowledge that is used for the consultation system.

In this paper, we focus on the case studies that aim to accomplish two goals through the experience of using EXPERT system;

I) Acquisition of knowledge from specific domain expert through a design of a small scale medical consultation system.

II) Evaluation of the system's performance through various case data.

2. Knowledge Representation

In order to design a medical consultation system, the knowledge is represented in the form of programs or data structure. For this purpose, the procedure for acquiring the knowledge is made by an analysis of an expert's explanation on a paticular problem domain.

The expertise is described in terms of varbal expression. This kind of explanation is not well structured, but rather loosely expressed. Therefore, the expertise must be reformulated according to a format of a programming language. In case of EXPERT, the knowledge is structured in a taxonomic classification scheme. If we select Glaucoma as medical domain, the taxonomy of diseases in Glaucoma is represented in the following lists which are shown in Fig.1.

```
**  HYPOTHESES
 *  TAXONOMY
RY  RYOKUNAISYOU
    (Glaucoma)
HGR  .HEISOKU-GUKAKU RYOKUNAISYOU
     (Angle Closure Glaucoma)
ZHR  ..ZOKUHATSU HEISOKU-GUKAKU RYOKUNAISYOU
     (Primary Angle Closure Glaucoma)
```

Fig.I Taxonomic Representation

This knowledge source is derived from Dr.Kitazawa audio instructional materils on Glaucoma. From the tape, the descriptions of glaucoma consists of three topics; 1) disease taxonomy 2) treatments 3) disease syndromes

and diagnosis. Therefore, the lists in Fig.I are correspond to the first topics of disease taxonomy.
The second which deals with medical treatment plan is transformed into the following lists in Fig.2. The expert's explanation is focussed on global treatment plans which are further divided into a small chunk of medical care, such as sequence of miotic therapies. This parts is made by the use of taxonomic scheme in the Fig.2.

```
    *  Treatment
     / RYOKUNAISHOU NO CHIRYOU PLAN
        (Glaucoma medical treatments Plan)
   KK  .SYUJUTSU
        (Surgery)
   SS  .SYUJUTSU GA KONPON DE ARU.
   XX  .YAKUBUTSU CHIRYOU GA 0M0 TO NARU.
        (Miotic therapies)
        ..Pilocarpine 1%
        ..Pilocarpine 2Z        further taxonomic
        ..Pilocarpine  3%       treatment plans
        ..Epinephrine 1%
```

Fig.2 Treatment descriptions

The third which is the most important topic is composed of two parts; a) syndromes b) diagnosis. In EXPERT formalism, syndromes are considerred as findings. Diagnosis is correspond to hypothesis which represents a reasoning strategy. Therefore, the third topic is to deal with a set of possible findings and a separate set of hypotheses.
Finding are facts about the patient with the particular phyosiological state. These facts are acquired from the expertise. The lists are shown in Fig.3 indicating the syndromes of Glaucoma.

```
   **  FINDINGS
    *  BEGIN
    *  NUMERICAL
       PATNO  PATIENT  ID  NUMBER:
    *  NUMERICAL/MIN-1/MAX-100
    .
    *  CHECKLIST
       SY0UJ0U:
         (Syndromes)
       ZU     ZUTU, ZUJUKAN
              (Headaches in dim light)
       GAN    GANTU
              (Ocular/head pain)
       KOS    KOUSI
              (rainbow vision)
       .
```

Fig.3 Checklists for Findings

The findings are obtained from the checklist which can be reported in the form of yes, no or associated with an observation. Hypotheses are inference rules which combine the findings into the associated physiological or disease state. In the EXPERT formalism, there are three types of rules for describing logical relationships among findings and hypotheses: 1) FF—(finding to finding rules) 2) FH—(finding to hypothesis rule) 3) HH—(hypothesis to hypothesis rules). Thus, the finding must be interpreted by a set of inference rules.
These examples are shown in Fig.4.

```
    **  RUIES
     *  FF Rules
        F(SIK,F)->  F(K0D,F)
        means;
        IF (complaints of decreased visual
            acuity)
        is false, then Shallow an chamber depth
        is false.
     *  FH Rules
        F(ZU,T)--»  H(RY,0.5)
        means;
        IF Ocular/Head pain is reported, then
        consider the hypothesis of Glaucoma
        with a confidence of 0.05.
        F(K0S,T)->  H(RY,0.08)
        means;
        IF Rainbow vision is reported, then
        consider the hypothesis of Glaucoma
        with a confidence of 0.08.
        .
```

Fig.4 Inference rules

The above examples show the rule of findings that are used for FF and FH rules. For the case of HH rule, the following lists in Fig.5 are one of the examples. Here, left hand side of HH rules contain assertions about findings.
In the example,
(F(IOP,20:*)&F(KAI,T)&F(KZY,F)) is the left hand side.

```
     *  HH Rules
     *  IF
        (F(IOP,@):*)AF(KAI,T)&F(KZY,F))
     *  THEN
        (1:F(BJH,T),F(BJZ,T),F(B0T,T)
           F(BJB,T))--> H(GKR,.99))
```

IF-part means;
F(I0P,20:*): Intraocular pressure is more than 20
F(KAI,T): Open angle is observed
F(KZT,T): Iris abnormalities is observed
THEN-part means;
F(BJH,T) :Di8c margin hemorrhage
F(BJZ,T) :Central Retinal Occlusion
F(B0T,T) :Central Retinal Artery Occlusion
The right hand side of THEN-parts means;
H(GKR,.99) ;
then consider hypothesis of primary open angle Glaucoma with a confidence of 0.99.

Fig.5 HH rules

Therefore, once IF-part conditions are satisfied, then, the system will determine an

expected FH rule which is assciated with hypothesis. This behavior is similar to human expert's diagnostic process. That is, the expert will diagnose a patient without a rigid order of checklist. But rather, doctor will proceed in a wholitic way. This process will be also true in the consultation system. Global inference deals with the entire view of the diagnostic inferences. In this case, the use of HH rule will facilitate the inference system with dual characteristics of both global and local views. One is FH rule which combines directly finding to hypothesis, and another is HH rule which finds out a proper hypothesis for a associated finding.

3. System Implementation

In order to improve the level of diagnosis and performance of the consultation system, the rule must be evaluated by the medical expert. This procedure is acomplished by gradual process through hand on experience of small scale working system. The system is implemented under the TOPS-20 operating system of DEC-20. The EXPERT system is transferred from Rutgers AIM group. The present model is stored by the use of interactive editing program of DEC-20. The preliminary results of model are shown in Fig.6 which indicates protocals of the final consultation.

```
                    —Summary—
Case   1: Visit   1        Date: 5/30/79

SYOU JOU:                    Symptoms
     ZUTU,ZUJUKAN            Ocuear/Head pain
     KOUSI                   Rainbow vision
     SIRYOKU-SYOUGAI         Decreased visual
                             acuity
KUSURI NO SIYOU :            Drug intorolerance
     NOTHING
BYOREKI:
   TOUNIYOU-BYO              Patient history of systemic
                             Disease Glycosuria

GUKAKU-IJOU:                 Angles abnormality
     KAIHOU-GUKAKU           Open
     GUKAKU-YUTYAKU

                  — Conclusions —
DIAGNOSES:
GKR  0.99   IENPATU KAIHOU-GUKAKU RYOKUNA:
            (primary open angle Glaucoma)

TREATMENTS:
KK   0.41   SYUJUTSU                    (surgery)
```

Fig.6 Summary and conclusions

For the above conclusion, HYPO command guides us a detail process of the maximum weighted hypothesis which is shown in Fig.7. In this case, HH rule supports the consultation of "Primary Open Angle Glaucoma". Therefore, the medical expert can ask the reasoning process of the consultations through the use of EXPERT commands. The rationale of EXPERT is referred to other paper (Weiss & Kulikowski,1979).

```
:HYP0

GKR  GENPATU KAIHOU-GUKAKU RYOKUNAISYOU
Direct confidence weight:   0.990 set by
                                  HH-Rule Table
1:
IF. . . . . .
CHOOSE 1:
Y ... INTRAOCULAR PRESSURE (MMHG) (IOP,20:1000)
THEN. . . . . .
CHOOSE 1:
Y ... SISINKEI-SENISOKU-SYOUGAI:
        BJERRUM-RYOKUIKI NO HIKAKUANTEN (BJH,
        TRUE)
            (Abnormality visual nerve system)

Y --> GENPATU KAIHOU-GUKAKU RYOKUNAISYOU
                        (GKR, 0.99)
                (Primary Open Angle Glaucoma)
Final weight:                          0.993
```

Fig.7 Reasoning process

4. Summary and conclusions

We have developed two types of medical consultation systems; one for small scale system that the medical expert can identify the whole structure so as to modify and check the knowledge, and another for large scale system that accepts various kinds of data from real clinical cases. The model that we focussed on this study is the former model, but simultanously, we are constructing the CASNET-type large scale consultation system, which is further put forward through the experiences of EXPERT case studies.

Reference

1) Pople,H.E. , Myers,j. , Miller,R. , "DIALOG: A Model of Diagnostic Logic for Internal Medicine" , Proc. of 4th IJCAI , pp.848-855 (1975)
2) Shortliffe,E.H. , "Computer Based Medical Consultations: MYCIN", Elsevier North-Holland Inc.(1976)
3) Weiss,S.M. , Kulikowski,C.A. , "EXPERT: A System for Devloping Consultation Models", CBM-TR-97 ,(Submitted to IJCAI-6), 1979.

HIERARCHICAL PRODUCTION SYSTEM

Riichiro Mizoguchi and Osamu Kakusho
The Institute of Scientific and Industrial Research
Osaka University
Suita, Osaka, JAPAN 565

Abstract

Production systems have been used for representing domain-specific knowledge in a number of successful AI projects. In this short paper, we present a general concept of hierarchical production system(HPS). The HPS is an extended version of the current production system, in which two hierarchical structures of rules are employed. In one hierarchical structure, heuristic rules for conflict resolution are embedded and in another one, domain-specific rules are represented. The introduction of the hierarchical rule structures makes the HPS to be a useful framework for constructing knowledge-based systems.

1. INTRODUCTION

In the field of AI, production systems(PS) have been used for representing domain-specific knowledge in a number of successful projects such as DENDRAL[5] and MYCIN[16]. A PS[1][2][6][10][11][14][15][19][20] consists of three components, data base(working memory or short term memory), a set of condition-action rules, and interpreter. In the simplest case the Interpreter operates by scanning the left hand side of each rule until one is found which is successfully matched against the data base. Then, the corresponding right hand side(action) is invoked and the data base is usually changed. In a pure PS all Interaction between rules is forced through a very narrow channel, i.e., the data base. Such restriction on interaction gives PS's "modularity", which is one of the most remarkable advantages of PS's. Thanks to the modularity PS's are superior to procedural approaches in flexibility and adaptability.

One draw back of PS's Is lack of programmability. Therefore, several PS's rather than pure ones adopt tags, markers, or conflict resolution methods to control the sequence of rule invocation. Consequently, however, they make it hard to understand what is intended.

In this short paper, we present a general concept of hierarchical production system(HPS). The HPS is an extended version of the current PS. It consists of several PS's arranged in a two-directional hierarchy to augment the ability of the system while retaining its modularity and flexibility.

2. HIERARCHICAL PRODUCTION SYSTEM

As pointed out by Davis[I], PS's have appropriate domains in which they show good performance. By using PS's for knowledge representation in designing problem solving systems, one must decompose the problem domain into almost independent states and embed the domain-specific knowledge into a system regardless of the manner in which it is to be used. So, the competence of the system depends largely on the results of the decomposition.

It is well known that PS's provide a useful framework for embedding domain-specific knowledge into a system with high intelligence. However, a pure PS is probably inefficient because of its excessive modularity. Therefore, certain modifications are made in several dimensions!1][3][7][11][15] in order to augment the competence of the system. In this short paper, we present an additional dimension in which PS's are to be augmented, that is, we propose a hierarchical production system consisting of the current PS's arranged hierarchically.

Decomposition of the problem domains is an important task in building an efficient problem solving system. In speech understanding system, e.g., Hearsay-II [4][8][12], the problem domain is divided into 6 levels such as phrase, word-

sequence, word, syllable, segment, and parameter levels. Kanade[9] has also introduced a multi-level representation of image understanding domain(object, subimage, region, patch, and pixel levels). Nine-level of pattern description is introduced into learning system by Soloway and Riseman[18]. In their systems learning is considered as knowledge-directed interpretation in order to deal with real world domain where a significant amount of knowledge is required. These multi-level representations of the problem domains are natural to the human intuition, and they help the system designer to consider the problem topdown.

In the GPS paradigm, abstraction spaces are employed by Sacerdoti[17]. In his ABSTRIPS system plan making is done as follows: Suppose that the system wants to solve a certain problem. It usually has a collection of actions whose preconditions and effects are completely known. Initially, it makes a rough plan ignoring detailed situations. In this case, the first plan is a sequence of the actions in the highest space. Although it may be inapplicable to the problem domain directly, the plan does not miss the major causal chain. For example, it may contain the action "GO TO A FROM B" regardless of the transportation media or of their subactions required to be performed beforehand. Then, it can refine the plan using the actions on the next lower level. This process is continued until a realizable sequence of actions are obtained. This "Step-wise refinement" of a plan Is useful especially when making a large scale plan.

The above observation shows that the hierarchical(multi-level) representation of problem domain and knowledge* plays a crucial role in dealing with complex problems.

Among the issues of PS design, conflict resolution is a major one. Davis[2][3] has discussed the conflict resolution problem by introducing meta-rules. Meta-rules are heuristic ones for guiding the selection of the next rule to be invoked from the conflict set and can be considered as knowledge concerning how to use the lower level knowledge. Furthermore, we can have meta-meta-rules. This hierarchy of rules enables us to embed the strategies Into the system while retaining the uniformity of representation. Goldstein and Grimson[7] and Hayes-Roth and Lesser[8] have also considered a similar problem. Although their representations are different, the underlying ideas are same, that is, they want to make their systems efficient by firing appropriate rules in every recognition-act cycle based on attached heuristic knowledge.

Taking the above two kinds of hierarchical knowledge structures into account, we obtain a general concept of hierarchical production system having the block diagram in Fig. 1.

