DOCUMENT RESUME

ED 143 360 IR 005 136

AUTHOR Brown, John Seely; And Others
TITLE Aspects of a Theory for Automated Student Modelling.
INSTITUTION Bolt, Beranek and Newman, Inc., Cambridge, Mass.
SPONS AGENCY Advanced Research Projects Agency (DOD), Washington, D.C.
REPORT NO BBN Report No. 3549; ICAI Report No. 4
PUB DATE May 77
CONTRACT MDA903-76-C-0108
NOTE 118p.

EDRS PRICE MF-$0.83 HC-$6.01 Plus Postage.
DESCRIPTORS Autoinstructional Methods; *Cognitive Processes; *Computer Assisted Instruction; Conceptual Schemes; Decision Making; *Diagnostic Teaching; *Educational Diagnosis; Individual Instruction; Logical Thinking; *Models; Problem Solving; Teaching Models; *Tutorial Programs

ABSTRACT

Automated Student Modelling explicates reasoning strategies, the representation of procedural skills, and underlying misconceptions as manifested in errors. A diagnostic model, based on a "procedural network" as opposed to a "semantic network," is presented which provides a technique both for modelling the underlying cognitive processes of a procedural skill and for finding a way to account for manifested errors in the performance of that skill. A technique is described for analyzing the problem solving trace or protocol of a student and then automatically synthesizing a model of the problem solving strategies and motivations which were used to arrive at the solution. The underlying theory captures the reasoning powers of a master tutor and is seen as useful for guiding a computer-based laboratory tutor and for measuring how problem solving performance is evolving. It differs from classical computer assisted instruction by focusing on techniques for teaching procedural knowledge and reasoning strategies which are best learned through hands-on tasks with the guidance of an automated intelligent tutor. (Author/DAG)

ED143360

BBN Report No. 3549
ICAI Report No. 4

Aspects of a Theory for Automated Student Modelling

John Seely Brown
Richard R. Burton
Catherine Hausmann
Ira Goldstein
Bill Huggins
Mark Miller

May 1977

Acknowledgements

We are indebted to Dr. Beatrice Farr for her many suggestions on how to improve the initial draft of theis report.

This research was supported in part, by the Advanced Research Projects Agency, Air Force Human Resources Laboratory, Army Research Institute for Behavioral and Social Sciences, and Navy Personnel Research & Development Center.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Government.

TR005136

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE — READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: BBN Report No. 3549
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Aspects of a Theory for Automated Student Modelling
5. TYPE OF REPORT & PERIOD COVERED: Technical Report
6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): John Seely Brown, Richard R. Burton, Catherine Hausmann, Ira Goldstein, Bill Huggins, Mark Miller
8. CONTRACT OR GRANT NUMBER(s): MDA903-76-C-0108
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Bolt Beranek & Newman Inc., 50 Moulton Street, Cambridge, Massachusetts 02138
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, VA 22209
12. REPORT DATE: May 1977
13. NUMBER OF PAGES: 85
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Army Research Institute for Behavioral and Social Sciences, 5001 Eisenhower Avenue, Alexandria, VA
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report): Approved for public release; distribution unlimited
18. SUPPLEMENTARY NOTES: This research was supported in part, by the Advanced Research Projects Agency, Air Force Human Resources Laboratory, Army Research Institute for Behavioral and Social Sciences, and Navy Personnel Research & Development Center.
19. KEY WORDS (Continue on reverse side if necessary and identify by block number): diagnostic models, student modelling techniques, procedural skills, procedural network, automated intelligent tutor
20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

This report documents some of our recent investigations into a theory for automatically inducing and using (structural) models of a student which explicate his reasoning strategies, his representation of procedural skills and his underlying misconceptions as manifested in his errors.

The first chapter discusses a diagnostic model based on the concept of a "procedural network" - a network having many of the properties of the older style semantic networks but which captures both the intensional and

DD FORM 1473 (1 JAN 73) EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) extensional (or executable) aspects of procedural skills. These diagnostic models provide not only a technique for modelling the underlying or deep structure aspects of a procedural skill but they also suggest that an important criterion for modelling cognitive processes and their related knowledge representation is that of finding a natural way to account for all possible manifested errors in human performance of that skill. The second chapter describes a considerably more complex theory/technique for analyzing the problem solving trace or protocol of a student and then automatically synthesizing a model of his problem solving strategies as well as the motivations or "plans" that he used to guide him in his solution. This theory captures the subtle reasoning powers of a master tutor and as such acts as a powerful modelling technique of a learner which is needed for guiding our computer-based laboratory tutor as well as providing a new methodology for measuring how a student's problem solving performance is evolving. This theory also forms a cornerstone for building information processing models of master tutors.

The instructional paradigm being developed is quite different from the classical CAI or CMI approaches. Here, we are focusing on techniques for teaching procedural knowledge and reasoning strategies which are best learned through hands-on laboratory or problem-solving tasks in which the student gets a chance to exercise his knowledge under the watchful and critical eye of an automated intelligent tutor. Our instructional systems attempt to mimic the capabilities of a laboratory instructor who works on a one-to-one basis with a trainee and who can carefully diagnose what the trainee knows, how he reasons, what kinds of deficiencies exist in his ability to apply his factual knowledge and so on.

Preface

This is the first of three reports [ICAI4,6,7] which document our recent investigations into a theory for automatically inducing and using (structural) models of a student which explicate his reasoning strategies, his representation of procedural skills and his underlying misconceptions as manifested in his errors. Our basic methodology has been to explore segments of the modelling problem in the context of particular knowledge domains, and to implement tentative theories in the form of prototype intelligent instructional systems. This methodology not only provides us a test for the completeness and usefulness of our theories, but equally important it provides us an opportunity to develop and experiment with tutorial strategies which utilize the kind of deep structure model of a learner which was, heretofore, impossible to draw upon.

Before proceeding, we should comment on why structural student models (as opposed to simpler, parametric models) are critical to the kind of instructional paradigms being developed under this Tri-service contract. One of the classical goals of CAI has been to produce adaptive instructional systems which transform textbook and classroom type learning into self-paced individualized instruction. Learner models for directing this kind of instruction require very little detail with respect to the reasoning capabilities and underlying knowledge representations of the particular learner. For example, parametric models based on a factor analysis of a student's performance, or Markov models based on observed transition probabilities, often capture all the information that is needed. Note, however, that the parameters of such models don't reveal very much about the infinite variety, subtlety and structure of the reasoning strategies and problem solving heuristics of the students; nor do they, in themselves, reflect any of his deep-seated misconceptions. In part this fundamental limitation arises from the fact that there are only a finite (and usually small) number of parameters which can represent only a finite number of <u>predetermined</u> "entities". In other words, these models are basically extensional with no generative capabilities.

The instructional paradigm being developed here is quite different from the classical CAI or CMI approaches. In particular, we are not focusing on techniques for teaching factual, textbook knowledge (which can often be competently handled by the frame-oriented CAI or CMI systems). Instead, we are focusing on techniques for teaching procedural knowledge and reasoning strategies which are best learned through hands-on laboratory or problem-solving tasks during which the student gets a chance to exercise his knowledge under the watchful and critical eye of an automated intelligent tutor. Our instructional systems attempt to mimic the capabilities of a laboratory instructor or "coach" who works on a one-to-one basis with a trainee and who can carefully diagnose what the trainee knows, how he reasons, what kinds of deficiencies exist in his ability to apply his factual knowledge and so on. The instructor then uses this inferred knowledge of the trainee to determine how best to critique and/or kibitz with him.

This report describes some techniques and a beginning theory for how a computer-based "intelligent" laboratory instructor (or on-the-job-site trainer) can extract and use such information about the learner. The first chapter discusses the concept of a diagnostic model, which is based on the concept of a "procedural network" - a network having many of the properties of the older style semantic networks but which captures both the intensional and extensional (or executable) aspects of procedural skills. These diagnostic models provide not only a technique for modelling the underlying or deep structure aspects of a procedural skill but they also suggest that an important forcing function for modelling cognitive processes and their related knowledge representation is that of finding a natural way to account for all possible manifested errors in human performance of that skill.

The second chapter describes a considerably more complex theory/technique for examining the problem solving trace or protocol of a student and automatically synthesizing, from the trace, a model of his problem solving strategies as well as the motivations or "plans" that he used to guide him in his solution. This theory begins to capture the subtle reasoning powers of a master tutor and as such not only acts as 1) a powerful learner modelling technique (useful for guiding our computer-based lab instructors as well as providing a methodology for measuring how a student's problem solving performance is evolving as a result of some instruction) but also as 2) a cornerstone for building information processing models of the skills of a master tutor.

TABLE OF CONTENTS

CHAPTER 1

DIAGNOSTIC MODELS FOR PROCEDURAL SKILLS[1]

"If you can both listen to students and accept their answers not as things to just be judged right or wrong but as pieces of information which may reveal what the student is thinking you will have taken a giant step toward becoming a master teacher rather than merely a disseminator of information." ---J.A. Easley, Jr. & Russel E. Zwoyer

Until recently our efforts in constructing "intelligent" knowledge-based instructional systems (ICAI) have been primarily focussed on endowing computers with sufficient expertise to answer a student's questions, critique his behavior, and in some cases, help him debug his own understanding. Although such expertise is necessary for sophisticated training systems, it is by no means the whole story. Master tutors have skills that transcend their particular field of expertise. One of their greatest talents is the artful synthesis of an accurate "picture" of a student's misconceptions from the meager manifestations reflected in his errors. An accurate picture of a student's capabilities is a prerequisite to any attempt at direct individual remediation. The pictures of students that teachers develop (in whatever form) are often called "models". The form, use and induction of such models for procedural skills is the topic of this chapter. In particular, we shall describe some initial efforts in the development and use of a representational technique called "procedural networks" as the framework for constructing <u>diagnostic models</u> of procedural skills. A diagnostic model attempts to capture a student's common misconceptions or <u>faulty behavior</u> as simple changes to (or mistakes in) a correct model.

This chapter consists of four sections. The first describes a domain of application and provides examples of the problems which must be faced with a diagnostic model. The second introduces procedural networks as a general framework for representing procedural knowledge underlying a skill

(1) A version of this chapter has been accepted for publication in the Proceedings of the National Association of Computing Machinery, 1977.

Sample of the student's work:

41	328	989	66	216
+9	+917	+52	+887	+13
50	1345	1141	1053	229

Once, you have discovered the bug, try testing your hypothesis by "simulating" that bug and predicting the results on the following two test problems.

446	201
+815	+399

The bug is really quite simple. In computer terms, the student, after determining the carry, forgets to reset the "carry register" and hence the amount carried is accumulated across the columns. For example, in the second problem 8+7=15, so he writes 5 and carries 1; 2+1=3 plus the one carry is 4. Lastly 3+9=12 but that one carry from the first column is still there -- it hasn't been reset -- so adding it in to this column gives 13. If this is the bug, then the answers to the test problems will be 1361 and 700. This "bug" is not so absurd when one considers that a child might use his fingers to remember the carry and forget to bend back his fingers, or counters, after each carry is added.

A common assumption among teachers is that students do not follow procedures well and that erratic behavior is the primary cause of a student's inability to perform each individual step correctly. Our experience has been that students are remarkably able procedure followers, but that they often follow the wrong procedures. One case encountered last year is of special interest in this regard. The student proceeded through a good portion of the school year with his teacher thinking that he was exhibiting random behavior in his performance of arithmetic. As far as the teacher was concerned there was no systematic explanation for his errors; and, we must admit that before we had "discovered" his bug we, too, thought that he was erratic. Here is a sample of his work:

$$\begin{array}{r}7\\+8\\\hline 15\end{array}\quad\begin{array}{r}9\\+5\\\hline 14\end{array}\quad\begin{array}{r}8\\+3\\\hline 11\end{array}\quad\begin{array}{r}6\\+7\\\hline 13\end{array}\quad\begin{array}{r}8\\+8\\\hline 16\end{array}\quad\begin{array}{r}9\\+9\\\hline 18\end{array}\quad\begin{array}{r}17\\+8\\\hline 25\end{array}\quad\begin{array}{r}19\\+4\\\hline 23\end{array}$$

$$\begin{array}{r}87\\+93\\\hline 11\end{array}\quad\begin{array}{r}365\\+574\\\hline 819\end{array}\quad\begin{array}{r}679\\+794\\\hline 111\end{array}$$

$$\begin{array}{r}923\\+481\\\hline 114\end{array}\quad\begin{array}{r}27,493\\+1,509\\\hline 28,991\end{array}\quad\begin{array}{r}797\\+48,632\\\hline 48,119\end{array}$$

There is a clue to the nature of his bug in the number of ones in his answers. Every time the addition of a column involves a carry, a one mysteriously appears in that column; he is simply writing down the carry digit and forgetting about the units digit! One might be misled by 17+8 which normally involves a carry yet is added correctly. It would seem that he is able to do simple additions by a completely different procedure -- possibly by counting up from the larger number on his fingers.

The manifestation of this student's simple bug carries over to other types of problems which involve addition as a subskill. What answer would he give for the following?

A family has traveled 2975 miles on a tour of the U.S. They have 1828 miles to go. How many miles will they have traveled at the end of their tour?

He correctly solved the word problem to obtain the addition problem 2975 + 1825 to which he answered 3191. Since his work was done on a scratch sheet, the teacher only saw the answer which is, of course, wrong. As a result, the teacher assumed that he had trouble with word problems as well as arithmetic.

When we studied this same student's work in other arithmetic procedures, we discovered a recurrence of the same bug. Here is a sample of his work in multiplication:

$$\begin{array}{r}68\\\times 46\\\hline 24\end{array}\quad\begin{array}{r}734\\\times 37\\\hline 792\end{array}\quad\begin{array}{r}543\\\times 206\\\hline 141\end{array}$$

$$\begin{array}{r}758\\\times 296\\\hline 144\end{array}\quad\begin{array}{r}2764\\\times 53\\\hline 2731\end{array}$$

There are really several bugs manifested here; the most severe one being that his multiplication algorithm mimics his addition algorithm. But notice that the bug in his addition algorithm above is also present in his multiplication procedure. The "carry unit" subprocedure bug shows up in both his multiplication and addition. For example, to do 68x46, in the first column he performs 8x6, gets 48 and then writes down the "carry" which in this case is 4, ignoring the units digit. Then he multiplies 6x4 to get 2 for the second column. All along he has a complete and consistent procedure for doing arithmetic. His answers throughout all of his arithmetic work are far from random. In fact they display near perfection with respect to his way of doing it.

A First Approximation to Representing Procedural Skills

In order to build a computer system capable of diagnosing aberrant behavior such as the above, the skill being taught must be represented in a form amenable to modelling incorrect as well as correct procedures. Additionally, the model should break the skill down into shared sub-skills in order to account for the recurrence of similar errors in different skills. We use the term diagnostic model to mean a representation that depicts a student's internalization of a skill as a variant of a correct version of the skill. For a representation of a correct skill to be useful as a basis for a diagnostic model, it must make explicit much of the tacit knowledge underlying the skill. In particular, it must contain all of the knowledge that can possibly be misunderstood by a student performing the skill, or else some student misconceptions will be beyond the diagnostic modelling capabilities of the system. For example, if the model of addition doesn't include the transcription of the problem, the system would never be able to diagnose a student whose bug was to write 9's which he later misread as 7's.

The technique we use to represent diagnostic models is a <u>procedural network</u>.[2] A procedural network consists of a collection of procedures (with annotations) in which the calling relationships between procedures are made explicit by appropriate links in the network. Each procedure node has two main parts: a conceptual part representing the intent of the procedure, and an operational part consisting of methods for carrying out that intent. The methods (also called implementations) are programs that define how the results of other procedures are combined to satisfy the intent of a particular procedure.[3] Any procedure can have more than one implementation which provides a way to model <u>different methods</u> for performing the same procedure (skill). For most skills, the network representation takes the form of a lattice. Figure 1 presents an example of how a part of the addition process is partially broken down into a procedural network. Conceptual procedures are enclosed in ellipses. The top procedure in the lattice is addition.[4] Two of the possible algorithms for doing addition are presented as alternative methods. In method 2, the columns are added from left to right with any carries being written below the answer in the next column to the left. If there are any carries, they must be added in a second addition. In method 1, (the

---

(2) This term has been used by Earl Sacerdoti [1975] to describe an interesting modelling technique for a partially ordered sequence of annotated steps in a problem solving "plan". Our use of procedural nets differs from, and is less developed, than his. The extensive treatment of the structure and use of our networks is being reported in a companion paper. [Burton and Brown, forthcoming]

(3) The language we have used is LISP. The particular programming language is unimportant from a theoretical standpoint because an implementation is non-introspectable. The modelling aspects of the network must occur at the conceptual procedure level. For example, the implementation of the subtraction facts table look up procedure in the computer is necessarily different from that in the student. However, the conceptual properties of the facts table procedure are the same in both. Those aspects which are the same (e.g., the invoking of other procedures, the values returned, the relevant side effects) are included in the network, while the implementation details, which may differ, are "swept under the rug" into the program. This is not a limitation, as any "implementational issue" can be elevated to the conceptual level by creating a new conceptual procedure in between the existing ones. The distinction between conceptual and implementation details can also be used to allow a single network to model a skill efficiently at different levels.

(4) This is a simplified representation intended only to demonstrate those features of the procedural network particularly relevant to the diagnostic task. The actual breakdown into subprocedures may be different in a particular network, and will be considerably more detailed.

standard algorithm) the columns are added from right to left with any carries being written above (and included in the column sum of) the next column to the left. Notice that these two methods share the common procedures for calculating a column sum and writing a digit in the answer, but differ in the procedure they use when carrying is necessary. One structural aspect of the network is to make explicit any subprocedures that can be potentially shared by several higher level procedures.

[insert Figure 1]

The decomposition of a complex skill into all of its conceptual procedures terminates in some set of primitives that reflects assumed elements of an underlying computational model. For addition, typical primitives are: recognizing a digit, being able to write a digit, and knowing the concepts of right, left, etc.. The complete procedure network (explicitly specifying all the subprocedures of a skill) can be evaluated or "executed", thereby simulating the skill for any given set of inputs. By itself, this merely provides a computational machine which performs the skill and is not of particular import. However, the possible "misconceptions" of this skill are represented in the network by "buggy" implementations associated with procedures in the decomposition. Each buggy version contains incorrect actions taken in place of the correct ones. An extension to the network evaluator enables the switching in of a buggy version of a procedure, thereby allowing the network to simulate the behavior of that buggy subskill. This provides a computational method for determining the external behavior of the underlying bugs.

