DOCUMENT RESUME

ED 207 579 IR 009 697

AUTHOR Miller, Mark L.; Goldstein, Ira P.
TITLE Overview of a Linguistic Theory of Design. AI Memo 383A.
INSTITUTION Massachusetts Inst. of Tech., Cambridge. Artificial Intelligence Lab.
SPONS AGENCY Advanced Research Projects Agency (DOD), Washington, D.C.; National Science Foundation, Washington, D.C.
REPORT NO LOGO-30A
PUB DATE Feb 77
GRANT NSF-EC40708X; ONR-N00014-75-C-0643
NOTE 38p.; For related documents, see IR 009 700-702.

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS Artificial Intelligence; Computational Linguistics; Computer Oriented Programs; *Design; Information Processing; *Linguistic Theory; Models; *Problem Solving; *Programing
IDENTIFIERS LOGO System; *Structural Planning and Debugging Editor

ABSTRACT

The SPADE theory, which uses linguistic formalisms to model the planning and debugging processes of computer programming, was simultaneously developed and tested in three separate contexts--computer uses in education, automatic programming (a traditional artificial intelligence arena), and protocol analysis (the domain of information processing psychology). In the education context, an editor has been implemented that encourages students to define and debug programs in terms of explicit design choices. The editor provides a structured programming environment based on a detailed theory of the processes involved in coherently structured problem solving. In the AI context, an automatic programmer called PATN was designed using an augmented transition network embodiment of the SPADE theory. This resulted in a unified framework which clarified work on planning and debugging by Sacerdoti and Sussman. In the psychology context, a parser called PAZATN has been designed that applies the SPADE theory to the analysis of programming protocols to produce a parse delineating the planning and debugging strategies used by the problem solvers. Hand-simulations of PATN and PAZATN on elementary programming problems and informal experiments with the SPADE editor demonstrate the effectiveness of the theory in accounting for a wide range of planning and debugging techniques. Twenty-six references are listed. (Author/LLS)

OVERVIEW OF A LINGUISTIC THEORY OF DESIGN

by

Mark L. Miller

and

Ira P. Goldstein

Massachusetts Institute Of Technology
Artificial Intelligence Laboratory

AI Memo 283A February 1977 Logo Memo 30A

Overview of a Linguistic Theory of Design[1]

Mark L. Miller and Ira P. Goldstein

The SPADE theory uses linguistic formalisms to model the program planning and debugging processes. The theory has been applied to constructing a grammar-based editor, in which programs are written in a structured fashion, designing an automatic programming system based on an Augmented Transition Network, and parsing protocols of programming episodes.

The SPADE 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 linear solution.

A complementary theory of rational bugs and associated repair techniques is also provided. SPADE introduces a new data structure to facilitate debugging -- the derivation tree of the program. The SPADE editor makes this structure available to the programmer.

The SPADE theory generalizes recent work in Artificial Intelligence by Sussman 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.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. The automatic problem solving aspect of this research was supported by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643, the educational aspect by the National Science Foundation under grant C40708X, and the protocol analysis aspect by the Intelligent Instructional Systems Group at Bolt Beranek and Newman, under contract number MDA 903-76-C-0108 jointly sponsored by Advanced Research Projects Agency, Air Force Human Resources Laboratory, Army Research Institute, and Naval Personnel Research & Development Center.

[1] This is an expanded version of a paper submitted to the Fifth International Joint Conference on Artificial Intelligence.

Table of Contents

1. A Multi-Faceted Approach

This paper provides an overview of three separate contexts in which the SPADE theory is being simultaneously developed and tested -- computer uses in education, automatic programming (a traditional AI arena), and protocol analysis (the domain of information processing psychology). Our experience has been that a powerful synergism results from this multi-faceted approach.

1. The Education Context: an editor has been implemented that encourages students to define and debug programs in terms of explicit SPADE design choices. The editor provides a structured programming environment based on a detailed theory of the processes involved in coherently structured problem solving.

2. The AI Context: an automatic programmer called PATN has been designed using an augmented transition network embodiment of the SPADE theory. This results in a unified framework which clarifies recent work on planning and debugging by Sacerdoti [1975] and Sussman [1975].

3. The Psychology Context: 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 strategies employed by the problem solver. PAZATN extends the series of automatic protocol analyzers developed at Carnegie-Mellon University [Waterman & Newell 1972, 1973; Bhaskar & Simon 1976].

Hand-simulations of PATN and PAZATN on elementary programming problems and informal experiments with the SPADE editor attest to the theory's cogency in accounting for a wide range of planning and debugging techniques [Goldstein & Miller 1976a,b; Miller & Goldstein 1976b,c,d].

2. A Linguistic Analogy

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

1. The concepts and algorithms of computational linguistics, though originally

intended to explain the nature of language *per se*, supply perspicuous yet powerful descriptions of complex computations in general. Problem solving, as a complex process, benefits from the application of these tools.

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 1975, Winograd 1975, Schank 1975]. Each phase captured some properties of language, but was incomplete and required generalization to more powerful and elaborate formalisms. Following this evolutionary sequence illuminates the complexity of language theory. We have pursued a similar evolutionary approach to clarify the complexity of problem solving processes.

To date, our theory of design has evolved through the following stages: we initially explored context free grammars for planning and debugging, and subsequently their generalization to ATN's; we 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, finally, our most recent work has 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 1976b], by studying the literature on problem solving [Polya 1957, 1965, 1968; Newell & Simon 1972; Sussman 1975; Sacerdoti 1975], and by enumerating techniques for finding procedural solutions to problems expressed as

