STRUCTURED PLANNINC AND DEBUCCINC

Mark L. Miller and Ira P. Goldstein
Artificial Intelligence Laboratory
Massachusetts Institute Of Technology
Cambridge, Massachusetts 02139

Abstract

The SPADE theory uses linguistic formalisms to model the program planning and debugging processes. The theory begins with a taxonomy of basic planning concepts covering strategies for identification, decomposition and reformulation. A handle is provided for recognizing interactions between goals and deriving a lincnr solution. A complementary taxonomy of rational bugs and associated repair techniques is also provided. SPA OK. introduces a new data structure to facilitate debugging -- the derivation tree of the program.

SPADE generalizes recent work in Artificial Intelligence by Suasman and Sacerdoti on automatic programming, and extends The theory of program design developed by the Structured Programming movement. It provides a more structured information processing model of human problem solving than the production systems of Newell and Simon, and articulates the type of problem solving curriculum advocated by Papert's Logo Project.

1. A Multi-Faceted Approach

The SPADE theory is being developed in three contexts:

1. Education: an editor called SPADEE-0 has bern implemented that encourages students to define and debug programs in terms of explicit SPADE design choices, thereby providing a highly structured programming environment.

2. AI: an automatic programmer called PATN has been designed using an augmented transition network embodiment of the SPADE theory. This results in a framework which unifies recent work on planning and debugging by Sacerdoti [75] and Sussman [75].

3. Psychology: a parser called PAZATN has been designed that applies the SPADE theory to the analysis of programming protocols. PAZATN produces a parse of the protocol that delineates the planning and debugging strategics employed by the problem solver. PAZATN extends the series of automatic protocol analysers developed at Carnegie-Mellon University [Waterman & Newell 72, 73; Bhaskar & Simon 76].

Hand-simulations of PATN and PAZATN on elementary programming problems and informal experiments with the SPADEE-0 editor attest to the theory's cogency in accounting for a wide range of planning and debugging techniques [Goldstein & Miller 76a,b; Miller & Goldstein 76b,c,d].

2. A Linguistic Analogy

In developing a representation for problem solving techniques, we have been guided by an analogy to computational linguistics, for three reasons.

1. Thc concepts and algorithms of computational linguistics, though originally intended to explain the nature of language *per .se*, supply perspicuous yet powerful descriptions of complex compulations in general.

2. Computational linguistics decomposes computations into syntactic, semantic, and pragmatic components. This decomposition clarifies the explanation of complex processes when viewed in the following manner: syntax formalizes the range of possible decisions; semantics the problem description, and pragmatics the. procedural relationship between the two.

3. Computational linguistics has undergone an evolution of procedural formalisms, beginning with finite state automata, later employing recursive transition networks (context free grammars), next moving on to augmented transition networks, and culminating in the current set of theories involving frames [Minsky 75, Winograd 75, Schank 75]. Each phase captured some properties of language, but was incomplete and required generalisation to more powerful and elaborate formalism*. Following this evolutionary sequence illuminates the complexity of language theory. We have pursued a similar evolutionary approach to clarify the complexity of prohlrm solving processes.

• To date, our theory of program design has evolved as follows: we first explored context free grammars for planning and debugging, and subsequently their generalisation to ATN's; we then examined the metaphor of protocol analysis as parsing, initially using the planning and debugging grammars to reveal the constituent structure of protocols and later using the derivations produced by the ATN formalism; and, most recently, we have studied the use of a chart-based parser to discover these analyses.

3. A Grammatical Theory of Planning

The basis for SPADE is a taxonomy of frequently observed planning concepts (fig. 1). We arrived at this taxonomy by introspection, by examining problem solving protocols [Miller & Goldstein 76b], by studying the literature on problem solving [Pglya 57, 65, 68; Newell & Simon 72; Sussman 75; Saecrdoli 75], and by enumerating techniques for finding procedural solutions to problems expressed at predicate calculus formulae [Emden & Kowalski 76] This last criterion demonstrates that the taxonomy is currently incomplete -- for example, techniques for handling disjunctions have not yet been analysed thoroughly enough to warrant inclusion. However, the taxonomy is adequate for a wide range of elementary programming problems.

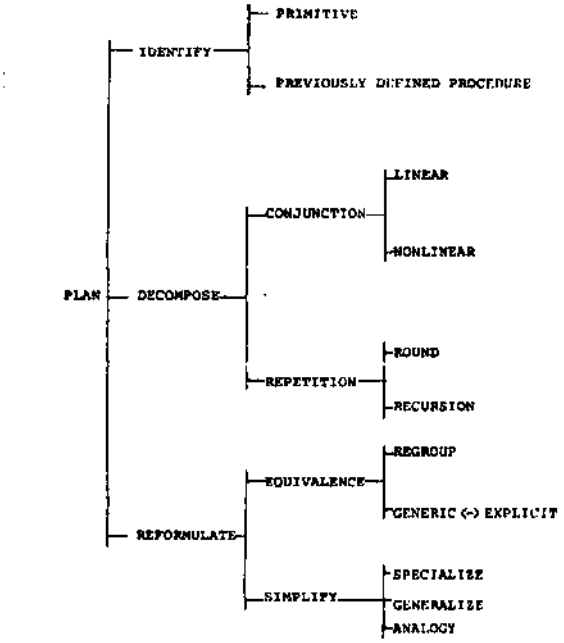


FIGURE 1
TAXONOMY OF PLANNING CONCEPTS

There are three major classes of plans in the taxonomy: identification, decomposition, and reformulation. Identification means recognizing a problem at previously solved. Decomposition refers to strategics for dividing a problem into simpler sub-problems. Reformulation plans alter the problem description, seeking a representation which is more amenable to identification or decomposition. The figure indicates how these classes of plans are further subdivided in the SPADE theory.

