DEDUCTIVE METHODS FOR LARGE DATA BASES

Charles Kellogg, System Development Corporation, Santa Monica, Calif.
Philip Klahr, System Development Corporation, Santa Monica, Calif.
Larry Travis, University of Wisconsin, fladison, Wisconsin

ABSTRACT

The design and prototype implementation of a deductive processor for efficient extraction of implicit information from explicit data stored within a relational data-base system is described. General statements (premises or inference rules) as well as queries are expressed in a canonical form as implications. From user queries, the system constructs skeletal derivations (proof plans) through the use of a predicate connection graph, a pre-computed net structure representing possible deductive interactions among the general statements. The system incorporates techniques for rapid selection of small sets of relevant premises (by proof planning); development and elaboration of proof plans; proof plan verification; use of proof plans as a basis for determining data-base access strategies; and instantiation of plans (i.e., turning proof plans into proofs) with retrieved data-base values. Examples of the current capability of the system are illustrated.

INTRODUCTION

The deductive processor (DP) described in this paper has been designed to interface with existing and emerging relational data management systems (RDMSs). Given this orientation, we have made a sharp distinction between specific facts (n-tuples) which reside in an RDMS data base and general statements (rule-based knowledge or premises) that are directly accessible to the DP. Since the number of general statements that may be required for a practical application is likely to be large (perhaps hundreds to thousands of premises), particular attention has been paid to the development of techniques for the rapid selection of relatively small sets of premises relevant to answering a user's specific request. Premise-selection techniques are automatically invoked when deductive support is necessary to respond to a user's request; otherwise, queries "fall through" the DP and directly drive the RDMS.

This "deductive inference by exception" principle suggests that the DP be viewed as an add-on or enhancement to existing data-base searching capabilities4 Such an enhancement can result in a major increase in the power of a data management system by providing a means for extracting and deriving implicit information from data bases of explicit facts. Further, as we shall see, the DP can aid a user in evaluating the utility and/or plausibility of an inferentially obtained answer by displaying the evidence on which the answer is based.

We briefly review some of the relevant work in the field of deductive question answering, outline our approach, describe the several components of our prototype DP, and illustrate by means of two examples the current operation of the system.

APPROACH

Previous approaches to adding deductive capabilities to data management have occurred primarily in the development of question-answering systems (Simmons1[4*'5] reviews many of these). The primary aeductive methods that have been used are set-inclusion logic, e.g., CONVERSE[3] and SYNTHEX[11]: techniques based on the "resolution" principle'10, e.g., QA32 and MRPfJ^; procedural-oriented deduction, e.g., SHRDLU18; and goal-oriented backward chaining, e.g., MYCIN

The primary difference between these systems and our DP is in our use of planning. Our system creates deduction plans to guide the generation of full deductions. We believe such planning to be essential for cutting through the massive number of dead ends and irrelevant inferences which have impaired the performance of earlier systems. Planning becomes even more important for systems involving large numbers of premises. Selection of a manageably small set of possibly relevant premises can be based on such planning.

To this end we have designed and implemented a deductive processor that first builds derivation skeletons which represent possible deduction plans. Once such plans are generated, the system will attempt to instantiate and verify the plans (examine substitutions for variables in premises). We have thus separated the premise-selection process from the process of verifying the consistency of variable substitutions.

The generation of derivation (proof) plans is centered around middle-term chaining^. This process finds implication chains from assumptions to goals through the premises. Middle-term chaining combines the processes of forward chaining from the assumptions in a query and backward chaining from the goals in a query. (In the case of no query assumptions, middle-term chaining defaults to backward chaining.) As chaining proceeds in the two directions, intersections are performed on the derived sets. When a non-empty intersection occurs, the system has found an implication chain from an assumption to a goal. The resulting chain is passed on to the proof plan generator, which extracts the premises whose occurrences are involved in the chain. Subproblems may result, requiring further deduction or data-base search. The examples presented below will illustrate these processes.

The chaining process does not operate on the premises themselves but on a net structure called tne predicate connection graph (PCG). This graph is abstracted from the premises. When a premise is introduced into the system, the implication connections existing among the predicate occurrences in the premise are encoded into the PCG. Further, the deductive interactions (i.e., unifications10) between predicate occurrences in the new premise and predicate occurrences in existing premises are pre-computed and encoded into the PCG. The variable substitutions required to effect the unifications are stored elsewhere, for latter use by the proof plan verifier. Thus, the PCG contains information on the implications within premises and the deductive interactions among the premises. During the generation of middle-term chains and proof plans, the system is aware of the existence of unifications among the premises, but it does not need to generate the unifications nor does it need to examine and combine the variable substitutions associated with the interacting unifications. The former is done by a pre-processor, while the latter is done by the verifier after proof planning.

Although some connection graphs used in theorem-proving systems also contain information on the unifications among general assertions (resolution clauses in these systems), they are not used as a planning tool as is the PCG. The PCG most resembles Sickel's clause interconnectivity graph"!3 in that both graphs represent the initial deductive search space and are not changed in the course of constructing deductions. Other graph procedures'''^involve adding nodes to graphs as deductions are formed. More detail on the PCG is given in Klahr^.