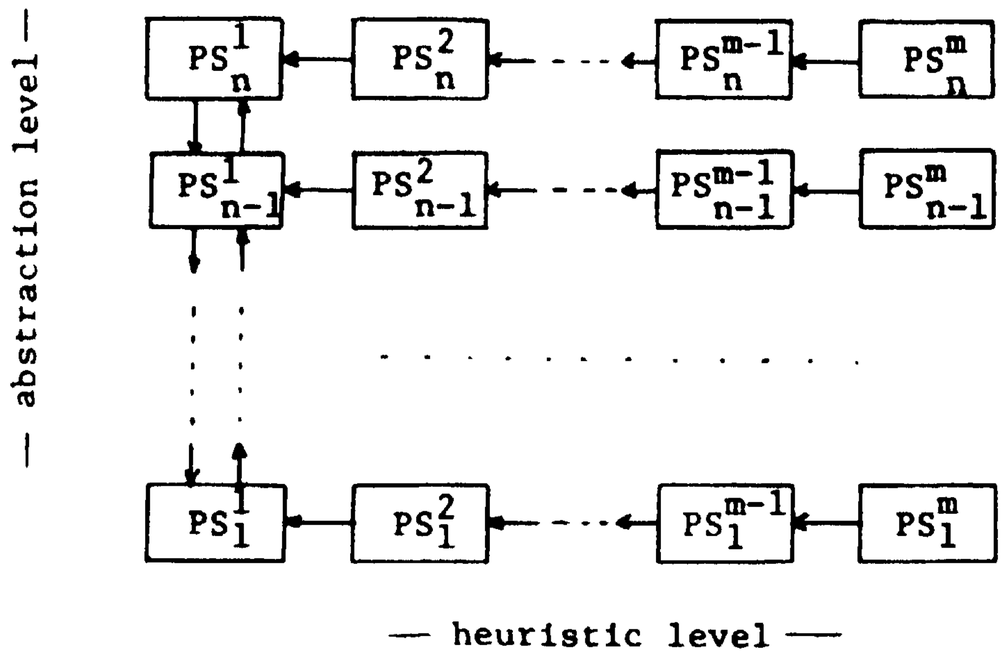


Fig. 1 Hierarchical production system, HPS_n^m. (HPS_1^1 corresponds to the current PS)

The HPS has the following characteristics:

1. The vertical dimension corresponds to the hierarchy of the domain-specific knowledge for step-wise construction of plans or something like that.
2. The horizontal dimension corresponds to the hierarchy of the meta-rules as strategies for conflict resolution.
3. Every component has a usual PS architecture.
4. Interaction between components is restricted to between neighboring ones.
5. Type of the interaction is only the access to the data base.

The HPS is designed to extend the current PS in a natural direction by introducing the hierarchical organization of rules, in which no ad-hoc techniques for conflict resolution are employed. In our system user-defined heuristics can be embedded in the form of rules as well as other knowledge. Furthermore, the hierarchy of the domain-specific knowledge helps the system to invoke appropriate rules, since a PS on a certain level can refer to the plan generated by the PS on the next higher level when determining which rule to fire. Thus, invocation of rules on every level is guided by its next higher level plans.

* The need of hierarchical representation of knowledge is recognized in knowledge representation paradigm[13].

3. CONCLUSIONS

A hierarchical production system has been proposed and its outline has also been described. The HPS enables us to represent the hierarchical domain-specific knowledge and several levels of heuristics.

The HPS has highly parallel computational structure in principle. In the ideal case where decomposition of domain-specific knowledge into a number of independent knowledge is made successfully, every rule and hence every component of the HPS can be activated in parallel. In practice, however, this is not the case. Therefore, we need a monitor or supervisor which arranges the flow of control to work the HPS.

REFERENCES

[1] Davis, R. and J. King: "An overview of production systems", Machine Intelligence, 8, pp. 300-332 (1977).

[2] Davis, R.: "Generalized procedure calling and content-directed invocation", Proc. of the Symposium on Artificial Intelligence and Programming Languages, pp. 45-54 (1977).

[3] Davis, R. and B. G. Buchanan: "Meta level knowledge: Overview and applications", Proc. of the 5th IJCAI, pp. 920-927 (1977).

[4] Erman, L. D. and V. R. Lesser: "A multi-level organization for problem solving using many, diverse, cooperating sources of knowledge", Proc. of the 4th IJCAI, pp. 483-490 (1975).

[5] Feigenbaum, E. A., B. G. Buchanan, and J. Lederberg: "On generality and problem solving———A case study involving the DENDRAL program", Machine Intelligence, 6, pp. 165-190 (1971).

[6] Forgy, C. and J. McDermott: "The OPS reference manual", Carnegy-Mellon Univ. Department of Computer Science (1978).

[7] Goldstein, I. P. and E. Grimson: "Annotated production systems — A model for skill aquisition", Proc. of the 5th IJCAI, pp. 311-320 (1977).

[81 Hayes-Roth, F. and V. R. Lesser: "Focus of attention in the Hearsay-II", Proc. of the 5th IJCAI, pp. 27-35 (1977).

[9] Kanade, T.: "Model representations and control structures in image understanding", Proc. of the 5th IJCAI, pp. 1074-1082 (1977).

[10] Lenat, D. B.: "Automated theory formation in mathematics", Proc. of the 5th IJCAI, pp. 833-842 (1977).

[11] Lenat, D. B. and J. McDermott: "Less than general production system architecture", Proc. of the 5th IJCAI, pp. 928-932 (1977).

[12] Lesser, V. R. and L. D. Erman: "A retrospective view of the Hearsay-11 architecture", Proc. of the 5th IJCAI, pp. 790-800 (1977).

[13] Minsky, M.: "A framework for representing knowledge", The Psychology of Computer Vision, McGraw-Hill, pp. 211-277 (1975).

[14] Newell, A. and H. A. Simon: "Human problem solving", Prentice-Hall (1972).

[15] Rychener, M. D.: "Control requirements for the design of production system architectures", Proc. of the Symposium on Artificial Intelligence and Programming Languages, pp. 37-44 (1977).

[16] Shortliffe, E. H.: "MYCIN: A rule-based computer program for advising physicians regarding antimicrobial therapy selection", Stanford Univ. Computer Science Report, CS-74-465 (1974).

[17] Sacerdoti, E. D.: "Planning in a hierarchy of abstraction spaces", Artificial Intelligence, 5, pp. 115-135 (1974).

[18] Soloway, E. A. and E. M. Riseman: "Levels of pattern description in learning", Proc. of the 5th IJCAI, pp. 801-811 (1977).

[19] Waterman, D. A.: "Generalization learning techniques for automating the learning of heuristics", Artificial Intelligence, 1, pp. 121-170 (1970).

[20] Waterman, D. A.: "Adaptive production systems", Proc. of the 3rd IJCAI, pp. 296-303 (1973).

FUZZY CONDITIONAL INFERENCES AND FUZZY INFERENCES WITH FUZZY QUANTIFIERS

Masaharu Mizumoto*, Satoru Fukami**, and Kokichi Tanaka***

* Department of Management Engineering
Osaka Electro-Communication University
Neyagawa, Osaka 572, Japan

** Yokosuka Electrical Communication Laboratory
Nippon Telegraph and Telephone Public Corporation
Yokosuka, Kanagawa 238-03, Japan

*** Department of Information and Computer Sciences
Osaka University
Toyonaka, Osaka 560, Japan

L.A. Zadeh and E.H. riamdani proposed methods for the fuzzy reasoning in which the antecedent involves a fuzzy conditional inference "If x is A then y is B" with A and B being fuzzy concepts.

This paper points out that the consequences inferred by their methods do not always fit our intuitions, and suggests some new methods which fit our intuitions under several criteria such as modus ponens and modus tollens. This paper also contains the discussion of the fuzzy inferences whose antecedents have fuzzy quantifiers such as "most", "some" and "many" using our new methods for fuzzy conditional inferences.

1. FUZZY CONDITIONAL INFERENCES

We shall consider the following form of inference in which a fuzzy conditional proposition is contained.

$$\begin{array}{ll} \text{Ant 1:} & \text{If x is A then y is B.} \\ \text{Ant 2:} & \text{x is A'.} \\ \hline \text{Cons:} & \text{y is B'.} \end{array} \qquad (1)$$

where x and y are the names of objects, and A, A', B and B' are the labels of fuzzy sets in universes of discourse U, U, V and V, respectively.

For this form of fuzzy conditional inference, several methods are proposed.

Let A and B be fuzzy sets in U and V, respectively, which are written as

$$A = \int_U \mu_A(u)/u \; ; \qquad B = \int_V \mu_B(v)/v \qquad (2)$$

and let x, ∪, ∩, 7 and ⊕ be cartesian product, union, intersection, complement and bounded-sum for fuzzy sets, respectively.

Then the following fuzzy relations are obtained from a fuzzy conditional statement "If x is A then y is B" in Ant 1 of (1). The fuzzy relations R_m, R_a are proposed by Zadeh [1], R_c is by Mamdani [2], and R_s, R_g, R_{sg} and R_{gg} are new methods proposed here.

$$R_m = (A \times B) \cup (7A \times V). \qquad (3)$$

$$R_a = (7A \times V) \oplus (U \times B). \qquad (4)$$

$$R_c = A \times B. \qquad (5)$$

$$R_s = A \times V \xrightarrow[s]{} U \times B \qquad (6)$$

$$= \int_{U\times V} [\mu_A(u) \xrightarrow[s]{} \mu_B(v)]/(u,v),$$

where

$$\mu_A(u) \xrightarrow[s]{} \mu_B(v) = \begin{cases} 1 & \ldots \mu_A(u) \le \mu_B(v), \\ 0 & \ldots \mu_A(u) > \mu_B(v). \end{cases}$$

$$R_g = A \times V \xrightarrow[g]{} U \times B \qquad (7)$$

$$= \int_{U\times V} [\mu_A(u) \xrightarrow[g]{} \mu_B(v)]/(u,v),$$

where

$$\mu_A(u) \xrightarrow[g]{} \mu_B(v) = \begin{cases} 1 & \ldots \mu_A(u) \le \mu_B(v), \\ \mu_B(v) & \ldots \mu_A(u) > \mu_B(v). \end{cases}$$

$$R_{sg} = (A \times V \xrightarrow[s]{} U \times B) \cap (7A \times V \xrightarrow[g]{} U \times 7B). \qquad (8)$$

$$R_{gg} = (A \times V \xrightarrow[g]{} U \times B) \cap (7A \times V \xrightarrow[g]{} U \times 7B). \qquad (9)$$

Then the consequence B' in Cons of (1) can be deduced from Ant 1 and Ant 2 using the max-min composition "o" of the fuzzy set A' in U and the fuzzy relation obtained above. Thus, we can have

$$B'_m = A' \circ R_m = A' \circ ((A \times B) \cup (7A \times V)),$$

$$B'_a = A' \circ ((7A \times V) \oplus (U \times B)),$$

and so on.

Table I Relations between Ant 2 and Cons under Ant 1 in (1), and the satisfaction of the relation under each method

	Ant 2	Cons	R_m	R_a	R_c	R_s	R_g	R_{sg}	R_{gg}
Relation I (modus ponens)	A	B	X	X	O	O	O	O	O
Relation II-1	very A	very B	X	X	X	O	X	O	X
Relation II-2	very A	B	X	X	O	X	O	X	O
Relation III	more or less A	more or less B	X	X	X	O	O	O	O
Relation IV-1	not A	unknown	O	O	X	O	O	X	X
Relation IV-2	not A	not B	X	X	X	X	X	O	O
Relation V (modus tollens)	not B	not A	X	X	X	O	X	O	X

In the above form of fuzzy conditional inferrence, it seems according to our intuitions that the relations between A' in Ant 2 and B' in Cons in (1) ought to be satisfied as shown in the left part of Table I. Relation II-2 has the result different from that of Relation II-1, but in Ant 1 if there is not a strong casual relation between "x is A" and "y is B", the satisfaction of Relation II-2 will be permitted. Relation IV-1 asserts that when x is not A, any information about y can not be deduced from Ant 1. The satisfaction of Relation IV-2 is demanded when the fuzzy proposition "If x is A then y is B" means tacitly the proposition "If x is A then y is B else y is not B." Relation V corresponds to modus tollens in which the form of this inference is

Ant 1: If x is A then y is B.

Ant 2: y is not B.

Cons: x is not A.

In Table I, it is noted that very A is defined as A^2, more or less A as $A^{0.5}$, not A as $\neg A$, and unknown as V.

The right part of Table I shows the satisfaction (O) or failure (X) of each Relation under the methods given in (3)-(9). It is assumed here that fuzzy sets A and B in (2) satisfy the conditions in the discussion of Relations I-III:

(i) $\{\mu_A(u) \mid u \in U\} \supseteq \{\mu_B(v) \mid v \in V\}$,

(ii) $\exists u \in U \quad \mu_A(u) = 0; \exists u' \in U \quad \mu_A(u') = 1$,

(iii) $\exists v \in V \quad \mu_B(v) = 0; \exists v' \in V \quad \mu_B(v') = 1$.

But in the discussion of Relation V, we use the condition (i)' instead of (i):

(i)' $\{\mu_A(u) \mid u \in U\} \subseteq \{\mu_B(v) \mid v \in V\}$.

2. FUZZY INFERENCES WITH FUZZY QUANTIFIERS

In this section we shall consider such inferences that the antecedents are quantified by the fuzzy quantifiers such as "most", "a few", etc.

Let us consider a simple form of such inference as

Ant 1: Most Swedes are blond.
Ant 2: Karl is a Swede. (10)
Cons: It is likely that Karl is blond.

In general, this form of inferences may be expressed in symbol as

Ant 1: qX are E.
Ant 2: x' is a member of X. (11)
Cons: It is p that x' is E.

where

$q \in Q$ = most + almost + some + a few + ...
$p \in P$ = likely + very likely + probable + ...

X is a certain set (X = {Swedes} in (10)), and E represents an attribute value of the element of X (E = blond in (10)). q ($\in Q$) is a fuzzy quantifier and can be interpreted as representing a fuzzy rate. p ($\in P$) is a linguistic probability and represents a subjective fuzzy probability (which is denoted by Pr(x' is E)) of the event "x' is E" over X. Since we can assume that p is determined by the fuzzy quantifier q alone, we introduce a function f which is a mapping from a rate space (say, the interval [0,100], with its element interpreted as %) into a probability space [0,1]. Let a fuzzy quantifier q be represented as a fuzzy set in the rate space, then we can get a fuzzy probability f(q) as a fuzzy set in [0,1] by applying the extension principle [1].

Using this function f, we can get the following statement from (11).

Ant 1': $x \subset X \longrightarrow Pr(x \text{ is } E) = f(q)$
Ant 2': $x' \in X$ (12)
Cons': $Pr(x' \text{ is } E) = f(q)$

Based on this discussion we shall consider the following form of a slightly complicate inference which includes fuzzy quantifier "most", and fuzzy attributes "tall" and "more or less tall".

Ant 1: Most tall men are well-built.
Ant 2: Tom is more or less tall. (13)
Cons: It is likely that Tom is well-built.

In general, this form of inferences may be represented in symbol as

Ant 1': x is A $\longrightarrow$ Pr(x is E) = f(q)
Ant 2': x' is A' (14)
Cons': Pr(x' is E) = p'

where A and A' are fuzzy attributes represented by fuzzy sets in a universe of discourse U (In the case of "tall" and "more or less tall", U will be, say, 150cm + 151cm + ... + 200cm). p' is a fuzzy probability which can be obtained from the consequence of the inference.