Inferring a Diagnostic Model of the Student

The problem of diagnosing a deep structure failure in a student's knowledge of a procedural skill can now be accomplished, at least theoretically, in a straightforward manner. Suppose, as in the examples on page 4, we are provided with several surface manifestations of a deep structure misconception or bug in the student's addition procedure. To

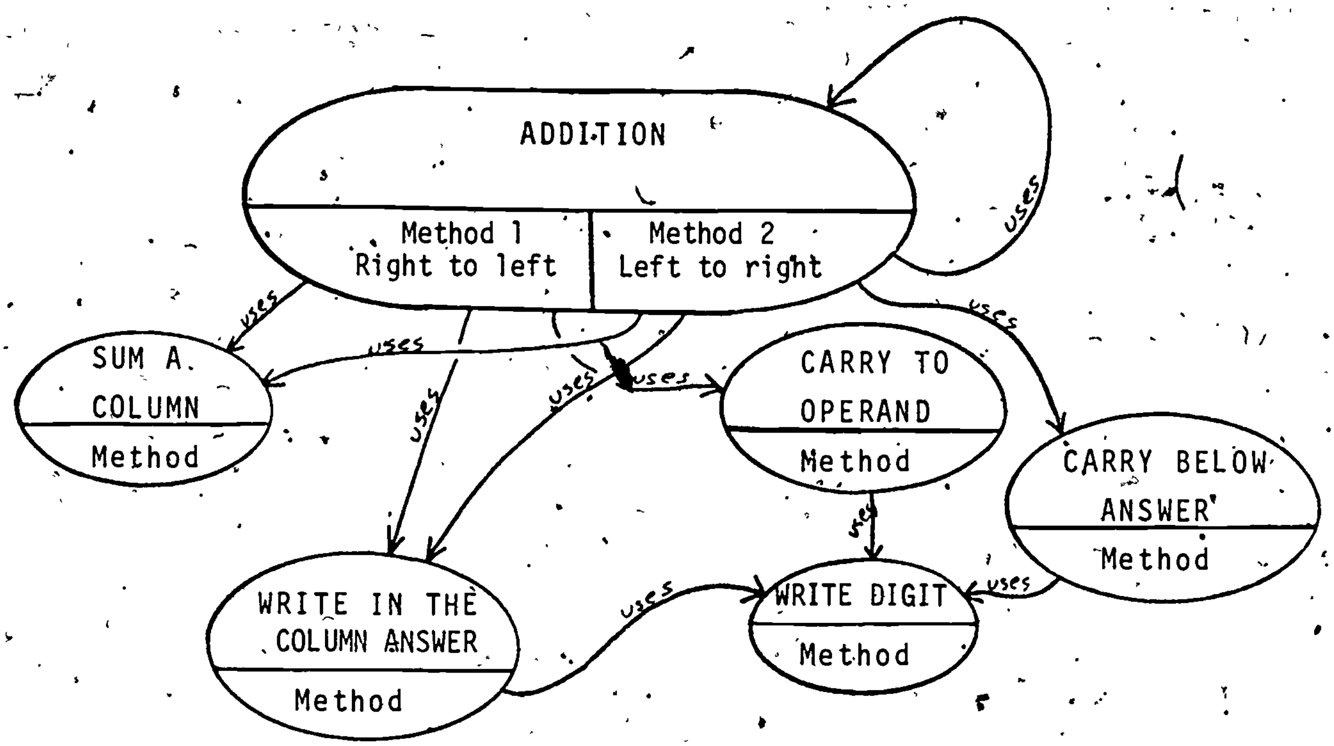


FIGURE 1
A Simplified Piece of a Procedural Network for Addition

uncover which possible subprocedures are at fault, we use the network to simulate the behavior of buggy subprocedures over the set of problems, and note those which generate the same behavior as exhibited by the student. To catch a student's misconceptions that involve more than one faulty subprocedure, we must be able to simulate various combinations of bugs.[5] For example, a student may have a bug in his carrying procedure as well as believing that 8+7 is 17 (a bug in his addition facts table). To model his behavior, both buggy versions must be used together. A <u>deep structure model</u> of the student's errors is a set of buggy subprocedures which, when invoked, replicate those errors. Each buggy version has associated information, such as the underlying teleology of the bug, specific remediations, explanations, examples and so on. These may be used by a tutoring system to help correct the student's problem.[6]

<u>Relationship of Diagnostic Models to Other Kinds of Structural Models.</u>

It is beyond the scope of this paper to discuss all the past and current work on structural models of students and how it relates to diagnostic models based on procedural networks. However, a few words are in order. Most previous and current research on this subject has been focussed on the intuitively appealing notion which postulates that if one has an explicit, well formulated model of the knowledge base of an expert (for a given set of skills or a problem domain) then one can model a particular student's knowledge as a contraction or simplification of the rules comprising the expert [Collins, Warnock and Passafiume 1975, Brown, Burton and Bell 1974, Burton and Brown 1976, Carr and Goldstein 1977]. Recently, Goldstein has articulated this concept in his Computer Coach

(5) Additional structure in the network helps resolve what combination of bugs are worth considering. In general, simulating or evaluating all simple and multiple bugs takes approximately 2 cpu seconds for the addition and subtraction procedural nets.

(6) West [1971] has broken down the diagnostic teaching task into four steps: 1) distinguish between conceptual and careless errors; 2) identify the exact nature of the conceptual error (bug); 3) determine the conceptual basis (cause) of the bug; and 4) perform the appropriate remediation. The system we describe has been directed towards problems (1) and (2). The buggy implementation nodes in the network provide the proper places to attach information relevant to problems (3) and (4).

research and has coined the term "overlay model" for capturing how a student's manifested knowledge of skills (rules) relates to an expert's knowledge base [Goldstein 1977]. In all these cases, the primary problem has been to develop techniques to discover 1) which skills were employed by the student in solving problems, 2) which skills were not used, and 3) which skills an expert would have used which the student did not.

The work reported in this paper differs in emphasis from such approaches in that the basic modelling technique focuses on viewing a structural model of the student not primarily as a simplification of the expert's rules but rather as a set of semantically meaningful deviations from an expert's knowledge base.[7] That is, each subskill of the expert is explicitly encoded, along with a set of potential misconceptions of that subskill. The task of inferring a diagnostic model then becomes one of discovering which set of variations or deviations best explains the surface behavior of the student. This view is in concert with (although more structured than) the approach taken by Self [1974] in which he models the student as a set of modified procedures taken from a procedural expert problem-solver.

We shall now consider examples of procedural skills in arithmetic, evaluations of the networks for these skills, and then we shall shift our focus to some pedagogical uses of the procedural network notion.

Procedural Knowledge Used in Subtraction

To provide an example indicative of the surprising amount of procedural knowledge needed to perform a simple skill, let us consider a more complete network representation of the subtraction of two numbers.[8] Figure 2 shows the links of the procedural network for subtraction that

(7) Because these deviations are based on both the student's intended goals and underlying teleology of the subskills, we have no automatic way to generate them (as opposed to what could be done if the deviations were based on the surface syntax of the rules). However, ongoing work by Goldstein and Miller [1976], Rich and Schrobe [1976] and Burton and Brown [forthcoming] will eventually help overcome this limitation.
(8) We have chosen just one of the several subtraction algorithms (the so-called "standard" algorithm) but the ideas presented here apply equally to others.

indicate which procedures a procedure may use. The network has been simplified by showing only one implementation of each procedure (i.e., the one taught in the "standard" algorithm).

[insert Figure 2]

The top most node represents the subtraction of two n-digit numbers. It may use the procedure for: setting up the problem, transforming it if the bottom number is greater than the top, and sequencing through each column performing the column subtraction. The implementation of the latter has to account for cases where borrowing is necessary and may call upon many separate subprocedures including taking the borrow from the correct place, scratching 0 and writing 9 if that place contains a zero, and so on. An important subprocedure is the facts table look-up where any of the simple arithmetic facts can be wrong, including the addition of 10 to a column digit, the subtraction of 1 during a borrowing operation, or any subtraction facts used during the processing of a column.

In principle, each of these subprocedures could have many buggy versions associated with it.[9] An example of a common bug is to calculate the column difference by subtracting the smaller digit from the larger regardless of which is on top. In another bug, the set-up procedure left-justifies the top and bottom numbers so that when the student is told to subtract 13 from 185, he gets 55. One interesting thing about the left justification bug is that the student will be faced with seemingly impossible problems (185-75) and may be inclined to change the direction in which he subtracts, borrowing from left to right instead of from right to left, or to change his column difference procedure to larger minus smaller, thereby eliminating the need to borrow. Thus, there can exist relationships between bugs such that one bug suggests others. A major challenge in identifying the procedural breakdown or description of a skill is to have the network naturally handle ramifications and interactions of

(9) On the average our network has two to three buggy versions for each correct version of a subprocedure.

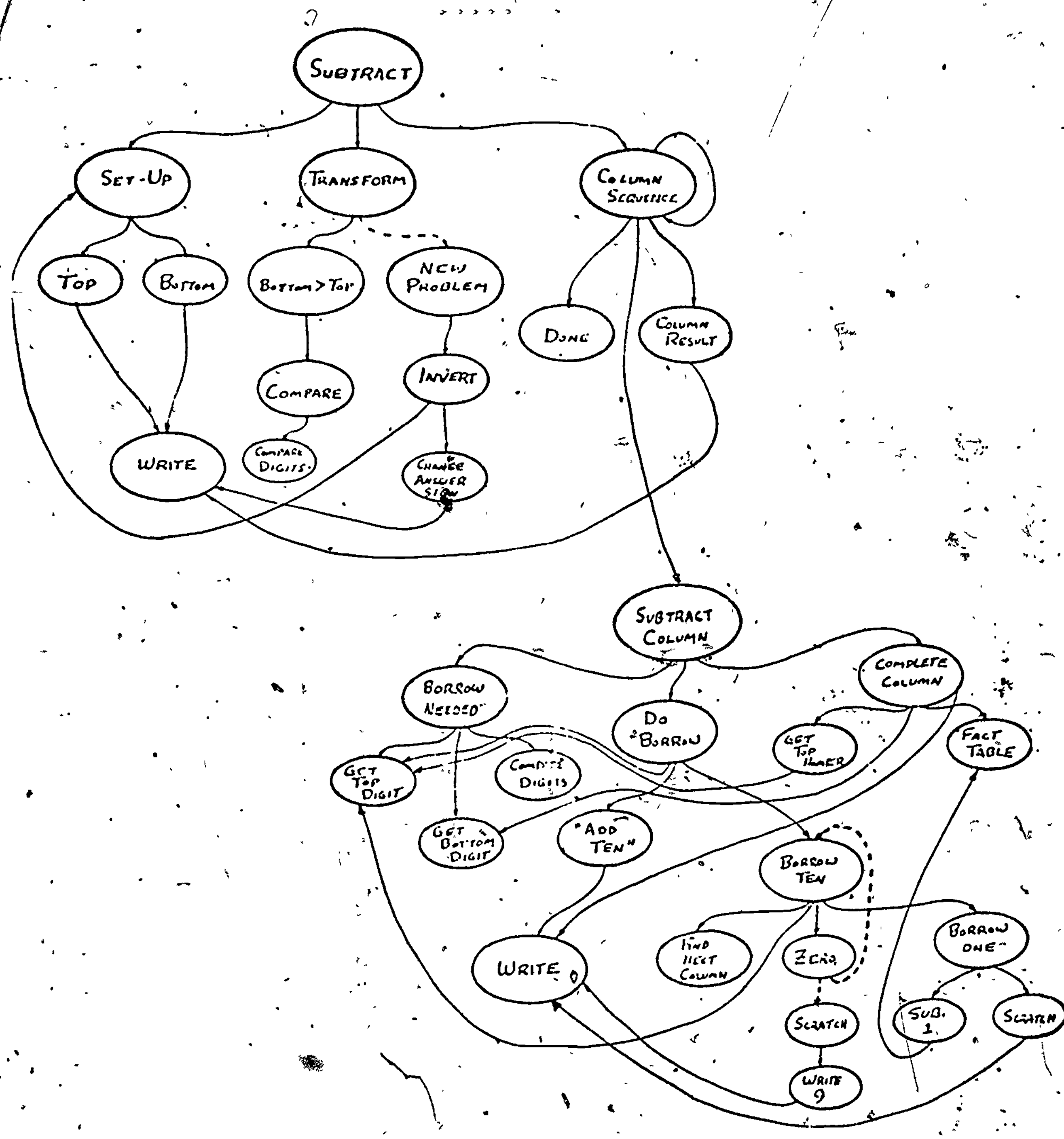


FIGURE 2

A Procedural Network for Subtraction

multiple bugs, as well as to provide a natural way to define and handle all common bugs.

Exhaustive Evaluation of the Network

Given a procedural network like the one in Figure 2, it is not always obvious how bugs in any particular subprocedure or several subprocedures will be manifested on the surface (i.e. in the answer) -- especially since bugs can have serious interactions or since a single buggy subprocedure can be used by several higher-order procedures in computing an answer. In fact, if asked to make predictions about the symptoms of a given bug, people often determine the symptoms by considering only the skills or subprocedures used in solving _one_ particular sample problem. As a result they often miss symptoms generated by other procedures that can, in principle, use or call on the given buggy subprocedure but which, because of the characteristics of the particular problem, weren't called. Yet if another sample problem were chosen, it would have caused the particular faulty subprocedure to have been used for a different purpose or in a different way, thereby generating different symptoms. Determining the complete set of symptoms for a bug is further complicated by the fact that sometimes a buggy subprocedure can be called by several higher order procedures in the midst of solving just one problem. It was this observation that first led us to consider the diagnostic value of this scheme for systematically verifying a conjectured bug.

In order to provide a feeling for the range of "answers" that can come from simple underlying bugs, we have included in Figure 3 the "answers" to a subtraction problem (15300-9522) using some of the bugs in the procedural network for subtraction. For example, the answer 14222 was generated by the bug which subtracts the smaller digit, in each column, from the larger. Appendix 4 gives one brief explanation of a bug that would generate each of the answers in Figure 3.

Figure 3

Manifestations of Some Subtraction Bugs

15300 -9522 95778	15300 -9522 27998	15300 -9522 24822	15300 -9522 16888	15300 -9522 16778	15300 -9522 14822	15300 -9522 14878
15300 -9522 14222	15300 -9522 14222	15300 -9522 14200	15300 -9522 10022	15300 -9522 10000	15300 -9522 8748	15300 -9522 7998
15300 -9522 6888	15300 -9522 6822	15300 -9522 5878	15300 -9522 5822	15300 -9522 5800	15300 -9522 5798	15300 -9522 5788
15300 -9522 5688	15300 -9522 5678	15300 -9522 5372	15300 -9522 4822	15300 -9522 4222		

Of course, a particular "answer" to a given problem can have more than one explanation or cause since there can be several distinct bugs that generate the same "answer". For example, a student may harbor many misconceptions and still get the correct answer to a particular problem. The need for teachers to thoroughly appreciate and strategically cope with the possible range of student bugs led us to construct a game called BUGGY.

BUGGY - An Instructional Activity

BUGGY is a computerized ~~game~~ based on the diagnostic interactions of a teacher and a computerized student. The teacher's role may be played by one or more persons. The teacher is presented with an arithmetic homework problem that the "student" has done incorrectly. The "student's" behavior is generated, using a procedural network, and manifests an underlying bug in one of the arithmetic subprocedures. The teacher's job is to diagnose the computerized student by providing strategic <u>test problems</u> for the "student" to solve in order to discover exactly what the underlying bug or misconception is. The problems given by the teacher are answered by the "student" using the bugged procedure. When the teacher thinks he knows the bug, he signals the computer program by pressing a "got it" key. BUGGY then asks the teacher to describe what he thinks the bug is. To make certain that he really has found the bug, a five-problem test is given in which the teacher must answer the problems in the same way that the student would do them, i.e. he must simulate the "student's" bug. Success is achieved when all five problems are done "correctly". Then the teacher proceeds to a new bug.

The following is a protocol of a team of teachers using BUGGY. The dialogue is interspersed with commentary about the issues that we feel are pedagogically important. Those lines typed by the team are marked with a vertical line in the left margin.

<u>Protocol of a Team Using BUGGY</u>

WELCOME TO BUGLAND.
I HAVE CHOSEN A BUG. HERE IS AN EXAMPLE OF THE BUG.

```
 17
 +5
 13
```

> The system presents an example in which the student doesn't get the right answer. At present, these initial examples are stored in the network linked to the buggy implementation.

NOW YOU CAN GIVE ME PROBLEMS TO DETERMINE THE BUG.

```
| 18    43
| +6   +79
| 15    23
```

> The team generates problems that are answered by the "student". Notice that the team's test problems are similar to the initial example generated by BUGGY. In general this is a poor diagnostic strategy, since it may leave other symptoms of a bug undiscovered.

| Got the bug!

> The team forms an initial hypothesis, and tells the system they have discovered the bug. The system now asks the team to describe the bug and then tests them on their understanding. Asking for a description forces the team to verbalize their sometimes fuzzy idea of what the bug is and often makes them realize how ill-defined or vague their idea actually is.[10]

(10) In the present system, the descriptions are saved but they are not currently used, since the understanding of these utterances is beyond the capabilities of any existing natural language system.

PLEASE DESCRIBE THE BUG YOU THINK I HAVE.

Student adds the number to be carried into the last column worked in.

This cryptic description is typical of those we have observed. Describing bugs in English is surprisingly difficult. What the team is trying to say is that if a column adds to more than 9, the units digit and the carry digit are added together and the result is written as the answer. For example, in the first column of the example problem, when 7 and 5 are added, 1 (the carry) and 2 (the units digit) are added to get 3. The same with 8+6, 2+9, and 3+9. (This bug could easily arise if the student were using method 2 from Figure 1 and mistakenly wrote the carry under the same column.) The hypothesis makes sense. Unfortunately, it is not the student's bug. Notice the possibly humorous, probably disastrous results that would have transpired if the team had stopped at this point and tried to tutor the student on the basis of their initial belief!

HERE ARE SOME PROBLEMS TO TEST YOUR THEORY ABOUT THE BUG.
WHAT IS:

To ensure that the team has found the correct bug, the system generates several problems for the team to solve. (At present, BUGGY has a heuristic problem generation program that gives a broad range of "different" problems. These are then filtered to try to get three problems which cause symptoms and two that give correct answers. This simple routine does surprisingly well at producing critical problems. So well, in fact, that a more "intelligent" routine which used meta-evaluation to determine the consistent set of bugs and then filtered the test problems to split the set was discarded as unnecessarily time-consuming.)

```
 21
+39
 51
```

The team's hypothesis is revealed as incorrect by the first of the five test problems. BUGGY presents its answer and then allows the team to enter more test problems.

THAT'S NOT THE BUG I HAVE.
USING MY BUG:

```
 21
+39
 15
```

TRY GIVING ME SOME MORE EXAMPLES.

```
   51     99    68
+1707    +99    +9
   21     36    23
```

After failing the test, the team gives a greater range of problems, specifically those using bigger numbers. How is it that 51+1707 can add up to 21? By broadening the scope of example problems they are able to reevaluate their initial hypothesis, dismiss it, and develop another one that is complete and consistent with respect to the new errors.

|Got the bug!

PLEASE DESCRIBE THE BUG YOU THINK I HAVE.

|The student is adding all the digits together.

HERE ARE SOME PROBLEMS TO TEST YOUR THEORY ABOUT THE BUG.
WHAT IS:

33	1091	8	28	90
+99	+60	+34	+70	+6
24	17	15	17	15

This time the team's hypothesis passes the test and they are given the description of the bug stored in the procedural network.

VERY GOOD. MY DESCRIPTION OF THE BUG IS:
THE STUDENT ALWAYS SUMS UP ALL THE DIGITS WITH NO REGARD TO COLUMNS.

Pedagogical Issues

One application of BUGGY and the "diagnostic model" view of procedural skills lies in the domain of instructor training. The realization that "random" errors are actually the surface manifestations of an underlying bug in a procedure is a major conceptual breakthrough for many instructors. Often behavior that appears to be random has a simple, intelligent, and complete underlying explanation. By proper diagnosis, remediation can be directed towards the specific weaknesses. The importance of this notion cannot be overstressed. Admitting the possibility of underlying bugs is critical to remediation in the classroom. Without the ability to diagnose procedural bugs, failure on a particular problem must be viewed as either carelessness or total algorithm failure. In the first case, the remediation consists of giving more problems, while in the second, it consists of going over the entire algorithm.[11] When a student's bug (which may only manifest itself occasionally) is not recognized by the instructor, the errant behavior must be explained as carelessness, laziness or worse. This causes the instructor to adapt his model of the student's capabilities, thereby mistakenly lowering his expectations. From the student's viewpoint, the situation is even worse. He is following what he believes to be <u>the</u> correct algorithm and, seemingly at random, gets marked <u>wrong</u>. This situation can be exacerbated by improper diagnosis. For example, Max subtracts 284 from 437 and gets 253 as an answer. Of course, says the instructor,"you forgot to subtract 1 from 4 in the hundreds place when you borrowed." Unfortunately Max's algorithm is to subtract the smaller digit in each column from the larger. Max doesn't have any idea what the instructor is talking about (he never "borrowed"!) and feels that he must be very stupid indeed not to understand. The instructor agrees with this assessment since none of his remediation has had any effect on Max's performance.