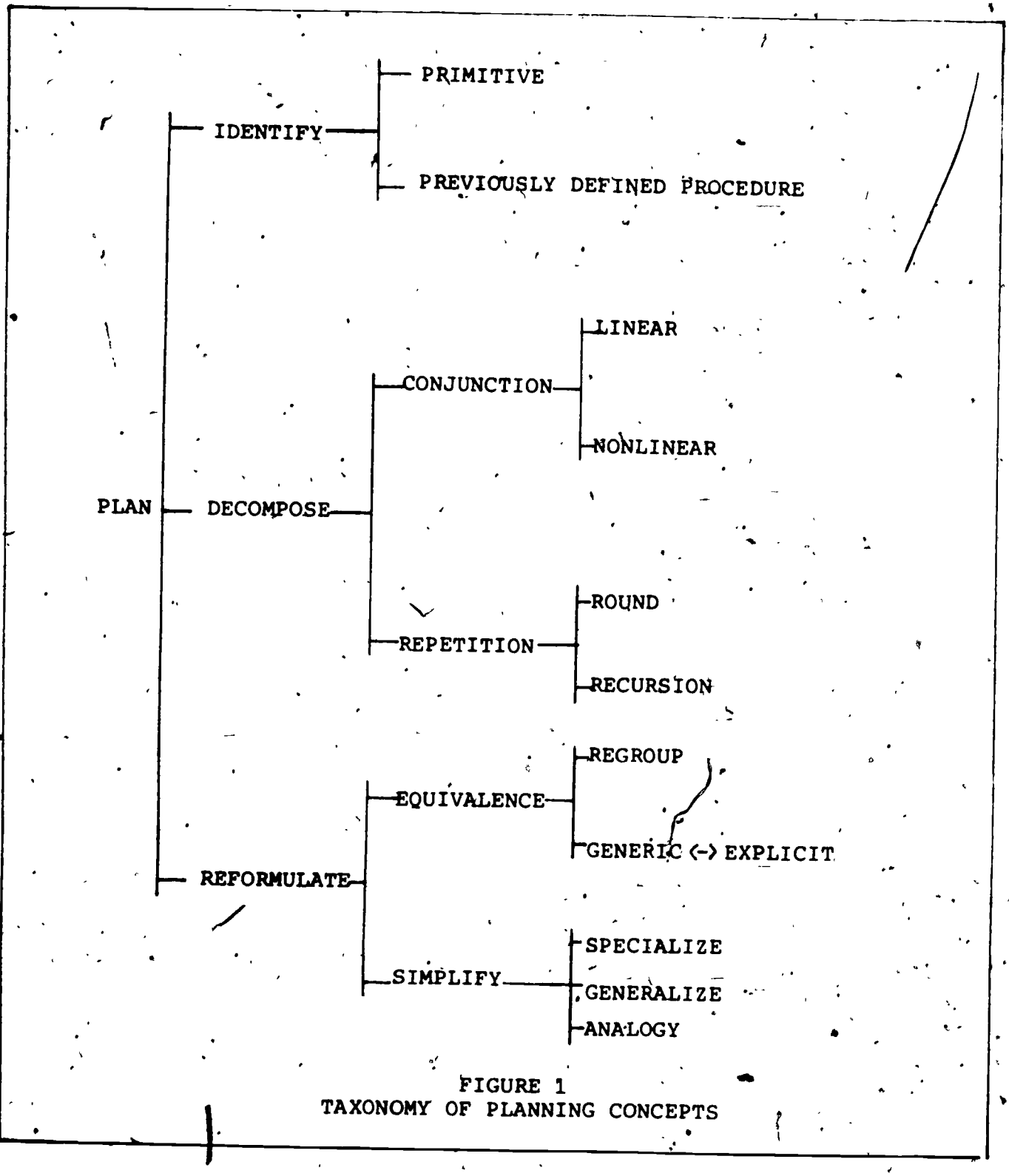


FIGURE 1
TAXONOMY OF PLANNING CONCEPTS

predicate calculus formulae [Emden & Kowalski 1976]. This last criterion demonstrates that the taxonomy is currently incomplete -- for example, techniques for handling disjunctions have not yet been analyzed thoroughly enough to warrant inclusion. However, the taxonomy is adequate for a wide range of elementary programming problems. Future research will investigate additional planning techniques.

There are three major classes of plans in the taxonomy: identification, decomposition, and reformulation. Identification means recognizing a problem as previously solved. Decomposition refers to strategies 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 selects the appropriate plan type, and then carries out 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 states which planning rules involve recursive application of solution techniques to subgoals: setup, interface, mainstep, cleanup, and parallel.

(The rules of the grammar are written using the following syntax: "|" is disjunction, "+" is ordered conjunction, "&" is unordered conjunction, "<...>*" is iteration, [...] is optionality, and a lower case English phrase enclosed in quotation marks (e.g., "repeat step") describes a lexical item which is not further expanded in the grammar.)

PLAN	-> IDENTIFY | DECOMPOSE | REFORMULATE
IDENTIFY	-> PRIMITIVE | DEFINED
DEFINED	-> "call user subprocedure" & PLAN
DECOMPOSE	-> CONJUNCTION | REPETITION
CONJUNCTION	-> SEQUENTIAL | PARALLEL
SEQUENTIAL	-> [SETUP] + <MAINSTEP + [INTERFACE]>* + [CLEANUP]
PARALLEL	-> <PLAN>*
SETUP	-> PLAN

```
MAINSTEP        -> PLAN
INTERFACE       -> PLAN
CLEANUP         -> PLAN
REPETITION      -> ROUND | RECURSION
ROUND           -> ITER-PLAN | TAIL-RECUR
ITER-PLAN       -> "repeat step" + SEQ
TAIL-RECUR      -> "stop step" + SEQ + "recursion step"
```

The SPADE theory is not restricted to any particular domain. However, to provide concrete examples, we have concentrated on problems from elementary Logo graphics programming [Papert 1971]. This domain was chosen because of the availability of extensive data on the performance of students writing "turtle" programs to draw simple pictures. The grammar rules for primitives in this domain are:

```
PRIMITIVE       -> VECTOR | ROTATION | PENSTATE
VECTOR          -> (FORWARD | BACK) + <number>
ROTATION        -> (LEFT | RIGHT) + <number>
PENSTATE        -> PENUP | PENDOWN
```

A typical task undertaken by beginners in the Logo environment is to draw a wishingwell picture using the computer.

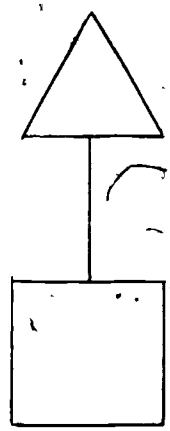


Fig. 2 -- A Logo Wishingwell

Fig. 3 illustrates a solution to the wishingwell problem with its hierarchical annotation according to our planning grammar.

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

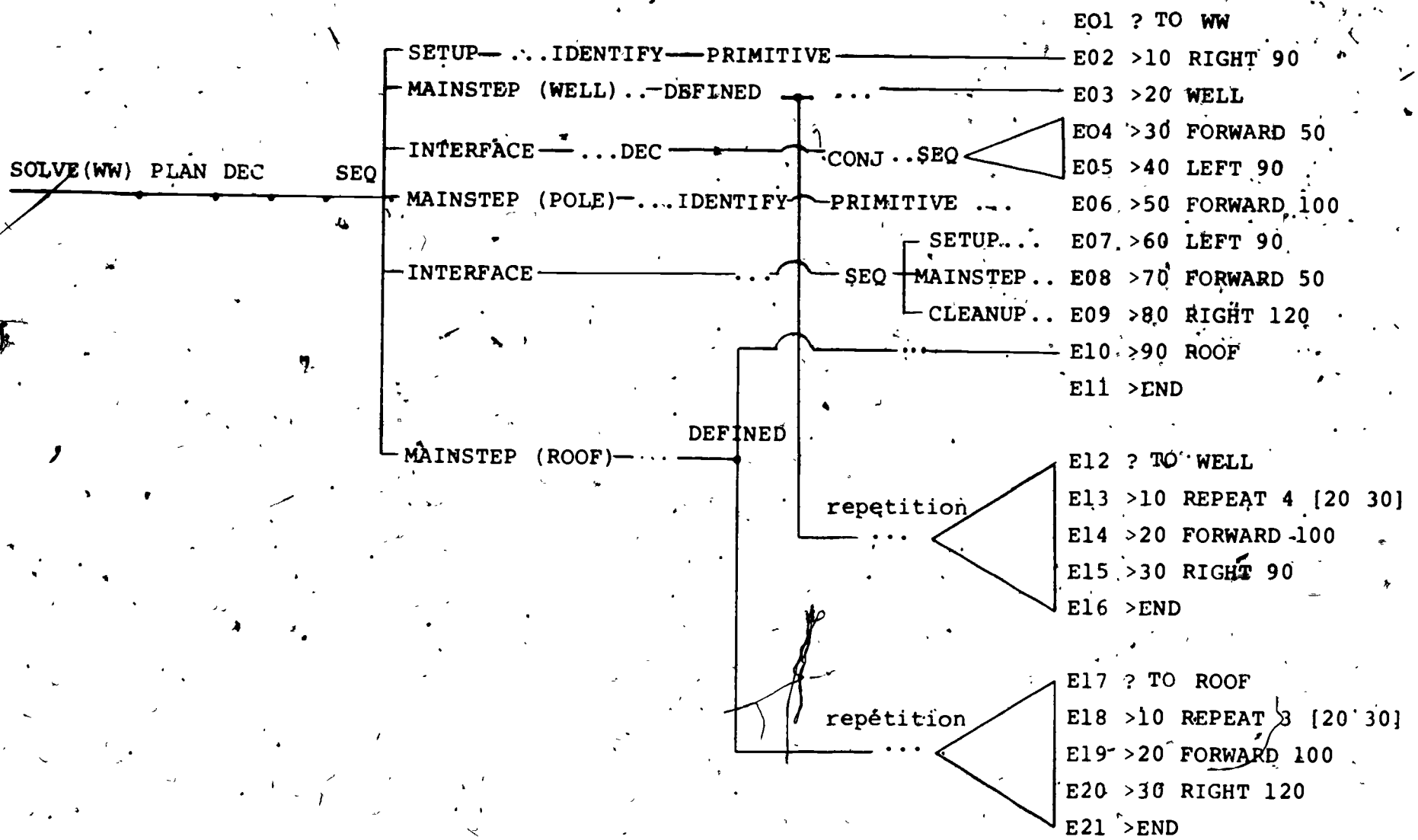


FIGURE 3 PATN'S HIERARCHICAL DERIVATION TREE FOR THE WISHINGWELL TASK

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 pragmatics as well.