For this type of inference, it may be thought that the relations in Table II ought to be satisfied between A' and p' under Ant 1' in (14).

Ant 1' of (14) can be read formally as

If x is A then Pr(x is E) is f(q). (15)

Thus we can deduce p' of (14) using the methods for fuzzy conditional inferences in Sec. 1.

If Ant 1' of (14), i.e., (15) translates into the fuzzy relation defined in (6), namely

$$R_s(A, f(q)) = A \times V \xrightarrow[s]{} U \times f(q) \qquad (16)$$

where U and V are universes of discourse of A and f(q), respectively, then the consequence Pr(x' is E) of (14) is given by

$$Pr(x' \text{ is } E) = p' = A' \circ R_s(A, f(q)). \qquad (17)$$

Thus it follows from (17) that Relations I°, II°-1, III° and IV° in Table II are satisfied.

Similarly, if (15) translates into the fuzzy relation given by (7), then we find that Relations I°, II°-2, III° and IV° are satisfied.

Example: Let us consider the following antecedents Ant 1 and Ant 2:

Ant 1: Most tall men are well-built.
Ant 2: Tom is very tall. (18)

where tall is a fuzzy set represented by

tall = 0.2/160 + 0.4/165 + 0.6/170 + 0.8/175 + 1/180 + 1/185 + 1/190,

with the universe of discourse, U, of tall being

U = 150 + 155 + 160 + 165 + ... + 185 + 190.

Table II Relations between A' and p'

	A'	p'
Relation I°	A	f(q)
Relation II°-1	very A	very f(q)
Relation II°-2	very A	f(q)
Relation III°	more or less A	more or less f(q)
Relation IV°	not A	unknown

Let the rate space of most be

0% + 10% + 20% + 30% + ... + 90% + 100%,

and let the probability space of f(most) be

0 + 0.1 + 0.2 + 0.3 + ... + 0.9 + 1,

and f(x) = x : 100 with x in the rate space, then when most is a fuzzy set in the rate space, that is,

most = 0.4/70% + 0.6/80% + 0.8/90% + 1/100%,

f(most) will be

f(most) = 0.4/0.7 + 0.6/0.8 + 0.8/0.9 + 1/1.

The antecedents Ant 1 and 2 can be rewritten from the notation of (14) and (15) as

Ant 1': If x is tall then Pr(x is well-built) is f(most).
Ant 2': Tom is very tall.

Therefore R_s(tall, f(most)) becomes

	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	1
150	1	1	1	1	1	1	1	1	1	1	1
155	1	1	1	1	1	1	1	1	1	1	1
160	0	0	0	0	0	0	0	1	1	1	1
165	0	0	0	0	0	0	0	1	1	1	1
170	0	0	0	0	0	0	0	0	1	1	1
175	0	0	0	0	0	0	0	0	0	1	1
180	0	0	0	0	0	0	0	0	0	0	1
185	0	0	0	0	0	0	0	0	0	0	1
190	0	0	0	0	0	0	0	0	0	0	1

Pr(Tom is well-built) = very tall $\circ$ R_s(tall, f(most))
= 0.16/0.7 + 0.36/0.8 + 0.64/0.9 + 1/1 = p'.

If this fuzzy probability can be approximated by the linguistic probability, say, very likely, then the Cons of Ant 1 and 2 of (18) will be

Cons: It is very likely that Tom is well-built.

REFERENCES

1. Zadeh,L.A. (1975). Calculus of fuzzy restrictions, in Fuzzy Sets and Their Applications to Cognitive and Decision Processes (ed. Zadeh, Tanaka et al.). New York: Academic Press, 1-39.
2. Mamdani, E.H. (1977). Application of fuzzy logic to approximate reasoning using linguistic system. IEEE Trans. on Computer, c-26, 1182-1191.

THE SYNTHESIS OF PROGRAMS BY ANALOGY

Robert Moll
Computer and Information Science Department
University of Massachusetts and
Hampshire College
Amherst, MA 01003

John Wade Ulrich
Computinq and Information Sciences Department
University of New Mexico
Albuquerque, NM

Analogical reasoning is the method by which solutions to old problems are adapted to solve new problems. In this paper we illustrate how analoqy can bo used in a non-trivial problem solving domain. Though we restrict ourselves here to the domain of computer programming, we argue that the principal idea is applicable to a broad class of problem solving situations.

1. PLANS AND ANALOGIES

A plan is a description of how a problem may be solved. We wish to adapt old plans to new problems. To do this we first identify certain details of a plan that are particular to one problem, and replace them with details from a second problem. This plan modification process is the essential part of our formulation of analogical reasoning. Reasoning by analogy is successful if application of the modified plan solves the new problem.

In order to automate analogical reasonina in the context of a transformational program synthesis system, plans must be described so that programming details that apply to particular problems may be easily identified and modified. We accomplish this by first specifying a small set of elementary transformations. Each elementary transformation, when appropriately instantiated, is applicable to a variety of programming situations. Then we introduce a means for combining elementary transformations into plans. The following is an example of a synthesis plan:

```
case L = ( )
    true: <unfold member(z,L), simplify>
    false: <unfold member(z,L);
        case atom(car(L))
            true: <simplify, fold>
            false: <simplify>>
```

The underlined expressions are formed from the predicates and terms of the specifications for the program islat(L), which returns true if the list L contains only atoms:

islat(L) <-- (all z) [member(z,L) implies atom(z)] where islist(L).

The words simplify, fold, case, test and unfold refer to elementary transformations. A plan is executed by applying the elementary transformations to selected portions of the program specification. If the final result is a well-formed program then the plan is successful. In the islat synthesis example the elementary transformations are applied as follows:

Step 1: Case L = ( ) [( ) denotes the empty list

The body of the program specification is replaced by

```
if L = ( ) then (all z)[member(z,L) implies
atom(z)]
   else (all z)[member(z,L) implies atom(z)].
```

Explanation:

The synthesis rule 'case B' causes the replacement of an expression A with an expression 'if B then A else A'.

Step 2: Unfold member(z,L) and Simplify.

The body of the specification becomes:

```
if L = ( ) then true else (all z)[member(z,L)
implies atom(z)].
```

Explanation:

The unfolding rule replaces member(z,L) with the body of the membership function:

```
if L = ( ) then false else if car(L) = z then
true
   else member(z,cdr(L)), where islist(L).
```

Under the case assumption L = ( ), member(z,L) simplifies to false. The simplifier further simplifies (all z)[false implies atom(z)] to true.

Step 3: Case atom(car(L))

```
if L = ( ) then true else
   if atom(car(L)) then (all z)[member(z,L)
   implies atom(z)]
     else (all z)[member(z,L) implies atom(z)].
```

Step 4: Unfold member(z,L) and Simplify

```
if L = ( ) then true else
   if atom (car(L)) then (all z)[member(z,
   cdr(L)) implies atom(z)]
     else (all z)[member(z,L) implies atom(z)].
```

Explanation:

Membership is unfolded once again, and simplification is performed under the assumption atom(car(L)).

Step 5: Fold

```
if L = ( ) then true else
   if atom(car(L)) then islat(cdr(L)) else
     (all z)[member(z,L) implies atom(z)].
```

Explanation:

The expression (all z)[member(z,cdr(L)) implies atom(z)] matches the top level specification of the program, with cdr(L) as a replacement for L. We replace the quantified expression with the recursive call islat(cdr(L)).

Step 6: Unfold member(z,L) and Simplify

```
if L = ( ) then true else
   if atom(car(L)) then islat(cdr(L)) else
   false
```

Explanation:

The final occurrence of the membership predicate is unfolded, and the expression is simplified to false under the assumption not(atom(car(L))). The result is the desired program.

2. SYNTHESIS BY ANALOGY

Now suppose we consider a problem which is quite similar to the islat synthesis problem: the synthesis from specifications of the predicate issubset(L1, L2). Issubset(L1, L2) has the specification

```
issubset(L1, L2) ← (all z)[member(z,L) implies
member(z,L2)]
   where islist(L1) and islist(L2).
```

This specification is syntactically analogous to the islat specification, and we can indicate the relevant analogical correspondences as follows:

```
                L : L1
member(z,L)     :  member(z,L1)
   atom(z)      :    member(z,L2)
```

Our next task is to attempt the synthesis of issubset using the islat synthesis plan and the analogical correspondences mentioned above. If we systematically replace the islat expressions with the corresponding issubset expressions we are left with the following plan for issubset:

```
case L1 = ( )
   true: <unfold member(z,L1), simplify>
   false: <unfold member(z,L1);
            case member(car(L1), L2)
               true: <simplify, fold>
               false: <simplify>>
```

When this plan is applied to the issubset specifications, the following program is produced:

```
issubset(L1, L2):
  if L1 = ( ) then true else
     if member(car(L1),L2) then issubset(cdr
     (L1),L2) else false.
```

As a second example of this principle consider the problem of determining if every member of a list L1 is less than some member of a second list L2. This problem can be specified as follows:

```
maxlists(L1,L2) ← (all z)[member(z,L) implies
  (exists w)[member(w,L2) and z < w)]],
    where islist(L1) and islist(L2).
```

Once again we indicate a family of analogical correspondence between the target specification and the islat specification:

```
                L : L1
member(z,L)     :  member(a,L1)
   atom(z)      :    exists w (member(w,L2) and
                     z < w)
```

We may introduce the abstraction:

```
bounded(w,L2) ← (exists w)[member(x,L2) and
z < w] where islist(L2)).
```

If we systematically substitute the maxlists expressions for the corresponding islat expressions in the islat synthesis plan we arrive at a successful plan for the synthesis of maxlists:

```
case L = ( )
    true: <unfold member(z,L1); simplify>
    false: <unfold member(z,L1);
       case bounded(car(L1), L2)
           true: <simplify, fold>
           false: <simplify>>
```

The code that is generated is:

```
  maxlists(L1, L2);
     if L1 = ( ) then false else
        if bounded(car(L1), L2) then maxlists
        (cdr(L1), L2)
           else false.
```

In this case synthesis is only partially complete. A subproblem, the synthesis of bounded (z,L1) must be solved in order to complete the synthesis.

We point out that the program synthesized is not the most efficient algorithm for computing maxlists. A more efficient algorithm would compute the max of LI first, and then check to see if some element of L2 exceeded this number.

3. CONCLUSIONS

We have developed a framework for studying analogical reasoning as a tool for automatic program synthesis, and we have implemented an interactive synthesis system. Our system has synthesized a variety of programming problems including those in the text. In addition to universal and existential quantifiers our specification language allows two other non-constructive operators, 'find' and 'findlistof*.

In order to give a mechanical formulation of analogical reasoning we have found it essential to distinguish between knowledge about particular problems and knowledge about problem solving. The first kind of knowledge is represented in the form of definitions and specifications. The second kind of knowledge is represented in the form of plans. The bridge between these two types of knowledge is the vocabulary of the problem being solved. If the design of a transformation-based system preserves the distinction between types of knowledge, then analogical reasoning as a problem solving device becomes possible. That is, a particular problem solving strategy, represented as a transformational sequence or plan, will have applicability to a variety of intuitively similar problems. The distinction between domain knowledqe and problem solving strategies can be enforced by requiring that transformation rules know only about definitions and how they may be manipulated. This is contrary to the frequent practice of embedding domain knowledge in the rules of a problem solving system.

4. BIBLIOGRAPHY

Boyer, R.S., and Moore, J.S., "A Lemma Driven Automatic Theorem Prover for Recrusive Function Theory", Proceedings of the Fifth IJCAI, Cambridge, Massachusetts, 1977, pp. 511-519.

Burstall, R.M., and Darlington, J., "A Transformation System for Developing Recursive Programs", JACM, Vol. 24, No.l, 1977, pp. 46-67.

Darlington, J., "A Synthesis of Several Sorting Algorithms", Research Report 23, Department of Artificial Intelligence, University of Edinburgh, Scotland, July, 1976.

Dershowitz, N., and Manna, Z., The Evolution of Programs: A System for Automatic Program Modification, IEEE Transactions on Software Engineering, Vol. SE-3, No.4, 1977.

Manna, Z., and Waldinger, R.J., "Knowledqe and Reasoning in Program Synthesis", Artificial Intelligence, Vol. 6, No.2, 1975, pp.175-208.

Manna, Z., and Waldinger, R.J., "Synthesis: Dreams ---> Programs", Technical Note 156, SRI International, Menlo Park, California, 1977.

Nelson, G., and Oppen, D., "Simplification by Cooperating Decision Procedures", Fifth ACM Symposium on Principles of Programming Languages, 1978.

Oppen, D., "Reasoning about Recursively Defined Data Structures", Stanford Artificial Intelligence Laboratory, Memo AIM-314, July, 1978.

Ulrich, J.W., and Moll, R., "Program Synthesis by Analogy", Proceedings of the Symposium on Artificial Intelligence and Programming Languages, Rochester, New York, 1977, pp.22-28.

AN EMPIRICAL METHOD THAT PROVIDES A BASIS FOR THE ORGANIZATION OF RELAXATION LABELING PROCESSES FOR VISION

Fanya S. Montalvo
Computer Science & Applied
Mathematics Department
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Naomi Welsstein
Department of Psychology
State University of New York
Buffalo, NT 14226

Relaxation labeling is a technique used in computer vision which parallels the operation of human vision* Weisstein et al. have developed an experimental method for investigating higher-level processes while maintaining a close tie to the underlying brain structure. Here we demonstrate that because of the close correspondence between relaxation labeling processes and neural network processes, the experimental results provide a basis for the design of relaxation labeling processes for vision.

1* RELAXATION LABELING

Relaxation labeling is a computer vision technique that corresponds closely to neural networks in the human brain [1,2,3,4]. It is a powerful technique whereby parallel, interacting processes may be uniformly represented at many levels while incorporating context in a natural way. In designing these networks for higher-level features the choice of labels and interactions between labels is not always clear.

Neurophysiology has thus far fallen short of clarifying the operation of higher-level symbolic processes in human vision. Single cell studies do not address the problems of distributed processing; information from electrically evoked potentials is at too large a scale to address Issues of feature representation within layers. The psychophysical method developed by Weisstein et al. [5,6,7,8] has the advantage of capturing aspects of higher-level processes while maintaining plausible links to the underlying neurophysiologies! structure* It makes the connection between the logical representation of visual features in a scene and the spatio-temporal organization of the brain, and thereby provides a basis for design specifications of relaxation labeling processes more complex than simple bar detectors*

The relaxation labeling process [4] is the iterative, parallel computation of a consistent array of location-specific labels representing features in the visual field* Compatibility functions specify which local features are most consistent with which neighboring features. For example, if we want to enhance straight line contours, we choose labels that represent straight line segments from a range of orientations and choose compatibility functions that strengthen neighboring labels of similar orientations and weaken labels of other orientations. In a higher-level system designed for office scenes [9], the compatibility functions strengthen homogeneous labeling (wall next to wall) and probable occurrences (wall next to door), and weaken improbable ones (picture next to floor). However, the system is far from being a general-purpose vision system in that the number of possible labels for surfaces is far too limited. A more general system would employ multiple levels of representation with interactions both within levels and across levels of the hierarchy [10].