BUGGY, in its present form, presents instructors with examples of buggy behavior and provides practice in diagnosing the underlying causes of errors. Using BUGGY, the instructor gains experience in forming theories about the relationship between the symptoms of a bug and the underlying bug itself. This experience can also be cultivated to make instructors aware that there are methods or strategies that they can use to properly diagnose bugs. There are a number of strategy bugs that instructors may have in forming hypotheses about a student's misconceptions. The development of a good "troubleshooting" strategy by an instructor can avoid these pitfalls. A common mistake is to jump too quickly to one hypothesis. Prematurely focussing on one hypothesis can cause a teacher to be unaware that there are many competing hypotheses that are just as likely, or possibly more likely. A common consequence of this is that the instructor only generates

(11) In computer programming metaphors, this corresponds to the debugging activities of resubmitting the program and throwing the whole program away and starting over from scratch because the computer must have made a mistake.

problems for the student that confirm his own incorrect hypothesis! For example, one student teacher was given the initial example (A) (shown following) after which he proceeded to generate example problems:

A	B	C
19	23	81
+9	+6	+8
199	236	818

At this point, he concluded that the bug was "writes the bottom digit after the top number." But his hypothesis failed when he was given the first test problem:

$$\begin{array}{r} 8 \\ +12 \\ \hline \end{array}$$

to which he responded 812. The bug actually is that single digit operands are concatenated on the end of the other operand, so that the correct buggy answer is 128. By presenting only examples with fewer digits in the bottom number, he got only confirming evidence for his hypothesis.

In some cases, an instructor may believe his hypothesis so strongly that he will ignore disconfirmations that exist or decide that these disconfirmations are merely random noise.[12] One way this can be avoided is by using the technique of differential diagnosis [Rubin 1975] in which one always generates at least two hypotheses and then chooses test problems that separate them.

Another important issue concerns the relationship between the language used to describe a student's errors and its effect on what a teacher should do to remediate it. Is the language able to convey to the student what he is doing wrong? Should we expect instructors to be able to use language as the tool for correcting the buggy algorithms of students? Or should we only expect instructors to be able to understand what the bug is and attempt remediation with the student using things like manipulative math tools? The following are quotes of student teacher hypotheses taken from protocols of BUGGY, which give a good idea of how difficult it is to express procedural ideas in English. The descriptions in parentheses are BUGGY's (prestored) explanations of the bugs.

(12) There is, of course, some amount of "processor failure" as students are often all too human.

"Random errors in carryover." (Carries only when the next column in the top number is blank.)

"If there are less digits on the top than on the bottom he adds columns diagonally." (When the top number has fewer digits than the bottom number, the numbers are left-justified and then added.)

"Does not like zero in the bottom." (Zero from any number is zero.)

"Child adds first two numbers correctly then when you need to carry in the second set of digits child adds numbers carried to bottom row then adds third set of digits diagonally finally carrying over extra digits." (The carry is written in the top number to the left of the column being carried from and is mistaken for another digit in the top number.)

"Sum and carry all columns correctly until get to last column. Then takes furthest left digit in both columns and adds with digit of last carried amount. This is in the sum." (When there are an unequal number of digits in the two numbers, the columns that have a blank are filled with the left-most digit of that number.)

What does this say to us? Even when one knows what the bug is in terms of being able to mimic it, how is one going to explain it to the student having problems? Considering the above examples, it is clear that anyone asked to solve a set of problems using these explanations would no doubt have real trouble. One can imagine a student's frustration when the teacher offers an explanation of why he is getting problems marked wrong, and the explanation is as confused and unclear as these are. For that matter, when the correct procedure is described for the first time, could it too be coming across so unclearly?

This issue is further complicated by the existence of another important issue: there are fundamentally different bugs which cause identical behavior! In other words, there can be several distinct bugs that are logically equivalent and always generate the same "answers". For example, here is a set of problems:

38	186	298	89
+46	+254	+169	+64
174	2330	2357	243

The underlying flaw in the student's procedure (his bug) can be described as "The columns are added without carries and the left-most digit in the answer is the total number of carries required in the problem." In this case, the student views the carries as tallies to be counted and added to the left of the answer. But another equally plausible bug also exists; the student is placing the carry to the left of the next digit in the top number instead of adding it to the digit (i.e. he is actually carrying ten times the carry digit). This generates the same symptoms. So even when the teacher is able to describe clearly what he believes is the underlying bug, he may be addressing the wrong one. The student may actually have either one of these bugs.[13]

We feel that all of the issues discussed above are as important for students learning procedures as they are for teachers. In particular, the diagnostic task of a player requires studying the structure of the procedural skill per se as opposed to merely performing it. This can be especially important if we are trying to get students not to just rotely memorize the procedural skill but to encode it in some semantically meaningful way.

Another reason for having students develop a language for talking about procedures, processes, bugs, etc. is that this language enables the student to talk about (and think about) procedures and the underlying causes of his own errors. This is important in its own right, but it also gives a student the motivation and the apparatus for stepping back and critiquing his own thinking, as well as saying something interesting and useful about his errors. This is especially important given the fact that there's been so little success in getting students to look over their own work (such as estimating answers) and to use this perusal to good advantage.

---

(13) This leads to an interesting question concerning how one can "prove" two different descriptions of bugs entail logically the same surface manifestations.

An Experiment using BUGGY

We have conducted an experiment to explore BUGGY's impact on student teachers. In particular, we wished to answer the question of whether exposure to BUGGY significantly improves the student teachers' ability to detect regular patterns of errors in simple arithmetic problems. The subjects were twenty-four undergraduate education majors from Lesley College in Cambridge. They were all volunteers who were not paid for their services. The 24 subjects were divided into twelve groups of two each.

Their exposure to BUGGY lasted approximately one and a half hours with most teams completing at least six different bug sessions. Both addition and subtraction bugs were presented. The first two bugs each team encountered were chosen from a list of simple bugs so as not to compound difficulties the subjects faced in just getting used to using a computer terminal and to BUGGY.

The effects of their exposure to BUGGY were measured by comparing each subject's performance on pre- and post-exposure tests. There were two such tests, labelled Red and Blue. The twenty-four subjects were randomly assigned to two groups. One group had the Red test before exposure, and the Blue test after, and the other group had them in reverse order. Each test had ten items, each item consisting of a set of four simple addition or subtraction problems with their "solutions". Seven of the items in each test contained "patterned" errors, such that the four solutions could all be arrived at as a result of a single misapplied rule -- for example, failure to carry when a column adds to more than 10. The other three items were "random" items in which there was no single explanation for all of the errors. (See Appendix 1 for the Red test.) For the experiment, BUGGY was modified so that <u>no</u> subjects were given bugs that occurred on their post-tests.

Results

The raw data generated by the tests are shown in Table 1. The items across the top (1P,2P,3R...) indicate the problem number and whether the correct problem description was random (R) or could be explained by a single bug-description or pattern (P). The subjects' responses were scored and assigned to four categories: PC, PI, PW, R, plus one extra category of Not Attempted (NA). The first letter stands for the type of response the subject made where P=pattern, and R=random. The second letter is the quality of the explanation the subject made on that item: C=consistent or complete (the subject's single explanation explains all the errors), I=inconsistent (the subject's explanation is not contradicted by any of the problems but does not explain all errors), and W=wrong (the subject's explanation is contradicted by at least one of the problems). For the case of "R", Random-Consistent is implied.

[insert Table 1]

First, let us compare the results of Pre and Post tests, combining the results across the two groups of subjects and across the Red and Blue tests. The distribution of responses is shown in Table 2 together with values for Chi-squared.

[insert Table 2]

There was a significant improvement on the patterned items. The number of correct responses for patterns (PC) rose (p=0.048 by one-tailed binomial test). The number of pattern descriptions disconfirmed by one of the solutions it was supposed to describe (PW responses fell significantly (p=0.02 by one-tailed binomial test). The number of random (R) responses, where a patterned bug was incorrectly described as a random error, also fell (p=0.047 by one-tailed binomial test).

The results on the Random test items also showed improved performance after exposure to BUGGY, although they fail to reach significance. The number of Random (R) responses for random items increased; the number of Pattern responses contradicted by at least one of the examples (PW) decreased; and the number of items not attempted (NA) fell, suggesting that speed increased slightly. (Almost all of the reduction in the number not attempted occurred on the final random items which were the last item in the Red test, and the next to last in the Blue test.) The number of pattern-inconsistent (PI) responses increased slightly in both patterned and random items, suggesting that the exposure to BUGGY increased the subjects' sensitivity to the presence of patterning.

TABLE 1

SUBJECT	RED PRE-TEST 1P	2P	3R	4P	5P	6P	7R	8P	9P	10R	BLUE POST-TEST 1P	2P	3P	4P	5R	6R	7P	8P	9R	10P
1	PC	R	PI	R	R	PI	R	R	PI	R	PC	PC	PC	PC	R	R	R	PC	PI	NA
2	PC	NA	PI	NA	PC	NA	NA	PW	NA	NA	PC	R	PW	R	NA	PI	PC	PC	PI	PC
3	PC	PI	R	R	PW	PW	NA	NA	NA	NA	PC	PC	R	PW	R	R	PC	PC	NA	NA
4	PC	PW	PI	R	NA	PI	NA	NA	NA	NA	PC	PC	PW	PC	R	NA	PC	PC	PI	NA
5	PI	PW	PI	PC	PW	PI	PI	PC	PW	PI	PI	PW	PI	PW	PI	PW	PI	PI	PI	PC
6	PC	PC	R	R	NA	PC	NA	NA	NA	NA	PC	PC	PI	NA	PI	NA	PC	PI	NA	NA
7	PI	NA	PI	R	PC	PI	NA	PW	PI	NA	PW	PI	PW	PC	R	NA	PC	PI	PI	R
8	PC	NA	NA	PC	NA	PI	NA	NA	NA	NA	PC	PC	NA	PC	NA	NA	PC	PW	NA	NA
9	PC	PC	NA	NA	PC	PC	NA	NA	NA	NA	PC	NA	PC	NA	NA	NA	PC	NA	NA	NA
10	PC	PC	R	R	PC	PC	NA	NA	NA	NA	PC	PC	PC	R	NA	NA	PC	NA	PI	NA
11	PC	NA	NA	R	NA	PC	R	NA	NA	NA	PC	PC	NA	NA	R	R	PC	PI	PI	NA
12	PC	R	R	PW	NA	PI	NA	NA	NA	NA	PC	PI	NA	PW	NA	NA	PC	NA	NA	[illegible]

SUBJECT	BLUE PRE-TEST 1P	2P	3P	4P	5R	6R	7P	8P	9R	10P	RED POST-TEST 1P	2P	3R	4P	5P	6P	7R	8P	9P	10R
13	PC	R	PW	PC	NA	NA	PC	PC	PI	NA	PI	NA	PI	PW	PC	PI	NA	NA	NA	PI
14	PC	R	PI	PC	NA	NA	PC	PC	PI	PI	PI	PI	PI	PW	NA	PI	NA	PC	NA	PI
15	PC	PW	NA	PC	R	NA	NA	R	PW	NA	PC	PC	PI	PC	NA	PI	PW	NA	R	PI
16	PC	NA	NA	PI	NA	NA	NA	NA	NA	NA	PW	PC	R	PC	NA	NA	NA	NA	NA	NA
17	PI	PW	NA	PW	R	NA	PC	PI	NA	NA	PC	PC	R	PC	PI	NA	NA	PW	NA	PI
18	PC	PW	PW	PW	PI	PI	PC	PW	NA	NA	PC	PC	R	PC	PI	PI	PI	PI	PI	NA
19	PC	NA	PW	PW	NA	PW	NA	NA	NA	NA	PC	NA	NA	NA	NA	PC	NA	NA	NA	NA
20	PC	PC	R	PC	PI	NA	NA	PW	NA	R	PC	PC	NA	PC	NA	NA	NA	PC	NA	R
21	PC	PI	PW	PW	NA	NA	PC	PW	NA	NA	PC	PC	NA	PC	NA	PC	NA	NA	NA	R
22	PC	PW	PW	PC	PW	PI	PC	PI	PI	PC	PI	NA	PI	PC	PC	PC	PI	PC	PC	PI
23	PC	NA	PC	NA	NA	PI	PC	PC	NA	NA	PC	PC	PI	PC	NA	PC	NA	NA	NA	NA
24	PC	PC	PW	NA	NA	PW	PC	PW	NA	NA	PC	PC	NA	PC	NA	PI	NA	NA	PI	NA

TABLE 2

Response	Patterned Items Pre-Test	Patterned Items Post-Test	Random Items Pre-Test	Random Items Post-Test
PC	55	75	-	-
PI	18	24	15 *	22 *
PW	27	13	4 *	2 *
R	16	7	9	13
NA	52	49	44	35
X^2	12.45		2.33	
DF	4		2	
P	P<0.02		N.S.	

*Combined for Chi-Square test

The foregoing conclusions depend on two assumptions implicit in the experimental design: that the two groups of subjects were equivalent, and that the Red and Blue tests were equivalent. To confirm that the two groups of subjects were equivalent, the responses obtained in the Pre-tests were combined with those from the Post-tests, for each group, as shown in Table 3.

[insert Table 3]

The two groups yielded very similar distributions of responses for both Patterned and Random items. The differences are not significant by Chi-squared test, and a large portion of the obtained Chi-squared values derive from the difference in the number of Random responses between the two groups, which appears in both the Patterned and in the Random test items.

The second assumption is that the Red and Blue tests are equivalent. The Pre- and Post-test responses are combined separately for the Red and Blue tests in Table 4.

[insert Table 4]

There is no difference between the two tests in the Random items, but the patterned items were significantly easier in the Blue test than in the Red test. The number of correct responses was greater for the Blue test, and the number not attempted was smaller, though neither difference is significant by one-tailed binomial test. On the other hand, there were significantly more internally-inconsistent errors (PW) on the Blue test (p=.04 by two-tailed binomial). This difference between the Red and Blue tests is unimportant as long as the pattern of differences is similar for both the Pre-test and the Post-test. Table 5 shows the distribution of responses to Patterned test items for Red and Blue tests separately for Pre-exposure and for Post-exposure applications. (Note that different groups of subjects are involved, so the validity of the conclusions depends on our earlier finding of no difference between the two groups.)

[insert Table 5]

TABLE 3

	Patterned Items S1-S12	Patterned Items S13-S24	Random Items S1-S12	Random Items S13-S24
PC	64	66	-	-
PI	21	21	17 }*	20 }*
PW	18	22	1 }	5 }
R	17	6	15	7
NA	48	53	39	40
X^2	5.94		4.06	
D.F.	4		2	
P	$P \simeq 0.2$		$P \simeq 0.2$	

*Combined for Chi-Squared test

TABLE 4

Response	Patterned Items		Random Items	
	Red	Blue	Red	Blue
PC	59	71	-	
PI	25	17	19	18
PW	13	27	1	5
R	12	11	12	10
NA	59	42	40	39
X^2	10.44		0.40	
DF	4		2	
P	$.02<P<.05$		$.90<P<.80$	

*Combined for Chi-Squared test

36

Table 5

Response	Pre-exposure		Post-Exposure	
	Red	Blue	Red	Blue
PC	24	31	35	40
PI	11	7	14	10
PW	9	18	4 }*	9 }*
R	11	5	1 }	6 }
NA	29	23	30	19
X^2	7.72		8.47	
DF	4		3	
P	$P \simeq 0.1$		$.02 < P < .05$	

*Combined for Chi-Squared test.

An inspection of Table 5 shows that the difference between the two tests is very similar for the Pre- and Post-exposure applications (with the single exception of the Random responses) and is certainly not large enough to cast doubt on the main conclusion. We can, therefore, conclude that exposure to BUGGY significantly improved the subjects' ability to detect regular patterns of errors in simple arithmetic problems.

Qualitative Impressions

The next question to be investigated concerned the issue of what the subjects (student teachers) themselves felt they gained from their exposure to BUGGY. In order to assess their impressions, we convened the entire group during the evening when they had finished using BUGGY. At that gathering, we first asked them to write their responses to two questions (discussed below) and then taped a final group discussion in which we sought their reactions to BUGGY, and their suggestions for its deployment with school-aged students. The following week, their professor held a similar group discussion (he also participated in the initial experiment) and reported back to us the consensus, which was consistent with what they had written.

Appendix 2 lists all the written responses to the question "What do you think you learned from this experience?". All 24 responded that they came away with something valuable. Many stated that they now appreciated the "complex and logical thought processes" that children often use when doing an arithmetic problem incorrectly. "It makes me aware of problems that children have and they sometimes think logically, not carelessly as sometimes teachers think they do." "I never realized the many different ways a child could devise his own system to do a problem." They also stated that they learned better procedures for discovering the underlying bug -- "I learned that it is necessary to try many different types of examples to be sure that a child really understands. Different types of difficulties arise with different problems." Several stated their mixed feelings about working with a computer. "Trying to beat the machine can be challenging." "I learned that computers are a very complicated piece of machinery. If one isn't experienced with the mechanism, then problems could result." And finally, "The types of analyses necessary to 'debug' student errors on the test (paper/pencil) seems more difficult than with the computer. But that doesn't make any sense. The 'analysis' ought to be the same. Perhaps the computer motivated my analytical ability."

Appendix 3 lists all written responses to the question "What is your reaction to BUGGY?" Many felt that "BUGGY could be used to sharpen a teacher's awareness of different difficulties with addition and subtraction." They felt that it might be of use in grade school, high school, or with special needs students, or even as a "great experience in beginning to play with computers."

Conclusion and Extensions

Although our experience shows that student teachers learn a significant amount from their use of BUGGY, the system should still be substantially extended. In particular, most of what the students learned while using BUGGY they learned or discovered, in some sense, on their own. BUGGY does no explicit tutoring. It simply challenges their theories and encourages them to articulate their thoughts.[14] The rest of the learning experience occurred either through the sociology of team learning or from what a person abstracted on his own. The next step in improving the educational effectiveness of BUGGY is to (1) implement an intelligent tutor to critique the example test problems the students create, (2) point out interesting facets of their debugging strategies and (3) isolate manifested weaknesses in their strategies. Our experience indicates that such a tutor would be very helpful in that it could keep students from getting caught in unproductive ruts and could help focus their attention on the structure of the procedures themselves.

---

(14) As a historical footnote, BUGGY was originally developed to explore the psychological validity of the procedural network model for complex procedural skills. During that investigation we realized the pedagogical potential of even this simple version of BUGGY as an instructional medium. More recent versions of this system have stressed instructional aspects by adding such features as assigning "costs" to student generated test cases, thereby encouraging him to optimally formulate and test his hypothesis.

Along these same lines, the "expert" portion of the procedural net should be made "articulate" in the sense of being able to explain and justify the subprocedures it uses. This would allow a student to pose a problem to the system and obtain a running account of the relevant procedures as the "expert" solves the problem.

Another area for extension concerns the psychological validity of the skill decomposition (and buggy variants) in the procedural network. Determining the proper functional breakdown of a skill into its subskills is critical to the psychological validity of the model and the resulting behavior of the system. If the breakdown of the skill is not correct, bugs that people would consider simple may be difficult to model, while those suggested by the model may be judged "unrealistic". From the network designer's point of view this leads to the issue of choosing or constructing one structural decomposition instead of another. We are just beginning to acquire a large data base of arithmetic errors from Stanford [Searle 1976] and will be testing to see how well our diagnostic model accounts for all of them. In particular, we are concerned not only with how many underlying bugs our current model captures, but also how many bugs our network predicts that never show up. A more subtle issue concerns the validity of the actual functional decomposition of the skills in the network. Measuring the "correctness" of a particular network is a problematic issue as there are no clear tests of validity, but issues such as the ease or "naturalness" of inclusion of newly discovered bugs and the appearance of combinations of bugs within a breakdown can be investigated.