Planning, according to the theory, is a process in which the problem solver sclccis the appropriate plan type, and then carries nut the subgoals defined by that plan applied to the current problem. From this viewpoint, the planning taxonomy represents a decision tree of alternative plans. The decision process can be modeled by the context free grammar given below. The grammar explicitly stairs which planning rules involve recursive application of solution techniques to subgoals: setup, interface, mainstep, cleanup, and

parallel.

The gremmer is written using the following syntex "I" is disjunction, V is ordered conjunction, "&" is unordered conjunction, "<..>'" is iterstton, [,] is optionality, end a lower csse phrese in quotation msrks (eg, "repest step"/ describes $ lexicel Item which is not further expended in the grsmmsr.

PLAN -> IOENTIFY | DECOMPOSE | REFORMULATE
IDENTIFY -> PRIMITIVE | DEFINED
DEFINED -> "call user subprocedure" & PLAN
DECOMPOSE -> CONJUNCTION | REPETITION
CONJUNCTION -> SEQUENTIAL | PARALLEL
SEQUENTIAL -> [SETUP] + (MAINSTE-P + [INTERFACE])* + [CLEANUP]
PARALLEL -> <PLAN>*
SETUP -> PLAN
MAINSTEP -> PLAN
INTERFACE -> PLAN
CLEANUP -> PLAN
REPETITION -> ROUND | RECURSION
ROUND -> ITER-PLAN | TAIL-RECUR
ITER-PLAN -> repeat step" + SEQUENTIAL
TAIL-RECUR -> "itop step" + SEQUENTIAL + "recur step"

The SPADE theory is not restricted to any particular domain However, to provide concrete examples, we have concentrated on problems from elementary Logo graphic* programming [Papert 71]. This domain was chosen because of the availability of extensive student performance data. The grammar rules for primitives in this domain arc:

PRIMITIVE -> VECTOR | ROTATION | PENSTATE
VECTOR -> (FORWARO | BACK) + "number"
ROTATION -> (LEFT | RIGHT) + "numbtr"
PENSTATE -> PENUP | PENDOWN

A typical task undertaken by beginners in the Logo environment is to draw a wishingwell picture using the computer (fig. 2). Fig. 3 illustrates a solution to the wishingwell problem with its hierarchical annotation according to our planning grammar.

FIGURE 2

The grammar characterizes the decision process involved in selecting plans from the taxonomy. We illustrate its utility in the next two sections by constructing an editor that embodies the grammar and analyzing debugging in terms of the grammar. Then we show how the grammar can be augmented to include not only the syntax of plans, but their semantics and pragmatic* a* well.

4. SPADEE-0. A Planning Assistant

One reason for calling our theory of planning and debugging structured is to emphasise the link between our research and the Structured Programming movement. Dahl, Dijkstra, and Hoare [72] call for a style of programming which reflects coherently structured problem solving; but a detailed formalization of what this style entails is lacking. Our efforts in this direction, therefore, supplement the work of Dijkstra and others. How can we judge, though, whether a particular gremmer of plans captures the planning dceisions involved in solving problems for some domain? One methodology is to incorporate the grammar into an editor (SPADEE-0) whose purpose is to augment and direct the capabilities of a human user. The critical question then becomes the extent to which the editing system aids or hinders the user.

Suppose a problem solver is defining a Logo program for drawing the wishingwell shown earlier. In SPADEE-0, this is accomplished by applying the planning.grammar in generative mode:

```
la. What 1s the name of your procedural
lb. >WW

2a. Tht rule is: PWK -> IDENTIFY | DECOMPOSE I REFORMULATE
    What now?
2b. >DECOMPOSE
```

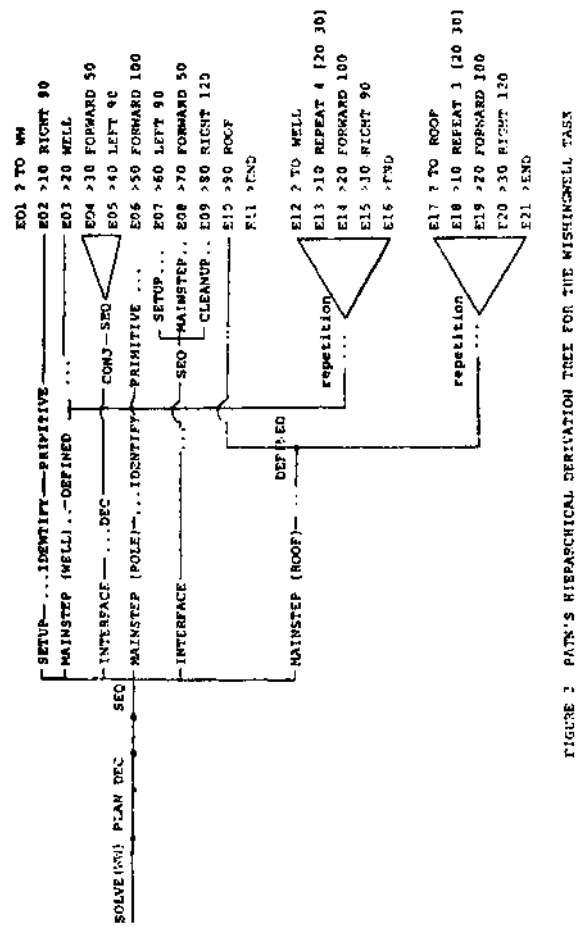


FIGURE 3 PAIN'S HIERARCHICAL DERIVATION TREE FOR THE WISHINGWELL TASK

3a. The rule is: DECOMPOSE -> CONJUNCTION | REPETITION.
3b. >CONJUNCTION

4a. The rule is: CONJUNCTION -> SEQUENTIAL ...
-> [SETUP] + <MAINSTEP+[INTERFACE]>* + [CLEANUP].
Oo you want to defint the optional SETUP?
4b. >Latar

SPADEE-0 thus encourages users to articulate their design decisions in top-down order. At the same time, the system allows the user to escape from this strict discipline if an alternative solution order seems preferable. This was illustrated by the user's "later" instruction, which suspends the current goal for subsequent solution.