REPRESENTATION OF INFORMATION

The basic representation for general assertions (premises) is the primitive conditional'?. This form is a normalized first-order predicate-calculus implication statement. The antecedent of the implication contains the assumptions (conditions) of the assertion; the consequent contains the goals of the assertion. Conjunctions, disjunctions, and negations can occur on either side of the implication. Each assumption and goal is a predicate occurrence consisting of a predicate (relation) and its argument terms (i.e., variables, constants, or functions).

The primitive conditional was chosen because general assertions are usually formulated in the form of "if....then..." implications. Users can easily express and understand general assertions in this form and can easily control and understand proofs involving them. Further, this form facilitates system discovery of deductive implication chains.

Variables and constants occurring in premises and queries may be categorized into specific domain classes. For example, a variable "x" might be specified as being a LABORATORY and the constant "Joe" as being a SCIENTIST. In attempting to match argument strings involving these terms, the system will not allow the substitution of Joe for x because they belong to different domains. The use of such semantic information eliminates certain deductive interactions among the premises and thus reduces the search space of possible deductions [5»8].

Semantic information in the form of user-supplied advice can also be given to the system. Advice most typically involves recommendations on the use of particular premises or predicates in finding deductions. For advised premises, the system will try using them whenever possible in the course of constructing a proof. For advised predicates, the system will try chaining through occurrences of them (in premises). In the case of negative advice, specified premises and predicates are avoided in proofs.

Advice may be given for a particular input query or stored in a permanent advice file which the system accesses for each query. Advice statements are in the form of condition-recommendation rules similar to the meta-rules used in MYCIN'. The conditions contain information about predicates, constants, and domain classes that may occur in query assumptions and goals. The conditions are matched against the input query and, if they are satisfied, the associated recommendations about the use of certain premises and predicates are activated. Internally, advice is transformed into premise and predicate alert lists (as well as negative alert lists for negative advice), which are accessed in the chaining and proof-planning processes.

In addition to the information used by the deductive processor, there is also a file of specific facts used by a data management system. This latter system searches for and retrieves specific facts needed to resolve subproblems resulting from premises. For our experiments with the prototype deductive processor, we have written a small LISP relational data-base management system. Facts are stored relationally as n-tuples associated with a predicate (relation) name. When a particular predicate occurrence becomes a subproblem, the system has three alternative methods for resolving it; the decision is based on how the user defined the various predicates known to the system. If a predicate is defined computationally by a procedure, the procedure is executed to determine the predicate's truth value. If a predicate is specified by the user as defined primarily by its data-base values, the unresolved predicate is left for data-base search. Otherwise, an unresolved predicate occurrence is given further deductive support through the premises. (Such predicate classification is currently mutually exclusive but need not be. An alternative control structure could try several methods for resolving each subgoal.) The examples below will show the interface between the deductive processor and the data management system, as well as examples of procedurally defined predicates.

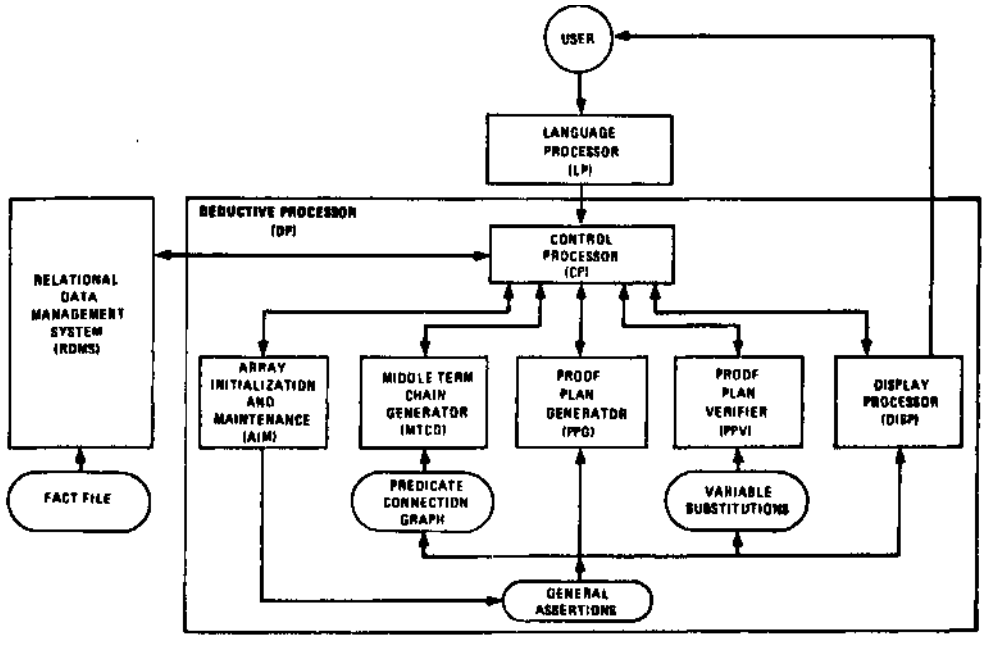

Figure 1. Deductive Processor Components

SYSTEM COMPONENTS

Figure 1 displays the various components of the deductive processor as well as its position in a deductive data management system. The language processor is currently not a part of our initial prototype environment but will be incorporated at a later date. The control processor shown in Figure 1 currently accepts premises and queries in primitive conditional form as well as user advice and commands. It accesses and coordinates the several system components described below.