Weissteln's method described here has been designed to map out the spatio-temporal scope and strength of interactions between processes that detect specific features of the visual input [7,8]. Characteristics of the response functions obtained indicate the types of processes involved, what they represent, when they occur, and how they interact, thus narrowing the choices for possible labels and compatibility functions.

2* THE METHOD

Two brief (10 msec*) visual stimuli, such as line segments, separated by a brief time interval are presented to a subject. The delay between the first stimulus (the target) and the second (the

*This work was supported in part by the Office of Energy Research of the U.S. Department of Energy under contract No. W-7405-ENG-48.

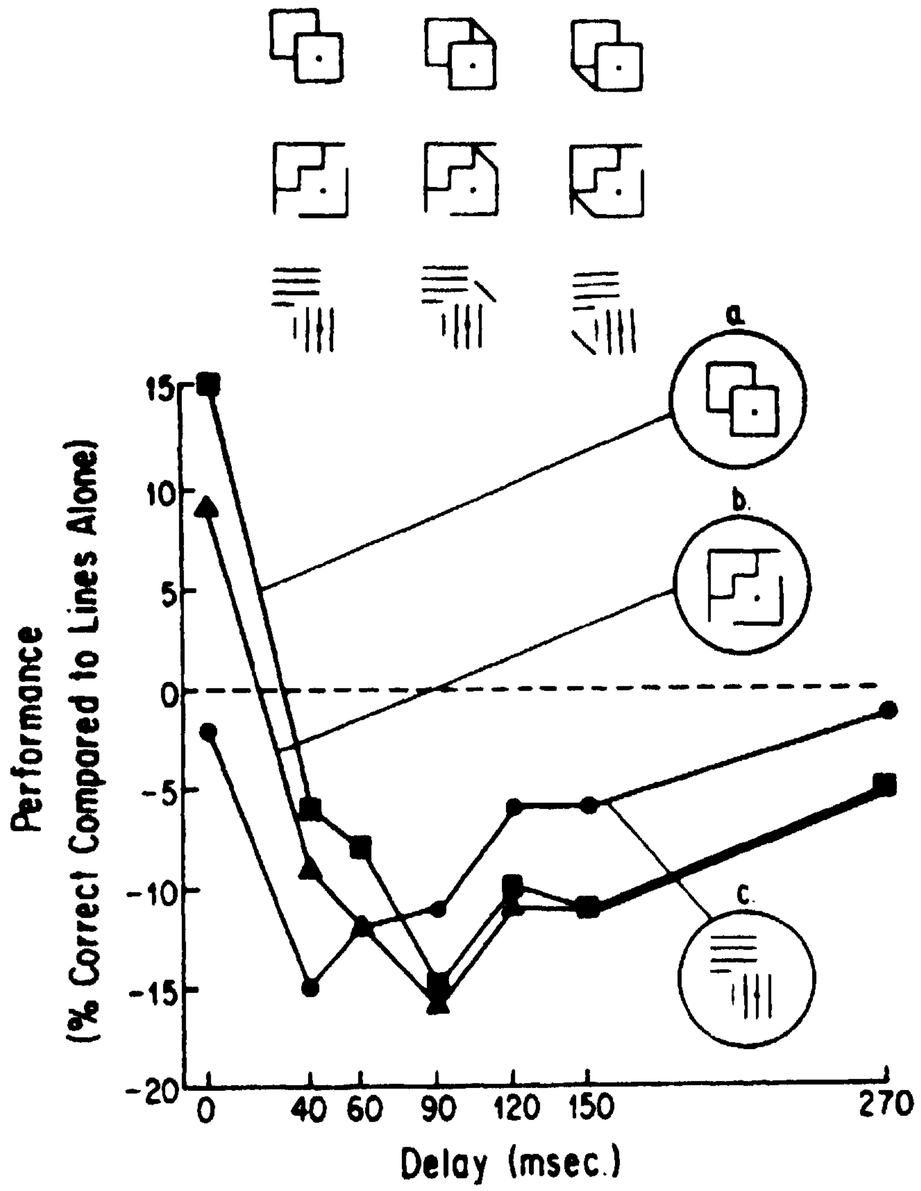


FIGURE 1. Three typical masking functions for masks pictured In circles. Masks, a, b, and c, are shown alone In the top column and with target lines embedded In them to the right. The accuracy for detection of a single line-segment target Is plotted In terms of the percentage difference from accuracy for a target presented alone (dashed baseline). (Data from Williams & Weisstein [7].)

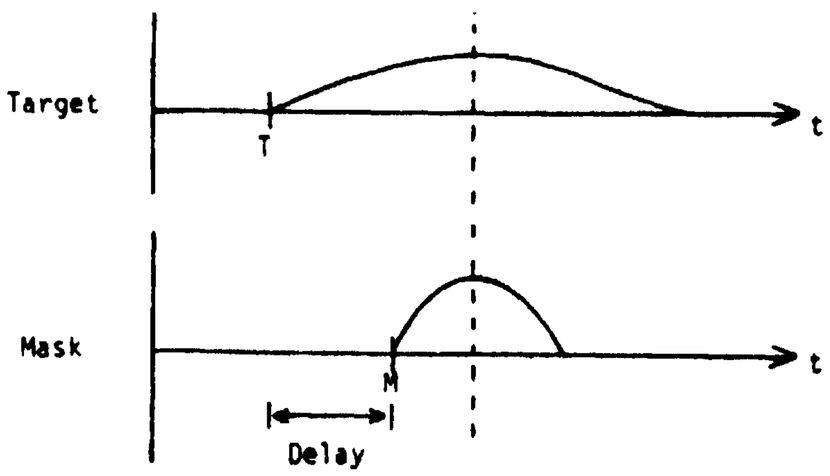


FIGURE 2. The typical time course of neural processes In response to a target and mask. If the response to a mask peaks faster, then the target must be presented before the mask for maximum masking to occur.

mask) Is varied while some measure of visibility Is taken [11). Plotting the accuracy of response against delay yields a time function of the Influence of the mask on the perception of the target. (See figure 1.) Usually, a minimum in this function occurs at a delay indicating the interval at which maximum masking occurs. For some stimuli enhancement rather than masking takes place. Both target and mask set up waveforms somewhere in the visual system that represent the processes enabled by their presentation. Each of these processes reaches some maximum and then decays to zero. Maximum negative interaction between target and mask processes occurs at the time interval that causes the peaks in their processing to coincide. Surprisingly, maximum masking generally occurs at a non-zero delay [12], implying that the time courses of the two processes differ. (See figure 2.)

In addition to time interactions, spatial Interactions can be measured by varying spatial separation instead of delay [12]. In this way, contrast enhancement curves corresponding to lateral inhibition have been mapped out by Growney and Velssteln [13]. Figure 3 shows some preliminary experimental results which correspond to heuristic compatibility functions chosen by

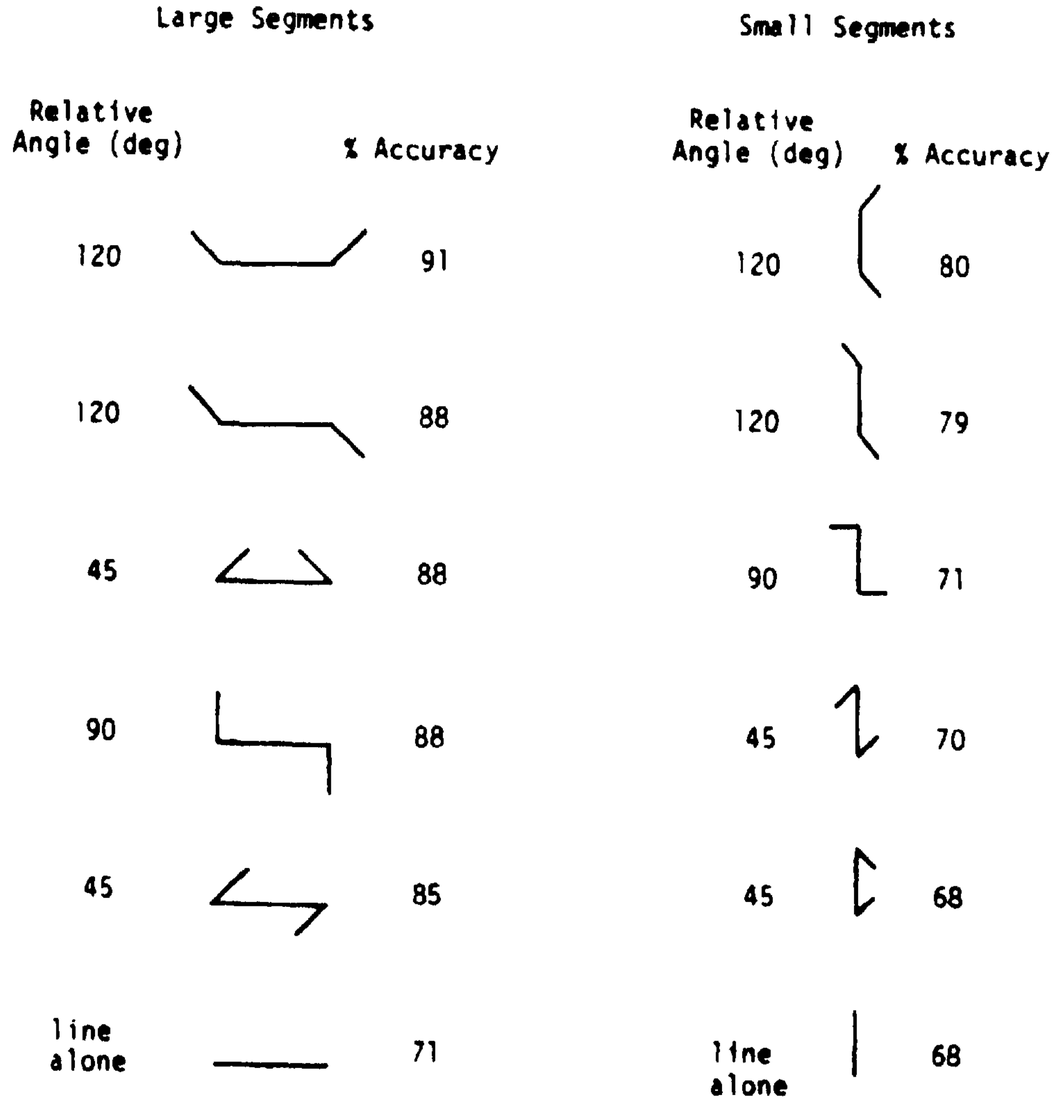


FIGURE 3. Percent accuracy for detection of line segment targets presented alone or with two flanking line segments as masks.

Zucker, Hummel, and Rosenfeld [1A]. Line segments In the context of relatively straight contours are enhanced, while segments in the context of more curved or non-continuing contours are suppressed. Thus, the strength of spatial interactions between stimuli provides an experimental basis for the design of compatibility functions. The scope of interactions and the size of features in the input can be used to infer either the scale at which local Interactions are taking place or the spatial scope and structure of compatibility functions. The choice of labels for nodes in the network can be narrowed down by a classification of features in the stimuli based on the characteristic masking functions they produce. For example, a number of masking stimuli which appear to be three-dimensional show enhancement at zero delay, while masks with absolutely no three-dimensional cues never show enhancement (Figure 1) [8,11]. Stimuli with fewer free endpoints and with less overall curvature between connected segments have a minimum at about 90 msec., while totally unconnected stimuli show a minimum at 40 msec. Key structural features of masks can be varied gradually in order to ascertain exactly which features make up the subjective impression of three-dimensionality and connectedness. For example, three-dimensionality may be a function of the number of T-junctions in the scene, support cues, perspective lines, or apparent solidity. We can vary detailed structural features of the masking context in order to isolate the primitives that make up these higher-level global features. Of course, we would expect the relaxation labeling process to occur at many levels of representation. However, the spatial scope of a feature and Its interactions gives some indication of where it is represented In the hierarchy.

3- CONCLUSION

We have described a psychophysical method that connects two areas of vision: one which addresses issues of symbolic representation, and one which deals with their underlying implementation In the brain. The sensory underpinnings to this method correspond closely to relaxation labeling processes from computer vision. We have a method for mapping out and characterizing symbolic processes at many levels of representation and measuring their interaction in space and time, for one general-purpose working system, the human brain. The measures of interactions between symbolic processes provide a basis for the design of compatibility functions in the relaxation labeling paradigm*

REFERENCES

[I] Arbib, M.A. "Artificial intelligence and brain theory." Comp.& Info.Sci. Tech.Rep. 75C-8, Univ. of Mass., Sept., 1975.

[2] Montalvo, F.S. "Consensus versus competition in neural networks*" Int.J.Man-Machine Studies 7 (1975) 333-346.

[3] Marr, D. & Poggio, T. "Cooperative computation of stereo disparity." Sci. 194 (1977) 283-287.

[4] Zucker, S.W* & Mohammed, J.L. "Analysis of probabilistic relaxation labeling processes*" Dept.of EE Tech.Rep. 78-3R, McGill Univ., Jan., 1978.

[5] Weisstein, N. "What the frog's eye tells the human brain." Psyc. Bull. 72 (1969) 157-176.

[6] Weis8tein, N. "Beyond the yellow Volkswagen detector and the grandmother cell." In R.L. Solso (Ed.) Contemporary Issues in Cognitive Psychology. Wash. DC: V.H.Winston, 1973, 17-51.

(7) Williams, A. & Weisstein, N. "The time-course of object superiority." Bull.of Psychon.Soc. 8 (1976) 260.

[8] Weisstein, N. & Magulre, W. "Computing the next step*" In E* Riseman & A. Hanson (Eds*) Computer Vision Systems. New York: Academic Press, 1978, 243-260.

[9] Tenenbaum, J.M. & Barrow, H.G. "Experiments in interpretation-guided segmentation." Artif. Intell. 8:3 (1977) 241-274.

[10] Zucker, S.W. "Vertical and horizontal processes in low-level vision." In E. RIseman & A. Hanson (Eds.) Computer Vision Systems. New York: Academic Press, 1978, 187-195.

[II] Williams, A. & Weisstein, N. "Line segments are perceived better in a coherent context than alone." Memory & Cogn* 6 (1978) 85-90.

[12] Weisstein, N. "Metacontrast." In Jameson & Hurvick (Eds*) Visual Psychophysics* Heidelberg: Springer-Verlag, 1972, 233-272.