SPADEE-0 was implemented by assigning an interpretive procedure to each grammatical operator. In essence, the editor is a bookkeeper for the user's goal tree. Though simple, the editor serves three useful purposes.

1. From an educational standpoint, the editor encourages students to articulate their problem solving strategies. The fundamental hypothesis of the Logo Project, as presented by Paperl [71], is that such articulate problem solving is beneficial to the learner. SPADEE-0, with its extreme form of articulation, provides an experimental vehicle for evaluating Papert's claim. Our experiment will he to test whether students exposed to SPADEE-0 learn Logo faster than controls whose problem solving is more tacit.

2. From an AI standpoint, its use will indicate whether the planning grammar is adequate, or whether certain plans are not present that competent problem solvers feel arc necessary,

3. From a psychological standpoint, we will collect transcripts of individuals using the editor and formulate pernonal grnmmars based on the particular rules usually employed by each user. The personal grammar will model the problem solving skills of that individual. In the past we have manually analysed protocols from standard Logo. SPADEE-0 protocols, with their explicit planning choices, should be

far more revealing.

5. RAID, A Debugging Assistant

SPADE ineludcs a theory of debugging. Sueh a theory is essential, since problem solvers must often formulate plans in the fare of imperfect knowledge and limited resources. Under such circumstances, even carefully reasoned judgments may be mistaken.

Given a grammatical theory of planning, debugging can be analyzed as the localixation and repair of errors in applying grammar rules during planning. Since our planning rules were constructed from operators for conjunction, disjunction, and oplionality, three basic classes of errors arise:

1. syntactic bugs, in which the basic grammar is violated, such as when a required *conjunct* is missing;

2. semantic bugs, in which a Romantic constraint arising from the particular problem is violated, such as when a syntactically *optional* constituent, needed because of the semantics of the particular problem, is missing;

3. pragmatic bugs, in which an inappropriate selection from a set of mutually exclusive *disjuncts* is made.

Fig. 4 exemplifies these bug types. Although additional

FIGURE ;4A - SYNTACTIC.\I,IA INCOKPIA'1
A NECESSARY roN.iur.rv ir MTr- lv<

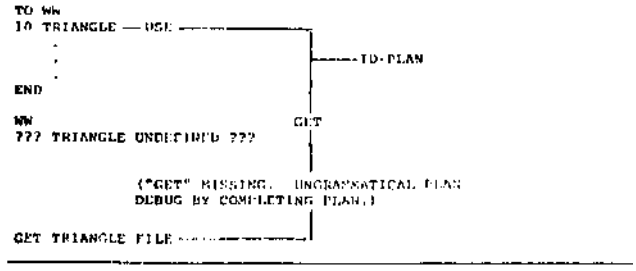


FIGURE 4B - SEMANTICALLY INCORRECT PLAN
AN OPTIONAL CONJUNCT IS MISSING

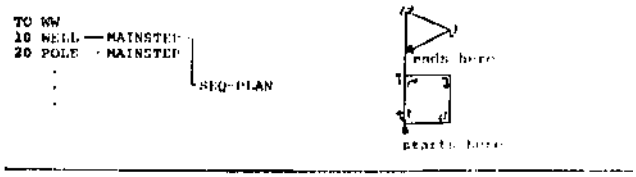
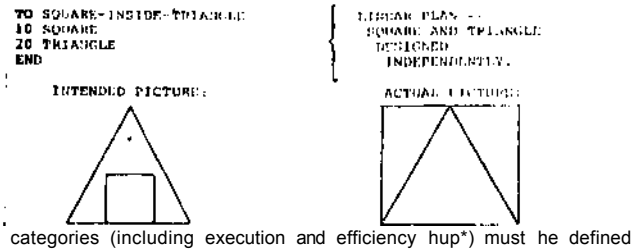


FIGURE 4C - PRAGMATICALLY INCORRECT PLAN
AN INCORRECT DISJUNCT HAS BEEN SELECTED



categories (including execution and efficiency hup*) must he defined to make this taxonomy of bugs complete, these classes are adequate lo characterize many examples which arise in elementary programming.

RAID is a debugging assistant for SPADKK-0 implemented on the basis of this taxonomy of errors. As SPADKK-0 prompts the user through the planning process, those choices which might lead to syntactic, semantic, or pragmatic bugs arc recorded by the system. In subsequent debugging episodes, RAID alerts the user to the relevant bug warnings, thereby aiding in diagnosis. Moreover, the SPADKK-0 planning editor can then be reinvoked to guide repair of the bug by resuming planning in the context of the culpable decision. An interaction with RAID that would arise from premature execution of a program is illustrated below:

The user has continued defining the wishingwell program begun earlier, with iti current definition as shown in fig. 4b. Although an initial SETUP and an INTERFACE have not yet been supplied, the user requests execution of the program.

Ib. >RUN WISHINGWELL

2a. OK. *(Fig. 4b shows the result.)* What now?
2b. >Diagnose

3a. The rule 1s: DIAGNOSE -> (PARSE|CODE|HO0ELI PROCESS)*.
Which typt of diagnosis do you want to use?
3b. >Parsa

4a. The rule 1i: PARSE -> SYNTAX|SEMANTICS!PRAGMATICS.
Which bug type do you want to consider?
4a. >Semant1ci

5a. The following optional steps are missing:
The SETUP and the WELL-POLE INTERFACE.
5b. >Repa1r

6a. The rule 1s: REPAIR -> COMPLETE \ REPLAC£.
6b. >Complete Intarface between well and pole.

7a. The rule is: COMPLETE -> PLAN -> IDENTIFY | ...