Array Initialization and Maintenance

Information abstracted from the premises is segmented into seven internal arrays. This segmentation contributes to good system structuring and increases processing efficiency. Each predicate occurrence is assigned a unique integer index. Information about a particular predicate occurrence is obtained from the array containing the kind of information needed by indexing into the array with the integer associated with the occurrence. The seven arrays are:

Premise Array: Each entry represents a premise and contains a list of the occurrences (i.e., occurrence indices) in the premise, the plausibility of the premise, and the premise itself, both symbolic (primitive conditional form) and English, for purposes of display.

Predicate Array: This array contains the relations known to the system. Associated with each relation is its support indicator, i.e., the method used to resolve the relation when it occurs as a subgoal (deduce, search data base, compute).

Predicate Occurrence Array: Each entry represents a predicate occurrence and contains the following information about the occurrence: its predicate name (index into predicate array), the premise in which it occurs (index into premise array), the sign of the occurrence (positive or negative), whether the occurrence is in an antecedent or consequent of a primitive conditional, the main connective governing the occurrence (i.e., conjunction or disjunction), and the numerical position of the occurrence within the premise. The information is compactly stored in a single one-word bit vector.

Arguments Array: The argument strings of the predicate occurrences are stored in this array in a one-to-one correspondence to the positions of the occurrences in the predicate occurrence array.

Links Array: Deductive dependencies within premises are stored in this array. Basically, these dependencies derive from implication connections among predicate occurrences within premises (Klahr[5]). This array is also indexed by occurrence integers. For each occurrence, a list of the occurrences it implies is stored in the entry corresponding to the occurrence's index.

Unifications Array: Each entry contains a list of the unifications (deductive interactions) associated with the given occurrence. The unifications array and the links array comprise the predicate connection graph.

Variable-Substitutions Array: The substitution lists association with unifications are stored in

a one-to-one correspondence to the position of the unifications in the unifications array.

Middle-Term Chain Generator

Each input query is broken down (based on the logical connectives in the query) into sets of assumptions (from query antecedents) and goals (from query consequents). The predicate connection graph is used to find deductive implication chains between assumptions and goals. "Wave fronts" are expanded out of assumptions and out of goals until an intersection is found, at which point the middle-term chain is identified and extracted.

Proof Plan Generator

For each middle-term chain generated, the system extracts the premises whose occurrences are part of the chain. Any subgoals resulting from the premises are set up as requiring deductive support through the premises, data-base search, or procedural computation. Subgoals are added to a proof proposal tree, which contains proof plans as they are being formed and developed. Proof plans having no remaining deduce subgoals are then passed on to the verifier.

Proof Plan Verifier

The variable substitutions required by the unifications in a proof plan are examined for consistency. If there are no clashes, i.e., no variable taking on more than one distinct constant value, then verification is successful. If there are any remaining subgoals requiring data-base support, the data management system is called to search the file of specific facts.

Display Processor

The user has a wide variety of display options available to monitor the operation of the deductive system. In particular, he can examine middle-term chains generated, proof plans formed, subgoals, proof plan verification, data-base search requests, data-base values returned, answers, completed proofs, and premises used in proofs.

COMPUTER EXAMPLES

In Figures 2 and 3 we illustrate examples of the current operation of our initial DP prototype interfaced to a small RDMS. (Both DP and RDMS are written in LISP 1.5 and operate on an IBM 370/158 computer.)

In the first example, we illustrate the generation of short inference and search/compute plans for the question, "What ships are closer to the Kittyhawk's home port than the Kittyhawk is?" The query is first shown in English and then in the primitive conditional symbolic form that our prototype currently recognizes. The query is expressed in terms of a conjunctive goal composed of the predicates CLOSER-THAN and HOME-PORT. Constants (e.g., Kittyhawk) are specified by being enclosed in parentheses, while variables (e.g., x and y) are not. One of the query goals (HOME-PORT) is to be given data-base support, i.e., it has been characterized as defined by data-base values, while the other goal (CLOSER-THAN) is to be deduced. Since the antecedent in the query is empty, middle-term chaining defaults to backward chaining. The system back-chains from CLOSER-THAN through premise 29. The plausibility (similar to certainty factors in MYCIN[16]) of the plan in this case is simply the plausibility of the single premise used. Premise plausibilities range from 1 to 99 and are set by the user.

Two new search requests (in addition to HOME-PORT) result from premise 29, as well as a compute relation containing functional arguments. Computations for the functions and the relation are delayed until values for the variables x and y have been found in the data base (i.e., values which satisfy the search requests).