[13] Growney, R.L. & Weisstein, N. "Some spatial characteristics of metacontrast." J.Opt.Soc.Am. 62 (1972) 690-696.

[14] Zucker, S.W., Hummel, R.A. & Rosenfeld, A* "An application of relaxation labeling to line and curve enhancement." IEEE Trans.Comp. C-26 (1977) 394-403 and 922-929.

Visual Mapping by a Robot Rover

Hans P. Moravec
AI Lab, Computer Science Dept.
Stanford University, Stanford, Ca., 94305

Introduction

The real world provides potentially huge quantities of noisy data. The lower levels of the perception process must extract reliable and meaningful measurements from this sullied wealth. Much existing work in vision evades the bulk of the noise problem, either by physically optimizing the nature, lighting and coloration of the viewed scene, or by starting with very high quality digitized photography.

This paper is about a situation in which the noise could not be ignored. The routines described extract very reliable three dimensional information from unconstrained scenes perceived by an imprecise vehicle transmitting pictures whose quality has caused some vision researchers to hold their (metaphorical) noses.

The cart's noise problems may be unusually severe, but are present to some degree in all real world sensors and effectors. For this reason our solutions may be generally useful as part of the solid foundations of ambitious systems well beyond robot explorers, as computer vision moves from optimized workspaces into the inherently dirty real world.

The underlying theme of the work is the use of redundancy (which the real world supplies in profusion) to counter noise. In many cases I was forced into this by performance failure of other approaches.

Setting

The Cart [1] is a rickety camera equipped rover, computer controlled over a radio link. This report describes a working program that has successfully used pictures obtained from the cart as it was driven, unguided, through cluttered real world environments to obtain navigational information and to build correct three dimensional maps of its surroundings and to plan obstacle avoiding paths to desired locations. The program builds a sparse 3D map of its surroundings, but makes no attempt to classify the objects it sees, except as obstacles or non-obstacles. Work underway will enable the program to actively keep the cart on the planned path.

In 1977 I abandoned as unworkable an approach [2] that tried to obtain depth information from a scene exclusively through motion stereo pairs. The matching errors in our very noisy pictures, combined with an uncertain baseline set by vehicle motion, often caused incorrect measurements from the routine which deduced vehicle motion from the perceived shift of features in the scene. The navigational model quickly became hopelessly inaccurate. Several attempts at software fixes proved fruitless.

New hardware, to extract more noisy data from the world, has overcome the problem. The cart is now equipped with a mechanism that uses a stepping motor to slide the TV camera horizontally on a 52 cm track in precise 0.25 mm steps. The cart system uses this arrangement to obtain nine views of the scene, at 6.4 cm intervals. These provide a highly redundant nine-eyed stereo baseline for reliable distance measurements even in the presence of much noise.

Overview

Mapping runs occasionally begin with a camera calibration step (at other times old calibrations are used). The cart is placed 3 meters in front of and pointed towards a 3 meter square pattern of dots on a wall. A program digitizes the pattern, and automatically locates the spots and determines the camera's focal length as well as distortions.

The cart is then placed on a clear path in a cluttered space (environments have included a large room cluttered with boxes and old machinery, and an outdoor roadway with parked cars, trees and signposts). A navigating program is begins by reading in a calibration file. It then slides the camera to the left limit of its track. A TV image is digitized, the camera then slides 6.4 cm to the right, another picture is taken, and so on for a total of nine images. The cart sits immobile for about ten realtime minutes while our timeshared KL-10 works on the pictures for 180 compute seconds. Then it lurches about a meter forward, stops, and repeats the camera motion.

As the cart moves slowly forward, meter by meter, the program generates and updates a three dimensional world model of features it has seen, and deduces the vehicle's motion through them. This model is displayed in map form. Initially the map is blank. As time (and the cart) proceeds more and more points are added. The map includes the cart's past trajectory, and a path to a desired destination that avoids all known obstacles.

The program is being extended to issue steering commands, intended to keep the cart on its planned obstacle avoiding path, before each lurch.

Digitizing (Temporal Redundancy)

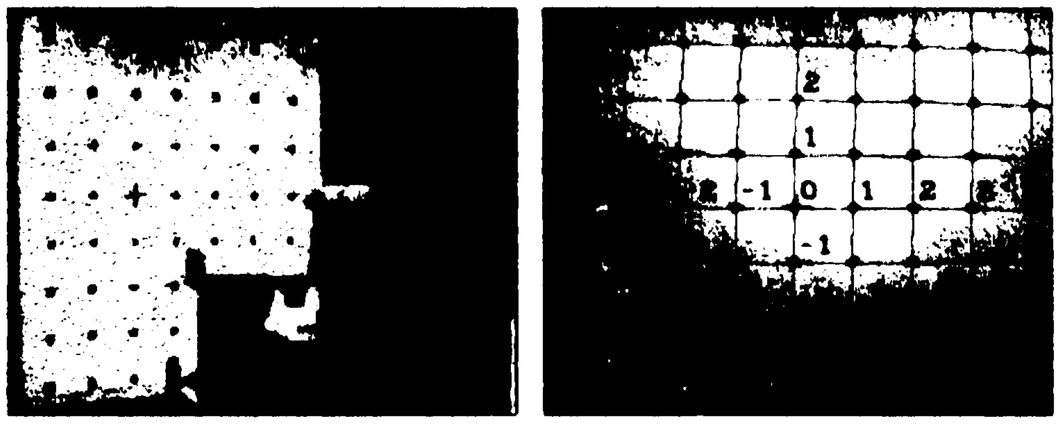
Digitizing images over a sometimes noisy radio link has its problems, but gaining adequate images is a crucial part of the cart system.

The picture is transmitted at 524 MHz, a wavelength of 57 cm. Reflections from nearby objects sometimes cancel the signal to a receiver. The programs are able to get their picture through any one of 4 TV receivers. They measure scanline to scanline noise on each in turn, and then pick the one with lowest noise. In future I may simply switch antennas to a single receiver.

Our decade old digitizer extracts a picture 280 samples wide by 240 high by 4 bits/sample from broadcast standard video. Four bits is only sixteen grey levels, and sometimes the noise level is high. Summing a number of frames can increase the intensity resolution and average out noise, but introduces new problems. The sync detector on the digitizer sometimes fails, especially when the signal is noisy, and the picture jitters or "rolls". Adding improperly registered pictures results in blurring. The problem has been solved by a program that digitizes a large number of frames (about 30), and by an efficient sampling of the pixels intercompares them to find the *subset* of pictures that are most nearly alike. These are then combined. Since most images have not suffered sync loss, the result is almost always a tolerable six bit picture.

Picture Reduction (Area Redundancy)

The cart vision programs make extensive use of reduced versions of input images. Digitized pictures are compressed repeatedly by linear factors of two, four pixels being added to become one, until the whole image has shrunk to a single pixel. The original image and all of its reductions are available to all subsequent steps.



The cart before its calibration array, and the digitized spot pattern, overlayed by an ideal grid distorted by the calibration polynomial.

Calibration (Redundancy in Regularity)

The cart camera's image is distorted by the details of its lens, and of its vidicon's scan. Its precision has been enhanced by an automatic focal length and distortion determining program.

The cart is parked a precise distance in front a square pattern of spots around a central cross. The program digitizes an image of the array, locates the spots and the cross, and constructs a 2D polynomial that relates the position of the spots in the image to their position in an ideal unity focal length camera, and another polynomial that converts points from the ideal camera to points in the image. These polynomials are used to correct the positions of perceived objects in later scenes.

The algorithm begins by determining its approximate spacing and orientation of the spot array. It trims the picture to make it square, reduces it by averaging to 64 by 64, calculates the Fourier transform of the reduced image and takes its power spectrum, arriving at a 2D transform symmetric about the origin, and having strong peaks at frequencies corresponding to the horizontal, vertical and half diagonal spacing, with weaker peaks at the harmonics. It multiplies each point $[i,j]$ in this transform by point $[-j,i]$ and points $[j-i,j+i]$ and $[i+j,j-i]$, effectively folding the primary peaks onto one another. The strongest peak in the 90 degree wedge around the y axis gives the spacing and orientation information needed by the next part of the process.

The directional variance interest operator described later is applied to roughly locate a spot near the center of the image. A special operator examines a window surrounding this position, generates a histogram of intensity values within the window and selects a threshold for separating the black spot from the

white background. It calculates the centroid and verifies the spot's shape by measuring the first and second moment. The region of found spots Is grown outward from this seed, using this operator guided by the Fourier determined orientation/spacing parameters.

The Fourier numbers are also uaed to construct a template for the expected appearance of the cross In the middle of the array. Each of the found spots Is matched to this cross template, and the closest match In the central portion of the picture Is declared to be the origin.

Two fourth degree least squares polynomials In two variables relating the actual positions of the spots to the Ideal positions are generated and written Into a file.

The program tolerates a wide range of spot parameters (about 3 to 12 spots across), arbitrary Image rotation, and Is very robust. It has worked without error on over 60 Images. The test pattern for the cart Is a 3 meter square painted on a wall, with 6 cm spots at 30 cm intervals. The program has also been used successfully with a small array (22 x 28 cm) to calibrate cameras other than the cart's.

Choosing Features {Redundancy in Boundaries)

The cart vision code deals with very simple primitive entitles, localized regions called features. A feature Is conceptually a point In the three dimensional world, but It Is found by examining localities larger than points In pictures. A feature Is good If It can be located unambiguously In different views of a scene. A uniformly colored region or a simple edge does not make for good features because Its parts are Indistinguishable. Regions, such as comers, with high contrast In orthogonal directions are best.

New features In Images are picked by a subroutine called the *Interest operator.* It tries to select a relatively uniform scattering, to maximize the probability that a few features will be picked on every visible object, and tries to choose areas that can be easily found In other Images. Both goals are achieved by returning regions that are local maxima of a directional variance measure.

Directional variance Is measured over small square windows. Sums of squares of differences of pixels adjacent In each of four directions (horizontal, vertical and two diagonals) over each window are calculated, and the window's Interest measure Is the minimum of these four sums.

Features are chosen where the Interest measure has local maxima. The feature Is conceptually the point at the center of the window with this locally maximal value.

The effects of noise are alleviated and the processing time Is shortened by applying the operator to a reduced Image. In the current program original Images are 240 lines high by 256 pixels wide. The Interest operator Is applied to the 120 by 128 version, on windows 3 pixels square.

Once s festure Is chosen, Its appearance Is recorded as series of excerpts from the reduced Image sequence. A 6x6 window is excised around the feature's location from each of the variously reduced pictures. Only a tiny fraction of the area of the original (unreduced) Image Is extracted. Four times as much of the x2 reduced Image Is stored, sixteen times as much of the x4 reduction, and so on until at some level we have the whole Image. The final result Is a series of 6 by 6 pictures, beginning with a very blurry rendition of the whole picture, gradually zooming in linear expansions of two to a sharp cioseup of the feature.

Finding FemturmM AgmJn {Redundancy in Continuity)

Deducing the 3D location of features from their projections In 2D Images requires that we know their position In two or more such Images.

The correlator Is a subroutine that takes a feature description (such as the Interest operator produces), and finds the best match In a different Image. Its search area can be an entire new picture, or a rectangular sub-window.

The search uses a coarse to fine strategy that begins In reduced versions of the pictures. Typically the first step takes place at the x16 (linear) reduction level. The 6 by 6 window at that level In the feature description, which covers about one seventh of the total area of the original picture, Is convolved with the search area In the correspondingly reduced version of the second picture. The 6 by 6 description patch Is moved pixel by pixel over the approximately 15 by 16 destination picture, and a correlation coefficient Is calculated for each trial position.

The position with the best match Is recorded. The 6x6 area It occupies In the second picture Is mapped to the x8 reduction level, where the corresponding region Is 12 pixels by 12. The 6 by 6 window In the x8 reduced level of the feature description Is then convolved with this 12 by 12 area, and the position of best match Is recorded and used as a search area for the x4 level.

The process continues, matching smaller and smaller, but more and more detailed windows until a 6 by 6 area is selected In the unreduced picture.

The work at each level Is about the same, finding a 6 by 6 window In a 12 by 12 search area. It Involves 49 summations of 36 quantities. In our example there were 6 such levels.

SI i dor StOtOO {Spatial Redundancy)

At each pause on Its computer controlled Itinerary the cart slides Its camera from left to right on the 62 cm track, taking 0 pictures at precise 6.5 cm Intervals.

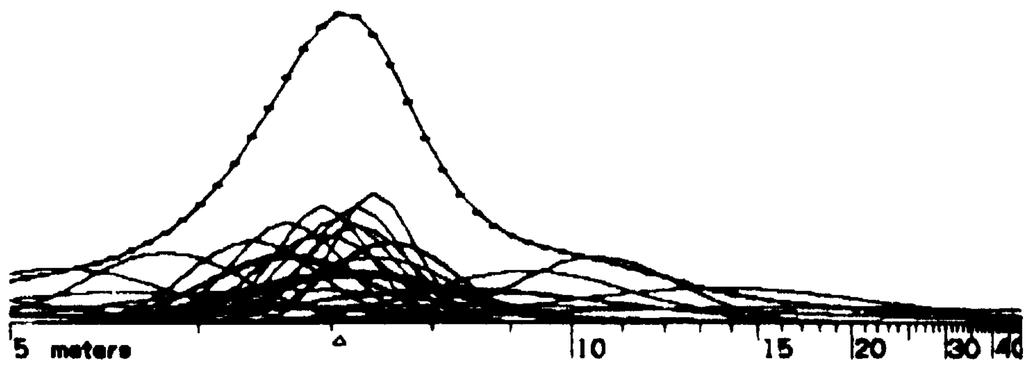
Points are chosen In the fifth (middle) of these 9 Images, either by the correlator, to match features from previous positions, or by the Interest operator.

The camera slides parallel to the horizontal axis of the (distortion corrected) camera co-ordinate system, so the parallax Induced apparent displacement of features from frame to frame In the 9 pictures la purely In the X direction.

The correlator looks for the selected features from the central Image In each of the eight other pictures. The search Is restricted to a narrow horizontal band. This has little effect on the computation time, but It reduces the probability of Incorrect matches.

In the case of correct matches, the distance to the feature la Inversely proportional to Ita displacement from one image to another. The uncertainty In such a measurement Is the difference In distance a shift one pixel In the Image would make. The uncertainty varies Inversely with the physical separation of the camera positions where the pictures were taken (the stereo baseline).

After the correlation step the program knows a feature's position In nine Images. It considers each of the 36 possible image pairings as a stereo baseline, and records the estimated distance to the feature In a histogram of Inverse distances. Each measurement adds a little normal curve to the histogram, with mean at the estimated distance, and standard deviation Inversely proportional to the baseline, modelling the distance uncertainty. The area under the each curve Is made proportional to the product of the correlation coefficients of the matches in the two Images (In central Image this coefficient Is taken as unity), reflecting the confidence that the correlations were correct. The area Is also scaled by the normalized dot products of X axis and the shift of the features In each of the two baseline images from the central Image. That Is, a distance measurement Is penalized If there Is significant motion of the feature In the V direction.