In this example, the interaction with SPADKK-0 was again guided by a grammar; this debugging grammar is given below.

```
DIAGNOSE -> <PARSE I CODE | MODEL | PROCESS)*
PROCESS -> <ASK | TRACE>*
CODE -> "print definition"
MODEL -> RUN
PARSE -> SYNTAX | SEMANTICS | PRAGMATICS
SYNTAX -> "review conjunctive choices 1n plan"
SEMANTICS -> "review optionel choices 1n plen"
PRAGMATICS -> "review disjunctive choices in plen"
REPAIR ■> COMPLETE | CORRECT
COMPLETE -> "solve for missing conjunct"
CORRECT -> "choose alternative disjunct"
RUN -> "run code line" + ["error msg"] + [DEBUG]
ASK -> "print definition" | "print value"
TRACE -> [SELF-DOC*] + RUN*
SELF-DOC -> "add break" | "add print" I "add trace"
```

The grammar is intended to include the traditional debugging strategics employed by programmers as well as the PARSK-bascd strategies made possible by the derivation tree.

For more complex programs, there will be too many potential bug locations for RAID's undirected advice to be helpful. Hence, following SPADKK-O's *structured bookkeeping* philosophy, we allow the user to set caveats during planning regarding those decisions which appear to be potential trouble spots. In this fashion, SPADKK-0 permits an exploratory style of problem solving, as illustrated below.

Suppose the wishingwell program is part of n large project. Then the number of optional constituents that have not been defined will he large. The user will probably have some insight into which of these is actually needed. To facilitate debugging, the user is encouraged to place caveats at those decisions he suspects may require later debugging.

6a. Do you want to define the optlonel SETUP?
6b. >No.

7a. Do you wish to record a caveat here?
7b. >Yes.

6a. OK. ROOF, POLE, and WELL are pending...

Another mechanism to make SPADEE's and RAID's advice more directed it to supply the semantics for problems. This it the topic of

the next section.

6. PATN - An ATN for Planning

While context free grammar* can represent a useful abstraction of planning decisions, they have limitations which prevent them from providing a complete theory. They provide no representation for the semantics of the prohlem nor for the pragmatics involved in choosing one plan over another. For this reason, we have designed and arc currently implementing PATN, an augmented transition network (ATN) problem solver. We have adopted the ATN formalism for the same reasons that led Woods [70] to introduce it into computational linguistics: the semantic and pragmatic limitations of contest free grammars.

Fig. 5 provides a global view of PATN [Goldstein & Miller 76b]

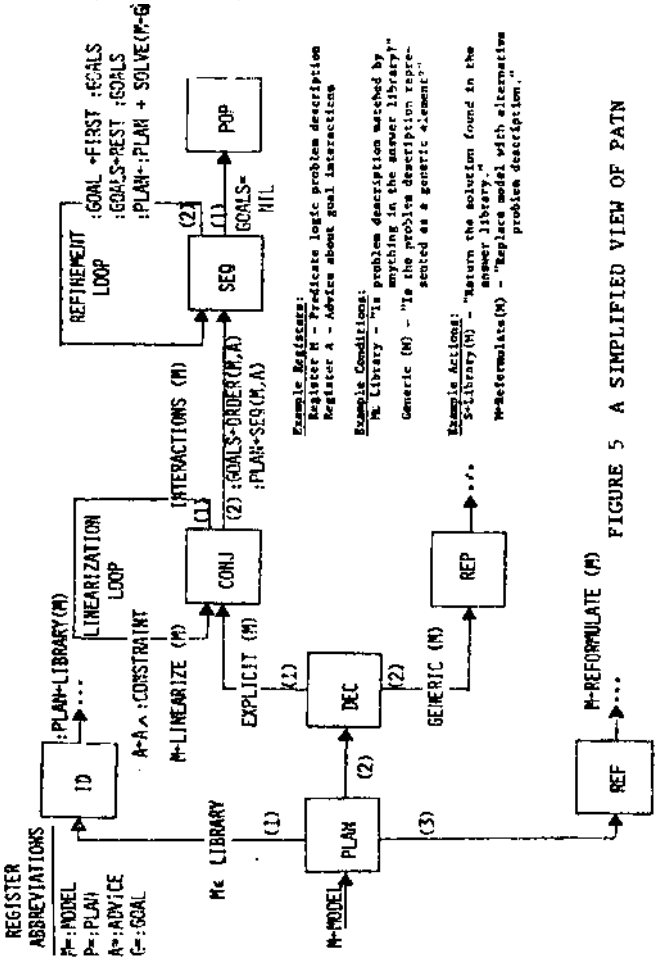


FIGURE 5 A SIMPLIFIED VIEW OF PATN

The topology of the network embodies the planning grammar. Registers contain descriptions of the problem, the solution, and various temporary constructs built during planning. Arc predicates supply pragmatic guidance by examining the registers and appropriately directing the planning process. For example, an identification plan cannot proceed if the problem description cannot be found in the answer library. PATN has been successfully hand-simulated on elementary Logo and Blocks World problems.

PATN allows us to elaborate our notion of a completed plan by defining an annotated *derivation tree.*. Associated with each node of the plan derivation is a snapshot of the values of the ATN registers at the point in the planning process when that node was created. A derivation tree reveals the constituent structure of the plan; these *semantic variables* reveal the semantic intent. A set of *pragmatic assertions* at each node, derived as instances of PATN's arc predicates, reveal the reasons for preferring a given plan over its competitors. Fig. 6 shows the annotated derivation tree for the wishingwell program given earlier.