We are also in need of a theory which explains what makes an underlying bug easy or difficult to diagnose. Simple conjectures concerning the depth of the bug from the surface don't seem to work, but more sophisticated measures might. It's hard to see how to predict the degree of difficulty in diagnosing a particular bug, without a precise information processing or cognitive theory of how people actually formulate conjectures about the underlying bug or cause of an error.

Finally, we note that we have left open the entire issue of a semantic or teleological theory of how bugs are generated in the first place. The need for such a theory is important for at least two reasons. First it could provide an interesting theoretical mechanism that would account for the entire collection of empirically arrived at bugs, and second, it provides the next step in a semantically based productive theory of student modelling.

CHAPTER 2

AUTOMATED PROTOCOL ANALYSIS - A TECHNIQUE FOR MODELLING AND MEASURING STUDENT PERFORMANCE[15]

SECTION I

The persistent theme throughout our research has been that for intelligent CAI programs to successfully tutor a student, they must be able to induce a model of the student's current knowledge and preferred interaction modes. Otherwise, computer-based tutors, regardless of the power of their embedded expert, risk transactions with the student that are inappropriate or annoying.

To address this student modelling problem, one must have some means for making hypotheses regarding the student's knowledge. The previous chapter described such a technique, namely diagnostic models built around procedural networks. This chapter discusses another technique that augments the previous one, and, unlike the previous one, assumes that the main source of data available to the ICAI tutor is the student's problem solving protocol or trace (as opposed to just his answer). This chapter proposes a theory and a computational approach for automating the protocol analysis task for the purpose of automatically inducing a structural model of the student's problem solving strategies. It then discusses the design of a computer system, named PAZATN, for carrying out this task.

In addition to providing us with a powerful technique for discovering a student's underlying reasoning strategies, automated protocol analysis also serves as a new means of measuring and testing the tutor's success. With it, we can determine whether successive protocols reflect improved problem solving competence on the part of the student. It can provide rigorous measures of the virtues of alternative tutoring modes. Finally, protocol analysis can also serve as a diagnostic tool for discovering gaps in the knowledge of a practicing problem solver and

(15) A substantially modified version of this chapter is appearing as a working paper by Goldstein and Miller.

direct a computer based assistant's attention to those areas that require assistance and review (e.g. an adaptive Job Performance Aid).

In designing such an automated protocol analysis system, we have drawn on concepts and algorithms from computational linguistics. While the protocols we consider relate to problem solving behavior, and not linguistic interactions, we nevertheless believe that there is a fruitful synergy between the concepts developed in the language understanding arena and the problems of ICAI.

Technical Statement of the Problem

Protocol analysis assigns one or more theoretical interpretations to a record of a subject's overt behavior on a problem solving task. Our concern is with problem solving tasks in which a student or subject interacts with an on-line computer terminal. For such tasks, the behavioral record is the sequence of keystrokes from the console session. The keystrokes are grouped into events, which are treated as unitary input/output transactions. An advantage over the most general analysis situation is gained by assuming that the dialogue occurs within the confines of a well-defined finite "menu" of legal responses. Our primary concern is to account for problem solving behavior; we do not attempt to solve the natural language understanding problem as a subprocedure.

For the purposes of this discussion, an interpretation is a structural description of the list of events, augmented by an assignment of values to a set of semantic context variables, and a set of pragmatic assertions, associated with each node of the description. The semantic variables and pragmatic assertions relate the subgoal structure of the problem solving protocol to the model, a formal description of the task to be accomplished. In applications of automatic protocol analysis, it is common to assume the existence of this formal problem description. It is not assumed that the student has internally represented the task in precisely the same fashion. These definitions are elaborated in section two.

In order to impose realistic bounds on the specification of the analyzer, it is also assumed that the protocol is "reasonable." That is, the protocol should represent a sincere attempt to solve the problem at hand, and should terminate exactly when this goal has been accomplished. Although "reasonable" is difficult to define more precisely, PAZATN's sensitivity to this assumption will be made clear in the ensuing discussion.

Determining the Validity of Theoretical Interpretations

The validity of the interpretations assigned by the analyzer may be ascertained in a variety of ways. Our philosophy is to utilize every available source of evidence. Since the synthetic problem solver employs identical descriptions, its heuristic adequacy is taken as suggestive, though by no means decisive, evidence. Introspection by human problem solvers is another source of weak confirming evidence.

The analyzer's ability to predict future behavior on the basis of past performance will provide the strongest corroboration. No formal experimentation has been carried out to date. Our plan is to employ the finished system for this type of rigorously controlled experimentation. Ultimately we hope to embed such analyzers in computerized tutors. This is an ambitious undertaking. When a prototype is available, though, the pedagogical efficacy of that system will provide a further check.

Review of the Synthetic Theory

Before examining the analyzer in detail, it will be helpful to briefly review the synthetic theory. The basis for the approach is a hierarchical classification of commonly observed planning and debugging techniques. According to the planning theory, when the problem solver confronts a problem, there are three major categories of plans which may be pursued. The problem may be solved by identification, that is, by recognizing it as a problem for which a solution already exists in some answer library. This type of plan may seem a bit trivial, but of course it is absolutely essential to avoid infinite regress.

Alternatively, the problem may be solved by decomposition, that is, by subdividing it into smaller, easier subproblems. These are each solved separately (by recursively calling the problem solving system), and then recombined in one of several specific ways, to produce a solution to the original problem.

If these strategies fail to produce a solution, the problem may be solved by reformulation, that is, by redescribing the goal in other terms which seem more amenable to solution. The reformulated problem must, of course, still be solved itself (recursively calling the problem solving system) by identification, decomposition, or further reformulation.

Each of these categories of planning concepts is further subdivided by the theory, as illustrated by Figure 1. Identifications may be accomplished by retrieval from a lexicon of primitive operations for the task domain, or by retrieval from an extensible answer library. Decomposition may be performed by Conjunction or by Repetition (among others). Reformulation may involve Equivalent models or Simplifications. Each of these, in turn, is elaborated still further.

The taxonomy is transformed into a procedural problem solver in the following manner. In order to represent semantic information, a finite set of registers is defined. These are used for storing flags and structures resulting from intermediate steps of the computation. At this point, the taxonomy can be thought of as a highly non-deterministic decision tree.

In order to increase the system's determinism, the nodes and links of the tree are taken to be the states and arcs of a recursive transition diagram. Arbitrary conditions over the contents of the registers are associated with the arcs, as preconditions for following them. Finally, arbitrary structure-building and register-setting actions are associated with the arcs, to be performed whenever they are followed.

Figure 1. The Planning Taxonomy

- PLAN
  - IDENTIFY
    - PRIMITIVE
      - SIN
      - COS
      - EXP
      - POLYNOMIAL
    - ANSWER LIBRARY
  - DECOMPOSE
    - CONJUNCTION
      - SUMS-RULE
      - CONSTANT-FACTORS
      - INT-BY-PARTS
      - PARTIAL-FRACTIONS
    - REPETITION
      - EXTENDED INT-BY-PARTS
  - REFORMULATE
    - EQUIVALENT
      - SUBSTITUTION
      - PYTHAGOREAN-RELATION
    - SIMPLIFICATION

For efficiency, some states with similar topology are merged, and a few additional arcs are added to provide for such features as iterative control, when recursively invoking the complete problem solver is unnecessary. Although we allow arbitrary conditions and actions, these are not chosen arbitrarily, but are carefully selected to reflect the semantics and pragmatics of the problem solving process.

The result of this metamorphosis is PATN's synthetic augmented transition network displayed in Figure 2.[16]

PATN has a particularly interesting property from the standpoint of protocol analysis. It views certain types of errors (bugs) as rational, in that they result from heuristically sound planning choices made in the absence of complete information, and is capable of producing partial solutions (i.e., traversing intermediate states) containing bugs of this type.

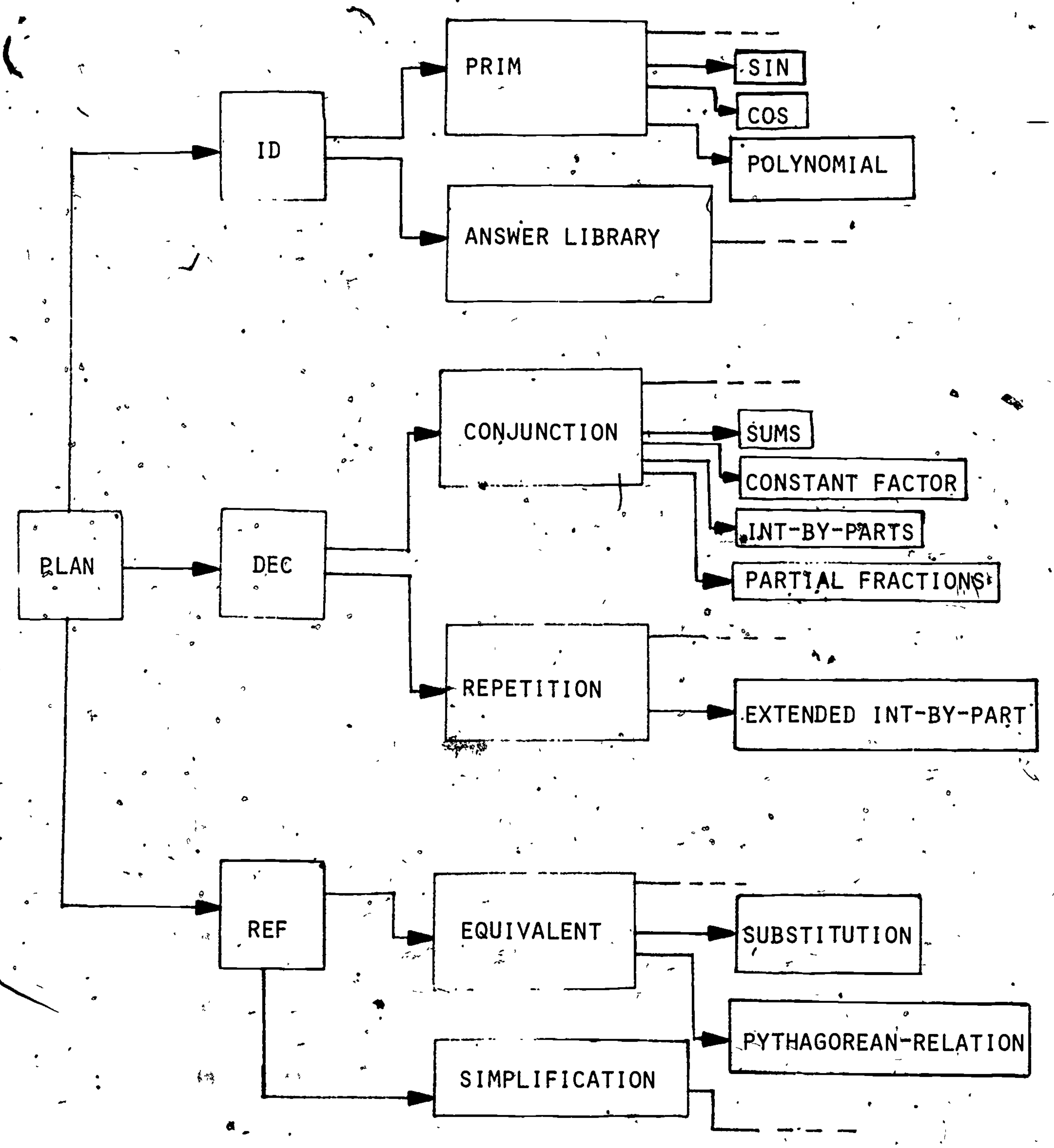
Design Considerations

A major insight of generative grammarians (e.g., Chomsky [1965]) was that in characterizing a set of phenomena, it is often helpful to conceptualize the formalism synthetically, and to view analysis as a process of inverting synthetic rules. Equivalently, analysis may be described as the selection of one or more plausible derivations from a potentially infinite collection of synthetic possibilities. In designing PAZATN, we have found it enlightening to view protocol analysis as parsing in this sense, where PATN is taken as the generative formalism.

Since the space of synthetic possibilities (both in language processing and in problem solving) is potentially infinite, it is critical that this space be characterized using a finite (reasonably small)

(16) PATN is an expert problem solving system, designed by Miller and Goldstein [1976] in which planning knowledge is modeled using augmented transition networks [Woods 1970]. This system serves as the cornerstone of a grammatical theory of problem solving which can act as a formalism for representing the knowledge of our Articulate Expert for mathematics and some aspects of electronics.

Figure 2. Planning ATN for Symbolic Integration



set of rules. In PATN, these rules take the form of an ATN. This is somewhat unusual, since in computational linguistics the ATN is commonly thought of as an efficient mechanism for inverting transformational rules, i.e., for analysis. PATN's synthetic ATN is a generator for the space of plans and debugging techniques which are relevant to the problem at hand.

Naturally, PAZATN is not prepared to understand protocols which PATN could not be made to generate eventually. The one exception to this is that buggy versions of various synthetic plans (including irrational bugs which would not be introduced by PATN) can often be recognized. Since PATN is presumably an effective procedure within its domain of competence, the analysis could, in principle, be performed by exhaustively enumerating the set of synthetic protocols, and selecting the first one which matches the input data. Unfortunately, this would take considerable time. Consequently, the primary consideration in the analyzer's design must be to ensure that this synthetic plan space is searched efficiently. Bottom up evidence from the actual protocol is used for this purpose.

An important design consideration is that the analyzer be able to take full advantage of the available sources of constraint. The protocol analyzer has access to an unusually strong set of expectations, namely the model. This is analogous to knowing the "gist" of what a speaker is going to say before parsing it. Consequently, the analyzer must be organized in such a manner that it is able to extensively utilize the top down synthetic guidance which can be provided by PATN.

This might suggest a design based on using PATN as a purely top down predictive analyzer. The difficulty is that, while we know the "gist" of the input, there is a tremendous diversity of potential realizations of a given model in terms of the form of the solution. So it is more like knowing the "theme" of a story, but not knowing whether the author will present the events in chronological order, via flashbacks, or in an order derived from some other organizing principle. The unguided PATN could generate scores of irrelevant synthetic solutions before stumbling upon one that matched the data. This factor leads to a somewhat elaborate dual organization for the analyzer, which enables it to reduce the diversity by considering bottom up evidence as well.

Another difficulty which must be faced, if PAZATN style analyzers are to be viable for eventual dynamic use in computerized tutoring, is that events must be examined in a single pass, in approximately left to right order. One could postpone this issue temporarily, but such a simplification might result in a design which could not be extended for applications because of fundamental, premature commitments. If the analyzer is forced to back up frequently, over many events, it is often likely to find itself "apologizing" for inappropriate tutorial remarks regarding prior events. Consequently, it must carry along any plausible alternative interpretations in parallel, until it has a clear basis for ruling them out. Conversely, the analyzer must have some capability for restricting the set of alternatives under active consideration, to ensure that excessive processing and storage resources are not consumed by low plausibility interpretations.

The organization of the protocol analyzer is a generalization and elaboration of the coroutine search plan-finding procedure used by Mycroft [Goldstein 1974, 1975]. The differences arise mainly from the need to take account of the considerations mentioned above. In particular, the protocol analyzer is intended to: (a) apply to more than a single task domain; (b) understand a wider range of event types (e.g., Mycroft was designed to analyze finished computer programs rather than protocols); (c) reap maximum advantage from the dynamic information available in the protocol regarding subgoal structure and development; and (d) embody the more coherent structured planning and debugging theory underlying PATN.

Overview

The PAZATN protocol analyzer is constructed on PATN's synthetic foundations by supplementing the synthetic ATN with a number of additional modules and data structures. One data structure is used to keep track of the set of plausible subgoals which have been proposed by PATN. Another is used to record the state of partially completed interpretations of the protocol. A preprocessor module is used to suppress uninteresting syntactic details and to perform preliminary segmentation. The preprocessor employs an event classifier to determine the syntactic class of each event of the protocol. Corresponding to each syntactic category, PAZATN must be supplied with an event specialist which embodies the requisite domain knowledge for assisting an event interpreter in associating an event of that type with some synthetic subgoal. Since a purely top down or bottom up strategy would be too inefficient, a scheduler module is necessary to direct the analyzer through a "best first" coroutine search.

Section two elaborates our notion of protocol analysis as a parsing process analogous to the natural language processing task. The third section provides a slightly simplified description of the organization of the automatic protocol analyzer. Section four refines this first order description of PAZATN's design. Finally, we present our tentative conclusions and plans for future work.

SECTION II

A GRAMMATICAL APPROACH TO PROTOCOL ANALYSIS

This section addresses the question: "What is it about PAZATN's approach to protocol analysis that makes it grammatical?" Central to the approach is the conjecture that various aspects of problem solving behavior can be studied approximately independently. Consider the underlying problem solver (i.e., the subject) whose behavior is to be analyzed. While we conceive of this problem solver as being an integrated procedural system, we nevertheless suppose, at least as a research strategy, that certain aspects may be factored out for separate study: the structural component, the semantic component, and the pragmatic component. These correspond, respectively, to the potential control paths, data flow, and branching conditions of a procedural problem solver. These aspects are modelled by the network of states and arcs, the registers, and the transition conditions of the augmented transition network. The next sub-section introduces an example protocol in order to illustrate PAZATN's analysis.

An Example Problem Solving Protocol

In this sub-section we provide a brief example of the type of problem solving protocol which PAZATN is to analyze, and the sort of analysis which it would provide. Imagine a situation in which a student (S) is interacting with a computerized educational environment such as SOPHIE. Suppose S is confronted with the the following problem:

> In an electrical circuit, the voltage at time "t" is given by
>
> $e(t) = r.\sin(wt)$,
>
> where r and w are arbitrary constants. Find the root-mean-square voltage for the time interval [a,b].

A segment from a hypothetical protocol, representing S's solution path on this problem, is shown in Figure 3. Before delving into the details of PAZATN's analysis, we provide an informal account of the student's solution.

The student was familiar with the definition of root- mean-square voltage, and hence began the protocol by writing down the relevant formula.

$$\text{E01:} \quad V_{rms}^{[a,b]} = \sqrt{\frac{1}{b-a}\int_a^b [e(t)]^2\,dt}$$

Figure 3 The Example Protocol Segment

E01: $$V_{rms}^{[a,b]} = \sqrt{\frac{1}{b-a}\int_a^b [e(t)]^2\,dt}$$

E02: $$= \sqrt{\frac{1}{b-a}\int_a^b [r^2 \sin^2(wt)]\,dt}$$

E03: $$= \int r^2 \sin^2(wt)\,dt$$

E04: $$= r^2 \int \sin^2(wt)\,dt$$

E05: $$= \int \sin^2(t)\,dt$$

E06: $$= \int u^2\,du$$

E07: $$= \frac{u^3}{3}$$

E08: $$= \frac{\sin^3(t)}{3}$$

Figure 3 The Example Protocol

E09: $\sin^2(t)\cos(t)$

E10: $\int \frac{u^2 du}{\cos(t)}$

E11: $\cos(t) = \sqrt{1 - \sin^2(t)}$

E12: $\int u^2[(1 - u^2)^{-1/2}]\, du$

E13: $\int \sin^2(t)\, dt$

E14: let $U = \sin(t)$, $dv = \sin(t)\, dt$

E15: $du = \cos(t)\, dt$, $v = -\cos(t)$

E16: $\int \sin^2(t)\, dt = -\sin(t)\cos(t) + \int \cos^2(t)\, dt$

E17: $\int \cos^2(t)\, dt = \int 1\, dt - \int \sin^2(t)\, dt$

E18: $2\int \sin^2(t)\, dt = t - \sin(t)\cos(t)$

Next, S substituted the particular definition for e(t) provided by the current problem statement.

$$E02: \quad = \sqrt{\frac{1}{b-a} \int_a^b [r^2 \sin^2(wt)]\, dt}$$

This resulted in a problem whose essence is integrating the function $\sin^2$. Some students might have remembered the formula for this indefinite integral, in which case the solution would have been straightforward. In this case, S knew only a few simple integrals and a few basic rules for decomposing complex integrals into simpler ones. In the next step S focused on this integration task.

$$E03: \quad = \int r^2 \sin^2(wt)\, dt$$

Then S applied the "sum of integrands" rule, eliminating the r^2 term.

$$E04: \quad = r^2 \int \sin^2(wt)\, dt$$

As a simplification, S decided to ignore the w term in the argument to the sin function.

$$E05: \quad = \int \sin^2(t)\, dt$$

At this point, S attempted to apply the substitution, u = sin(t), hoping to convert the integrand to a polynomial, one of the primitive integrals which was known. However, the student committed the common error of failing to substitute for the differential term.