Results of a nine-eyed stereo ranging on a single feature. The small curves are from individual image pairs, the beaded curve is the final histogram. The feature is 7.1 meters distant.

The distance to the feature Is indicated by the largest peak In the resulting histogram, If this peak Is above a certain threshold. If below, the feature Is forgotten about.

This adding of normal curves and locating the peak In the sum Is an efficient way of finding the strongest cluster of mutually agreeing measurements. The distance measurements from different pairings of Incorrect matches are usually inconsistent. When the normal curves from 36 pairs are added up, the correct matches agree with each other, and build up a large peak In the histogram, while Incorrect matches spread themselves thinly. Two or three correct correlations out of the eight will usually build a peak sufficient to offset a larger number of errors.

In this way eight applications of a mildly reliable operator interact to make a very reliable distance measurement.

***Motion Stereo* (*Redundancy of Rigid Motions*)**

The cart navigates exclusively by vision. It deduces Its own motion from the apparent 3D shift of the features around it.

After having determined the 3D location of objects at one position, the computer drives the cart about a meter forward.

At the new position it slides the camera and takes nine pictures. The correlator Is applied In an attempt to find all the features successfully located at the previous position. Feature descriptions extracted from the central Image at the last position are searched for In the central Image at the new stopping place.

Slider stereo then determines the distance of the features so found from the cart's new position. The program now knows the 3D position of the features relative to Its camera at the old and the new locations. It can deduce Its own movement by finding the 3D co-ordinate transform that relates the two.

There can be mis-matches In the correlations between the central Images at two positions. Before the cart motion Is deduced, the feature positions are checked for consistency. Although it doesn't yet have the co-ordinate transform between the old and new camera systems, the program knows the distance between pairs of positions should be the same In both. It makes a matrix In which element $[i,j]$ Is the absolute value of the difference In distances between points i and j In the first and second co-ordinate systems divided by the expected error (based on the one pixel uncertainty of the ranging).

Each row of this matrix Is summed, giving an indication of how much each point disagrees with the other points. The idea Is that while points In error

disagree with virtually all points, correct positions agree with all the other correct ones, and disagree only with the bad ones.

The worst point is deleted, and its effect is removed from the remaining points in the row sums. This pruning is repeated until the worst error is within the error expected from the ranging uncertainty.

After the pruning, the program has a number of points, typically 10 to 20, whose position error is small and known. These are recorded in a world model.

From the 3D camera co-ordinates of these sanitized points at both positions, the program finds a three dimensional rotation and translation that, if applied to the co-ordinates of the features at the first position, minimizes the sum of the squares of the distances between the transformed first co-ordinates and the raw co-ordinates of the corresponding points at the second position, each term divided by the square of the *a priori* expected error.

The error expression is expanded. It becomes a function of the rotation and translation, with parameters that are the weighted averages of the x, y and z co-ordinates of the features at the two positions, and averages of their various cross-products. These averages need to be determined only once, at the begining of the transform finding process.

To minimize the error expression, its partial derivative with respect to each variable is set to zero. It is easy to simultaneously solve the three linear equations thus resulting for the vector offset, getting the optimal offset values for a general rotation. This gives symbolic expressions (linear combinations of the rotation matrix coefficients) for each of the three vector components. Substituting these values into the error expression makes it a function of the rotation alone. This reduced error expression is used in all the subsequent steps.

Minimizing the error expression under rotation is surprisingly difficult, mainly because of the non-linear constraints in the 3D rotation matrix.

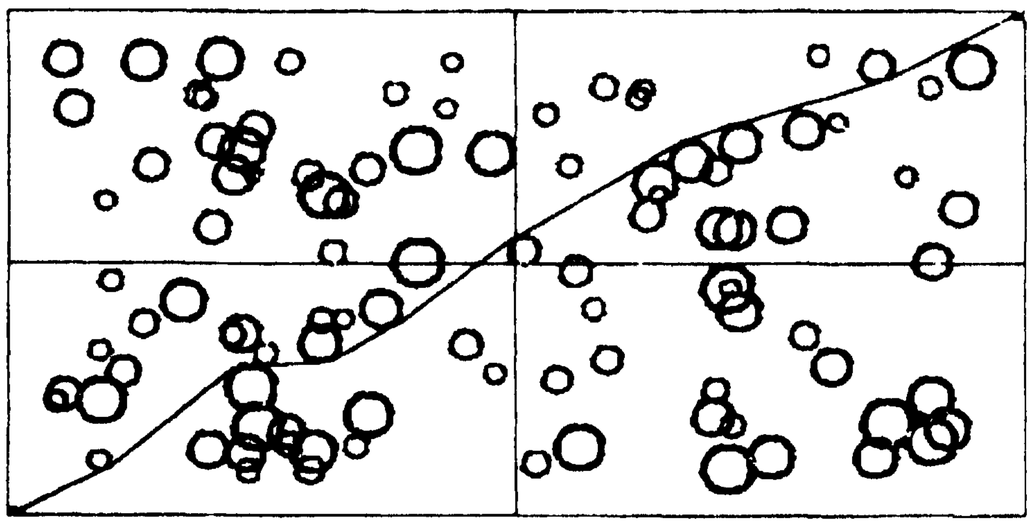
The program begins by ignoring the non-linearities. It solves for the general 3D linear transformation, nine elements of a matrix, that minimizes the least square error. The derivatives of the error expression with respect to each of the matrix coefficients are equated to zero, and the nine resulting simultaneous linear equations are solved for the nine coefficients. If the points had undergone an error-free rigid rotation and translation between the two positions, the result would be the desired rotation matrix, and the problem would be solved.

Because there are errors in the determined position of the features, the resulting matrix is usually not simply a rotation, but involves stretching and skewing. The program ortho-normalizes the matrix. and converts its 9 independent linear elements to unavoidably non-linear expressions in three parameters that uniquely characterize a rotation.

This new error expression is differentiated with respect to each of the three rotation parameters, and the resulting expressions are equated to zero, giving us three non-linear equations in three unknowns. These are solved by Newton's method.

Four or five iterations usually brings the parameters to within our floating point accuracy of the correct values. In the occasional case of non-convergence, the program picks a new starting point, and tries again, up to 100 times. The rotation with the smallest least square error ever encountered during such a search is returned as the answer.

Since the summations over the co-ordinate cross-products are done once and for all at the begining of the transformation determination, each iteration, involving evaluation of about a dozen moderately large expressions and a 3 by 3 matrix inversion, is relatively fast. The whole solving process, even in cases of pathological non-convergence, takes one or two seconds of computer time.



The path planner at work. Point of origin is lower left, destination is upper right. Obstacles are indicated by circles. This field was randomly generated. The apparent radius of the obstacles in a real case would be augmented by the cart's radius.

Path Planning

The features in the cart's 3D world model can be thought of as fuzzy ellipsoids, whose dimensions reflect the program's uncertainty of their position. Repeated applications of the interest operator as the cart moves cause virtually all visible objects to be become modelled as clusters of overlapping ellipsoids.

To simplify the problem, the ellipsoids are approximated by spheres. Those spheres sufficiently above the floor and below the cart's maximum height are projected on the floor as circles. The cart itself is modelled as a 3 meter sphere. The path finding problem then becomes one of maneuvering the cart's 3 meter projected circle between the (usually smaller) circles of the potential obstacles to a desired location.

It is convenient (and equivalent) to conceptually shrink the cart to a point, and add its radius to each and every obstacle. An optimum path in this environment will consist of either a straight run between start and finish, or a series of tangential segments between the cylinders and contacting arcs (imagine loosely laying a string from start to finish between the cylinders, then pulling it tight).

There are four possible paths between each pair of obstacles, because each tangent can approach clockwise or counterclockwise. The program finds a near optimal obstacle avoiding route to the desired goal in $O(n^3)$ time, using an extension of a shortest path in a graph algorithm first given by Dijkstra [3]. Because our tangent space cannot be modelled exactly as a simple graph, because of the route dependent arc lengths on the circular part of the paths, the program finds good paths, but not always the best.

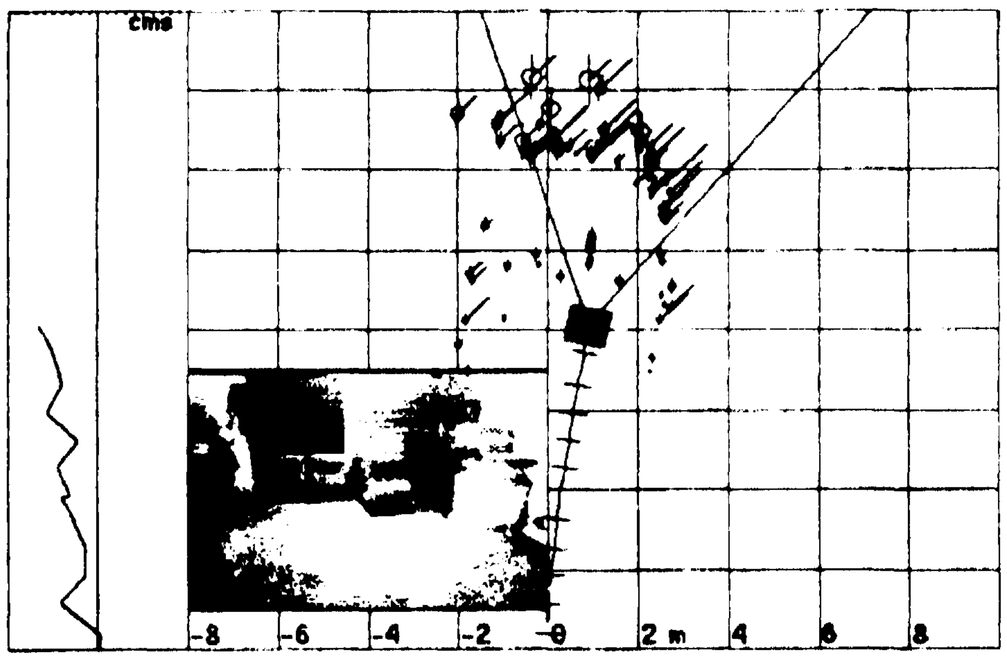
Results

Because of a hardware problem with the steering electronics, the program has thus far driven the cart without steering control, in nearly straight lines only. During these runs it has built correct world maps, and planned (but not executed) good obstacle avoiding paths.

The longest run thus far has been for 14 nine-picture sets, a total distance of about 10 meters. During this run, made in a large cluttered room, the program inserted 120 features into its world model, of which about half were potential obstacles, most of the other half being marks on the floor. I was able to find no errors beyond the known position uncertainties in the resulting map. The parts of the room that were in the cart's field of view for the bulk of the run are densely represented, with no possible paths falsely seeming to be obstacle-free. Several shorter runs gave similarly robust results.

The steering problem is being repaired, and I expect a long outdoor run, in which the cart will maneuver around several obstacles, in the near future.

A longer description of this work is available from the author.



Cart's world model at the end of a 14 lurch run. Photo is view from final position. Features are maked by circles, indicating position uncertainty, on the ground. 45 degree lines growing out of circles are height of features above ground. Cart's position is marked by black square. Forward pointing rays are its field of view. Tracks behind it are its itinerary, each stop is marked by a bar. Graph on left is its estimate of own height above ground. It shows the limiting accuracy (a few centimeters) of its self-position model.

References

1. R.A. Schmidt, "A Study of the Real-Time Control of a Computer Driven Vehicle", Stanford AI Memo 149, August 1971.

2. H.P. Moravec, "Towards Automatic Visual Obstacle Avoidance", *Proceedings of the 5th IJCAI*, MIT, Cambridge, Mass., 1977, p. 584.

3. E.W. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerische Mathematik*, 1, 269-271 (1959).

OPERATIONALIZING HEURISTICS: SOME AI METHODS FOR ASSISTING AI PROGRAMMING

Jack Mostow
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Frederick Hayes-Roth
Information Sciences Department
The Rand Corporation
1700 Main Street
Santa Monica, California 90406

Machine-aided heuristic programming is a paradigm for incorporating domain knowledge In Intelligent teak performance programs. In this paradigm, a system interactively assimilates a natural language description of i task advice on how to perform it, and definitions of the domain concepts in terms of which the advice la expreeeed. The) system translates this input into an internal representation, operationalizes the assimilated knowledge to make It effective, integrates different pieces of advice, and applies them to the performance of *he task. A typed applicative LISP-like language is used for the internal representation of domain knowledge. Operationally Ion la defined in terms of transforming well-defined but non-effective expressions Into effectively executable ones. Several techniques for performing this process mechanically are presented and applied to en example drawn from the domain of the card game Hearts. A system to operationalize Hearts advice is currently being implemented ee en Instantiation of the advocated paradigm.

1. Introduction

In a longer version of this paper [4], we advocate machine-aided heuristic programming as a paradigm for training computers to do new tasks, and formulate a set of research problems involved in building a machine-aided heuristic programming system (hereafter abbreviated "mahps") capable of assimilating conceptually diverse knowledge and putting it to effective use. In this paradigm, an expert interactively gives the mahps advice about how to do a task, plus definitions of the concepts in terms of which that advice is expressed. For example, in training a mahps to play the card game Hearts,** the expert might advise it to avoid taking points. Such advice is *parsed* into a syntax tree which is *interpreted* into an Internal representation. Advice may then be *operationaliied,* either *static ally* — independent of a particular game state — or *dynamically* — exploiting current information arising from an actual game. Once operationalized, the advice can be applied to produce recommendations relevant to the decision at hand.

Diverse advice is *integrated* by combining these recommendations, either rationally or heuristlcally, and *applied* to the system's game decisions and to its interpretations of its opponents' behavior. At any point, the system can pose questions to the expert using the rule

> If x is an important question which has resisted solution, explain it to the user (preferably with a small set of multiple choice answers) end ask for the answer or a piece of relevant advice.

The expert monitors the system's behavior and improves it by providing additional advice.

In this paper, we focus on the automatic operationalization of advice. We assume that the advice has been interpreted into an unambiguous internal representation of its meaning, and suggest some approaches to the problem of transforming that representation so that it can be effectively applied to the task.

2* Representation of Conceptual and Heuristic Knowledge

Our knowledge representation is a typed lambda-calculus combining an applicative LISP-like language [3] with a schema-like structure. The basic unit of representation ia the *concept,* which is a named lambda-expression. Schematic structure associated with each concept is described in [4]. Briefly, It provides type information about the concept and its lambda-variables, and defines inheritance relations between general and specific concepts

*This work was supported in part by the National Science Foundation, RCN 6869, Grant No. MCS77-03273 to the Rand Corporation, in part by the Graduate Fellowship program support to the first author, and in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-78-C-1551. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

**The card game Hearts is used throughout this paper for purposes of illustration; the rules of the game are given in [4].