The system sends the three search requests to the RDMS, which finds two ships, the Forrestal and the Gridley, that are closer to the Kittyhawk's home port (San Diego) than the Kittyhawk is. The system then displays the proof that led to the first answer (the Forrestal). A proof using the other answer would be identical to this one except that Gridley would replace Forrestal in the proof, and the distance between the Gridley and San Diego would replace 310 (the distance between the Forrestal and San Diego). The symbols G2, G3, etc., represent nodes in the proof proposal tree and are used here for reference. G2 and G3 represent the original goals as also shown in the inference plan. G5, G6, and G7 are subgoals that resulted from premise 29, which was used to deduce G2. Thus, these three subgoals are indented below G2.

The middle-term-chaining and proof-planning processes are more evident in the example in Figure 3. The input query contains two assumptions (DAMAGED and DESTINATION) and one goal (TRANSPORT). Taurus and NY are constants; Cargo and x are variables. The query asks the system to find values for x that satisfy the query. The variable x is restricted to range over ships. (This is an example of a domain class specification for a variable. Such domain specifications could also have been used in the previous example.) In the course of developing deductions, the system will not allow values to be substituted for x that belong to domain classes other than ships.

The inference plan shown in Figure 3 has already been verified. To see the planning mechanism more clearly, we will refer to Figure 4. The first middle-term chain generated connects the DESTINATION assumption to the TRANSPORT goal via premise 23. This is shown by the unifications u_1 and u_2 in Figure 4. The predicate occurrences involving the relations AVAILABLE and OFFLOAD become subproblems. The former is to be given data-base support; the latter is deduced by a middle-term chain from the DAMAGED assumption through premises 7 and 15. The chain is shown in Figure 4 by the unifications u_3, u_4, and u_5. The

```
•WHAT SHIPS ARE CLOSER TO THE KITTVHAWK'S HOME PORT
•THAN THE KITTYHAWK IS?

QUERY ((OIMP<AND(CLOSER-THAN X (KITTYHAWK) Y)
                (HOME-PORT (KITTYHAWK) Y))))
INFERENCE PLAN:
   DEDUCE    G2 *CLOSER-THAN X KITTYHAWK Y
   SEARCH    G3 •HOME-PORT KITTYHAWK Y
PREMISES USED: (29)       PLAN PLAUSIBILITY:   99
SEARCH/COMPUTE PLAN:
   SEARCH       •SHIPS KITTYHAWK
   SEARCH       •SHIPS X
   SEARCH       •HOME-PORT KITTYHAWK Y
   COMPUTE      *GREATER-THAN (DISTANCE-BETWEEN KITTYHAWK Y) (
                 DISTANCE-BETWEEN X Y)
ENTERING DATA BASE
DATA-BASE SEARCH SUCCESSFUL

ANSWER SUMMARY —
VARIABLES:
(X Y)
ANSWERS:
(FORRESTAL SAN-DIEGO)
(GRIDLEY SAN-DIEGO)

PROOF DISPLAY:
   DEDUCED   G2 *CLOSER-THAN FORRESTAL KITTYHAWK SAN-DIEGO
   FACT      G5 ••SHIPS KITTYHAWK
   FACT      G6 ••SHIPS FORRESTAL
   COMPUTED  G7 ••GREATER-THAN 378 310
   FACT      G3 •HOME-PORT KITTYHAWK SAN-DIEGO
PREMISES USED: (29)       PROOF PLAUSIBILITY:   99
TYPE PREMISE NUMBER TO DISPLAY, OR 'ENO':
29
((ALL X79) (ALL X80) (ALL X81)
 (AND (SHIPS X79) (SHIPS X80))
  (GREATER-THAN (DISTANCE-BETWEEN X79 X81)
   (DISTANCE-BETWEEN X80 X81)))
 IMP (CLOSER-THAN X80 X79 X81>)
PLAUSIBILITY:   99
TYPE PREMISE NUMBER TO DISPLAY, OR 'END':
ENO
END DISPLAY
```

Figure 2. Deduction Involving Deduce, Data-Base Search, and Compute Predicates

```
•IF THE TAURUS WERE DAMAGED WHILE DESTINED FOR NEW
•YORK WITH A CARGO, WHAT SHIPS COULD TRANSPORT THE
•CARGO TO NEW YORK?

QUERY(((WHAT (SHIP . X))
        (AND (DAMAGED (TAURUS))
             (DESTINATION (TAURUS) (NY) CARGO))
        IMP (TRANSPORT X CARGO (NY))))
INFERENCE PLAN:
   DEDUCE     GI *TRANSPORT SHIP#X X75 NY
   ASSUME        •DESTINATION TAURUS NY X75

   DEDUCE     G3 ••OFFLOAD TAURUS X75 X72
   ASSUME        **DAMAGED TAURUS
   MID-TERM      ••RETURNS TAURUS X72

PREMISES USED: (23 7 15)       PLAN PLAUSIBILITY:   80
SEARCH/COMPUTE PLAN:
   SEARCH       -HOME-PORT TAURUS X72
   SEARCH       *CARRY TAURUS X75
   SEARCH       -AVAILABLE SHIP#X X72
ENTERING DATA BASE
DATA-BASE SEARCH SUCCESSFUL

ANSWER SUMMARY —
VARIABLES:
(X)
ANSWERS:
(PISCES)
(GEMINI)

PROOF DISPLAY:
   DEDUCED   GI *TRANSPORT PISCES 01L NY
   ASSUME       *DESTINATION TAURUS NY OIL

   DEDUCED   G3 ••OFFLOAD TAURUS OIL FREEPORT
   ASSUME       ••DAMAGED TAURUS
   MID-TERM     **RETURNS TAURUS FREEPORT

   FACT      GII***HOME-PORT TAURUS FREEPORT
   FACT      G12**«CARRY TAURUS OIL
   FACT    G4    ••AVAILABLE PISCES FREEPORT
PREMISES USED: (23 7 15)       PROOF PLAUSIBILITY:
END DISPLAY
```