E06: $= \int u^2\, du$

In a sense, the bug was fortuitous, since it converted the integrand to a simple polynomial.

E07: $= \frac{u^3}{3}$

The final step of S's substitution plan was to re-substitute for the temporary variables, restoring the solution to include only those terms which were mentioned in the original problem statement.

E08: $= \frac{\sin^3(t)}{3}$

At this point, S became suspicious of the substitution -- the result seemed too simple. As a check on its validity, S differentiated the expression.

E09: $\sin^2(t)\cos(t)$

Here, S realized the mistake in E06, and re-executed the substitution. This time S correctly substituted for the differential term, except that the expression used was still in terms of t, not u.

E10: $\int \frac{u^2\, du}{\cos(t)}$

The appropriate next step is to rid the expression of t. S accomplished this using the pythagorean relation.

E11: $$\cos(t) = \sqrt{1 - \sin^2(t)}$$

E12: $$\int u^2[(1 - u^2)^{-1/2}]\,du$$

Actually, at E12, S has derived the canonical $u = \sin(t)$ substitution formula. However, the resulting subproblem was also unfamiliar. It did not appear to S to be sufficiently simpler than the original problem.

The substitution plan therefore failed to produce the desired result. Hence, S retreated to the $\sin^2(t)$ formulation, and tried a new approach - integration by parts.

E13: $$\int \sin^2(t)\,dt$$

E14: let $U = \sin(t)$, $dv = \sin(t)\,dt$

E15: $du = \cos(t)\,dt$, $v = -\cos(t)$

E16: $$\int \sin^2(t)\,dt = -\sin(t)\cos(t) + \int \cos^2(t)\,dt$$

Integration by parts resulted in what appears, at first, to be an equally hard problem - integrating $\cos^2(t)$.

E17: $$\int \cos^2(t)\, dt = \int 1\, dt - \int \sin^2(t)\, dt$$

But once again, the student applied the pythagorean relation, this time leading to an equation which did allow solving for the desired integral.

E18: $$2\int \sin^2(t)\, dt = t - \sin(t)\cos(t)$$

Event 18 still does not represent a complete solution to the original problem. S might still have forgotten, for example, to correct for the simplification introduced at event E05, or might have incorrectly evaluated the limit terms for the definite form of the integral. However, this segment of the protocol is sufficient to serve as our example of the form of PAZATN's analysis.

Structural Descriptions

The result of PAZATN's protocol analysis is a set of data structures representing these several aspects of the problem solving behavior. The first is a description of the subgoal structure of the protocol. This data structure is similar to the context free deep structures (or base components) of natural language parsing. It summarizes the arc transitions which presumably were followed by the generating ATN. The set of legal structural descriptions may be characterized by a context free grammar. To apply PAZATN to a wide range of protocols, a thorough analysis of the specialized problem-decomposition techniques relevant to the particular domain is necessary. The reduced grammar illustrated in Figure 4 is adequate for analyzing the subgoal structure of the segment of protocol introduced above. While this grammar is typical of the sort we envision, by no means does it represent a complete task analysis.

Figure 4. The Context Free Grammar

```
SOLVE          -> PLAN + [DEBUG]
PLAN           -> IDENTIFY | DECOMPOSE | REFORMULATE
IDENTIFY       -> PRIMITIVE | ANSLIB
PRIMITIVE      -> SIN | COS | EXP | POLY | ...
DECOMPOSE      -> CONJUNCTION | REPETITION
CONJUNCTION    -> INT-BY-PARTS | PARTIAL-FRACTIONS |
                      SUM-RULE | CONSTANT-FACTOR | ...
REPETITION     -> EXTENDED-INT-BY-PARTS | ...
REFORMULATE    -> EQUIVALENT | SIMPLIFICATION
EQUIVALENT     -> SUBSTITUTION | PYTHAGOREAN-RELN
DEBUG          -> <[DIAGNOSE] + [REPAIR]>*
DIAGNOSE       -> D-PLAN | D-PROCEDURE | D-MODEL | D-PROCESS
D-MODEL        -> CHECK-DERIVATIVE
REPAIR         -> EDIT | SOLVE
```

Figure 5 indicates the structural description of this protocol which PAZATN is intended to produce. Such structural descriptions capture one aspect of problem solving behavior. They can be used to provide formal answers to certain questions which heretofore might have been discussed only in a more intuitive way. As an example, the parse tree makes it apparent, by inspection, that the student is comfortable with integration by parts; however, the incorrect first attempt to use substitution, and the subsequent failure to apply it on a second appropriate occasion (at E12), provide evidence that this student requires additional practice using substitutions.

Semantics and Pragmatics

Although the sort of description discussed in the previous section is useful for answering certain questions, it does not tell the whole story. Even to make such structural descriptions intelligible to the reader, it is necessary to provide some semantic and pragmatic commentary. The synthetic theory of planning and debugging provides the basis for more complete and precise semantic and pragmatic annotation.

Semantic annotation is defined to be the values of the ATN registers associated with each node of the structural description. These relate the behavior to the formal problem description. Pragmatic annotation is defined to be a record of the justifications for selecting a given arc transition rather than its competitors. In analysis, this pragmatic annotation is a hypothesis about the subject's reasons for using a particular approach. These hypotheses are based on both PATN's arc conditions (when the recommended synthetic transitions have been made) and heuristic inferences from the available data.

The following is a typical set of registers which would be employed by PATN to define the semantic context of a node in the problem solving tree. Some of these are not "primitive," since they are derivable from one or more of the others. It is possible that additional

Figure 5. Structural Description of the Example Protocol

```
SOLVE(integrate r²sin²(wt)) ;top level of integration task
    PLAN
      DECOMPOSE
        CONJUNCTION
          CONSTANT-FACTOR ;r²  E04
          INTEGRAL-TERM
            SOLVE   (integrand = sin²(wt))
              PLAN
                REFORMULATE
                  SIMPLIFY ;ignore w
                    E05
                    SOLVE (integrand = sin²(t))
                       PLAN
                         DECOMPOSE
                            REFORMULATE
                              SUBSTITUTION (u = sin(t)) ... E06
                                SOLVE
                                  PLAN
                                    IDENTIFY
                                      PRIMITIVE ... E07
                                RESTORE-INITIAL-TERMS
                                    E08
                       DEBUG
                         DIAGNOSE
                           D-MODEL
                             CHECK-DERIVATIVE
                               E09
                         REPAIR ;first attempt fails
                             EDIT ... E10
                                  SOLVE ... REF ... PYTH, E11, E12
                         REPAIR
                             SOLVE ; for the sin²(t) integral again
                                 PLAN
                                   DECOMPOSE
                                      INT-BY-PARTS (u = sin(t))
                                        E14, E15, E16
                                        SOLVE (integrand = cos²(t))
                                          PLAN
                                            REFORMULATE ...
                                              PYTH, RELN, ... E17, E18
```

semantic variables may be added in future research, perhaps in tailoring PATN to particular domains. The list below is adequate for our current purposes.

1. ?TREE is that part of the parse tree attached to the current node ("below" it).

2. ?PROCEDURE is the terminal solution procedure as defined so far. This reflects the state of the plan after any debugging events have been taken into account.

3. ?EFFECT is a domain-oriented description of the actual performance obtainable by the solution as defined so far. Since a partially solved problem may contain references to currently unsolved subgoals, ?EFFECT may be unassigned at a given node.

4. ?PROTOCOL is the "fringe" of ?TREE. That is, it is the list of terminal events dominated by a given node.

5. ?PLAN is a collapsed version of the subtree associated with ?PROCEDURE. That is, ?PLAN corresponds to the notion of the plan of a finished solution. The concept of collapsing a parsed protocol into a plan is elaborated in other reports by the authors.

6. ?MODEL is the set of predicates which ?PROCEDURE is intended to accomplish. For a correct solution ?EFFECT will be a special case of ?MODEL.

7. ?ADVICE is a list of planning and debugging suggestions generated by the synthetic pragmatics of PATN. For example, in solving a novel integral by partial fractions, when it is not known for certain whether such a decomposition is valid, a record of the fact that the partial fractions arc transition may have been inappropriate, is appended to the current contents of ?ADVICE. This helps to guide the debugging component in

diagnosing the underlying cause of later model violations.

8. ?TITLE is the symbolic name of the solution currently being developed. This aids in the detection of self-referential (recursive) plans. An example of its use in the example protocol is when the integration-by-parts led to a second occurrence of the integral of $\sin^2$. Sometimes, as it happened here, a self-reference results in a solution; at other times, it may indicate a circularity in the solution path.

9. ?GIVENS is a list of the names and types of the given data, and assumptions which may be made regarding them by the subplan below a given node. This is used, for instance, in the detection of inconsistencies between the definitions of subgoals and their usage.

10. ?VIOLATIONS is the list of model predicates which are not satisfied by the ?EFFECT achieved by ?PROCEDURE. This register is set by a separate performance annotation module.

Let us briefly consider a few examples of the values of these registers at various nodes of the structural descriptions for the hypothetical problem solving protocol presented earlier. For the SOLVE node corresponding to E03, ?MODEL is as shown in Figure 6.

Prior to E09, the ?VIOLATIONS register at the PLAN node for the substitution was:

```
(NOT (= (EXPR E05) (EXPR E06)))
```

Since the integration task is eventually solved, ?VIOLATIONS is empty at its SOLVE node, since solutions include debugging. The same is not true for the corresponding PLAN node.

Figure 6. Problem Description (Model) for Top Level Integral

$\exists(f(t))$ such that

$$\frac{df(t)}{dt} = r^2 \sin^2(wt),$$

and

$$``\int" \in f$$

The pragmatics provides rationales for the various planning choices in the protocols. These are derived from the synthetic arc conditions when applicable. For example, the reason for integration by parts being attempted on the integration task was that the integrand was in the form of a product of two terms.

(REASON (INT-BY-PARTS E13)
    (EQ (FORM (INTEGRAND E05)) 'PRODUCT))

The reason for each buggy event in the protocol is the same as the reason for what might have been the corresponding correct version of the event, but flagged by a note stating that the attempt was buggy.

Debugging operations localize (or repair) the cause of some violation. The reason for E09, for example, is to verify that the integration satisfied its specifications (i.e., that the derivative of the results give the original expression). In this case, the underlying cause of the violation was the omission of an essential cleanup step (the differential term). The repair was to solve for the missing term and incorporate it at the appropriate point in the solution:

(REASON E10 (REPAIR E06))

REASONS are represented by assertions involving instantiated arc predicates of this sort, attached to each node of the structural description.

Discussion

The example protocol discussed in this section illustrates the analyses which PAZATN is designed to generate. In keeping with the grammatical metaphor, these analyses have three aspects: structural (syntax), semantic (purposes), and pragmatic (reasons). The structural analysis is represented as a parse tree. The semantic and pragmatic information is represented as annotation (variables and assertions) associated with each node of the parse tree.

Some readers might object that these three aspects alone do not constitute a complete analysis of a protocol. Perhaps some essential dimension of the subject's problem solving performance has been overlooked. If there are useful questions about the behavior which are not captured by these aspects, we would have to agree. However, our working hypothesis is that there are not. Hence, we believe that part of our contribution in this research is our recognition of the appropriateness of a linguistic analogy.

A precise definition of protocol analysis has been provided, along with a brief example of the form of this analysis. We now turn our attention to the design of PAZATN, a scheme for performing such analyses automatically.

SECTION III

ORGANIZATION OF THE PAZATN PROTOCOL ANALYZER

General

In this sub-section we describe the general organization of the protocol analyzer. Later sub-sections present additional detail. The analyzer would consist of the following data structures and modules: PATN, the PLANCHART, the DATACHART, the preprocessor, the event classifier, the (domain specific) event specialists, the event interpreter and the scheduler. Figure 7 provides a block diagram. After reviewing the analyzer's input/output specifications, we consider each of these components in turn. Section four refines the first order description provided in the current section. Since the event specialists are domain specific, we will not provide details in this report.

The analyzer receives the model as input. It is a formal statement of the top level goal, and the protocol, which is a list of input/output events. It has been assumed that the protocol is "reasonable," in that it represents a sincere attempt to accomplish the task, and that it terminates exactly when this goal has been satisfied. The design is robust in this respect: it relies only slightly on these simplifying assumptions. Consequently, it is our expectation that

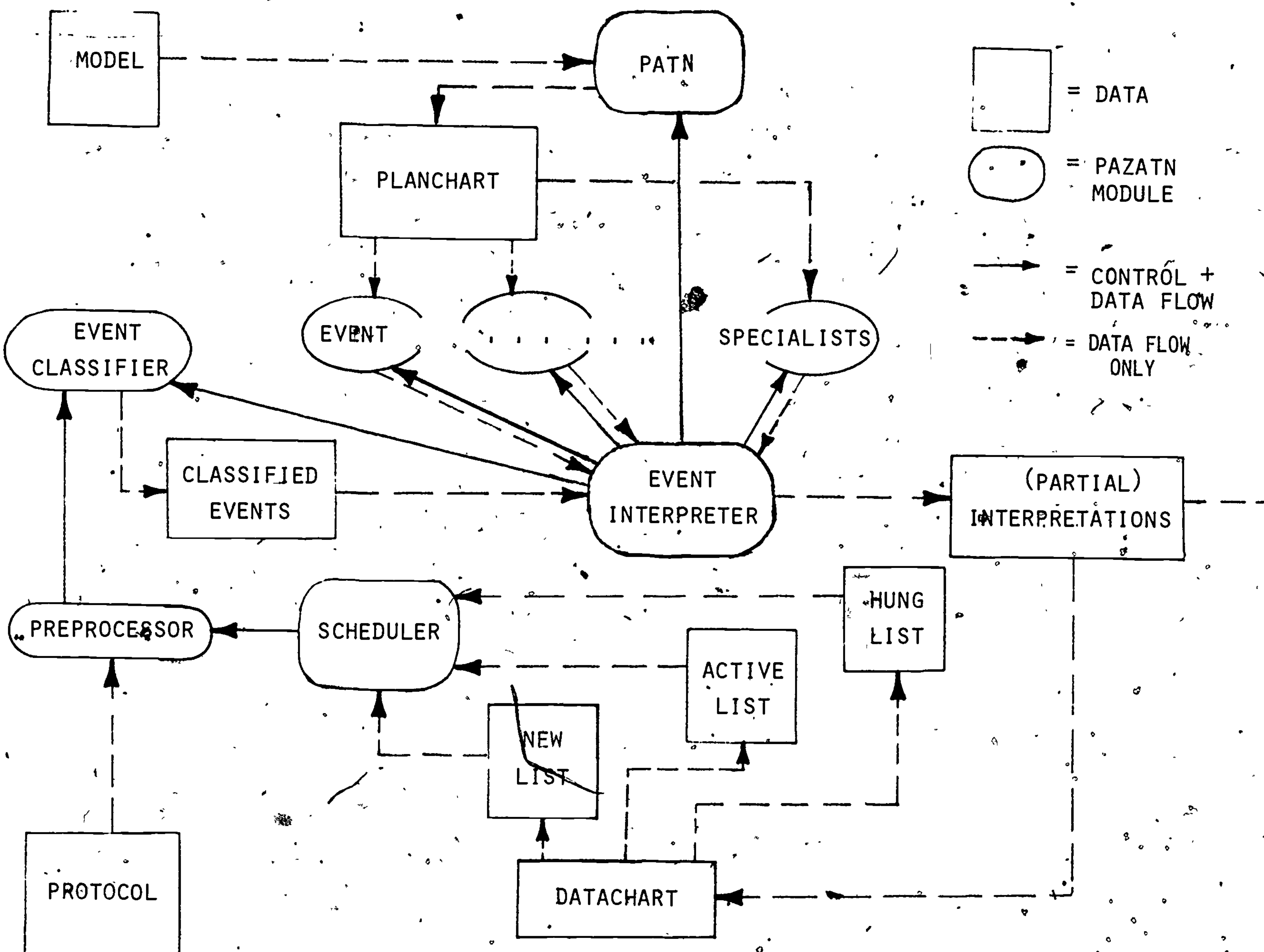


Figure 7. Block Diagram of PAZATN

the analyzer will also prove to be useful (although it may perform less efficiently) for less than ideal protocols, such as where the subject/student makes a sensible start but fails to complete the project.

The output of the analyzer is a set of one or more plausible interpretations of the protocol, where an interpretation is defined as the assignment of a structural description (or "parse") to the list of events, augmented by an assignment of values to the set of semantic variables, as well as by a collection of pragmatic-reason assertions, for each node of the description. In order to discuss the representation of interpretations, and the manner in which they are discovered, it is necessary to introduce the roles of the ATN and PLANCHART in the analysis process.

Augmented Transition Network (ATN)

To understand the central role of the ATN, one need only remember that the analyzer is little more than a procedure for selecting those synthetic solutions to the stated problem which most closely match the input data. However, the space of possible solution paths is represented intensionally (as opposed to extensionally) by the ATN. We require the ATN to generate complete protocols, even to the level of events corresponding to the typing in of detailed instructions to the computer monitor. Some of these requirements are superfluous for the expert version of the problem solving system. Hence, we plan to employ a slightly modified version of PATN in the analyzer (but the differences are not otherwise important).

There is a question as to whether the expert version of the ATN will eventually succeed in spanning the entire space of reasonable non-expert behaviors, provided that each of its preferred approaches is successively rejected by the analyzer. The expert version of PATN would have the interesting property of being capable of producing partial solutions which contain certain "rational bugs." Furthermore, it will be seen that the spanning requirement does not rule out the analysis of "inexplicable" (or "irrational") bugs -- such as typographical errors or memory lapses -- provided that they can be recognized as deviant versions of some rational synthetic behavior. Consequently, we tentatively assume that PATN is indeed such a spanning model in this extended sense.

The ATN would perform arc transitions partially as a result of PATN's synthetic pragmatics and partially as a result of analytic guidance. For example, the ATN may expand the plan for a subgoal which might not have been pursued in the pure synthetic system, because analytic criteria have established that this is probably a subgoal of the subject/student. The ATN then suggests how one might go about solving it.

The PLANCHART

As the analysis progresses, there are a number of reasons for needing an extensional represention of the ATN process, as it operates upon the particular problem. Consequently, a complete trace of the synthetic computation is kept for examination by the analyzer. This data structure is called the PLANCHART. The most obvious reason for creating such a representation is to avoid repeated calculations; but important additional uses for the PLANCHART will appear in the course of the discussion.

In fact, the PLANCHART includes not only plans, but nodes of other types such as debugging episodes. As its name suggests, the PLANCHART is a chart [Kay 1973], a network-like data structure which compactly represents many combinations of subexpressions. This data structure is an efficient representation for PATN's current set of partial solutions and their structural descriptions. Rather than generating the entire solution space at once, which would be impractical even if the space happened to be finite, the ATN expands this PLANCHART incrementally as additional possibilities are needed by the analyzer.