4. SPADE-0, A Planning Assistant

One reason for calling our theory of planning and debugging *structured* is to emphasize the link between our research and the Structured Programming movement. Dahl, Dijkstra, and Hoare [1972] 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. The SPADE-0 editing environment serves as an example.

How can we judge whether a particular grammar of plans captures, at some level of abstraction, the set of planning decisions involved in solving problems for some domain? One methodology we employ is to incorporate the grammar into an editor (SPADE-0) whose purpose is to augment and direct the capabilities of a human user. The critical question then becomes determining the extent to which such a system aids or hinders the user.

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

1a. What is the name of your top level procedure?
1b. >WW

2a. The rule is: PLAN -> IDENTIFY | DECOMPOSE | REFORMULATE.
What now?
2b. >DECOMPOSE.

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

4a. The rule is: CONJUNCTION -> SEQUENTIAL.

The rule is: SEQUENTIAL -> [SETUP] + <MAINSTEP + [INTERFACE]>* + [CLEANUP].
Do you wish to define an optional SETUP?
4b. Later.

5a. ...

In this way, SPADE-0 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. "Later" suspends the current goal for later solution.

SPADE-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. Despite its simplicity, we see three useful applications of the program.

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 Papert in *Teaching Children Thinking* [1971], is that such articulate problem solving is beneficial to the learner. The SPADE editor, with its extreme form of articulation, provides an experimental vehicle for evaluating Papert's claim: do students exposed to SPADE progress faster in learning Logo 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 are necessary.

3. From a psychological standpoint, we will collect transcripts of individuals using the editor and formulate *personal grammars* 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 analyzed protocols from standard Logo. SPADE protocols, with their explicit planning choices, can be far more revealing.

5. RAID, A Debugging Assistant

SPADE includes a theory of debugging. Such a theory is essential, since problem solvers must often formulate plans in the face 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 localization and repair of errors in applying grammar rules during planning. Since our planning rules were constructed from operators for conjunction, disjunction, and optionality, 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 semantic 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 provides examples of these bug types. Although additional categories (including execution and efficiency bugs) must be defined to make this taxonomy of bugs complete, these classes are adequate to characterize many examples which arise in elementary programming.

RAID is a debugging assistant for SPADE-0 implemented on the basis of this taxonomy of errors. As SPADE-0 prompts the user through planning process, those choices which might lead to syntactic, semantic, or pragmatic bugs are recorded by the system. In subsequent debugging episodes, RAID alerts the user to the relevant bug warnings, thereby aiding in diagnosis. Moreover, the SPADE-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:

FIGURE 4 - SYNTACTICALLY INCORRECT PLAN
A NECESSARY CONJUNCT IS MISSING

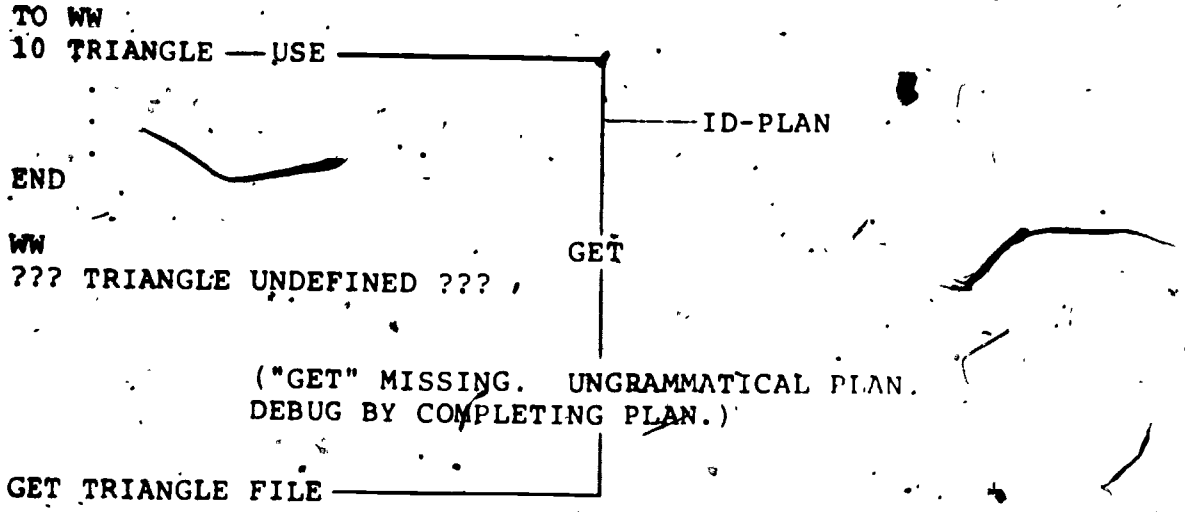


FIGURE 4 - SEMANTICALLY INCORRECT PLAN
AN OPTIONAL CONJUNCT IS MISSING

FOR EXAMPLE: "WW" MISSING INITIAL SETUP, AND INTERFACE FOR POLE.

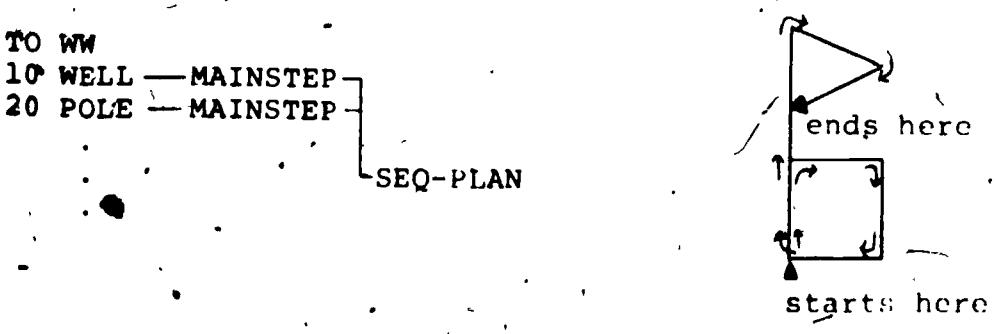
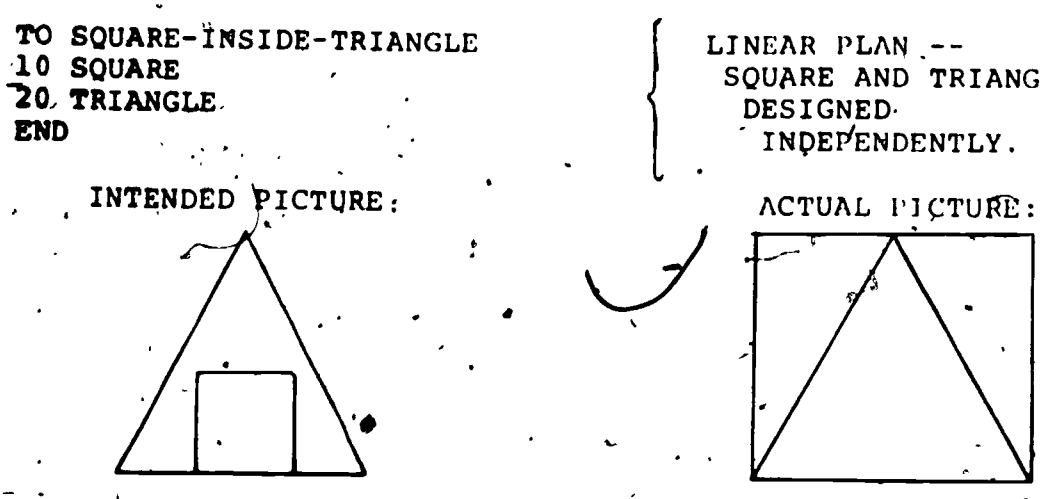


FIGURE 4 - PRAGMATICALLY INCORRECT PLAN
AN INCORRECT DISJUNCT HAS BEEN SELECTED



The user has continued defining the wishingwell program begun earlier, with its 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.