Figure 3. Deduction Using Middle-Term Chaining

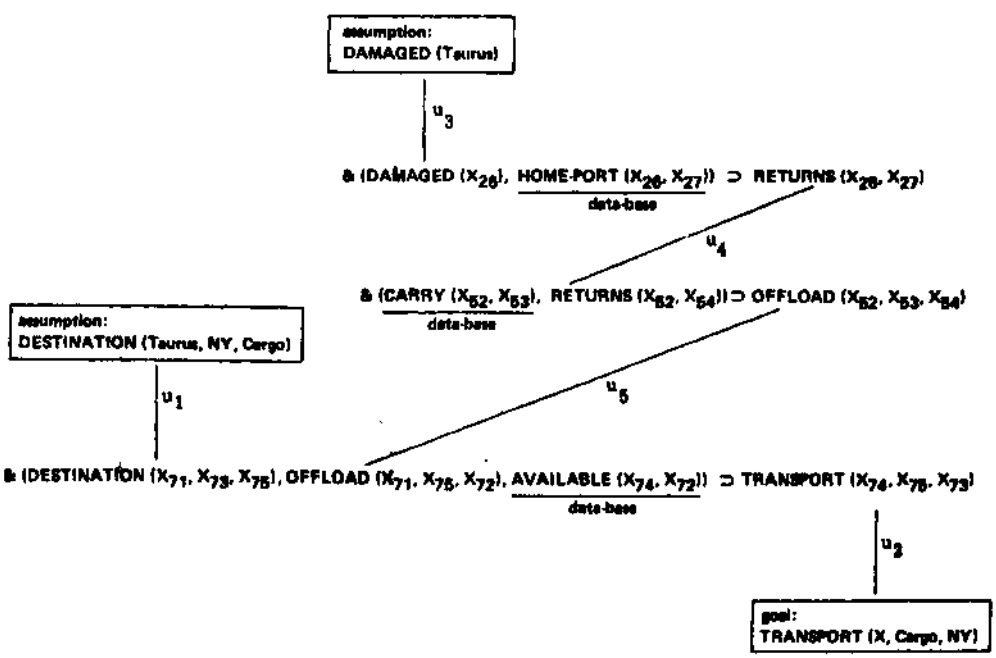

Figure 4. Proof Plan for Query in Figure 3

two new subproblems are to be given data-base support. Thus the plan generated uses three premises and contains three subproblems requiring data-base search. The plausibility of the plan is currently calculated by a fuzzy intersection (the minimum of the plausibilities of the premises involved$_{19}$).

The plan is then verified with variable substitutions inserted in the plan and in the search requests (Figure 3). Note the variable constraints in the search requests. The variable x_{72} represents the home port of Taurus; values found for this variable must be the same as those found for $x_{,2}$ in the AVAILABLE search request. The proof aisplay is given for the first answer found (the Pisces).

In Figure 4 we note that the unifications u, and u_5 were computed when these premises were first entered into the system and stored in the PCG. Also stored in the PCG were the implication connections within the premises, e.g., between DAMAGED and RETURNS, between RETURNS and OFFLOAD, and between DESTINATION and TRANSPORT. The unifications U-,, u-, and u_2 were computed after query input (because they involve predicate occurrences in the query) and serve to locate possible middle-term-chain end points. Once these end points were identified, only the PCG was used for middle-term chaining.

SUMMARY AND FUTURE PLANS

We have described a deductive system specifically designed to provide inferential capability for a data management system. From a set of general assertions, the system generates skeletal derivations or proof plans in response to given input queries. These plans are then used to trigger data-base search requests for the specific facts needed to instantiate and thus complete proof plans, turning them into proofs and answers. General information is thus used to guide and direct the proof-planning process and to identify subproblems that may be resolved by data-base search or by computation. (Or subproblems may be left open in the display of incomplete proof plans to the user, thus identifying information which cannot be found within the system but which the user may be able to supply from without.)

We are currently expanding the prototype along several different dimensions in line with our goal of eventually incorporating the deductive processor into an operational data management system and language processor environment. A number of improvements in man-machine interaction and user displays are being made in order to allow users to have more direct and flexible control of the proof-plan-generation and data-base-search processes. Additional semantic constraints oh the generation of plans will be introduced through the use of a semantic net to further restrict the range of variables, as well as through extensions to the existing semantic-advice condition-recommendation formalism. Work in these two critical areas of improved user and semantic control of deductive processes is being supplemented by additional investigations into the encoding and integration of incomplete and plausible knowledge.