The PLANCHART resembles an AND/OR goal tree (see Figure 8, for an example). However, there are a greater variety of node types,

Figure 8. Example Planchart: Like an AND/OR Goal Tree

```
SOLVE
   PLAN
      CHOOSE
         DECOMPOSE
            CONJUNCTION
               CHOOSE
                  INT-BY-PARTS
                     SOLVE (integrate cos^2(x))
                        .
                        .
                        .
                  DERIVATIVE-DIVIDES

         REFORMULATE
            EQUIVALENT
               PYTHAGOREAN-RELATION
                  SOLVE(integrate cos^2(x))
                     .
                     .
                     .
                  SOLVE(integrate h(x))
```

These would be identical.

rather than just AND and OR. This allows the PLANCHART to represent such concepts as whether a set of conjuncts need to be accomplished in the specified order, or whether any order will do, allowing a greater variety of synthetic combinations to be expressed parsimoniously. For concreteness, we take the PLANCHART to be a LISP S-expression. However, each subexpression is unique-ized; that is, EQUAL subgoals refer to physically identical structures. The reasons for this are explained shortly.

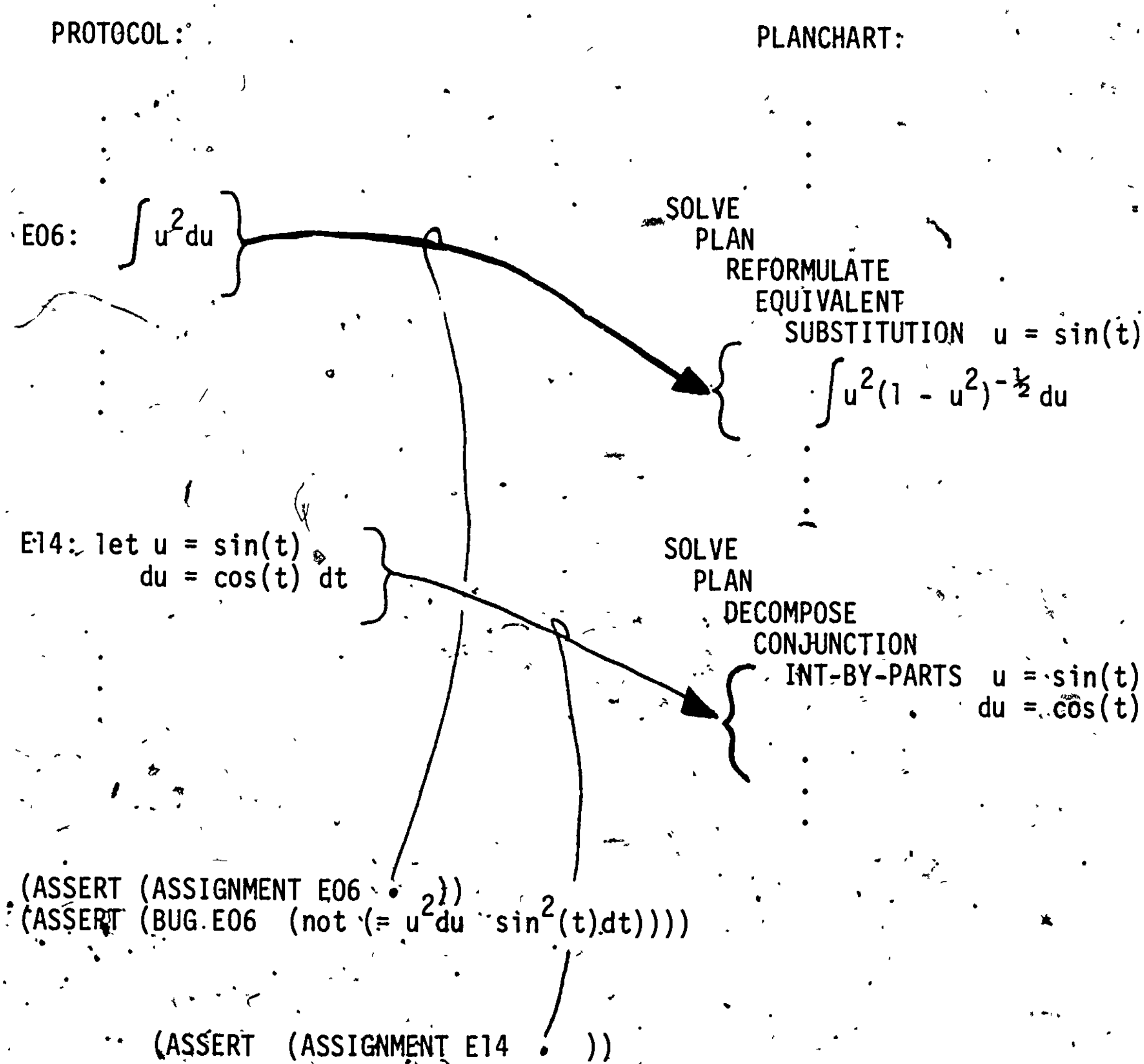
The analysis process is closely tied to modifications of this data structure. In particular, the structural description assigned to a protocol corresponds to a subtree of the PLANCHART starting from the root, (the top level SOLVE node) to the individual protocol events corresponding to a subset of the leaves. Consequently the structure building actions of the analysis system are performed entirely by the ATN.

The Representation of Interpretations

In view of the above remarks, it should be clear that an interpretation of an event can be defined simply as an assignment of that event to a leaf of the PLANCHART (Figure 9). Similarly, an interpretation of the protocol corresponds to a complete association list of such event assignments, and a partial interpretation is an association list containing assignments for a subset of the events in the complete protocol. As a consequence of the left-to-right processing order, a typical partial interpretation contains assignments for the first M out of N events.

Notice, though, that a given PLANCHART leaf may be a member of more than one structural description, due to the structure sharing mentioned earlier. This is an advantage. Genuine ambiguities need not be treated as explicit alternatives. The analyzer does not commit itself to an arbitrary decision. All possibilities are carried along, implicitly, at no extra cost. It is possible, but unlikely, that

Figure 9. Interpreting Events by Assignment to PLANCHART Leaves



73

the complete association list for the entire protocol will likewise have multiple structural description pathways through the PLANCHART. Each of these, technically, should be considered a different interpretation. Nevertheless, it is sensible to lump them together, since this situation can only occur when the data have been inadequate to distinguish them.

In order to be assigned to a given leaf of the PLANCHART, it is not necessary for the data event to identically match the corresponding synthetic event. The assignment merely reflects the heuristic judgment of the analyzer that the actual data event was intended to serve the same role as the associated synthetic event. Consequently a synthetic event (i.e. a single PLANCHART leaf) actually stands for an equivalence class of data events, with various plausibilities.

For an interpretation to be plausible, the data event must be very "similar" to the assigned synthetic event. There are exactly two ways in which the events may differ: (a) the data event is an alternative, equivalent realization of the synthetic event; or (b) the data event is a "buggy" realization of the synthetic event. The plausibility of assignments of type (b) depends on three factors. One factor is the intrinsic, essentially syntactic, similarity. Misspellings which differ by only one or two characters are an example. The second factor is knowledge of common bug types. Since "rational" bugs would appear as distinct leaves of the PLANCHART, here we speak of the "irrational" variety. Since there is, at present, no compelling theory to account for such bugs, the evidence must be of a statistical nature, and not necessarily the same for each individual. The third factor is the context in which the bug occurs. This is determined by the status of neighboring leaves. We return to these questions later.

The DATACHART

A partial interpretation is said to split when it proposes more than a single PLANCHART assignment for its next event. Some method for keeping track of the analyzer's alternative partial interpretations is needed. Ideally, it should take advantage of the fact that, following a split, the event interpretations prior to that split remain the same: the common ancestry should be preserved. The DATACHART serves this function.

The DATACHART may be thought of as a context-layered data base, such as that provided by CONNIVER [Sussman & McDermott 1972]. PAZATN would record partial interpretations in CONNIVER-like contexts. Suppose that two interpretations have identical assignments for the first M events, and then split. The split corresponds to a single context layer having two descendants. Assertions corresponding to the shared part of the interpretation are automatically inherited from the parent context layer (Figure 10).

Whenever an event assignment is to be made whose plausibility does not exceed some threshold, the following actions are performed:

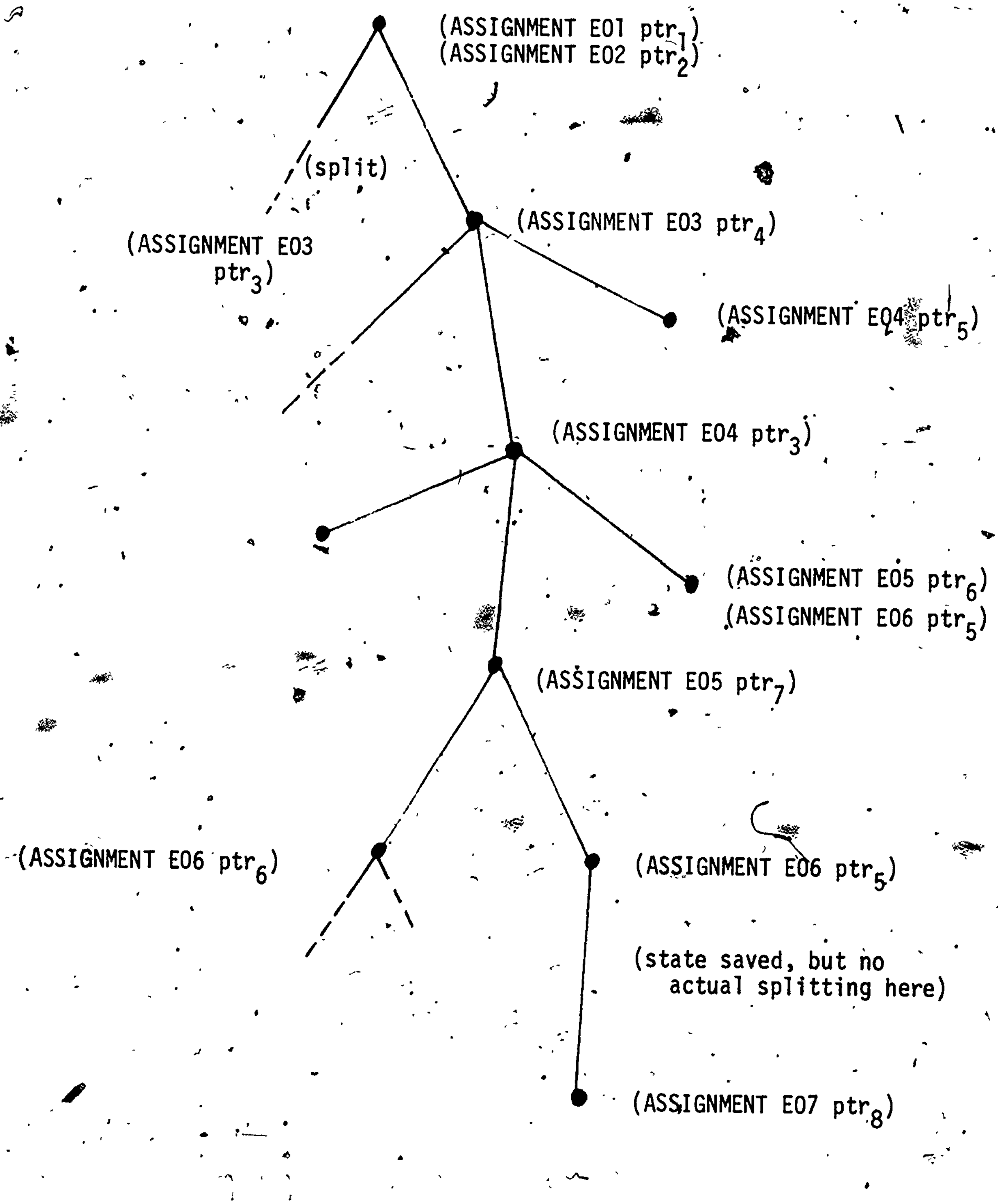
(1) An assertion is added to the current context, indicating which assignment is about to be made. This ensures that the same possibilities will not be repeatedly pursued.

(2) A PUSHCONTEXT is executed, creating a new subcontext which will inherit prior assignments from the parent context. This ensures that changes which reflect the uncertain continuation of the interpretation will not affect the state information in the parent.

(3) The uncertain assignment is performed in the new subcontext. The normal operations associated with event interpretation (described below) are carried out.

(4) A handle to this context is placed on a list of NEW partial interpretations. This ensures that it will be scheduled for at least one cycle of further investigation.

Figure 10. Inheritance of Shared Partial Interpretations



(5) A POPCONTEXT is executed. The parent context of the new interpretation is then re-examined to determine if alternative assignments should also be considered. If so, the above sequence of operations is carried out for each. When no further alternatives seem worth considering at the present time, the parent context is placed on a list of HUNG interpretations.

With this technique, it is not necessary to explicitly list all of the possible alternative interpretations for a given event. Note that, after the PUSHCONTEXT, the HUNG layer represents, not a single partial interpretation, but an indefinite number of implicit alternatives to the partial interpretations explicitly represented by its offspring. Even after it is HUNG, the parent context contains the necessary state information for generating additional possibilities, should it ever need to be reactivated.

Incremental PLANCHART Expansion

Consider the situation in which an active partial interpretation can find no acceptable assignment for its next event in the PLANCHART. There are two actions possible: either (a) conclude that the current partial interpretation is a dead end, and move it to the HUNG list; or (b) conclude that the PLANCHART has not been expanded sufficiently to account for the current data.

In case (b), the analyzer passes control to PATN, which expands those subgoals most likely to be relevant to this interpretation. Since the PLANCHART is kept in the GLOBAL context, other interpretations may also benefit from the additional growth. This is the only situation in which the PLANCHART is expanded. (This rule is modified slightly in the next section.) Limited, incremental growth ensures that a minimum number of irrelevant synthetic solutions are generated.

Unfortunately, deciding whether (a) or (b) is actually the case, may be difficult. The difficulty is compounded by the fact that a given data event need not be an exact match to a PLANCHART leaf in order to be assigned to it; it could be a buggy version, or an equivalent construct. There are three technical problems: (1) choosing between cases (a) and (b) above for a given leaf; (2) locating the relevant existing leaves which ought to be considered in view of possible equivalence and bugginess; and (3) locating the relevant existing partial interpretations which might be able to "make use" of newly generated PLANCHART leaves, especially in view of possible equivalence and bugginess.

Now, if the analyzer is too miserly in allowing PLANCHART growth, an event might be interpreted as a buggy version of an existing leaf, when only slight growth would have allowed it to match a new leaf exactly. But if the analyzer is too eager to expand the PLANCHART, the number of irrelevant synthetic solutions considered could be enormous.

We plan to provide the analyzer with a number of strategies for dealing with these problems. One strategy, which has already been introduced, handles the case where the relevant events are EQUAL; this is the unique-izing of subexpressions. But unique-izing is inadequate to deal with buggy or equivalent versions. Another strategy employs a hash coding scheme, where the contents of the buckets are pointers into the PLANCHART.

Markers and Marker Propagation

A third set of strategies for dealing with the difficulties of the previous section relies on a system of PLANCHART markings and marker propagations. The marker scheme is of interest because it is also used to produce the final structural description, by selecting a subtree of the PLANCHART. The assignment of a data event to a PLANCHART leaf can be thought of as marking that leaf.

Now recall that the PLANCHART is essentially an elaborated AND/OR goal tree. Each non-terminal node type represents an ATN state, each of which specifies either a conjunction or a disjunction of subgoals, with possible sequencing constraints. Consequently, we can allow markers to propagate upward through the PLANCHART according to three rules:

1. MPR-1. If the parent of a marked node is a disjunctive type (e.g., CHOOSE), the parent is marked;
2. MPR-2. If the parent of a marked node is a conjunctive type (e.g., SEQ), and the siblings of the marked node are also marked, the parent is marked (note that if there were constraints on the ordering, but the events appeared in the wrong order, the siblings would probably not have been marked);
3. MPR-3. If no higher plausibility interpretation can be discovered, under certain conditions a propagation may be postulated when neither rule MPR-1 nor rule MPR-2 is completely satisfied. (This third propagation rule is designed to allow structurally ill-formed ["ungrammatical"] plans to be analyzed, but with lessened plausibility.)

Top down MOD plans (see below) however, are handled specially. The solution for the top level problem should be propagated when it is finished, even though the solutions for the subproblems have not yet been encountered; but the expectation for the subproblem solutions remain in effect, and cause subsequent propagations when they occur. This is indicated by using two different marker symbols in later diagrams.

The marker propagation status is local to a partial interpretation and its offspring. Notice that it indicates which synthetic subgoals are expected, and which are satisfied. An upward propagation corresponds to what might be termed a reduction in a bottom up parsing scheme. The propagation of markers is intended to allow the analyzer to efficiently draw inferences about the probable solution path represented by the protocol, with respect to a particular assignment of events.

At intermediate stages in the analysis, these PLANCHART markers provide evidence concerning the plausibility of alternative interpretations. This is especially important when additional PLANCHART growth is under consideration. The following guidelines follow immediately:

PLR-1. An event assignment which would result in a propagation is more plausible than one which would not.

PLR-2. An event assignment which would result in a long chain of propagations is more plausible than one which would result in a shorter chain.

PLR-3. A completed interpretation (one which has interpreted the final protocol event) which propagates a marker to the top level SOLVE node is much more plausible than one which does not (a consequence of the "reasonableness" assumption).

PLR-4. An event assignment to a conjunction dominated leaf, many of whose siblings are marked, is more plausible than an assignment to such a leaf only a few of whose siblings are marked. A similar rule holds for plausibly marking non-terminal nodes.

PLR-5. No leaf should be marked by more than one event. More generally, a node dominated by a marked node should not be marked. One exception is that if the dominating marking was via marker propagation rule MPR-3 (or the USE nodes of top down MOD plan), and if the new marking would have allowed a propagation via MPR-1 or MPR-2, then the node may be marked. The other exception is that if the marking was the result of a buggy assignment, and the new marking is the correct version of that assignment, the node may be marked.

PLR-6. Assignments which result in propagations by propagation rule MPR-3 are much less plausible than assignments which result in propagations by rules MPR-1 or MPR-2.

These heuristic guidelines help the analyzer to: (a) determine whether it is propitious to allow additional PLANCHART growth; (b) select the preferred interpretation for an event; and (c) select the preferred structural description of the protocol, which is a subtree of the final PLANCHART.

The marker propagation scheme provides a precise notion of expectations. A constituent is expected to the extent to which it would result in propagations. For example, consider an Identification Plan for solving a subproblem. If the subproblem had previously been solved and saved in a file, it is expected that a command retrieving the solution from the file will occur. The PLANCHART would contain an unordered conjunction of subgoals, one to add a use of the solution to the subproblem to the solution to the top level problem, and one to retrieve the solution to the subproblem from the file. After an event had been assigned to the former, the latter would be expected because its occurrence would result in a propagation at least as far as the Identification Plan node.

Suppose that an expectation (such as the Identification Plan example) fails to be satisfied after many events. One possibility is that the partial interpretation which expects it is just on the wrong track, and should be abandoned. A second possibility is that the overall subgoal structure is correct, but the subject has proceeded to re-solve the problem via Decomposition or Reformulation, perhaps because the existing solution had some undesirable property. If this second possibility was in fact the case, then when the subproblem's solution was completed, the resulting propagation would "turn off" the aberrant expectation, since it would then be dominated by a marked node.

A third possibility is that the student/subject is actually using an ungrammatical plan. If a file retrieval is not performed as expected, it could be that the student simply forgot to do it, or thought that it was unnecessary, mistakenly believing that its solution was already present in the workspace. The fact that a plan is ungrammatical does not make it unanalyzable, however. When the end of a solution to a subproblem is encountered, some propagation ought to occur under every ACTIVE interpretation. If such an event is followed by events which are analyzed as diagnosis, then the most plausible propagation is forced, even if this is only possible via rule MPR-3. The plausibility of this interpretation will be greatly increased if the missing event eventually does occur as a result of subsequent error correction.

The Event Classifier

The event classifier module contains the syntactic knowledge necessary to distinguish the various domain-specific event types. The event classifier is one of the few components of PAZATN which would need to be redefined for each domain. In assigning an interpretation to an event, a variety of semantic and pragmatic evidence may ultimately be considered by the analyzer; but the domain-specific event classifier is deliberately restricted to syntactic evidence (and timing data, for a few cases such as those mentioned earlier).