1a. ... What now?
1b. >RUN WISHINGWELL.

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

3a. The rule is: DIAGNOSE-> <PARSE | CODE | MODEL | PROCESS>*.
What now?
3b. >Parse

4a. The rule is: PARSE -> SYNTAX | SEMANTICS | PRAGMATICS.
What now?
4a. >Semantics

5a. The following optional constituents are missing in WW:
- The SETUP and the INTERFACE between WELL and POLE.
What now?
5b. >Repair

6a. The rule is: REPAIR -> COMPLETE | REPLACE.
6b. >Complete interface between well and pole.

7a. The rule is: PLAN -> IDENTIFY | DECOMPOSE | REFORMULATE
7b. > ...

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

```
DIAGNOSE    -> <PARSE | CODE | MODEL | PROCESS>*
PROCESS     -> <ASK | TRACE>*
CODE        -> "print definition"
MODEL       -> RUN
PARSE       -> SYNTAX | SEMANTICS | PRAGMATICS
SYNTAX      -> "review conjunctive choices in plan"
SEMANTICS   -> "review optional choices in plan"
PRAGMATICS  -> "review disjunctive choices in plan"
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" | "add trace"
```

This grammar represents the traditional debugging strategies employed by programmers supplemented by the PARSE-based strategies allowed by the existence of the SPADE derivation tree.

For more complex programs, there will be too many potential bug locations for RAID's undirected advice to be helpful. Hence, following SPADE-0'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, SPADE permits an exploratory style of problem solving, as illustrated below.

Suppose the wishingwell program is part of a large project. Then the number of optional constituents that have not been defined will be 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 wish to include the optional initial SETUP in WW?

6b. >No.

7a. Do you wish to record a caveat here?

7b. >Yes.

8a. OK. Your pending goals are ROOF, POLE and WELL. What now?

8b. > ...

Another mechanism to make SPADE's and RAID's advice more directed is to supply the semantics for problems. This is the topic of the next section.

6. PATN -- An Augmented Transition Network for Planning

While context free grammars can represent a useful abstraction of planning decisions, they have limitations which prevent them from providing a complete theory of design. They provide no representation for the semantics of the problem nor for the pragmatics involved in choosing one plan over another. For this reason, we have designed and are currently implementing PATN, an augmented transition network (ATN) problem solver. We have adopted the ATN formalism for the same reasons that [illegible] Woods [1970] to introduce it into computational linguistics: the semantic and pragmatic limitations of context free grammars.

Fig. 5 provides a global view of PATN [Goldstein & Miller 1976b]. 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 *assertions* at each node, derived as instances of PATN's arc predicates, reveal the *pragmatic reasons* for preferring a given plan over its competitors. Fig. 6 shows the annotated