ACKNOWLEDGEMENTS

The research reported here has been supported by the Advanced Research Projects Agency of the Department of Defense and is monitored by the Office of Naval Research under Contract N00014-76-C-0885.

REFERENCES

1. Davis, R., Buchanan, B., and Shortliffe, E. Production rules as a representation for a knowledge-based consultation program. Artificial Intelligence, 8, 1977, 15-45.

2. Green, C. C. Theorem proving by resolution as a basis for question-answering systems. In Machine Intelligence 4, Meltzer, B. and Michie, D. (Eds.), Edinburgh University Press, Edinburgh, 1969, 183-205.

3. Kellogg, C. H., Burger, J., Diller, T., and Fogt, K. The CONVERSE natural language data management system: current status and plans. Proceedings of the Symposium on Information Storage and Retrieval. ACM. New York. 1971. 33-46.

4. Kellogg, C, Klahr, P., and Travis, L. A deductive capability for data management. In Systems for Large Data Bases, Lockemann, P. C. and Neuhold, E. J. (Eds.), North Holland, Amsterdam, 1976, 181-196.

5. Klahr, P. The Deductive Pathfinder: creating derivation plans for inferential question-answering. Ph.D. Dissertation, Computer Science Department, University of Wisconsin, Madison, December 1975.

6. Klahr, P. Planning techniques for rule selection in deductive question-answering. In Pattern-Directed Inference Systems, Waterman, D. and Hayes-Roth, F. (Eds.), Academic Press, New York, 1977.

7. Kowalski, R. A proof procedure using connection graphs. Journal of the ACM, 22, 1975, 572-595.

8. McSkimin, J. R. The use of semantic information in deductive question-answering systems. TR-465, University of Maryland, Colleqe Park, 1976.

9. Minker, J., Fishman, D. H., and McSkimin, J. R. The Q* Algorithm—a search strategy for a deductive question-answering system. Artificial Intelligence, 4, 1973, 225-243.

10. Robinson, J. A. A machine-oriented logic based on the resolution principle. Journal of the ACM, 12, 1965, 23-41.

11. Schwarcz, R. M., Burger, J. F., and Simmons, R. F. A deductive question-answerer for natural language inference, Communications of the ACM, 13, 1970, 167-183.

12. Shostak, R. F. Refutation Graphs. Artificial Intelligence. 7. 1976, 51-64.

13. Sickel, S. A search technique for clause interconnectivity graphs. IEEE Transaction on Computers, C-25, 1976, 823-835.

14. Simmons, R. F. Answering English questions by computer: a survey. Communications of the ACM, 8, 1965, 53-69.

15. Simmons, R. F. Natural language question-answering systems: 1969. Communications of the ACM, 13, 1970, 15-30.

16. Shortliffe, E. H. Computer-Based Medical Consultations: MYCIN. American Elsevier, New York, 1976

17. Travis, L., Kellogg, C, and Klahr, P., Inferential question-answering: extending CONVERSE. SP-3679, System Development Corporation, Santa Monica, Calif., 1973.

18. Winograd, T. Understanding Natural Language. Academic Press, New York, 1972.

19. Zadeh, L. A. Fuzzy sets. Information and Control, 8, 1965, 338-353.

SPARK: A SYSTEM FOR PARALLEL REPRESENTATION OF KNOWLEDGE

Gerald A. Wilson
Computer Science Laboratory
Naval Research Laboratory
Washington, D.C. 20375

In the System for Parallel Representation of Knowledge (SPARK) the ingredient of concern is not the high-level, human-like, modeling of knowledge, but the compact, efficient, and effective internal representation and use of the knowledge. SPARK employs a knowledge base representation technique which has been shown to be as much as seven times more efficient for information retrieval than some other relational representations. At the same time, this technique, called the Parallel Representation (PAR) Technique, can also compact the knowledge base by a factor of two or more. What distinguishes SPARK from data management systems is that this efficient and effective retrieval mechanism also provides a powerful deductive inference capability.

Two types of parallelism are employed in SPARK, one achelved by data structures anc the other by parallel processing. Both are made possible by the distinction made between the "structure" and the "content" of data. In human problem solving the structure is the general concept while the content is the sets of items which, when combined with the structure, make one or more instances of that concept. Thus "transporting A from X to Y" is a concept while "carrying the block from the floor to the table" Is an instance of that concept with the content: carrying; block; floor; and table. For the basic constructs (the individual facts and inference rules of the knowledge base) PAR employs templates to represent the structure and sets to specify the content. For example, the collection of facts about objects supported by the table would be given by:

```
((R,X,Y){[supports]/R,[table]/X,
        [block,cone,lump,hammer] /Y»
```

where (R,X,Y) is the template specifying the concept of a binary relation with two independent arguments. The sets associated with R, X, and Y can be used to form specific instances by appropriate substitutions, in this case simply ordered cross products of the sets. To index the knowledge base PAR employs meta-templates and super-sets in a corresponding manner. The indexing structure proovides a compact form which facilitates efficient search and retrieval. Thus the representation Is parallel because any single symbol appearing in a PAR structure can represent an unbounded number of instances of that symbol in the knowledge base.