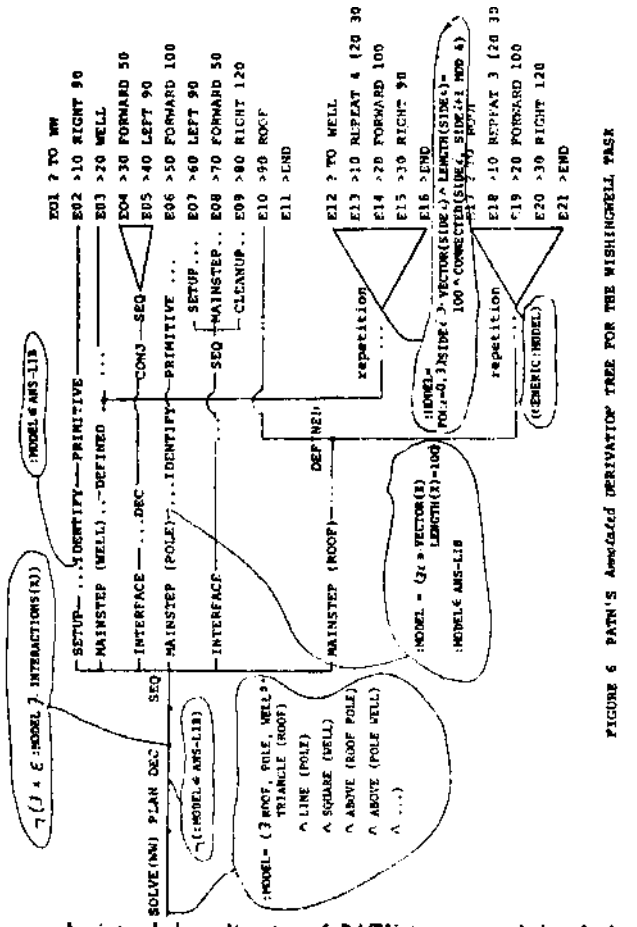


FIGURE 6 PATN'S *Annotated* DERIVATION TREE FOR THE WISHINGWELL TASK

An intended application of PATN is as a module of the SPADEE editor for providing the user with additional advice. The following hypothetical dialogue with SPADEE-1 (the SPADEE-0 editor augmented by PATN) illustrates this.

2a. Would you like to define • model?
2b. >WI5HIN6WELL

3a. OK. WISHINGWELL found 1n Modal Library.
PLAN -> IDENTIFY | DECOMPOSE | REFORMULATE
What now?
3b. >ROOF POLE WELL

4a. Ara thasa subgoals for LINEAR DECOMPOSITION?
4b. >YES

5a. The pending mainittpi ara ROOF, POLE, WELL.
5b. >ROOF

6a. In tht model ROOF 1s deierlbed as a triangle.
TRIANGLE 1s a procedure 1n the answer library.
Would you Uka to use 1t?
6b. >YES

7e. OK. The ROOF 1s complete. Pending goals are:
POLE, WELL, and the preparatory steps...

PATN's Linearisation Loop provides a superstruclure for handling dependent, interacting subgoals. The approach is to view a non-linearity as a property of the problem description. That is, even though one description of the problem leads to an unsuccessful refinement process because of interactions between subgoals, another description can succeed by making the dependent constraints explicit. For example, consider the SQUARE-INSIDK-TRIANCLE of fig. 4C If the problem description is:

SQUARE X;
TRIANGLE Y;
INSIDE X Y;

then a decomposition that draws the square and triangle independently and then attempts to fit them together lo achieve the inside relation will fail. However, a problem description of the following form allow* a successful decomposition:

SQUARE X, WITH SIDE * 100;
TRIANGLE Y, WITH SIOE - 300;
CENTER OF X ■ CENTER OF Y.

The 1NTKRACTIONS predicate is a conjunction of tests on the model register. Kach test is responsible for detecting a given non-linearity. A corresponding action modifies the model, adding new statements lo make the interaction explicit. The REFINKMKNT loop is the repository for what Sussman [75] calls the Critics Gallery. Thr theoretical progress of PATN is lo integrate the Critics Gallery concept into a theory of planning. In Sussman's HACKER, the critics gallery and library of programming techniques were separate modules: there was no integrated theory.

Of course, at any point in time the system may be unaware of a given type of non-linearity. In such cases, the absence of an interaction test will lead to a sequential decomposition that ultimately fails. The design of a program for debugging such failures is the subject of the next section.

7. DAPR — An ATN for Debugging

PATN can make mistakes. That is, PATN will sometimes introduce what we term *rational hug** into its plans, due to making arc transitions with imperfect knowledge of subtleties or interactions in the task domain. Hence, PATN must be equipped with a complementary debugging module, DAPR (fig. 7).

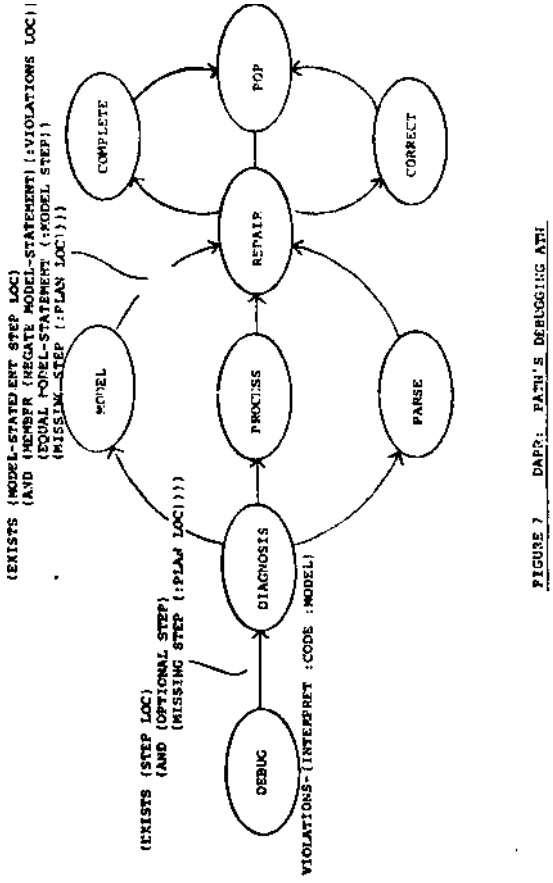


FIGURE 7 DAPR'S DEBUGGING ATN

DAPR's task it easier than that of RAID: DAPR must analyze the closed set of bug types to which PATN is subject, whereas RAID is intended to assist human programmers in finding and correcting a wide assortment of buys. DAPR employs three diagnostic techniques: model, process, and plan diagnosis. Model diagnosis is the basic technique. It amounts to comparing the effects of executing a plan to a formal description of its goals, to determine if, and in what fashion, the plan has failed. Another DAPR technique, based on Susctnan's HACKKR [75], is examining the state of the process at the time of the error manifestation. Plan diagnosis, a DAPR first, involves examining the *caveat** left by the planner as various nodes were constructed.