The event classifier can be invoked in three modes. In the normal mode (which is used by the preprocessor) its input is an event, and its output is that event's primary syntactic class. For most events, this is sufficient. The second mode of operation is used by partial interpretations which find the primary syntactic class of the event to be questionable, but have a specific alternative class under consideration. In this second mode, the classifier is called with an event and a proposed alternative category. The classifier returns with a numerical summary of the syntactic evidence relevant to the proposed reclassification. The third mode is employed when the primary class is questioned, but no alternative readily suggests itself. The classifier returns with an exhaustive rank-ordered list of the syntactic categories and their (syntactic) plausibilities.

Event classification would be performed using straightforward pattern matching. The details, being domain specific, are generally uninteresting and are not given here.

The Event Interpreter and Event Specialists

The event interpreter is the module responsible for category independent operations of event interpretation. This includes the context saving and restoration sequence described in the DATACHART section, the actual processing required for marker propagation, and the marker status plausibility computations. The rationale for grouping these activities into a separate component is modularity: they are routinely required, and common to every category of event interpretation.

The event interpreter is the "inner loop" of the analyzer. It is invoked by the scheduler with two arguments: a handle to a partial interpretation, and a data event from the protocol. In cooperation with one or more event specialists, it attempts to explain that data event in the context of that partial interpretation. This may result in the creation of one or more additional (descendant) partial interpretations. When event interpretation is complete, control returns to the scheduler.

A collection of domain specific event specialists [ESP's] are responsible for category dependent operations of event interpretation. Each specialist contains the requisite knowledge for analyzing events of a particular syntactic type. The event interpreter invokes an ESP with an event (and an implicit assumption regarding its syntactic category) in the context of a given partial interpretation. The specialist is free to assign any interpretation to the event which is consistent with the categorization assumption. However, a given specialist is not free to consider the possibility that the category assumption is incorrect.

If the event specialist does not return with a sufficiently plausible event assignment, the event interpreter will then consider the possibility that the syntactic category which has been postulated for the event may be incorrect. Whenever an event is interpreted as buggy, expectations for diagnosis and repair are generated at the request of the event interpreter. The details of the ESP's for particular task domains are not given here; examples of ESP's for the LOGO graphics domain are presented in [Miller & Goldstein 1976d].

The Scheduler

The remaining module to be considered is the scheduler. The job of the scheduler is to drive the analysis through a best first coroutine search of the space of partial interpretations. Ultimately it arrives at one or more plausible completed interpretations.

The state of each interpretation is represented by assertions in its context layer. For example, one fact which the scheduler needs to know about an interpretation is how far along it is in processing the protocol. (Note that not all interpretations are equally far along.) This progress is represented by an assertion of the form:

(INPUTMARKER= <event#>)

which means that the input marker is sitting immediately after the <event#>'th input event.

Another set of facts which are needed are the event assignments. These are assertions of the form:

(ASSIGNMENT <event#> <leafptr>)

which means that the <event#>'th event has been assigned to the PLANCHART leaf referenced by <leafptr>. Note that at most a few of these assignment assertions are explicitly present in a given layer; the rest are inherited from higher up in the context hierarchy.

The scheduler maintains three lists of partial interpretations (handles into the context hierarchy): the NEW list, the ACTIVE list, and the HUNG list. Every partial interpretation which has been discovered is on one of these three lists. Typically interpretations on the ACTIVE and NEW lists are further along in processing the input. Those on the HUNG list will not make progress unless a sufficient number of currently ACTIVE interpretations become HUNG, at which time some HUNG interpretations may be reactivated.

The basic difficulty which is faced by the scheduler is to ensure that interpretations which have a reasonable likelihood of succeeding continue to make progress, while those that are likely to fail do not consume valuable resources. ACTIVE interpretations are pursued in parallel, while HUNG interpretations are available should backup become necessary. The size of the ACTIVE set is a global parameter of the analyzer. It should be chosen to be just large enough to ensure that backup will be infrequent, but not so large that progress is forestalled. A fundamental hypothesis is that the ATN plus the event specialists provide sufficient information to constrain the likely interpretations to a moderately small number.

The scheduler operates by cycling through the ACTIVE list, allowing each partial interpretation to process one input event. Then the plausibility of each modified interpretation is recomputed, and the ACTIVE and HUNG lists are updated. NEW interpretations (resulting from the splitting of ACTIVE interpretations on the previous cycle) are automatically moved to the ACTIVE list, to ensure that they receive at least one quantum of processing before being HUNG. The plausibility of a partial interpretation increases with each additional event accounted for. (This provides for automatic attenuation of older HUNG interpretations.)

This coroutine search process continues until at least one ACTIVE interpretation has processed the last input event with high plausibility. To be highly plausible, a finished interpretation should not have dangling expectations, but be a successful solution of the original problem. If the first successful interpretation is not sufficiently better than every other candidate, some of the better alternatives may also be pursued until they become implausible or determine that in fact the protocol may successfully be interpreted in more than one way.

SECTION IV

REFINING THE ANALYZER

Overview of Refinements

This section examines two broad classes of refinements to the PAZATN protocol analyzer's basic design. The first class is a set of elaborations to the slightly simplified description of the previous section, which will be included in our first implementation. The second category consists of some possible alternatives to the organization presented here. Our purpose in outlining this second category is to provide the reader with a flavor of the issues involved.

Our overall scheme for doing protocol analysis is to use PATN to generate expectations, and then to define a recognition process that attempts to match these expectations to a protocol. This parsing process can be refined by utilizing several ideas that have proven effective in problem solving and language parsing programs, including lookahead (e.g., [Aho & Ullman 1972]), least commitment (e.g., [Sacerdoti 1975]) and differential diagnosis (e.g., [Rubin 1975]). Some of these have parallels in the synthesis process. Here we examine their role in analysis.

We also briefly examine some techniques for improving the applicability of the analysis scheme to use in dynamic tutoring. One strategy is to replace the expert ATN by a modified version, which more closely models the idiosyncratic problem solving behavior of the individual student. Another strategy is to introduce pruning procedures to reduce the amount of storage required by the analyzer. Still another is to provide heuristics for dynamically adjusting parameters of the recognition process in accord with the pragmatics of a tutoring session.

Finally, we explore a number of issues related to possible alternative design choices. The possibility of organizing PAZATN as an analytic ATN [AATN] instead of as a coroutine searcher is discussed. This approach might offer greater clarity and modularity, decoupling matters of efficiency from formal theoretical concerns. Limitations of the breadth of the synthetic theory are also considered. Finally, the question of episode based analysis -- performing the analysis in larger chunks -- is raised.

Lookahead and Least Commitment

Lookahead and least commitment are related search strategies designed to avoid premature decisions based on inadequate evidence, and the resultant need to back up. Lookahead consists of briefly examining later events in the input string prior to interpreting the current event. Least commitment consists of postponing a decision regarding the proper interpretation of the current event until further evidence is gathered from later events.

Recall that PATN as an AI expert system always engages in strict top down problem solving. The top level plan is completely defined before the solutions for subproblems are attempted. Human problem solving is not this uniform. Alternatives to pure top down planning need to be incorporated by allowing variations on the order in which goals are pursued.

A goal may be expanded before a subgoal, representing top down planning. Or, once the need for a particular subgoal has been established, that subgoal may be expanded before ascertaining which other subgoals are needed for the main goal, representing bottom up problem solving. Figure 11 illustrates a top down expansion, while Figure 12 illustrates bottom up.

A bottom up or mixed solution order is a good example of the possibility for misleading mismatches between expectations and protocol events. Least commitment helps to minimize this. The net effect is that

Figure 11. Top Down Expansion

$$\int \left[f(x) + g(x) \right] dx$$

$$\int f(x)dx + \int g(x)dx$$

$$\left[uv + \int vdu \right] + \int g(x)dx$$

$$\left[uv + \int vdu \right] + \left[h(x) + C \right]$$

Figure 12. Bottom Up Expansion

$$\int [f(x) + g(x)]\, dx$$

$$\int f(x)dx + \int g(x)dx$$

$$\int f(x)dx = [uv + \int vdu]$$

$$\int vdu = \ldots$$

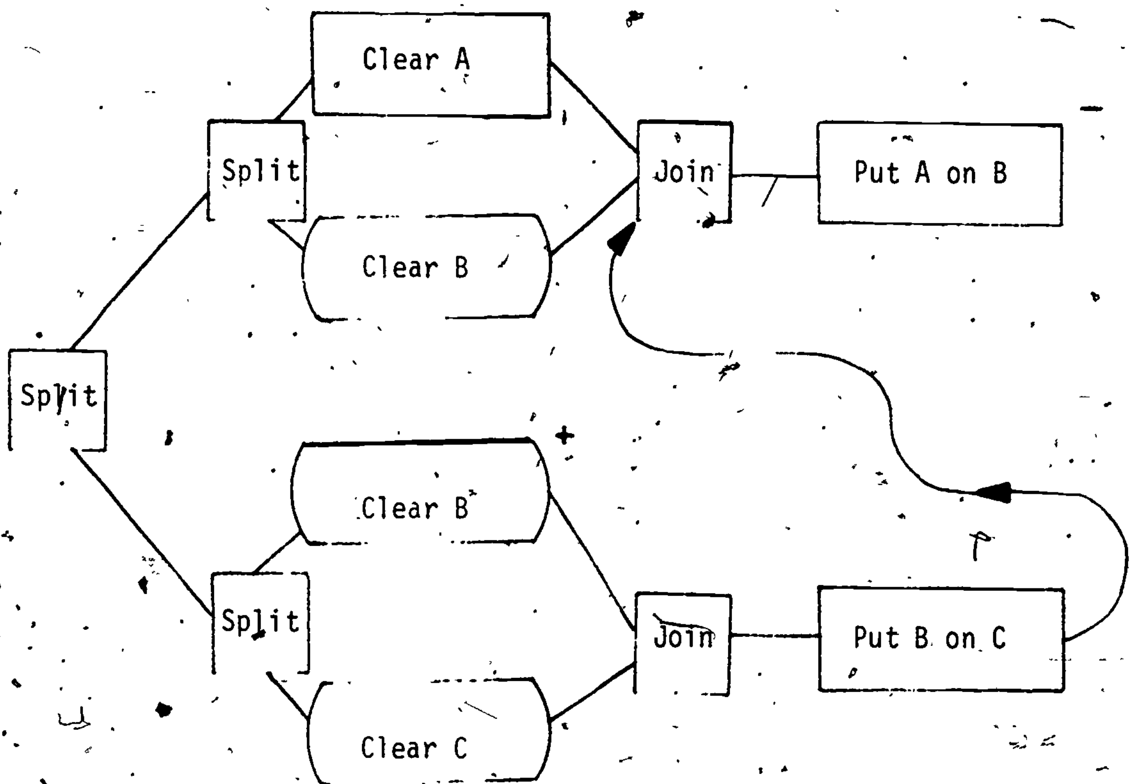
at those decision points where the choice is essentially arbitrary (such as in the particular sequence for accomplishing a SET plan) PATN generates a disjunctive set of possibilities, rather than making an arbitrary selection. Thus, at any point in the parsing process, a set of alternative expectations may be present. This avoids a blind depth first top-down analysis, and reduces costly backup.

We have already seen some use of these techniques by PATN. The primary application of least commitment, in the synthetic component, is the avoidance of arbitrary ordering decisions. As currently designed, PATN can optionally be instructed to produce procedural nets [Sacerdoti 1975]. Figure 13 illustrates how purely sequential solution procedures, unlike procedural nets, overspecify the ordering constraints. The virtue of the procedural net representation for PAZATN is that, when an ordering would be arbitrary, there is no reason to expect the student to choose the same path as PATN. By postponing the decision, a greater number of interpretations can be implicitly represented by a single PLANCHART marking.

Examples of the techniques occur in the analytic component as well. Some difficulties which are encountered in designing event specialists, for example, can be resolved by the use of demon procedures [Charniak 1972]. In certain situations a demon would be created to represent an event assignment which depends on subsequent events. When the relevant events are finally encountered, the demon would then fire, completing the assignment on the basis of the additional information.

One effective application of least commitment in the analytic component is the sharing of substructures in the PLANCHART. This allows ambiguous collections of event assignments -- those which have more than a single structural description -- to be economically stored. Rather than committing the analysis to one or another structure, the decision is postponed until some event provides evidence clearly favoring one or the other. Implementing this policy does not

Figure 13. Procedural Nets versus Sequential Procedures



A Procedural Net For Building A Tower
After Criticism to Resolve Conflicts
[Based on Sacerdoti, 1975, p. 15]

require special action. It is an automatic consequence of the analyzer's data structures.

PAZATN can also benefit from a type of lookahead which has not been presented so far. Previously it was claimed that PLANCHART growth was to be limited to those cases in which a plausible active interpretation could not find an acceptable assignment for its next event. This statement was an expository simplification, and is not strictly true.

The primary objective of PAZATN's control structure is to cause the strongest sources of constraint to be utilized first. This is to prevent unguided search in a potentially large space. Thus, when there is clearcut bottom up evidence of a particular constituent, that evidence should be examined. Likewise, when a top down decision is straightforward, that route should be pursued prior to making less certain analytic assumptions.

Therefore, instead of severely restricting PATN's activity, as previously stated, we actually intend to allow it some freedom to exploit strong sources of top down constraint. Some synthetic decisions are virtually forced by the form of the model. There is no reason to interrupt PATN when it is about to make such a decision. This can be viewed as a type of lookahead, in that even before the event interpreter has "noticed" any deficit, the synthetic component has predicted the necessity for -- and accomplished -- appropriate PLANCHART growth.

PAZATN's analysis process is actually designed to begin by synthetic examination of the model. This top down investigation proceeds until some decision point is reached for which the synthetic basis is uncertain in some fundamental way. At that point, control switches to the analytic component. Likewise, whenever the ATN is invoked, it is allowed to proceed so long as its choices follow from firm criteria. This reduces the overhead of constantly switching between event interpretation and plan synthesis. Operations would proceed with fewer interruptions, in slightly larger units.

Despite its virtues, though, least commitment could be overdone. The result would be such a large disjunction of expectations that no guidance could be obtained. Moreover, the relationship between the system's formal model and the student's intuitive model is tenuous. The analyzer strikes a balance between overly committing itself, and stubbornly refusing to take decisive action. This is accomplished by avoiding overcommitment in the course of a given decomposition strategy, but requiring bottom up evidence to change the formulation of the model. The next section describes the differential diagnosis knowledge that would be used to request such reformulations.

Differential Diagnosis

We have already encountered a use of demon procedures by the analyzer; this was to handle the problem of the assignment of a given event depending primarily on the assignment of some future event. Another use of demons, which we did not consider, is to perform differential diagnosis in deciding between two interpretations, or in recovery of an appropriate explanation when a given approach becomes hung. In those situations where even the use of least commitment fails to produce a successful set of expectations, differential diagnosis knowledge should direct PAZATN to produce a new set of expectations. There are two situations where differential diagnosis is appropriate. One is the use of explicit diagnostics for unsuccessful category assignments. The second, and most significant, is the reformulation of the problem description to achieve consistency with bottom up evidence.

In our first order description of the event specialists, we imposed the stringent requirement that no specialist ever consider the applicability of another specialist; this job was left to the event interpreter. Sometimes this requirement can be artificial. When a piece of category specific knowledge is able to diagnose the appropriateness of some other ESP, then that piece of knowledge belongs within the specialist for that category.

Likewise, differential diagnosis is used to select the proper subset of a disjunctive set of expectations (such as is produced using the least commitment policy). Conversely, when none of the alternative expectations matches the protocol, the analyzer requests that PATN perform a reformulation consistent with that evidence. The following are some examples of demon templates, which can be instantiated to realize this behavior in specific situations.

DDR-1. If the current protocol segment uses a named subproblem whose model has been firmly established, and if that model corresponds to a disjunctive subset of the current expectations, then select that subset. If no expectation corresponds to the model of this segment, reformulate the current problem description in such a way that this model is among the expected subgoals.

DDR-2. If the effects produced by the current protocol segment match a disjunctive subset of the current expectations, select that subset. If not, consider a reformulation that uses a model satisfied by the segment effects as a subgoal. (The possibility that the current segment is an error must also be considered.)

DDR-3. If the subject states that the current segment corresponds to a certain subgoal, select that subgoal. If that subgoal is not among the current expectations, reformulate the model so that it is.

DDR-4. If the current segment accomplishes the effects of an expected subgoal, but not by a plan that matches current expectations (e.g. via different control structure) then reformulate for this part, in terms of a model corresponding to the control structure observed in the protocol. Generic/explicit conversion [Miller & Goldstein 1976b] could be handled by this rule, for instance.

DDR-5. If the effects of the current segment violate only a few model predicates under the current interpretation, but the segment has a sub-segment structure that does not correspond to expectations, then reformulate. If there are too few segments, try regrouping into compound parts. If there are too many segments, try disecting model parts which contain multiple sub-parts.

This list is not exhaustive. However, it does suggest how differential diagnosis demons could be useful in refining the basic analyzer.

Tailoring the ATN to the Individual

In previous sections, it has been assumed that PATN is a spanning model, in other words, that the ATN is capable of exhaustively enumerating the space of reasonable problem solving behaviors (within its chosen domain). To this definition is added the caveat that "irrational bugs" such as typing errors are often understandable as buggy versions of one of these intended synthetic solutions.

It might seem that the caveat leaves the definition so weak as to be vacuous. But it is at least thinkable, if not probable, that some human problem solvers might display genuinely irrational intent. This does not refer to deliberately trying to mislead the analyzer -- "hacking the system". In PATN terminology, such problem solvers would have a deviant ATN. Their protocols would be more difficult, if not impossible, to analyze.

In what ways can an ATN be incorrect? One error would be to have a variant of the optimal pragmatic arc constraints. A characteristic example would be an ATN with an overly developed critic on the linear planning arc. A problem solver, having encountered several cases in which an initially linear attack led to bugs, might reach the general conclusion that all problems require a non-linear approach. Consequently, any problems which appeared to be linear might be reformulated to ensure the introduction of non-linearities.

Such an approach, of course, misses the valuable guidance in understanding the complexities of novel tasks, which is offered by the failure of the linear plan. This quirk is common among novices in the programming domain, for example. Relations, which by all accounts of "style" in programming ought to be accomplished via an interface step, will be accomplished as part of the definition of an adjacent main step. For example, a WISHINGWELL is defined as a TOP, a POLE, and a WELL, where the setups for each are included in the subprocedures.

More serious would be to have missing, or extra arcs. A novice programmer, whose prior experience was in the BASIC language, would probably be missing the recursion arc for achieving round plans. Consequently all problems involving generic models would be solved by iteration. Those problems for which iteration is truly inadequate, such as drawing arbitrarily deep binary trees, would be unsolvable.

Even more catastrophic would be to have missing, or extra states. Suppose one wished to apply PAZATN to the analysis of protocols produced by some other Artificial Intelligence program. It is likely that reformulation would not be one of its solution techniques; the relevant states would probably be missing entirely.

Moreover, the class of "rational" bugs should really be seen as relative to the problem solver's computational resources. Suppose there were certain systematic limitations on the ATN, such as an upper bound on the size of the structures contained in (or pointed to by) its registers. Some bugs which formerly might have been termed "irrational", in that they might have been avoided by consulting the critics gallery for example, become "rational." This is because a plan involving oversimplification, followed by debugging, may place less stringent demands on the limited resource. Rationality, by definition, is measured with respect to some estimate of utilities, costs, and risks.

Very likely, it is possible to handle most protocols produced by such non-ideal problem solvers without significantly modifying PAZATN's design: It is easy to generate example solutions which PATN would be loathe to produce, but which PAZATN, using the PATN ATN, can nonetheless understand. Whether compelling counterexamples can be found is an open question.