The second type of parallelism is multi-processing made possible by the meta-templates and super-sets of the index structure. The meta-templates are cannonlcal B-trees which partition the knowledge base into disjoint collections of data. When a query pattern matches n meta-tempates, n independent processes may be created to complete the retrieval match, thus performing many retrievals in parallel.

SPARK, with the PAR Technique, is not posed as a panacea for all knowledge base management problems. Several constraints were assumed in the development:

(1) Very large knowledge bases (more than 10**12 bits) are to be commonly employed.

(2) There is a significant degree of interrelationship among the elements of the knowledge base. If the knowledge base is viewed as a collection of n-tuples, then any distinct argument of a tuple has a high probability of appearing in multiple tuples.

(3) Search and retrieval are the preponderance of knowledge base operations.

(A) Search and retrieval may be equally likely for any combination of arguments, i.e. a query n-tuple may have instantiated any combination of argument positions, the remaining positions being left free.

(5) The representation must allow semantic (domain specific) constraints to be used In the search and retrieval process.

(6) Sets should be treated as sets.

(7) The representation should facilitate the use of inference.

These constraints appear to be quite general and representative of a large variety of realistic knowledge bases.

The Parallel Representation Technique employed in the SPARK system is posed as an approach to intelligent knowledge base management (i.e. management employing inference) for very large knowledge bases. Preliminary results from a simplified model and analysis of the technique indicate the potential for significant storage and search processing savings over some other relational representations. It is significant to note that the space savings due to the knowledge base compression ability of PAR do not cause an increase in the effort required to search the knowledge base on the average. The search mechanism can accomplish its task more efficiently in fact. This is due primarily to the elimination of any conflict between the manner in which the information is stored and the manner in which it is utilized by the search mechanism. Because the PAR Technique is intended as an internal representation of information it can be adapted to many different high level external representations.

The implementation and testing of SPARK is continuing at the Naval Research Laboratory. Once the system is fully operational experimentation will be made with large practical knowledge bases to further determine the strenghts and weaknesses of SPARK and the PAR Technique.

APPROXIMATE RESPONSES FROM A DATA BASE QUERY SYSTEM: AN APPLICATION OF INFERENCING IN NATURAL, LANGUAGE

Aravind K. Joshi and S. Jerrold Kaplan
University of Pennsylvania
Department of Computer and Information Science
The Moore School of Electrical Engineering
Philadelphia, Pennsylvania 19104

Ronald M. Lee
Department of Decision Sciences
The Wharton School
University of Pennsylvania
Philadelphia, Pennsylvania 19104

Our' goal is the development and application of various techniques for generating approximate responses to data base queries. An "approximate response" is a response other than a direct answer to the question. Approximate responses are frequently referred to by linguists as "indirect answers" or "replies" (e.g. in BS76). What is approximate is not so much the response as the relationship between the response and the initial query. Our approach is to regard an interaction between a user and a data base as a discourse, having the properties and constraints normally associated with human dialog. (Conversational Postulates of Grice (G67) are examples of such constraints.) Many of the conventions of human dialog can be implemented through approximate responses which, for instance, 1) aid a user in formulating a suitable alternative query when the precise response to the initial query would be uninteresting or useless; 2) inform a user about the structure or content of the data base when the user is unfamiliar with its complexities; and 3) summarize at an appropriate level, eliminating unnecessary detail.

Natural language (NL) query systems *are* of benefit to users who are only partially familiar with the structure and/or content of the underlying data base. Such "naive" users are typically hampered by their lack of knowledge in formulating a query which will retrieve the. desired information. We believe that NL can do more than simply provide the user with a convenient, higher-level replacement for a formalized query syntax. NL questions frequently embed information about the user's understanding of the structure of the data. This information can be exploited to inform and guide the user in the use of the data base.

Of particular interest to us is the key role that shared knowledge between conversants plays in the effectiveness of human dialog. As observed in (CH75), dialog tends to proceed with statements which offer a specific piece of 'new' information to the conversation which is differentiated from information considered as 'given' or already known to the other party. Breaches of this 'Given-New Contract' can point to the need for additional background information to be supplied in order for communication to be effective. We believe that this observation can be effectively utilized within the context of queries to a data base system. Our approach here is to pay special attention to the 'given' information contained in the user's questions in the form of presuppositions. If these turn out to be false, we interpret this as a signal that the user misunderstands some aspect of the data base's structure or content and is in need of additional clarification. An approximate response explicitly contradicting the failed presupposition and perhaps suggesting an alternative is appropriate, as it is in human dialog. Such a response serves to correct the users mis impressions and provide suggestions for alternatives, hopefully relevant and useful ones.

A presupposition of a sentence S can be broadly defined as any assertion that must be true in order for S to be meaningful. In the case of questions, the presupposition must be true for a direct answer to be meaningful.