DAPR will also be used to provide additional guidance to RAID. This illustrates the synergism possible when educational, psychological and AI facets of a cognitive theory are studied in an integrated fashion. This integration is further exemplified in the next section when we apply the SPADK theory to protocol analysis.

8. PAZATN, a Protocol Analyser

As soon as one has an heurislically adequate theory of program design, it is natural to ask, "Can the theory provide an account of how *people* design programs?". An experimental technique we employ for answering this question is the analysis of protocols collected during problem solving sessions. By adopting this methodology we follow the precedent established in seminal studies conducted at Carnegie Mellon University [Newell & Simon 72; Waterman & Newell 72, 73; Rhaskar & Simon 76]. Our work extends their approach along three dimensions.

1. With the exception of the recent Rhaskar & Simon effort, the CMU studies have been restricted lo very limited domains such as cryplarithmelic Rather than limiting the task domain, we limit thr range of responses. Typically protocols are transcriptions of think-aloud verbalisations; we focus on the more restricted interactions arising from a problem solving session at a computer console. The analysis task in this setting is lo interpret user actions — editing, executing, tracing, etc. — in terms of the SPADK theory of planning and debugging.

2. The CMU theory centers on the *production systemn model.* Although productions arc Turing universal, they encourage a less hierarchical, less local program organization than the linguistic formalisms of the SPADK theory. In PATN, each arc transition, consisting of a predicate and an action, can be thought of as a production. However, PATN organizes these productions into local contexts, each of which consists of the arcs exiting from a given node. Not all of the arc productions arc present at any moment in time; an arc is present only when the problem solver is at the relevant node. In the production systems discussed in *Human Problem Solving* [Newell & Simon 72], all of the productions are always present and are tested in serial order.

3. CMU analyses are based on the *problem behavior graph.* Pursuing an analogy to computational linguistics, we define an interpretation of a protocol to be a *parte tree* supplemented by semantic and pragmatic annotation. The parse tree characterizes the constituent structure of the protocol. Semantic and pragmatic annotation — variables and assertions attached to nodes of the parse tree -- formalize the problem description and the rationale for particular planning choices. Annotated parse trees closely reflect the local structure of PATN's linguistic problem solving machinery, leading more directly to inferences regarding individual differences than is evident from problem behavior graphs.

Ruvcn Brooks [75] applied the CMU approach lo the programming domain, developing a model of *coding* -- the translation of high level plans into the statements of a particular programming language — and testing the model by analyzing protocols. His model is a set of production rules whose conditions match the patterns of plan elements and whose actions generate code statements. Protocols are analyzed manually, with the experimenter attempting to infer the plan which is then expanded by the production system into code paralleling that of the protocol. The processes of understanding the problem, generating the plan, and debugging arc not formalized. SPADE goes beyond this in that it can be used to parse protocols and that the parse constitutes a formal hypothesis regarding not only the coding knowledge but also the planning and debugging strategies employed by the problem solver.

[Miller & Goldstein 76b] provides an example of such analysis being performed by hand. The example is a segment from a protocol several hundred lines long in which a high school student uses Logo

to draw the letters of his name. By examining the grammar rules present in the derivation, we can observe various properties of the student's problem solving such as: reliance on certain planning choices to the exclusion of others (e.g., the student employed iteration, but aever recursion); the misuse of certain optional constituents (e.g., a setup was usually included in each procedure even when it was unnecessary); and certain situations where his problem solving violates the grammar and hence is susceptible to syntactic errors (e.g., programs were often executed before their subprocedures had been defined).

Just as a context free grammar is incomplete as a theory .of planning, likewise a parse is only a partial analysis of a protocol. The theory of annotation developed in the PATN work led us from describing only the s*yntactic* structure lo more complete analyses of protocols: an interpretation of a protocol is the selection of a particular *annotated* PATN plan derivation. Fig. 8 shows such an

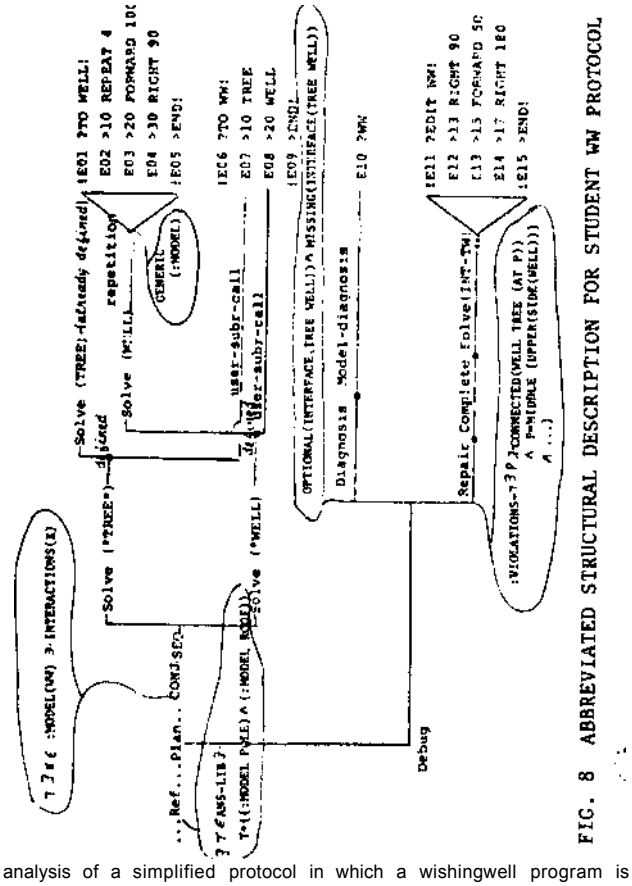


FIG. 8 ABBREVIATED STRUCTURAL DESCRIPTION FOR STUDENT WW PROTOCOL

analysis of a simplified protocol in which a wishingwell program is defined, executed and debugged.