A type is simply a class concept. Since one class may be a subclass of another, an entity may belong to more than one type. For instance, ME (the machine's card-playing persona) is both a PLAYER and a COMPUTER. Two types are said to *match* if they overlap. Thus PLAYER matches HUMAN since it is possible to be both human and a player.

Although there is no assignment operation in this representation, every expression is implicitly a state function, i.e., its interpretation depends on the state of the world it describes. Actions are represented as functions from a state to an *event.* A *history* is an actual or hypothetical sequence of states and events. When an action is performed, the history of what happens during its execution Is recorded so it can be referred to subsequently. Thus at any point in time one can refer to the PAST — the history of events that have already occurred, the PRESENT -- the set of events that have begun and not yet ended, and the FUTURE — the as yet undetermined history about to occur.

2.1. Meta-functions

Our representation provides a limited number of built-in meta-functions, characterized by their lack of strict type (some of them take a type as a parameter) and their need to examine some of their arguments without evaluating them. These meta-functions include the following (optional parameters and their default values appear in brackets):

(EXISTS x S P) = there exists x in S such that P

(SET-OF x S P) = the set of x in S such that P

(THE x S P) = the unique x in S such that P

(SOME x S [P]) = any x in S [such that P]; any expression containing this construct is potentially non-deterministic

(ALL e) = set of possible values for the non-deterministic expression e; thus (ALL (SOME x S)) = S and (SOME x (ALL e)) = e

(SUM V x S [P]) = the summation of V(x) for x in S [such that P(x)]

(# S) = the number of elements in the set S

(GIVEN P x) = the value of x given that P is true

(PR P) = probability that P is true

(PR P Q) = conditional probability of P given that Q is true; thus (PR P Q) = (PR (GIVEN Q P)) = (GIVEN Q (PR P))

(CASE (P_1 V_1) ... (P_k V_k) [V]) = the unique (GIVEN P_i V_i) such that P_i is true [otherwise V]

(IF P_1 V_1 ... P_k V_k V) = the first (GIVEN P_i V_i) such that P_i is true, otherwise V

(SET x_1 ... x_k) = the set $\{x_1, ..., x_k\}$

(SCENARIO x_1 ... x_k) = the event sequence x_1 ... x_k; each x_i can be either a single event or a scenario

(EACH x L A) = (SCENARIO A(x_1) ... A(x_k)) where L = (LIST x_1 ... x_k)

(DO scenario state) = simulate the effects the scenario would have on the specified state

(ACTIONS-OF agent [state = PRESENT]) = the set of possible actions the agent can do at present [or in a specified state]

(DURING history event$_1$ ... event$_k$) = true iff the scenario event$_1$ ... event$_k$ occurs during history

(CURRENT event) = history of the current incarnation of a generic event such as TRICK

(CHANGE P_1 P_2 state) = altered state produced by making P_1 false and P_2 true

(AFTER event) = state following a specified event

2.2. Example concept definitions

An essential feature of this representation is its *extensibility.* A mahps must be able to assimilate new concepts defined by the expert in terms of existing ones. The following definitions of concepts used in later examples illustrate the representation.

```
(DE TRICK NIL
    (SCENARIO (EACH P (ORDER-OF-PLAY)
                      (PLAY-CARD P))
              (TAKE-TRICK (TRICK-WINNER))))

(DE PLAY-CARD (P)
    (PLAY P (SOME C (LEGAL-CARDS-OF P))))

(DE CARD-OF (P)
    (THE C CARDS
         (DURING (CURRENT TRICK)
                 (PLAY P C))))

(DE LEGAL-CARDS-OF (P)
    (SET-OF C CARDS (LEGAL P C)))

(DE LEGAL (P C)
    (AND [HOLDS P C]
         [CASE ((FOLLOWING P)
                (OR [VOID P (SUIT-LED)]
                    [= (SUIT-OF C)
                       (SUIT-LED)]))
               ((LEADING P)
                (OR [CAN-LEAD-HEARTS P]
                    [NEQ (SUIT-OF C) H]))]))
```

```
(DE TRICK-WINNER NIL
    (THE P PLAYERS
         (DURING (CURRENT TRICK)
                 (PLAY P (WINNING-CARD)))))

(DE TAKE-TRICK (P)
    (EACH C (CARDS-IN-POT) (TAKE P C)))

(DE TAKE (PLAYER CARD)
    (MOVE PLAYER CARD POT (PILE-OF PLAYER)))

(DE PLAY (PLAYER CARD)
    (MOVE PLAYER CARD (HAND-OF PLAYER)
          POT))

(DE MOVE (AGT OBJ ORIG DEST)
    (CAUSE AGT
           (CHANGE (AT OBJ ORIG)
                   (AT OBJ DEST))))

(DE CAUSE (AGENT EVENT)
    (SOME ACT (ACTIONS-OF AGENT)
          (LEAD-TO ACT EVENT)))

(DE POINTS (CARD)
    (CASE ((= CARD QS) 13)
          ((= (SUIT-OF CARD) H) 1)
          ((= CARD JD) -10)
          0))

(DE AVOID (AGENT EVENT)
    (SOME ACT (ACTIONS-OF AGENT)
          (SMALL (PR-LEAD-TO ACT EVENT))))

(DE SMALL (X) (< X 0.1))

(DE PR-LEAD-TO (EVENT1 EVENT2 [H = FUTURE])
    (PR (LEAD-TO EVENT1 EVENT2 H)))

(DE LEAD-TO (EVENT1 EVENT2 [H = FUTURE])
    (GIVEN (DURING PRESENT EVENT1)
           (DURING H EVENT2)))
```

Heuristics are concepts too. A goal like "avoid taking points" can be represented as a predicate to be satisfied by an agent:

```
(DE AVOID-TAKING-POINTS (P)
    (AVOID P (TAKE-POINTS P)))
```

2.3. Discussion

A key feature of our approach is the separation of representation from implementation. The definition of a concept encodes only enough information to specify its meaning precisely. The task of converting this definition into an effective implementation is factored out into the operationalization process. Essentially we are retaining LISP-like semantics but replacing EVAL with a collection of heuristic methods for evaluating expressions. We feel that the traditional popularity of LISP as an implementation language has obscured its potential as a powerful, concise representation of knowledge.

3. Operationallzation

This section suggests several methods for operationallzation, which we define as transforming well-defined but non-effective expressions to make them effectively executable. The expressions used in the examples in this section use the representation described in Section 2. An expression may be non-effective in any of several ways:

- It may depend on non-observable information that can be deduced, or on uncertain data that can be predicted only probabilistically.

- It may depend on events that have not yet occurred but can be reasoned about.

- It may be defined intensionally over an infinite or combinatorially large domain and hence infeasible to compute systematically.

- It may be defined in terms of a function which can only be computed by special-case reasoning because it has no known general formula.

- It may be too complex to evaluate exactly but may still be amenable to approximation.

Operationallzation seeks to resolve such difficulties by combining appropriate methods: means-ends rules, special-case reasoning, simplifying assumptions, search, etc. It involves two important subproblems:

1. *Mapping expressions to methods:* Match an expression to a suitable method for operationalizing it, and decide how to instantiate the premises of the method in terms of the expression.

2. *Representation of AI methods:* Represent inference methods in a form that can readily apply to novel problems. This requires formalization of techniques which, although widely used, have not previously been characterized with sufficient precision to permit automatic application to a wide range of tasks.

Given an Internal representation of what "avoid taking points" means, the mahps must *operationalize* that concept, i.e., figure out <u>how</u> to avoid taking points.

3.1. Goal-specific operationalization rules

The simplest way is to use some method provided by the expert, such as

To avoid taking points, play a low card.

This method can be represented by a means-end rule that essentially ignores the underlying conceptual structure of "avoid taking points" and simply treats it as a goal symbol. Such rules enable user-provided domain-specific operationalization of concepts.

3.2. Systematic evaluation

Systematic evaluation can be used as an operationalization technique. In this approach, concepts *ere* treated as functions, and expressions are systematically *expanded* until known quantities are reached. Functions are applied to arguments by the rule

To operationalize an expression (F e_1 ... e_n) where F has no known operationalization, substitute e_1 ... e_n for the corresponding lambda-variables in the functional definition of F.

Thus

```
AVOID-TAKING-POINTS

= (AVOID ME (TAKE-POINTS ME))

= (SOME ACT (ACTIONS-OF ME)
        (SMALL (PR-LEAD-TO ACT
                           (TAKE-POINTS ME))))
```

The subexpression

```
(PR-LEAD-TO ACT (TAKE-POINTS ME))
```

can be expanded to

```
(PR (DURING FUTURE (TAKE-POINTS ME))
    (DURING PRESENT ACT))

= (GIVEN (DURING PRESENT ACT)
         (PR (DURING FUTURE (TAKE-POINTS ME))))
```

Systematic evaluation of this expression is precluded by the lack of a general definition that can be substituted for PR. In general, this approach encounters difficulties when there is inadequate knowledge to evaluate an expression, or when systematic evaluation is computationally exorbitant. Heuristics proposed elsewhere [4] for dealing with such difficulties apply special-case knowledge about partially-defined functions or introduce simplifying assumptions that permit approximate solutions.

3.3. General operationalization methods

(This section uses fault tree analysis as an example of a general operationalization method and shows how it can be used to operationalize "avoid taking points." The purpose of this exercise is to suggest the feasibility of performing such reasoning automatically by applying simple search and manipulation operations to a knowledge base of user-defined concepts.)

A reasonable way to operationalize "avoid taking points" is to apply knowledge about how to avoid things in general. For instance, one method for avoiding an event is fault tree analysis: figure out what conditions can cause the event, and prevent them. Analysis of the game rules shows that taking points only occurs by winning a trick in which points have been played. Fault tree analysis might suggest trying to lose the current trick if it is likely to have points.*

Note what reasoning is required to do this sort of Operationalization. The idea of using fault tree analysis can be suggested by instantiating a rule like

If your goal is (operationalize (avoid x))
then find a necessary condition for x and make its negation a goal.

The analysis itself might be performed as follows. (This description omits numerous fruitless branches taken along the way; the intermediate steps of the derivation given below require some search to find.) First the definition of TAKE-POINTS is expanded in terms of TAKE:

```
(TAKE-POINTS ME)

= (TAKE ME (SOME X CARDS (HAS-POINTS X)))
```

Next the mahps looks for EVENT concepts in which TAKE occurs. (This simply requires the ability to iterate over the set of defined concepts.) The only such concept is

```
(DE TAKE-TRICK (P)
    (EACH C (CARDS-IN-POT) (TAKE P C)))
```

Thus TAKE-TRICK Is the only way the mahps knows about of taking a card. In particular,
(TAKE ME (SOME X CARDS (HAS-POINTS X») implies

```
(AND [TAKE-TRICK ME)
     [EXISTS C (CARDS-IN-POT)
               (HAS-POINTS CARO)))
```

---

*A sophisticated player will save low cards to use during the latter part of the game, when the risk of taking points normally increases. It would be rather difficult for a mahps to infer this subtle tactic from the advice "avoid taking points"; it is much more reasonable for the expert to suggest it explicitly.

The conjunct

```
(EXISTS C (CARDS-IN-POT)
          (HAS-POINTS CARD))
```

in this expression is the definition of TRICK-HAS-POINTS. Similarly, the only known case of TAKE-TRICK occurs in

```
(DE TRICK NIL
    (SCENARIO (EACH P (ORDER-OF-PLAY)
                      (PLAY-CARD P))
              (TAKE-TRICK (TRICK-WINNER))))
```

so (TAKE-TRICK ME) implies (= ME (TRICK-WINNER)) In short, (TAKE-POINTS ME) occurs iff

```
(AND [= ME (TRICK-WINNER)]
     [TRICK-HAS-POINTS])
```

To avoid (TAKE-POINTS ME), the mahps can add to its goals the proposition

```
(IF (TRICK-HAS-POINTS)
    (NOT (= ME (TRICK-WINNER))))
```

Each step of this derivation can be accounted for by a simple operation, such as lambda-substitution or unification matching. It appears quite feasible to automate this sort of analysis as, say, a set of rules for when to perform each operation.

3.4. Weak methods

(This section uses heuristic search as an example of a standard AI "weak method" [5] and applies it to the operationalization of "avoid taking points." It is argued that the reasoning involved in so doing might feasibly be performed mechanically. Some of the problems involved in applying a general method to a given problem are discussed, and some techniques for solving them are proposed.)

AI provides several weak methods applicable to the operationalization problem. One such method is heuristic search (hereafter abbreviated "HSM"), which can be used to find a sequence of elements satisfying a given property [5]. Figure 1 shows (in quasi-SAIL syntax) a generic "procedural" for HSM which yields an effective search procedure when the underlined components are instantiated by the components of a particular search problem. Figure 2 gives a schematic description of these components. For example, it specifies "operator" as the type consisting of all functions from a node to a node.

Let us see how the mahps might operationalize "avoid taking points" in terms of heuristic search. Suppose it has encountered difficulties in operationalizing

```
(SMALL (PR-LEAD-TO ACT (TAKE-POINTS ME)))
```

and applies

> If you're having trouble operationalizing a predicate P, find a non-trivial predicate Q that implies P, and operationalize Q as an approximation for P.

with P = (SMALL (PR x)) and Q = (NOT x). The mahps is now left with the problem of operationalizing

```
(NOT (LEAD-TO ACT (TAKE-POINTS ME)))
```

```
- (NOT (GIVEN (DURING PRESENT ACT)
               (DURING FUTURE
                       (TAKE-POINTS ME))))

- (NOT (EXISTS H (ALL FUTURE)
                 (DURING H ACT TAKE-POINTS ME)))
```

```
node sequence procedure HSM-search =
begin "search"
node n, n';
operator op;
search-space:=[inode];
repeat
if   (n:=select-node(search-space)) = NIL
then return failure
! search-space exhausted
else if prune-test(n)
     then remove n from search-space
     ! prune nodes not worth considering
     else begin "consider node"
          if   (op:=select-op(n)) = NIL
          then remove n from search-space
          ! no operators left to apply to n
          else begin "expand node"
               n':=op(n);
               Link(n,n');
               if   solution-test(n')
               then return Path-to(n');*
               ! (inode, ..., n') is solution
               if   retain-test(n')
               ! prune if not worth retaining
               then put n' in search-space;
               end "expand node";
          end "consider node";
end "search";
```

Figure 1: Procedural for heuristic search

*Path-to(n') finds the path (inode, ..., n') by tracing back along the links established by Link(n,n').