Presuppositions come in many forms. There are presuppositions which are primarily syntactic (JW77). Others deal with implied restrictions on the size, or a claim about the completeness of the answer set (BS76). Of particular interest in a data base context *are* those presuppositions of an NL question which are implied by a corresponding formal query to a given data-base structure. We have observed that each stage in the execution of a formal query, except for the final one, has an interpretation as a presupposition of the NL question. If a particular stage of execution returns a null set, the corresponding presupposition has failed and can be explicitly contradicted, rather than returning an obviously uninformative or misleading null response.

Consider the query
"WHICH LINGUISTICS MAJORS GOT A GRADE OF B OR BETTER IN CS500?"
Assuming a suitable structure for the data (see Figure one), a corresponding formal query might perform the following operations: 1) Find the set of students and restrict it to linguistics majors; 2) Find the set of courses and restrict it to CS500; 3) Find the class list (set of students) associated with the result of 2; M) Restrict the class list of 3 to those with grades ^ B; and 5) Intersect 4 with 1 to produce the response. An empty set at each stage could be used to produce the following approximate responses contradicting the failed presuppositions: 1) There are no linguistics majors; 2) There is no course "CS500"; 3) No students were enrolled in CS500; and 4) No students received a grade of B or better in CS500. A failure in the final stage leads to the direct answer NONE. It is worth noting that different data structures will reveal different presuppositions. For instance, a different data base might produce the response "No linguistics majors took CS500."

* This work is partially supported by NSr Grant MCS 76-19466.

We wish to thank Peter Buneman, Rob Gerritsen, and Ivan Sag for many fruitful discussions.

Another type of approximate response deals with the generation of a response to a substitute query. For instance, "Is Venus the fourth planet?" may be responded to by "No, it is the second planet."(see (L77) for similar examples). A determination of the focus and topic of the question can be used to generate an appropriate alternative, as opposed to (say) "No, Mars is the fourth planet." Syntactic and contextual cues are under investigation to determine the topic and focus in the face of partial information. Careful construction of the formal query can provide a relevant piece of alternative information forfree by selecting the most appropriate access path to the desired information.

An important convention of human conversation is that no participant monopolize the discourse, so that control can be shared. One implication of this is that all responses given in a conversational mode must be short. Thus where the system would otherwise respond with a lengthy list, we would prefer to be able to return a non-enumerative, or "intensional" response". Lengthy response sets could be summarized, or defined by a characteristic or attribute. For instance, the question "Which employees engage in profit sharing?" may be answered by listing the extension of a set containing (perhaps) 10,000 names, or by the intensional response "All vice-presidents." The summary might be computed from the data or infered from the data base schema, and can be used to avoid unnecessary and distracting detail. In these cases, the response may implicitly incorporate the restrictions of the question. For instance, a response to "Which students were invited to the party?" of "The girls living in West Philadelphia." clearly implies that only those girls who are students were invited (KH 73).

Conversations also allow hypothetical questions, or questions about the structure of the world (in our case, the data base). Questions such as "Can supervisors profit share?" may be answered affirmatively by the contents of the data base (finding an instance), or negatively by noting that the data base structure precludes such a possibility. If neither of these alternatives are successful, an approximate response of "maybe", or "I don't know" may be returned, (since many constraints to the data base may be imposed by the logic of the updating programs or organizational procedures).

Finally, conversations admit answers of a statistically approximate nature. "What percentage of welfare recipients are single mothers?" may be sufficiently answered by "About 80%". This concept is of use in the execution of queries on very large data bases, when precise responses are both unnecessary and expensive. If the user is willing to accept an approximate response which is within a given confidence level, this can frequently be computed for a fraction of the cost of a complete one.

Existing data base systems could be described as "stonewalling", giving only limited, precise answers, which inhibited browsing and query formulation, Approximate responses, as they are used in human dialog, can significantly increase the usefulness and convenience of data base query systems.

References:

(BS76) Belnap, N.D. and Steel, T.B., The Logic of Question and Answers, Yale University of Press, New Haven, 1976.

(CH75) Clark, H.H. and Maviland, S.E., "Comprehension and the Given-New Contract," in Discourse Production and Comprehension (ed. R. Freedle), Lawrence Erlbaum, Hillside, N.J., 1975.

(G67) Grice, H.P., "The Logic and Conversations", in The Logic of Grammar (eds. D. Davidson and G. Harman), Dickinson, Encino, Callf., 1975.

(KH73) Keenan, E.L. and Hull, R.D., "The logical presuppositions of questions and answers," in Prasuppositionen in Philosophic Und Linguistik"(eds. J.S. Petofi and D. Franck), Athenaum Verlag, Franfurt, 1973.

(JW77) Joshi, A.K. and Weischedel, R., "Computation oi a subclass of inferences: presupposition and entailment," American Journal of Computational Linguistics, January 1977.

(L77) Lehnert, W., "Human .and computational question answering", Cognitive Science, vol. .1, no. 1, 1977.

In the relational formalism:

STUDENTS(STUDENTS,MAJ0R)
OFFERINGS*(COURSES,SEQUENCER)
ENROLLMENTS (SEQUENCED , STUDENT# ,GRADE)

Figure 1

* NOTE: SEQUENCE# uniquely identifies an offering of a course.