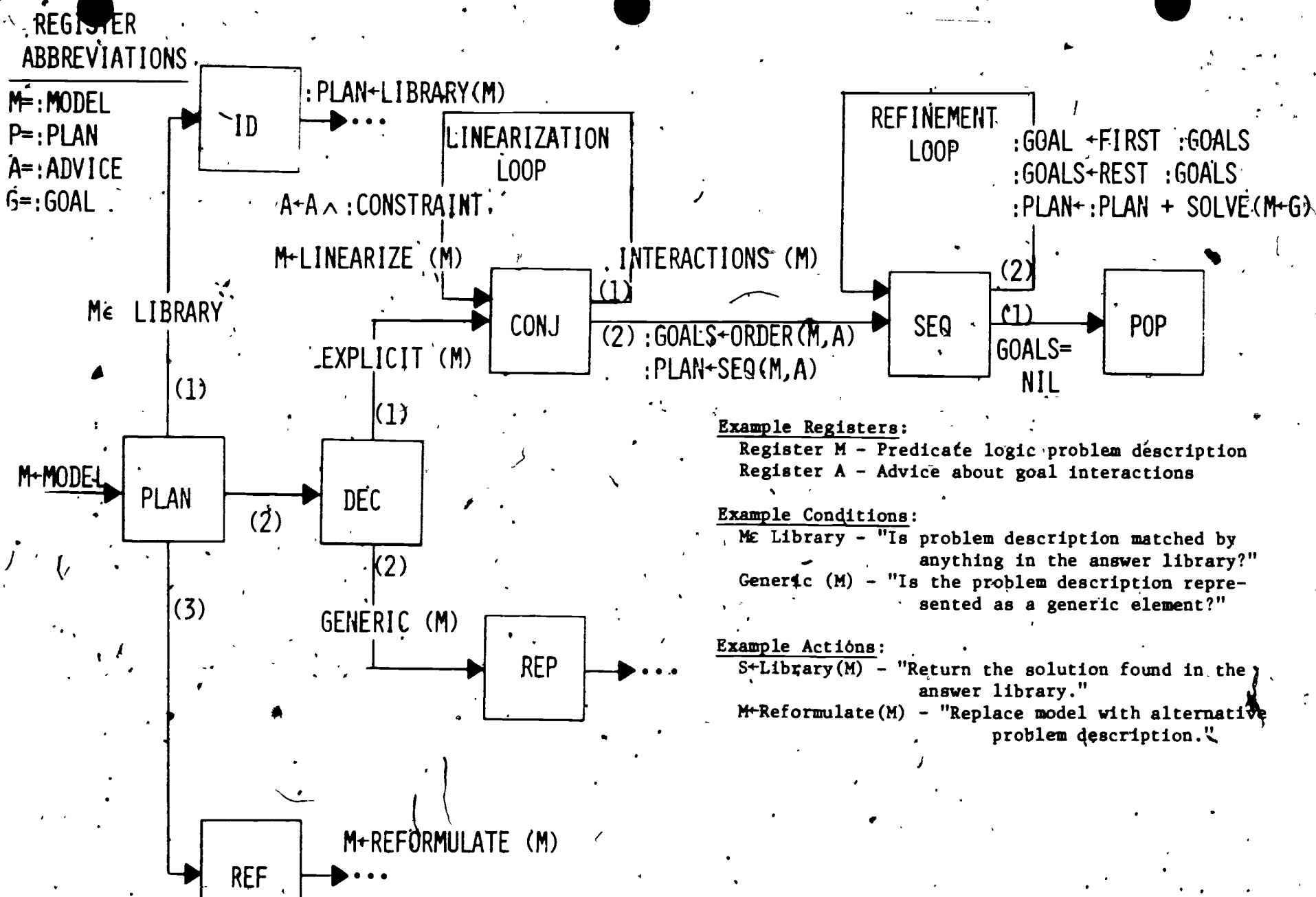


FIGURE 5 A SIMPLIFIED VIEW OF PATN

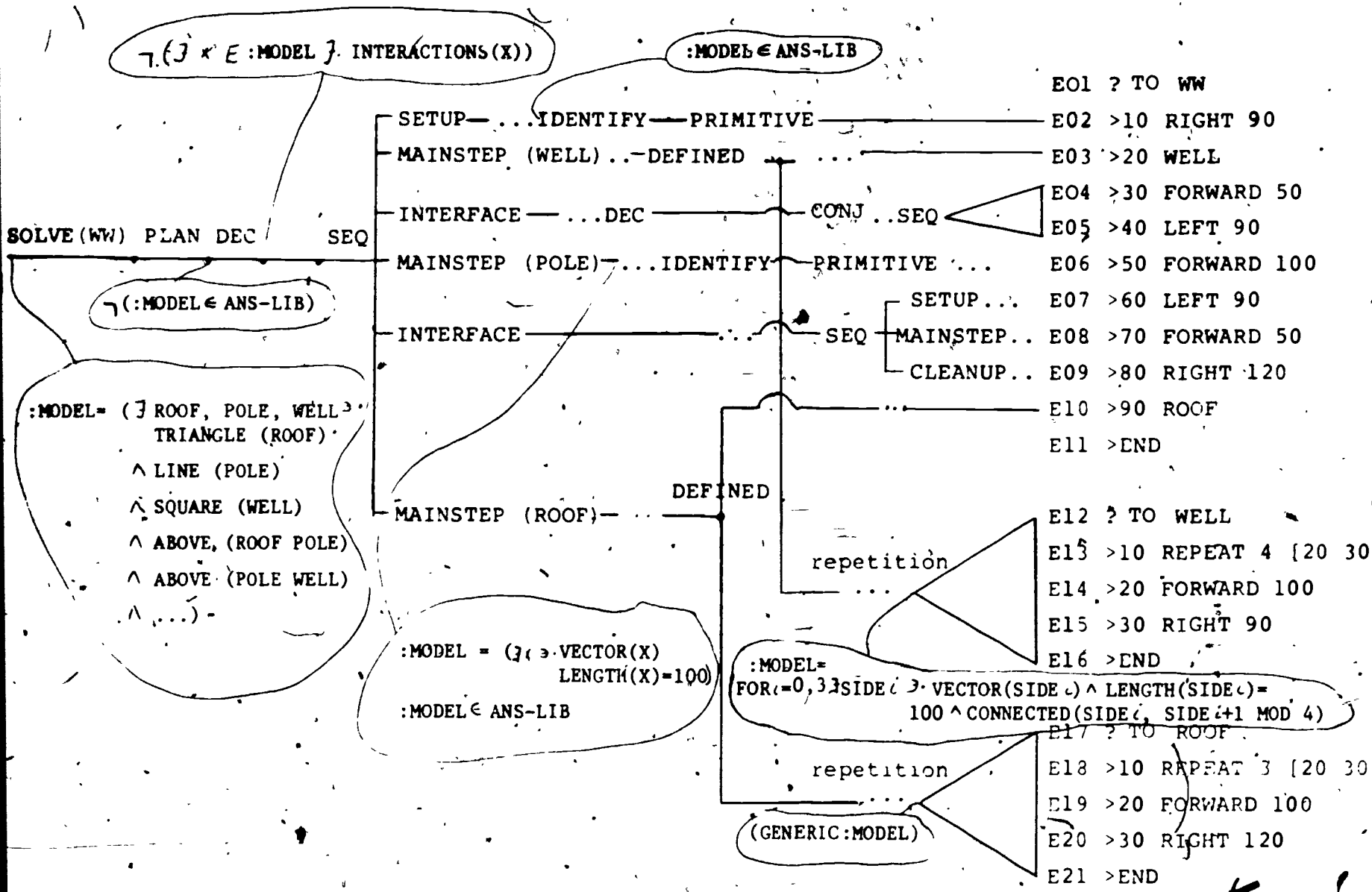


FIGURE 6 PATN'S *Annotated* DERIVATION TREE FOR THE WISHINGWELL TASK

derivation tree for the wishingwell program given earlier.

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

1a. Tell me the name of your toplevel procedure?
1b. >WW

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

3a. OK. WISHINGWELL model found in Model Library. The rule is:
PLAN -> IDENTIFY | DECOMPOSE | REFORMULATE. What now?
3b. >ROOF POLE WELL

4a. These subgoals correspond to model parts.
Are you pursuing a LINEAR-DECOMPOSITION plan?
4b. >YES

5a. The pending subgoals are: ROOF, POLE, WELL and the preparatory steps.
What now?
5b. >ROOF

6a. In the WW model, ROOF is described as a triangle.
There is a TRIANGLE procedure in the answer library.
Would you like to use it?
6b. >YES

7a. OK. The ROOF is complete. Your pending goals are:
POLE, WELL, and the preparatory steps. What now?
7b. > ...

PATN's Linearization Loop provides a theoretical superstructure 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-INSIDE-TRIANGLE 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 to achieve the inside relation will fail. However, a problem description of the following form allows a successful decomposition:

```
SQUARE X, WITH SIDE = 100;
TRIANGLE Y, WITH SIDE = 300;
CENTER OF X = CENTER OF Y.
```

The INTERACTIONS-predicate is a conjunction of tests on the model register. Each test is responsible for detecting a given non-linearity. A corresponding action modifies the model, adding new statements to make the interaction explicit. The REFINEMENT loop is the repository for what Sussman [1975] calls the Critics Gallery. The theoretical progress of PATN is to 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 Augmented Transition Network for Debugging

PATN can make mistakes. That is, PATN will sometimes introduce what we term *rational bugs* 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).

While RAID was designed to assist human programmers in finding a variety of bugs (primarily by plan diagnosis), DAPR was designed to analyze PATN's bugs. 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 Sussman's HACKER [1975], is examining the state of the process at the time of the error manifestation. Plan diagnosis is a DAPR first. It is accomplished by examining the *caveats* variable associated with various nodes of PATN's annotated plan derivation.

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 SPADE theory to protocol analysis.

8. PAZATN, an Automatic Protocol Analyzer

As soon as one has an heuristically adequate theory of design, it is natural to ask, "Can the theory provide an account of how *people* solve problems?". 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 protocol analysis studies conducted at Carnegie Mellon University [Newell & Simon 1972; Waterman & Newell 1972, 1973; Bhaskar & Simon 1976]. Our work extends their approach along three dimensions.