To decide whether such a scenario H exists, the mahps can use heuristic search. This method might be suggested by partial-matching the sub-expression

```
(EXISTS H (ALL FUTURE)
          (DURING H ACT (TAKE-POINTS ME)))
```

HSM	type	METHOD
problem	type	PROBLEM
	def	(FIND S (SEQUENCES-OF node) (ADMISSIBLE S))*
node	type	TYPE
search-space	type	node set
inode	type	node
operator	type	TYPE
	def	function with arg: node, result: node
operators-on	type	function with arg: node, result: operator set
test	type	TYPE
	def	predicate with arg: node
solution-test	type	test
prune-test	type	test
retain-test	type	test
select-node	type	generator** from search-space
select-op	type	generator from operators-on(node) with arg: node
solution	type	node sequence
	def	(HSM-search)

Figure 2: Scheme for heuristic search

*Defined by

```
(DE ADMISSIBLE (S)
    (AND [= S1 inode]
         [solution-test S(LENGTH S)]
         [FORALL I (BETWEEN 2 (LENGTH S))
                 (EXISTS OP (operators-on S(I-1))
                         (= (OP S(I-1)) SI))]))
```

**A *generator* is a procedure that returns an element of its choice set each time it is called. The choice set may be expanded by other processes between calls to the generator.

against the HSM problem description

```
(FIND S (SEQUENCES-OF node) (ADMISSIBLE S)
```

The facets of HSM can then be instantiated by identifying corresponding residuals in the partial match [1], using

> To operationalize an expression E in terms of a method M, partial-match E against M's problem description and instantiate the components of M with the corresponding components of the expression.

For instance, the correspondence between (ALL FUTURE) and (SEQUENCES-OF node) suggests using game states as nodes in the search, since each future is a sequence of game states. However, additional information, not explicit in the expression, is needed to formulate an effective search. For instance, what are the operators?

The answer to this question is not explicit in the expression, but the HSM schema gives a clue: it specifies the operators as functions from a node to a node, i.e., from a state to a state. By examining the logical types of the concepts it knows about, the mahps can find that actions are functions from a state to an event. By applying the rule

> To view a concept c as an entity of type X, find a known function F whose domain contains c and whose range matches X, and view c as (F c).

the mahps can view any action (A x_1 ... x_k) as the composite function (AFTER (A x_1 ... x_k)) from a before state to an after state.

Next the mahps can decide which actions should be used as operators in the search. An adequate set of operators in this case is given by (PLAY (PLAYER) C) for C in (LEGAL-CARDS (PLAYER)), where (PLAYER) denotes the player whose turn it is to play in the current state. A more efficient search might use abstract operators like

```
(OVERPLAY (PLAYER) ME)

= (PLAY (PLAYER)
        (SOME C CARDS
              (AND [HIGHER C (CARD-OF ME)]
                   [= (SUIT-OF C)
                      (SUIT-LED)]))))


(SLUFF-POINTS (PLAYER))

= (PLAY (PLAYER)
        (SOME C CARDS
              (AND [HAS-POINTS C]
                   [NEQ (SUIT-OF C)
                        (SUIT-LED)]))))
```

as suggested by

> To reduce a search space, suppress operator details that are irrelevant to the search criterion, and use the resulting abstractions in place of the original operators.

Such abstract operators have been used to increase planning efficiency in problem-solving systems [6, 7\

Other information absent in the expression includes the functions select-node and select-op. The simplest solution to this problem is to use random selection functions. However, this would result in an inefficient search. A better choice can be made by applying

> When selecting nodes and operators in a search, select first those most likely to satisfy the search criterion.

Thus when assigning successive values to the non-deterministic expression (SOME C (LEGAL-CARDS P», the mahps should choose first those cards most likely to satisfy (TAKE-POINTS ME), namely cards that underplay Its own choice or that break suit. This ordering preference can be solicited from the expert by or derived by applying fault tree analysis to the search criterion. If (TAKE-POINTS ME) occurs in some plausible scenario (i.e., a scenario consistent with ME's knowledge about legal play and the distribution of unplayed cards), this would be detected early in the search. If it doesn't, however, the whole combinatorial space would be searched. This could be avoided in a couple of ways. First, the mahps might simply prune branches of the search considered unlikely to lead to (TAKE-POINTS ME), using

> **If a search node is unlikely to lead to a solution, prune it.**

Second, it could apply the general rule

> **If you search for something and look at the most likely candidates without finding it, its probability of being there is small.**

Thus an incomplete, unsuccessful heuristic search for

```
(SOME H (ALL FUTURE)
        (DURING H ACT (TAKE-POINTS ME)))
```

could itself serve as evidence for the hypothesis (SMALL (PR-LEAD-TO ACT (TAKE-POINTS ME))).

3.5. Dynamic operationalizition

(This section illustrates how information arising from an actual task situation can be used to simplify the problem of operationalizing a generally applicable concept or heuristic. A hypothetical game situation is described, and a line of reasoning a human player might use in that situation is presented. The problem of automatically generating a similar line of reasoning is examined, and some general heuristics useful in solving it are proposed.)

Let's see how a mahps might operationalize "avoid taking points" in the context of a specific game situation such as the following: The order of play is Joe, then ME (the machine), then Ann. Joe has just led the 15th trick of the round with the King of diamonds. The machine has two diamonds (the 10 and the Ace) and knows that the only outstanding diamond Is the Jack, and that the only point card still out is the Queen of spades. Clearly it behooves the machine to compare carefully the consequences of playing the 10 or the Ace.

A person in this situation might reason as follows: "I can certainly avoid taking points by playing the 10, since then Joe will take the trick. If I play the Ace and Ann has the Jack, shell have to play it, so I won't take points. But if she doesn't have the Jack and she does have the Queen of spades, she might stick me with it. Otherwise I'm safe."

Mow might a similar line of reasoning be generated automatically? To start with, the mahps expands the definition of AVOID-TAKING-POINTS to

```
(SOME ACT (ACTIONS-OF ME)
          (SMALL (PR-LEAD-TO ACT
                             (TAKE-POINTS ME))))
```

This generates the goal of evaluating (SMALL (PR-LEAD-TO ACT (TAKE-POINTS ME))) for each ACT in (ACTIONS-OF ME). In the situation at hand, (ACTIONS-OF ME) consists of (PLAY ME 10D) and (PLAY ME AD). It is easy to evaluate (PR-LEAD-TO (PLAY ME 10D) (TAKE-POINTS ME)) as zero by applying fault tree analysis (3.3): (NEQ 10D (WINNING-CARD)) follows from (LOWER 10D KD) and the definition

```
(DE WINNING-CARD NIL
    (HIGHEST (SET-OF C (CARDS-IN-POT)
                       (= (SUIT-OF C)
                          (SUIT-LED)))))
```

It is trickier to evaluate (PR-LEAD-TO (PLAY ME AD) (TAKE-POINTS ME)). Expanding the definition of PR-LEAD-TO and substituting (CURRENT trick) for FUTURE yields

```
(PR (DURING (CURRENT TRICK)
            (TAKE-POINTS ME))
    (DURING PRESENT (PLAY ME AD)))
```

```
= (PR (DURING (SCENARIO (PLAY JOE KD)
                        (PLAY ME AD)
                        (PLAY ANN
                              (SOME C
                                    (LEGAL-CARDS ANN)))
                        (TAKE-TRICK (TRICK-WINNER)))
                      (TAKE-POINTS ME)))
```

Consider the sub-expression (TAKE-POINTS ME) in this expression. (TAKE-POINTS ME) occurs iff (AND (- (TRICK-WINNER) ME) (TRICK-HAS-POINTS)). (PLAY ME AD) implies (- (TRICK-WINNER) ME) since no higher card can be played, so (TAKE-POINTS ME) reduces to (TRICK-HAS-POINTS) in the case at hand. Furthermore, since (CARDS-IN-TRICK) - (SET KD AD (CARD-OF ANN)), (TRICK-HAS-POINTS) reduces to (PLAY ANN (SOME C CARDS (HAS-POINTS C)))). This expression Instantiates the definition

```
(DE UNLOAD-POINTS NIL (P)
      (PLAY P
            (SOME C CARDS
                  (AND [HAS-POINTS C]
                       [NEQ C (WINNING-CAR0)])))))
```

since (- (TRICK-WINNER) ME) precludes (PLAY ANN (WINNING-CARD)). Thus (TAKE-POINTS ME) reduces to (UNLOAD-POINTS ANN).

Now consider the sub-expression (SCENARIO ~). Evaluating it requires determining (LEGAL-CARDS ANN). Since (NOT (LEADING ANN)), (LEGAL-CARDS ANN) simplifies to

```
(SET-OF C (HAND-OF ANN)
          (OR [VOID ANN DIAMONDS]
              [- (SUIT-OF C) DIAMONDS)))
```

Since JD is the only outstanding *diamond,* (- (SUIT-OF C) DIAMONDS) reduces to (- C JD) and (VOID ANN DIAMONDS) reduces to (NOT (HOLDS ANN JD)). Thus (LEGAL-CARDS ANN) reduces to (IF (HOLDS ANN JD) (SET JD) (HAND-OF ANN)).

The resulting expression

```
(PR
 (DURING (SCENARIO (PLAY JOE KD)
                   (PLAY ME AD)
                   (PLAY ANN
                         (IF (HOLDS ANN JD)
                             JD
                             (SOME C (HAND-OF ANN))))
                   (TAKE-TRICK ME))
         (UNLOAD-POINTS ANN)))
```

can be simplified to

```
(PR (DURING (PLAY ANN
                  (IF (HOLDS ANN JD)
                      JO
                      (SOME C (HAND-OF ANN))))
            (UNLOAD-POINTS ANN)))
```

since (UNLOAD-POINTS ANN) can only occur as part of (PLAY ANN ...). The IF construct can be factored out of this expression, yielding

```
(IF (HOLDS ANN JD)
    (PR (DURING (PLAY ANN JD)
                (UNLOAD-POINTS ANN))
    (PR (DURING (PLAY ANN
                      (SOME C (HANO-OF ANN)))
                (UNLOAD-POINTS ANN)))))
```

The falsity of

```
(DURING (PLAY ANN JO)
        (UNLOAD-POINTS ANN))
```

follows from (- (POINTS JD) -10). We still have the problem of evaluating

```
(PR (DURING (PLAY ANN (SOME C (HANO-OF ANN)))
            (UNLOAD-POINTS ANN)))
```

This expression can be simplified by worst-case analysis using the rule

```
(PR (DURING (f ... (SOME x S)...) bad-event))-
(PR (EXISTS x S (DURING (f... x ...) bad-event)))
```

Applying this rule yields

```
(PR (EXISTS C (HAND-OF ANN)
              (DURING (PLAY ANN C)
                      (UNLOAD-POINTS ANN))))
```

which reduces to

```
(PR (EXISTS C (HANO-OF ANN)
              (HAS-POINTS C)))
```

```
- (PR (HOLDS-POINTS ANN))
```

In other words, worst-case analysis assumes

```
(GIVEN (HOLDS-POINTS ANN)
       (DURING (CURRENT TRICK)
               (UNLOAD-POINTS ANN)))
```

Combining the above simplifications, we have (PR-LEAD-TO (PLAY ME AD) (TAKE-POINTS ME)) - (IF (HOLDS ANN JD) 0 percent (PR (HOLDS-POINTS ANN))).

Of course, the rules of the game prevent the mahps from inspecting Ann's hand in order to evaluate (HOLDS ANN JD) and (HOLDS-POINTS ANN). This illustrates a particular type of operationalization problem: evaluation of well-defined expressions based on unobservable data. There are various ways to cope with this problem. Sometimes an expression can be evaluated Indirectly by deduction from data that art observable. For Instance, if the mahps remembers that Ann has shown void in diamonds, it can infer that (HOLDS ANN JD) is

false. Another way to cope with unobservable data is to treat it probabilistically. This approach is discussed in some detail in [4].

3.6. Mapping expressions to methods

This section has described several methods which might be used to operationalize the expression (AVOID ME (TAKE-POINTS ME)). An important issue in operationalization is mapping expressions to methods. This mapping involves not only finding a method appropriate to a given expression but also determining the correspondence between the elements of the expression and the elements of the method: the mahps must figure out how to satisfy the informational requirements of the method given the information contained in the expression or otherwise inferrable. The examples in this section involved a variety of techniques for doing this mapping. These include goal-specific means-end rules (3.1), expanding expressions in terms of lower-level concepts (3.2), instantiation of general rules (3.3), partial-matching expressions against problem descriptions associated with Known methods (3.4), viewing concepts as related concepts (3.4), and Introduction of simplifying approximations (3.5).

Intuitively, the further removed the method from the expression, the weaker the method, since it exploits less specific knowledge about the expression. Thus fault tree analysis can be expected to produce a better (more efficient) operationalization of "avoid taking points*[1] than heuristic search, since the fault tree solution operates at the level of scenarios, while the heuristic search solution operates one conceptual step lower at the level of sequences.

4. Current and Future Work

The body of previous research related to this work is too large to cite here, but is discussed elsewhere [2, 4].

The first author's dissertation (in progress) focuses on operationalization. We have operationalized several examples of Hearts advice by hand and are formulating general rules to account for the derivations. A working program generates such derivations interactively. At each step it determines which rules are applicable and applies the one ranked highest according to some simple criteria. In case of a tie, the user selects from among the best-ranked candidates. In addition we have defined a procedural for heuristic search and are working towards a formalization of fault tree analysis. The representation used in this paper has been implemented, including most of the meta-functions described in Section 2.1. An interpreter capable of executing this representation has been implemented in LISP.

Acknowledgments

This work has benefited from discussions with our colleagues at the Rand Corporation and Carnegie-Mellon University. Stan Rosenscheln was especially helpful In clarifying many knowledge representation issues.

References

[1] Hayes-Roth, F. "The Role of Partial and Best Matches in Knowledge Systems." In D. A. Waterman and F. Hayes-Roth (eds). *Pattern-Directed Inference Systems.* Academic Press: New York, 1978, pp. 557-574.

[2] Hayes-Roth, F., Klahr, P., Burge, J., and Mostow, D.I "Machine Methods for Acquiring, Learning, and Applying Knowledge.- Technical Report P-6241, The Rand Corporation, Santa Monica, Calif., 1978.

[3] McCarthy, J., et al. *LISP 1.5 Programmer's Manual* MIT Press: Cambridge, Mass., 1963.

[4] Mostow, D.J., and Hayes-Roth, F. "Machine-Aided Heuristic Programming: A Paradigm for Knowledge Engineering.[1]* Technical Report N-1007, The Rand Corporation, Santa Monica, Calif., 1979.

[5] Newell, A. "Heuristic Programming: Ill-Structured Problems." In J. Aronofsky (ed.), *Progress in Operations* Research. Wiley: New York, 1969, pp. 363-414.

[6] Newell, A., and Simon, H.A. *Human Problem Solving.* Prentice-Hall: Englewood Cliffs, N. J., 1972.

[7] Sacerdoti, E.D. "Planning in a Hierarchy of Abstraction Spaces." *Artificial Intelligence.* 5, (1974) 115-135.