PAZATN is a chart-based parser [Kay 73; Kaplan 73] being implemented to interpret protocols in terms of PATN's annotated plan derivations [Milter & Coldstein 76d]. It will operate by causing PATN to deviate from its preferred approach in response to bottom-up evidence (fig. 9). By taking advantage of parsing strategies developed in research on speech understanding [Lesser ct al. 75; Paxton A Robinson 75], as well as the economical chart representation of ambiguities, PAZATN has been successfully band-simulated on ten I/Ogo protocols.

PAZATN will operate by matching PATN-generated plans with protocol data. Two charts.will be used to represent alternative interpretations. The PLANCHART keeps track of the set of plausible subgoals which have been proposed by PATN. Kig. 10 shows a planchart for a wishingwell in which PATN has proposed two alternative decompositions. The structure is a chart because it shares substructures, as exemplified by the common solution to the WELL subgoal pointed to by both wishingwell decompositions'. The DATACHART records the state of partially completed interpretations.

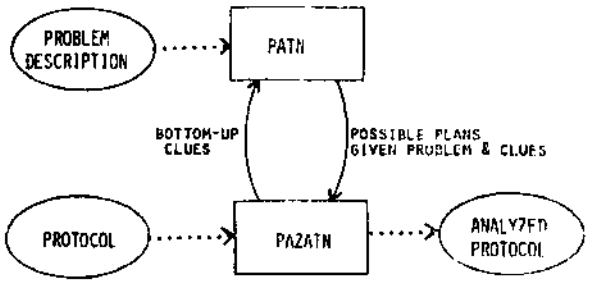


FIGURE 9 TOP LEVEL ORGANIZATION OF THE PROTOCOL ANALYZER

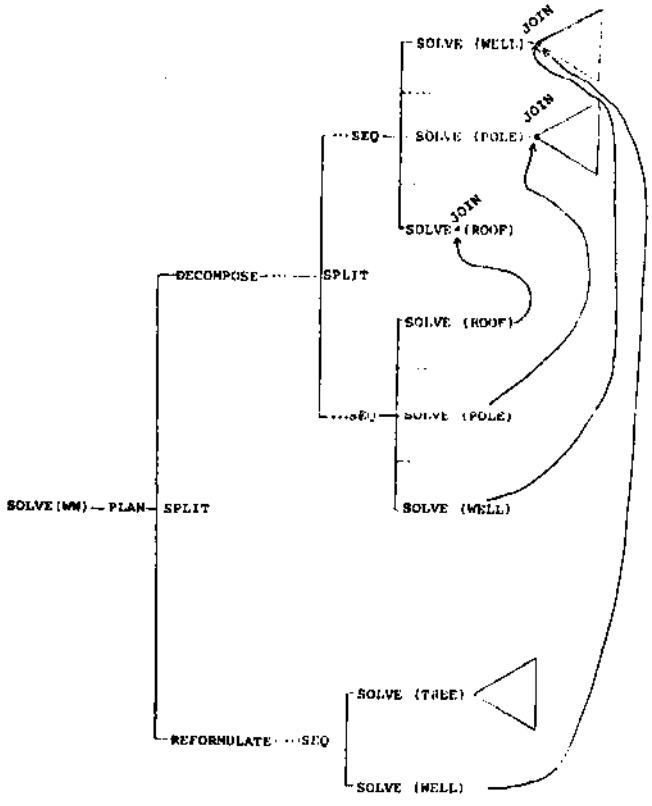


FIG. 10 PARTIAL PLANCHART OF ALTERNATIVES FOR WW

Fig. 11 shows how the datachart links events into the planrhart for a PAZATN interpretation of the wishingwell protocol given earlier.

These charts are grown as follows. First PAZATN requests PATN to generate its most plausible plan. This plan is inserted into the PLANCHART. Then protocol events are analyzed one hy one, and matched with subgoals in the PATN plan. The match is recorded in the DATACHART. If no plausible matches are found, PATN is asked to generate the next most plausible plan. The PLANCHART is thereby extended. Common subgoals share the same structure in the chart.

At first, PAZATN will be implemented interactively, with the user -- a psychologist analyzing a protocol — directing PAZATN to select different PATN plans. This follows the incremental implementation strategy used in two of the CMI) protocol analyzers [Waterman & Newell 72; Rhaskar & Simon 76]. PAZATN, even in its early interactive stages, should provide strong evidence regarding PATN's adequacy as a cognitive theory.

PAZATN will also be tested in the SPADEE. contest. Below is

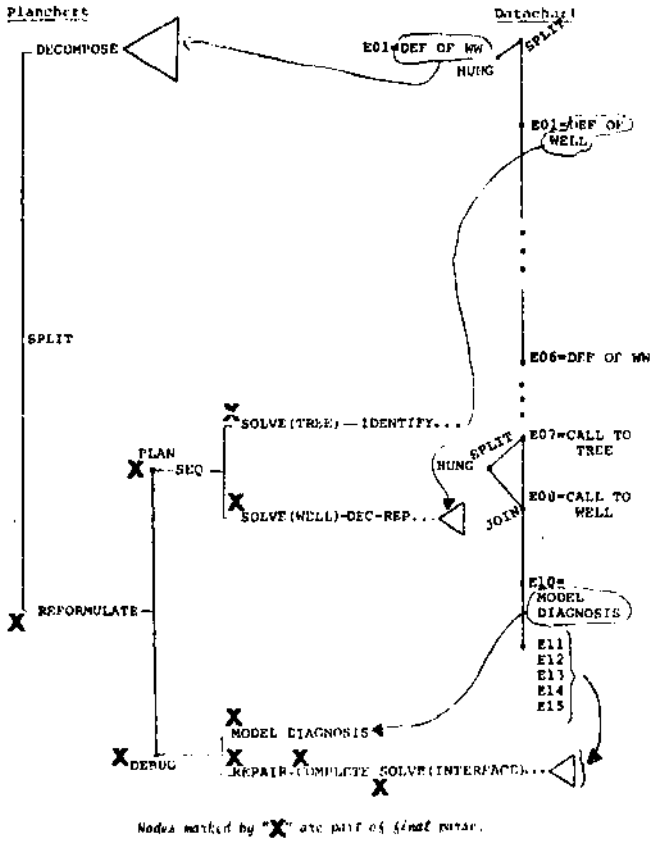


FIGURE 11 DATACHART LINKING PROTOCOL EVENTS TO PLANCHART LEAVES

a hypothetical dialogue with SPADEE-2, representing the original 'SPADEE-O augmented by both PATN and PAZATN.