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

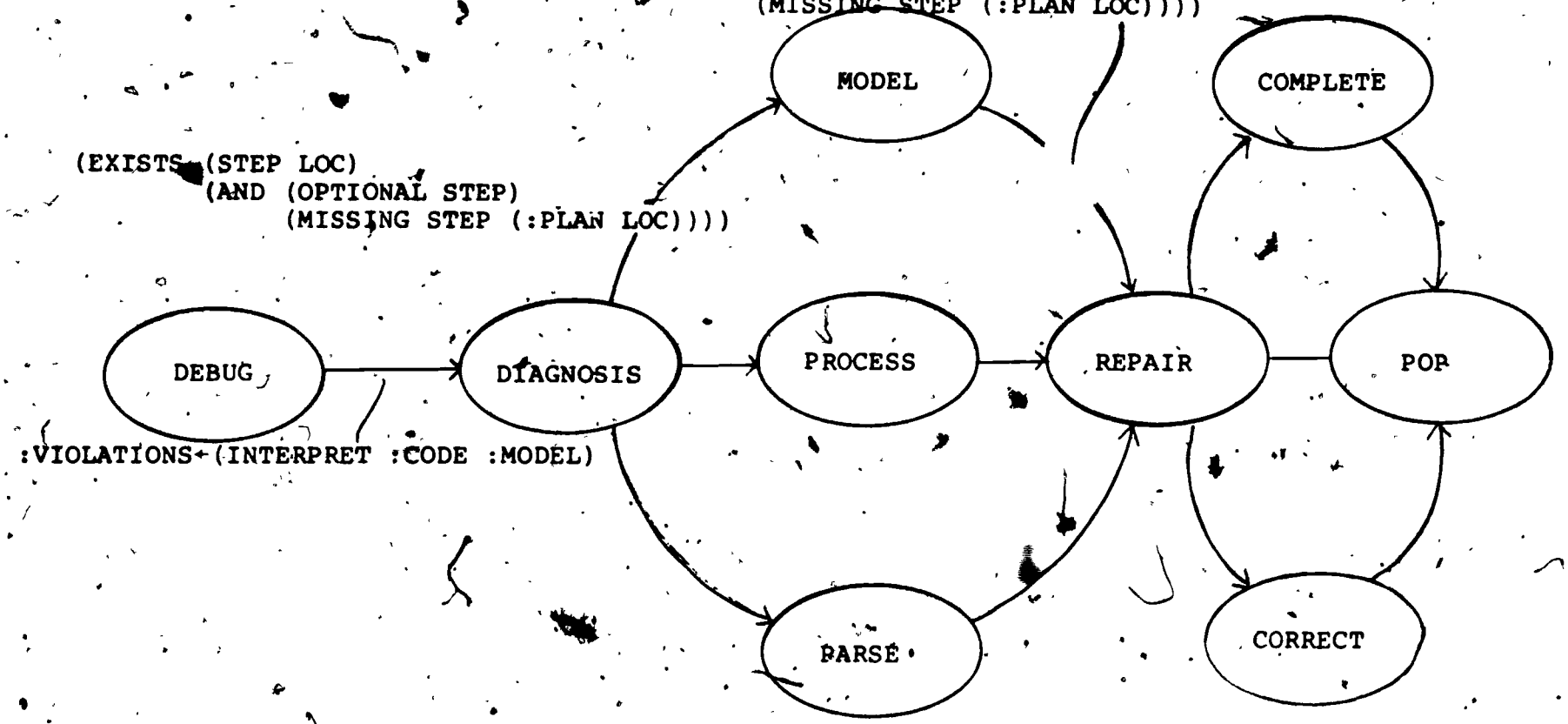


FIGURE 7 DAPR: PATN'S DEBUGGING ATN

2. The CMU theory centers on the *production systems model*. Although productions are Turing universal, they tend to result in a less structured program organization than the linguistic formalisms of the SPADE 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 are 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 1972], 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 *parse 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.

Ruven Brooks [1975] applied the CMU approach to 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 are 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 1976b] provides a detailed 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 the Logo turtle to draw the letters of his name.

By examining the grammar rules present in the derivation, we can readily 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 never 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 *syntactic structure* to 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 analysis of a simplified protocol in which a wishingwell program is defined, executed and debugged.

PAZATN is a chart-based parser [Kay 1973; Kaplan 1973] being implemented to interpret protocols in terms of PATN's annotated plan derivations [Miller & Goldstein 1976d]. It will operate by causing PATN to deviate from its preferred approach in response to bottom-up evidence (fig. 9). By taking advantage of powerful parsing strategies developed in research on speech understanding [Lesser et al. 1975; Paxton & Robinson 1975], as well as the economical chart representation of ambiguities, PAZATN has been successfully hand-simulated on approximately 10 Logo 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. Fig. 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. Fig. 11 shows how the datachart links events into the planchart for a PAZATN interpretation of the wishingwell protocol given earlier.

These charts are grown as follows. First PAZATN requests PATN to generate its

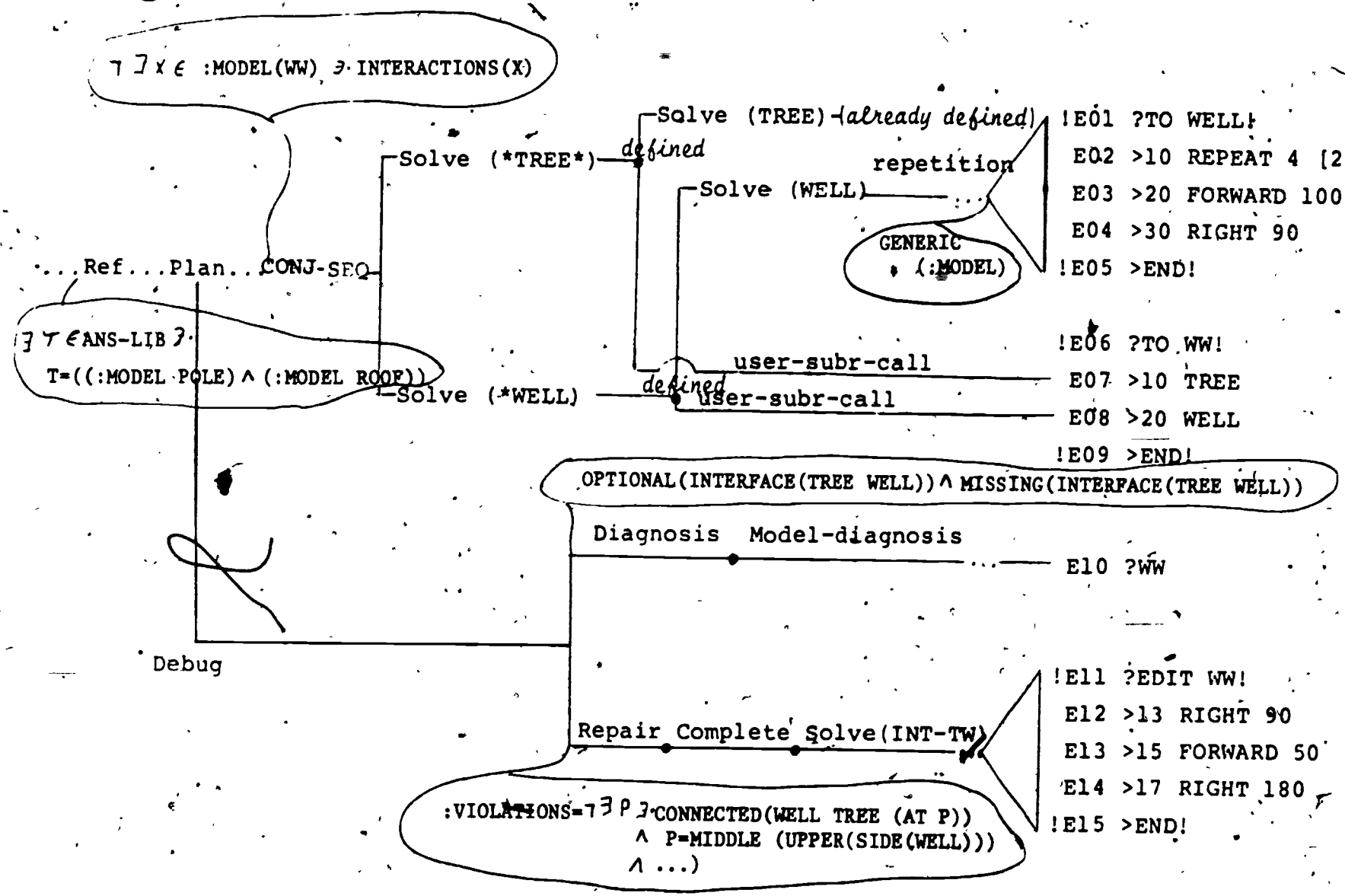


FIGURE 8 ABBREVIATED STRUCTURAL DESCRIPTION FOR WW

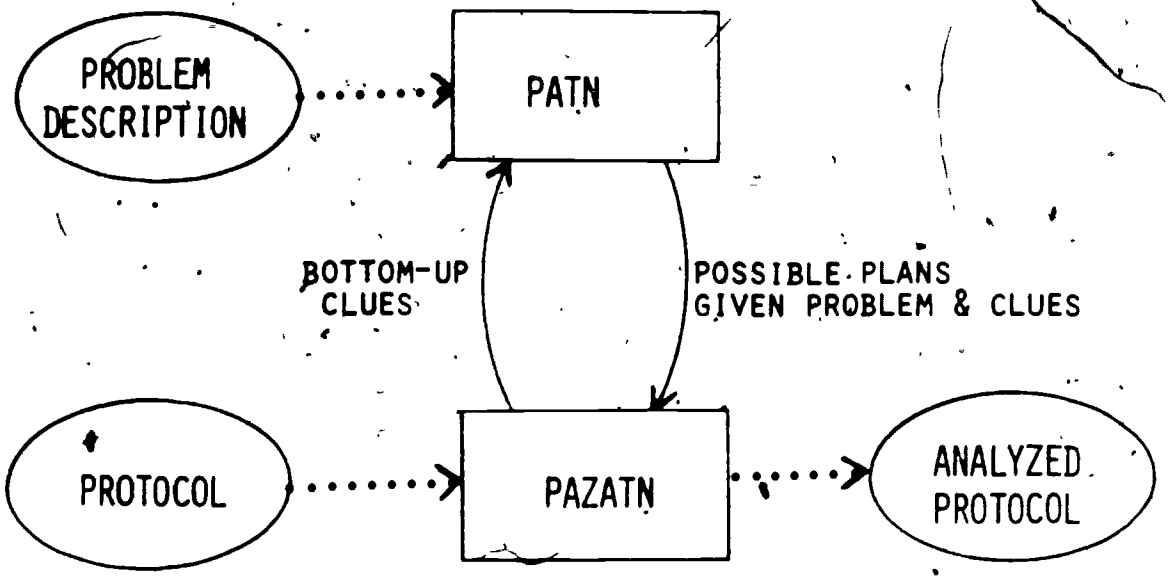


FIGURE 9 TOP LEVEL ORGANIZATION OF THE PROTOCOL ANALYZER

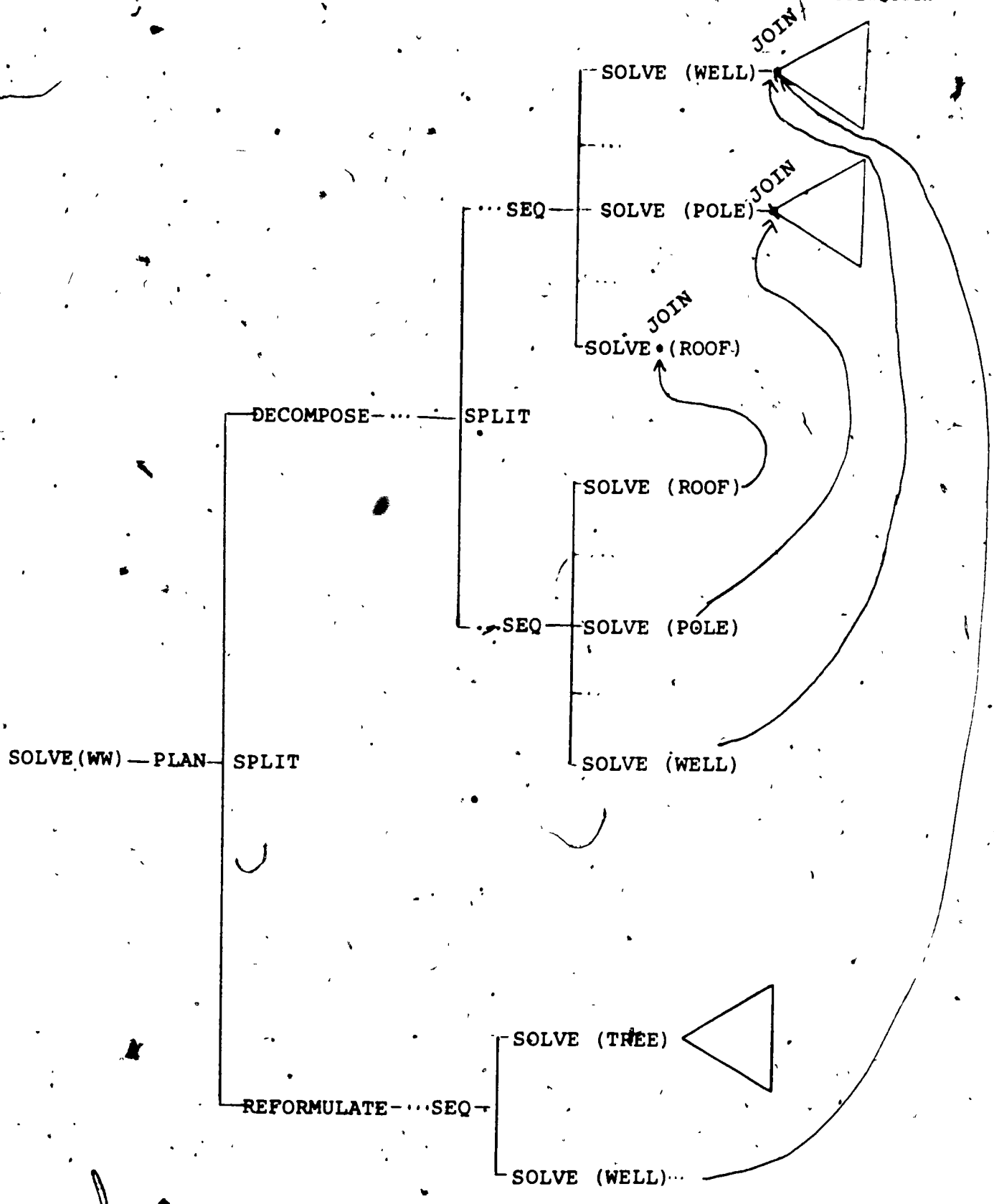
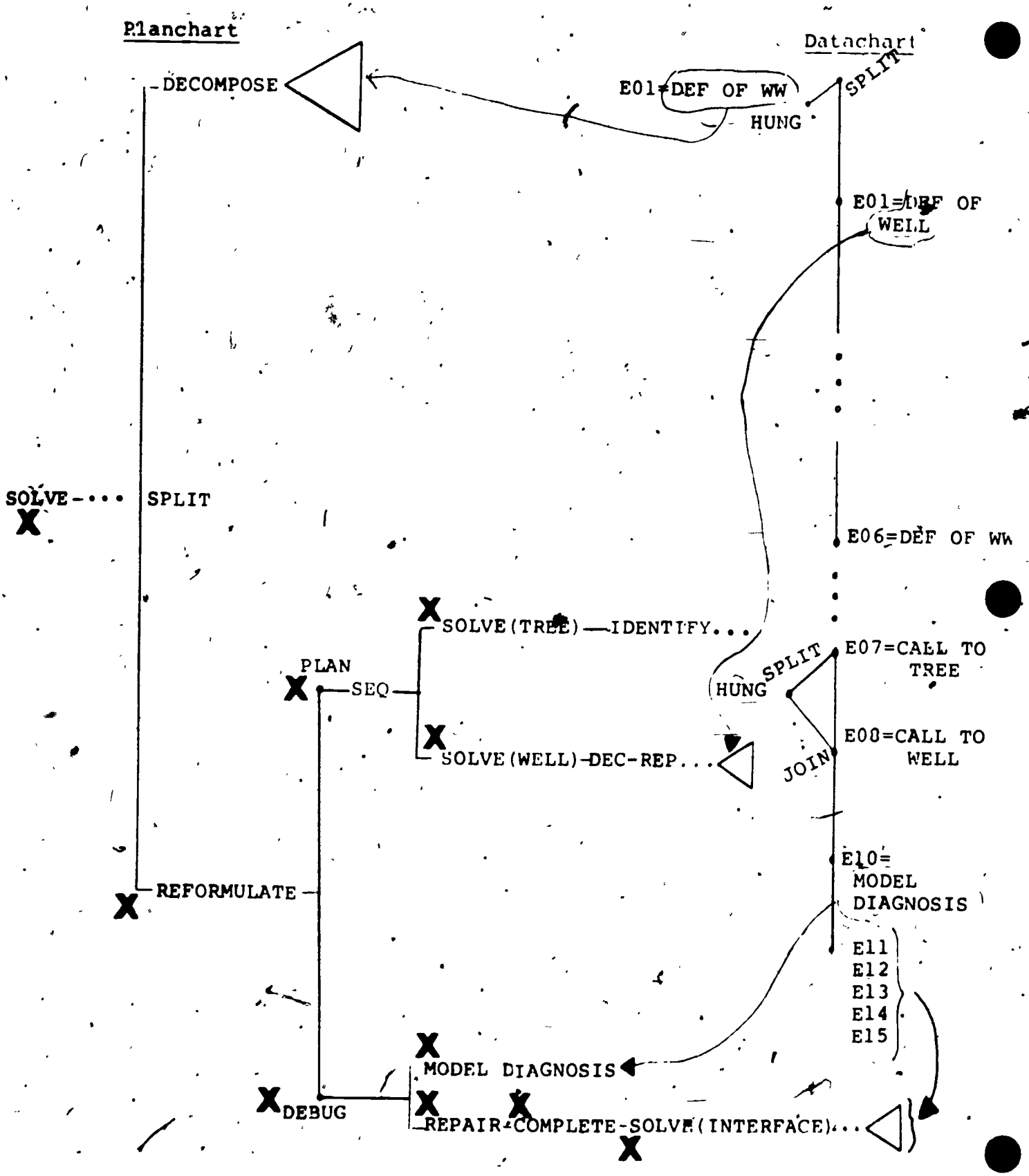


FIGURE 10 PLANCHART OF ALTERNATIVES FOR WW



Nodes marked by "X" are part of final parse.