Ia. Solving for WISHINGVEIL. Pending subgoais ara:
ROOF, POLE. WELL, interfaces. What now?
Ib. >SQUARE

2a. OK. WELL has baen solvad by a call to SQUARE.
SQUARE has already btan solved. What now? •

PAZATN will increase the editor's flexibility in handling ambiguous events, and in alleviating what might seem to some users to be an executive allocation of time and effort to the planning phase

9. Conclusions

The use of tools from computational linguistics — grammars, ATN's, derivation trees, parsing algorithms, charts -- has led to a perspicuous representation for a theory of planning and debugging. Computational linguistics is also responsible for suggesting the propitious decomposition of problem solving processes into components involving syntactic, semantic and pragmatic knowledge.

Our multi-faceted approach — studying problem solving in the three distinct contexts of AI, education, and psychology -- holds out the possibility of a synergistic effect. Rut proof of this must await further experimentation. Although all of the programs have been designed and hand-simulated, as of this writing only the SPADKK-0 editor has been implemented. Furthermore, the theory has not yet been exercised in enough contexts to prove its generality. However, at least for the three domains in which the theory has been explored — Logo, the Blocks World, and elementary calculus — it has provided a unified treatment of plans and bugs, a significant stride for a theory of program design.

The automatic problem solving aspect was supported by the Advanced Research Protects Agency of the Department of Defense under Office of Neval Research contract NOOO14-75-C-O643, the educational aspect by the National Science Foundation under grant C40708X, and the protocol analysis aspect by the Bolt Beranek A Newman Intelligent Instructional Systems Croup under contract MDA 903-76-C-0108 jointly sponsored by Advanced Research Projects Agency, Air Force Human Resources Laboratory, Army Research Institute, and Naval Personnel Research A Development Center.

References

Bhasker, R, and H Simon, 1976 "Problem Solving in Semenlically Rich Domains An Example, from Engineering Thermodynamics" Carnegie-Mellon U, CIP Working Paper 314

Brooks, R, 1975 A Model of Human Cognitive Behavior in Writing Code for Computer Programs Carnegie-Mellon U, Report AF0SR-TR-1084

Dahl, 0 J , E. Dijkstrs and CAR Hoare 1972. Structured Programming London, Academic Press

Emden, M Van, and R Kowelski, 1976 "The Semantics of Predicate Logic as a Programming Language " JACM 23 4, pp 733-742

Goldstein, I, and H Miller, 1976a AI Baaed Personal Learning Environments MIT AI Memo 384

Goldstein, I, and M Miller, 1976b Structured Planning and Debugging A Linguistic Theory of Design MIT AI Memo 387

Kaplan, R, 1973 "A General Syntactic Processor" in R Rustin (ed), Natural Language Processing, NY, AlgOrithmicS Press, pp 193-241

Key, M, 1973. "The MIND System" in R Rustin (ed), Natural Language Processing, NY, Algorithmics Press, pp 155-186

Lesser, V, R Fennell, L Ermen and DR Reddy, 1975 "Organization of the Heersoy II Speech Understanding System" IEEE Transactions on Acoustics, Speech, and Signal Processing Assp-23:, pp. 11-24

Miller, M, and I Goldstein, 1976b Parsing Protocols Using Problem Solving Grammars MIT AI Memo 385

Millar, M, and I Goldstein, 1976c SPADE A Grammar Based Editor For Planning and Debugging Programs MIT AI Memo 386

Miller, M, and I Goldatein, I976d PAZATN A Linguistic Approach To Automatic Analysis of Elementary Programming Protocols MIT AI Memo 388

Minsky, M, 1975 "Frame-Systems A Framework for Representation of Knowledge" in P Winston (ed), The Psychology of Computer Vision, NY, McGraw-Hill

Newell, A, and H. Simon, 1972. Human Problem Solving N.J, Prentice-Hall

Papert, S, 1971 Teaching Children Thinking MIT AI Memo 247

Paxton, W, and A. Robinson, 1975. "System Integration and Control in a Speech Understanding System * AJCL 5, pp. 5-16

Polya, G, 1957 How to Solve It NY, Doubleday Anchor Books

Polya, G, 1965 Mathematical Discovery (Vols 142) NY, Wiley and Sons

Polye, G, 1968 Mathematics and Plausible Reasoning (Vols t&2) NJ, Princeton U. Press

Sacerdoti, E, 1975 "The Nonlinear Nature of Plans" 4IJCAI, Tbilisi, Georgia. USSR, pp 206-218

Schank, R, 1975. "Using Knowledge to Understand" in R Schank & B Nesh-Webber, Theoroticet Issues in Natural Language Processing, pp 117-121

Suss man, G, 1975 A Computational Model of Skill Acquisition NY, American Elsevier

Waterman, D, and A Newell, 1972 Preliminary Results with a System For Automatic Protocol Analysis Cernegie-Mellon U, CIP Working Paper 211

Waterman, 0, and A Newell, 1973 "PAS-II An Interactive Task-Free Version of An Automatic Protocol Analysis System" SIJCAI, Stanford, Ca, pp 431-445

Winograd, T., 1975. "Frame Representations and the Declarative-Procedural Controversy" in 0. BobrOw A A Collins, Representation and Understanding, Academic Press, pp. 185-2)0.

Woods, W, 1970 "Transition Network Grammars for Natural Language Analysis" CACM 1310, pp 591-606