FIGURE 11 DATACHART LINKING PROTOCOL EVENTS TO PLANCHART LEAVES

most plausible plan. This plan is inserted into the PLANCHART. Then protocol events are analyzed one by 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 CMU protocol analyzers [Waterman & Newell 1972; Bhaskar & Simon 1976]. 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 SPADE context. Below is a hypothetical dialogue with SPADE-2, representing the original SPADE-0 augmented by both PATN and PAZATN.

1a. We are now solving for a WISHINGWELL. Your pending subgoals are:
ROOF, POLE, WELL, and the interfaces. What now?
1b. >SQUARE

2a. OK. I am assuming that WELL has been solved by identifying it with the previously solved SQUARE procedure. What now?
2b. > ...

PAZATN will increase the editor's flexibility in handling ambiguous events, and in alleviating what might seem to some users to be an excessive allocation of time and effort to the planning phase. Ultimately we expect SPADE-2 to provide a programming environment of value to professional as well as student programmers.

9. Conclusions

An outline of the SPADE theory of design has been presented. One theme has been the synergism achievable in a multi-faceted research effort. Programs have been designed and are being implemented for testing SPADE in educational, psychological and AI contexts. The AI dimension involves the development of a spectrum of possible

theories of cognition. The psychological and educational dimensions supply feedback regarding which parts of a theory are rejected by competent human problem solvers. Furthermore, the psychological dimension would be incomplete were it not to address the issue of learning, while the educational dimension characterizes the trajectory of the student's cognitive state through time.

A second theme has been the exploitation of concepts and algorithms from computational linguistics: grammars, ATN's, derivation trees, search strategies from speech understanding, chart-based parsers. Computational linguistics is also responsible for suggesting the propitious decomposition of problem solving processes into components involving syntactic, semantic [redacted] pragmatic knowledge.

We believe that our unified approach to AI, psychology and education represents a new research paradigm offering the potential for considerable progress in all three fields. But much remains to be done. Although all of the programs have been designed and hand-simulated, only the SPADE-0 editor has been implemented. Furthermore, while SPADE has been applied to Logo graphics, blocks world and elementary calculus problems, it has not yet been exercised in enough contexts to prove its generality. If this research project succeeds, a new clarity will have been brought to the study of problem solving. Even if it fails, the reasons for the failures should provide useful insights. In any case, it has already unified the treatment of plans and bugs, a significant stride for a theory of problem solving.

10. References

Bhaskar, R., and Herbert A. Simon, February 1976. "Problem Solving in Semantically Rich Domains: An Example from Engineering Thermodynamics." (draft of paper to appear in *Cognitive Science*), Carnegie-Mellon University, C.I.P. Working Paper 314.

Brooks, Ruven, May 1975. *A Model of Human Cognitive Behavior in Writing Code for Computer Programs*. Carnegie-Mellon University, Report AFOSR-TR-1084.

Dahl, Ole-Johan, Edsger Dijkstra and C.A.R. Hoare, 1972. *Structured Programming*. London, Academic Press.

Emden, M.H. Van, and R.A. Kowalski, October 1976. "The Semantics of Predicate Logic as a Programming Language." *Journal of the ACM*, Volume 23, Number 4, pp. 733-742.

Goldstein, Ira P., and Mark L. Miller, December 1976a. *AI Based Personal Learning Environments: Directions For Long Term Research*. MIT Artificial Intelligence Laboratory, Memo 384 (Logo Memo 31).

Goldstein, Ira P., and Mark L. Miller, December 1976b. *Structured Planning and Debugging: A Linguistic Theory of Design*. MIT Artificial Intelligence Laboratory, Memo 387 (Logo Memo 34).

Kaplan, Ronald M., 1973. "A General Syntactic Processor." in Randall Rustin (ed.), *Natural Language Processing*, Courant Computer Science Symposium 8 (December 20-21, 1971), New York, Algorithmics Press, pp. 193-241.

Kay, Martin, 1973. "The MIND System." in Randall Rustin (ed.), *Natural Language Processing*, Courant Computer Science Symposium 8 (December 20-21, 1971), New York, Algorithmics Press, pp. 155-188.

Lesser, V.R., R.D. Fennell, L.D. Erman and D.R. Reddy, February 1975. "Organization of the Hearsay II Speech Understanding System." in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Volume Assp-23, Number 1, pp. 11-24.

Miller, Mark L., and Ira P. Goldstein, December 1976b. *Parsing Protocols Using Problem Solving Grammars*. MIT Artificial Intelligence Laboratory, Memo 385 (Logo Memo 32).

Miller, Mark L., and Ira P. Goldstein, December 1976c. *SPADE: A Grammar Based Editor For Planning and Debugging Programs*. MIT Artificial Intelligence Laboratory, Memo 386 (Logo Memo 33).

Miller, Mark L., and Ira P. Goldstein, December 1976d. *PAZATN: A Linguistic Approach To Automatic Analysis of Elementary Programming Protocols*. MIT Artificial Intelligence Laboratory, Memo 388 (Logo Memo 35).

Minsky, Marvin, 1975. "Frame-Systems: A Framework for Representation of Knowledge." in Patrick Winston (ed.), *The Psychology of Computer Vision*, New York, McGraw-Hill.

Newell, Allen, and H. Simon, 1972. *Human Problem Solving*. Englewood Cliffs, New Jersey, Prentice-Hall.

Papert, Seymour A., 1971. *Teaching Children Thinking*. MIT Artificial Intelligence Laboratory, Memo 247 (Logo Memo 2).

Paxton, William and Ann Robinson, 1975. "System Integration and Control in a Speech Understanding System." in *American Journal of Computational Linguistics*, Volume 5, pp. 5-18.

Polya, George, 1957. *How to Solve It*. New York, Doubleday Anchor Books.

Polya, George, 1965. *Mathematical Discovery* (Volumes 1&2). New York, John Wiley and Sons.

Polya, George, 1968. *Mathematics and Plausible Reasoning* (Volumes 1&2). New Jersey, Princeton University Press.

Sacerdoti, Earl, September 1975. "The Nonlinear Nature of Plans." in *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, USSR, pp. 206-218.

Schank, Roger C., June 1975. "Using Knowledge to Understand." in R. Schank & B. Nash-Webber, *Theoretical Issues in Natural Language Processing* (Workshop Proceedings), pp. 117-121.

Sussman, Gerald Jay, 1975. *A Computational Model of Skill Acquisition*. New York, American Elsevier.

Waterman, D.A., and A. Newell, May 1972. *Preliminary Results with a System For Automatic Protocol Analysis*. Carnegie-Mellon University, C.I.P. Working Paper 211.

Waterman, D.A., and A. Newell, August 1973. "PAS-II: An Interactive Task-Free Version of An Automatic Protocol Analysis System." in *Advance Papers of the Third International Joint Conference on Artificial Intelligence*, Stanford, California, pp. 431-445.

Winograd, Terry, 1975. "Frame Representations and the Declarative-Procedural Controversy." in D. Bobrow & A. Collins, *Representation and Understanding: Studies in Cognitive Science*, Academic Press, pp. 185-210.

Woods, William A., October 1970. "Transition Network Grammars for Natural Language Analysis." *Communications of the ACM*, Volume 13, Number 10, pp. 591-606.