Nevertheless, a drastic reduction in search would result if the problem solver's quirks were turned to advantage. In tutoring the same student day after day, for example, consistent failure to use a certain type of plan should suggest to PAZATN that it is pointless to continue to look for it (except perhaps as a last resort). Consequently, our intention is to replace the expert ATN by an idiosyncratic version tailored to the individual. Once such an idiosyncratic ATN has been constructed, it can also be used, in tutoring applications, as a student model for the selection of tutorable issues.

Further Improvements in Applicability to Dynamic Tutoring

Although an automatic protocol analyzer is a valuable tool in its own right, the authors are particularly concerned that PAZATN's structure be amenable to applications involving real time, on-line tutoring. This constraint imposes strong limitations on the design, most notably the restriction that events be processed in a single pass in approximately left to right order. Moreover, the system must be sufficiently responsive so as not to interfere with the student's progress. Naturally this consideration is less critical in the ex post facto exhaustive study of the protocol for theoretical and experimental purposes.

To these ends, this section considers additional improvements to PAZATN. The tailoring of the ATN to the individual, discussed in the last section, is one improvement. Two further improvements are presented. One is the introduction of pruning heuristics to reduce the amount of storage required by the analyzer. The other aspect is the dynamic adjustment of key parameters of the recognition process, to increase the system's responsiveness without degrading the accuracy of its interpretations.

In order to assure reliability and the capability to recover from initially erroneous interpretations, PAZATN keeps a record of every partial interpretation which has been discovered. These are kept on three lists: NEW, ACTIVE, and HUNG. Furthermore, every local ambiguity can potentially cause PAZATN to save the state of the interpretation, in the event that splitting this interpretation becomes necessary. This cautious style might result in a very long HUNG list.

One technique for dealing with this contingency is to provide heuristics which reduce the amount of unnecessary splitting. The avoidance of overly cautious saving of states and splitting of interpretations is not a complete solution, however. Unless reliability is dangerously sacrificed, there are inevitably going to be a substantial number of local ambiguities for which these precautions are required. Only after examining later evidence will the doubtful status of other alternatives be firmly established. Furthermore, it is not enough that such low plausibility interpretations cease to consume processing time. Their continued existence implies that the analyzer will be "hanging on" to large quantities of storage in the form of assertions in CONNIVER context layers (or their equivalent).

For this reason, PAZATN should include a mechanism for pruning very implausible interpretations. The maximum allowable size of the HUNG list, HMAX, is a parameter of the system. When HMAX is exceeded, the lowest plausibility interpretation is deleted. This is based on a heuristic assumption that, at most, HMAX interpretations will have sufficient plausibility to warrant further consideration.

Unfortunately, it is entirely possible that a prunable context layer has non-prunable offspring. This is possible because the prunable context layer implicitly represents the set of (typically implausible) alternative interpretations other than those explicitly represented by its (typically more plausible) offspring. Since these offspring are inheriting assertions from the prunable interpretation, the garbage collector will not be able to reclaim its space, except in the case that all the offspring have also been pruned.

Fortunately, most context layers would probably have exactly one subcontext. This is because the typical event would be sufficiently ambiguous, to warrant maintaining a potential for splitting, but not so ambiguous to cause any other alternative implicit in the parent context to actually be pursued. The pruning procedure is designed to detect this situation. When a context layer with exactly one non-pruned subcontext is selected for pruning, this indicates that the subcontext may be finalized. Consequently, the parent context layer may be spliced out of the hierarchy altogether, and its space reclaimed. This helps to impose an upper bound on the storage required by PAZATN.

We now turn our attention to another potential inefficiency bug in the current design of PAZATN. This is that the size of the ACTIVE list required to prevent frequent back up may be large. If so, the system could simply be too slow for practical use in tutoring. PAZATN requires some technique for increasing the responsiveness of the system, while maintaining the effective size of the ACTIVE list.

The solution is to dynamically vary those parameters which determine the size of this list. (The actual size would be determined by a number of factors, including minimum size, maximum size, and minimum plausibility for inclusion.) The capability for variation would allow PAZATN to carry along a small working set of interpretations when the student is rapidly typing. Whenever the student paused to think or rest, the higher plausibility HUNG interpretations could be updated. In this way, should one of these be reactivated later, less back up would be required.

An elaboration of this refinement takes advantage of the primary underlying reason for avoiding back up. The greatest danger of backup in the tutoring application is that some previous suggestion or criticism may turn out to have been inappropriate. This danger can be reduced as follows. Naturally, the system should always require a high degree of confidence in its interpretation prior to intervening. This should be supplemented by filtering any remarks so as to be appropriate under all reasonably plausible alternative interpretations. (Introspection suggests that human tutors employ a similar heuristic.)

Furthermore, immediately prior to the remark, the size of the working set should be increased, and the reactivated interpretations brought up to date. It should then be verified that those marginal interpretations are unlikely to invalidate the planned remarks. This implies that normally the system would be highly responsive; but if delays were to be experienced, they would occur only when the student was about to be interrupted for tutoring anyway.

Design Issues and Alternatives

The careful reader may have noticed that PAZATN is somewhat independent of the detailed form of the synthetic formalism. Although tremendous leverage for analysis is obtained by the postulation of an effective synthetic theory, little use is made of the fact that PATN is specifically organized as an Augmented Transition Network. For example, the possibility that the debugging component is organized differently has not been completely ruled out by anything which has been said so far.

It does make a difference that the synthetic component plans and debugs by making a series of pragmatic choices, which can be summarized by the tree structured PLANCHART. Furthermore, it is essential that the system is capable of generating, not one solution, but an entire space of progressively less favored solution paths. Also, an implicit assumption runs throughout the analyzer's design that the linguistic analogy is fruitful -- that the solution path consists of structural, semantic, and pragmatic elements. It may be that any synthetic formalism satisfying these constraints is trivially equivalent to an ATN. Such questions are notoriously difficult to answer.

It is probably a virtue that PAZATN is somewhat decoupled from this issue, but one could construe it as a defect. One could argue that somehow the design of the analyzer may be failing to take full advantage of the claims of the theory. A possible alternative design would be to organize PAZATN as an analytic version of the ATN. This "AATN" would have numerically valued arc conditions, representing the plausibility computations of the analytic pragmatics. Note that the event specialists are to be organized internally as decision trees. It is only a small step to reformulate this decision tree structure as a subgraph of an ATN.

It might seem that employing an AATN instead of a coroutine searcher might commit the analyzer to a less powerful automatic backtrack type of control structure. This is not necessarily the case. Depending upon the implementation, the ATN formalism per se carries no irrevocable control structure assumptions. One may traverse the diagram according to any of a wide variety of search strategies. In this respect, the AATN would be attractive, offering greater perspicuity by decoupling efficiency issues from theoretical concerns.

Nevertheless, the AATN design for PAZATN has not been pursued. Although it is possible, in principle, to employ a mixture of top down and bottom up strategies with an ATN, it is more natural to conceptualize an ATN parser as a top down backtracker. To understand their bottom up use, PUSH arcs must be thought of as "IF-REDUCE" arcs; POP arcs must be thought of as "REDUCE" arcs. This felt counterintuitive.

An important issue in the design concerns the breadth of the synthetic theory. There are of course particular lacunae, such as conditional plans, which have been deliberately, but only temporarily, ignored. The greater threat comes from the unknown. Even the youngest children display an incredible richness in their problem solving behavior. PATN's origins are at least partly empirical. But some phenomena, perhaps those most in need of investigation, may have been lost in the process of formalization. This remains a topic for investigation.

A final design issue warrants mention here. PAZATN operates by individually processing each event. But perhaps this leads to too local a perspective. Perhaps larger sized chunks of protocol should be examined at once. In other words, an episode based analyzer might be preferable. The event based design has been selected because it is the simplest, most straightforward approach.

SECTION V

TENTATIVE CONCLUSIONS AND PLANS FOR FUTURE WORK

Recapitulation

In this report we have investigated the problem of analyzing problem solving protocols. The result of this investigation is a preliminary design for PAZATN, a domain independent framework for automatic protocol analysis. The foundation for the approach was a grammatical theory of problem solving as a structured process of planning and debugging. This lead us to the definition of an interpretation as an assignment of a structural description to a list of events, augmented by semantic and pragmatic annotation associated with each node. The foundation for the approach was a grammatical theory of problem solving as a structured process of planning and debugging. This lead us to the definition of an interpretation as an assignment of a structural description to a list of events, augmented by semantic and pragmatic annotation associated with each node.

A key ingredient in the design is a synthetic problem solving system called PATN. PATN employs an augmented transition network to represent fundamental planning concepts, including techniques of identification, decomposition, and reformulation. PAZATN is somewhat decoupled from the ATN representation per se. However, considerable leverage for the analysis process is obtained from PATN's ability to generate successively less preferable solution paths, by a series of pragmatically guided planning decisions, as well as from PATN's characterization of certain bugs as errors in these planning choices.

The analysis procedure has been designed to obtain maximal advantage from both top down synthetic guidance and bottom up analytic constraints. Analysis proceeds by a coroutine search of a space of plausible partial interpretations. The PLANCHART, a data structure resembling an AND/OR goal tree, is used to keep track of synthetic expectations. By careful selection of the representational scheme, this structure achieves considerable storage economy. It is incrementally expanded by the synthetic ATN when existing expectations are inadequate in view of the protocol data. The DATACHART, a data structure analogous to a context layered CONNIVER data base, is used to keep track of the state of alternative partial interpretations.

The analogy to computational linguistics has turned out to be fruitful, providing insights into the parsing process developed in research on language understanding and speech recognition. The value of this analogy is illustrated by the adoption of several search strategies and representational techniques. For example, the chart representation is utilized to economically store well-formed substructures. Partial knowledge of structure and of the status of synthetic expectations is recorded using a scheme of PLANCHART markings and marker propagations. These would allow for considerable efficiency both in storage and in the drawing of inferences regarding possibly ambiguous structural descriptions. Likewise, the basic outlines of PAZATN have been refined by the incorporation of search heuristics prevalent in computational linguistics, including lookahead, least commitment, and differential diagnosis. These would allow the analyzer to proceed with reasonable assumptions when necessary, and yet modify its interpretation in response to anomalies. Ideas for replacing the expert ATN by a version tailored to the individual were discussed. Major design issues and alternatives were also examined.

Although PAZATN is not yet a working program, the design is sufficiently specific so as to be hand simulable. The next phase of the research is to implement and experiment with a prototype analyzer.

Generality of PAZATN

The design of PAZATN is of interest in that it suggests a paradigm for protocol analysis which may be applicable to many domains. Although an operational PAZATN system for a particular task domain requires considerable domain specific knowledge -- a necessity if significant power is to be attained -- its knowledge is extremely modular. This domain specific knowledge is restricted to the event classifier, the event specialists, the lowest levels of PATN, and the answer library. The other modules of PAZATN, which have been emphasized in this report, make no domain specific assumptions in their operation. This suggests that PAZATN systems could be constructed for a variety of domains by supplying "plug-in" modules for these domain specific components.

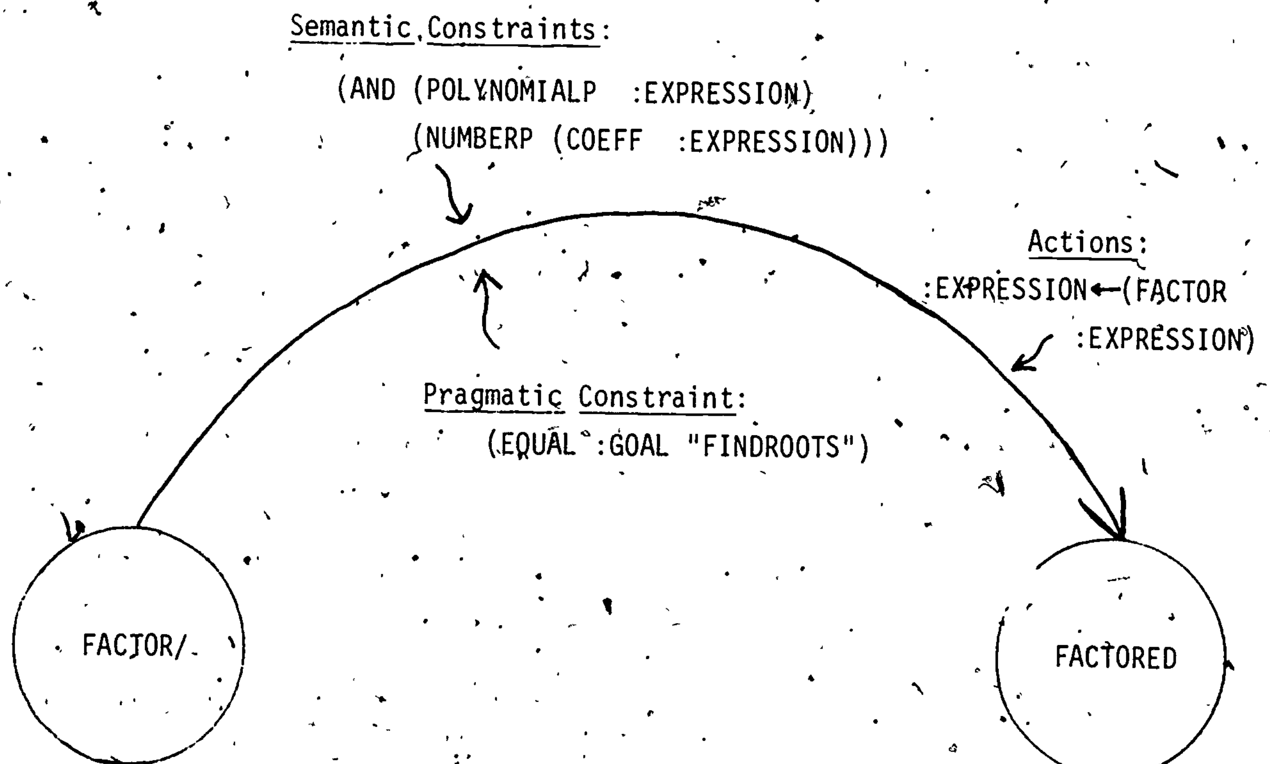
In our early work, a text by Donaghey & Ruddel [1975] was found to be useful in organizing knowledge of elementary algebra into procedural rules. It was found that many students demonstrated an understanding of the rules, and often were able to apply them correctly. Their hardest problem was to recognize the appropriateness of a given rule to a particular problem situation. For example, in actual student protocols, it was observed that students would multiply out an expression, and then, only a few lines later, factor it again. This haphazard application of inverse operations inevitably leads to careless errors, by increasing the length and subjective difficulty of the task.

These algebraic rules can be modeled by a PATN-based synthetic problem solver. Each algebraic transformation operation can be associated with an arc transition on an ATN subgraph. Associated with each transition is a set of semantic and pragmatic constraints on its applicability. For example, to follow the factoring arc, the semantics require that the ?EXPRESSION register to be a polynomial in a single variable with numerical coefficients. The pragmatics indicate that this is an appropriate transition when the goal is to determine the roots of the polynomial (see Figure 14). While many students will have learned the syntax of the transitions, which is usually all that is taught, their weaknesses often lie in not knowing the appropriate semantic and pragmatic constraints.

A feature of programming environments, which has been helpful in thinking about the PAZATN system for that domain, is that a great deal of the student's reasoning is manifest in the protocol. Not all CAI environments share this property. PAZATN would have more difficulty with domains for which the "bandwidth" of the analyzer's window into the student's thinking is low. This might be a problem in applying the paradigm to WUMPUS [Stansfield and Carr 1976], WEST [Brown and Burton 1976], or SOPHIE [Brown et al. 1976]. For example, in the electronic troubleshooting scenario, the student requests a particular measurement, but provides no indication of the pragmatics -- the reasoning which led to that measurement rather than another. Since there are many routes by which the misguided troubleshooter could have arrived at the requested measurement, a precarious chain of statistical inferences from multiple trials is required to pinpoint the student's underlying confusion.

Probably this would pose problems for any analyzer. Hence, the extent to which the student's reasoning is articulated suggests itself as a dimension along which to evaluate designs for future CAI environments. Note that this is a property not only of the domain, but also of the particular scenario used. For example, in the electronics domain, one can envision a design scenario which would closely mimic the alleged virtues of the programming world. (It would be essential to contrast the reasoning strategies required for debugging an erroneous design to those needed for troubleshooting a faulty component in a properly

Figure 14. Subgraph of Algebra ATN



designed circuit.) Another possibility is to ask the student to explain his reasoning. The major stumbling block to such an undertaking at the present time lies not in inadequate theories of problem solving, but in the understanding of natural language.

REFERENCES

Aho, A.V., & Ullman, J.D. The theory of parsing, translation, and compiling, Volume 1: Parsing. Englewood Cliffs, New Jersey: Prentice-Hall, 1972.

Barr, A. et al. A rationale and description of the basic instructional program. Psychology and Education Series, Stanford University, Technical Report 228, April 1974.

Brown, J.S., & Burton, R.R. Systematic understanding: Synthesis, analysis, and contingent knowledge in specialized understanding systems. In D. Bobrow and A. Collins (Eds.), Representation and Understanding: Studies in Cognitive Science, New York: Academic Press, 1975.

Brown, J.S., & Collins, A. Artificial intelligence and learning strategies. To appear in H.F. O'Neil (Ed.), Learning strategies. New York: Academic Press, 1978, in press.

Brown, J.S., Burton, R.R., & Bell, A.G. SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI) (Final Report). AFHRL-TR-74-77, October 1974.

Brown, J.S., Burton R.R., Miller, M.L., DeKleer, J., Purcell, S., Hausmann, C., & Bobrow, R. Steps toward a theoretical foundation for complex knowledge-based CAI (Final Report). Bolt, Beranek and Newman, August 1975.

Brown, J. S., Rubinstein, R., & Burton, R.R. Reactive learning environment for computer assisted electronics instruction. Bolt Beranek and Newman Inc, Report 3314, (ICAI Report 1), October 1976.

Burton, R.R., & Brown, J.S. A tutoring and student modelling paradigm for gaming environments. In Proceedings for the Symposium on Computer Science and Education, Anaheim, California, February 1976.

Carbonell, J., & Collins, A. Natural semantics in artificial intelligence. In Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, 1973.

Carr, B., & Goldstein, I. Overlays: A theory of modelling for computer aided instruction. Massachusetts Institute of Technology, AI Memo 406, February 1977.

Charniak, E. Toward a model of children's story comprehension. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 266, December 1972.

Chomsky, N. Aspects of the theory of syntax. Cambridge, Massachusetts: The M.I.T. Press, 1965.

Collins, A., Warnock, E., & Passafiume, J. Analysis and synthesis of tutorial dialogues. In G.H. Bower (Ed.), Advances in Learning and Motivation, Vol. 9, 1975.

Donaghey, R., & Ruddel, J.A. Procedures of elementary algebra. New York: Academic Press, 1975.

Goldstein, I. The computer as coach: An athletic paradigm for intellectual education. Massachusetts Institute of Technology, AI Memo 389, January 1977.

Goldstein, I.P. Understanding simple picture programs. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 294, September 1974.

Goldstein, I.P., & Miller, M.L. AI based personal learning environments: Directions for long term research. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 383 (Logo Memo 30), October 1976.

Goldstein, I.P., & Miller, M.L. AI-based personal learning environments: Directions for long term research. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 384 (Logo Memo 31), December 1976a.

Goldstein, I.P., & Miller, M.L. Structured planning and debugging: A linguistic theory of design. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 387 (Logo Memo 34), December 1976b.

Kay, M. The MIND System. In Randall Rustin (Ed.), Natural Language Processing, Courant Computer Science Symposium 8 (December 20-21, 1971), New York: Algorithmics Press, 1973, pp. 155-188.

Krauss, R.M., & Glucksberg, S. Social and nonsocial speech. Scientific American, 1977, 236(2), 100-105.

Miller, M.L., & Goldstein, I.P. Overview of a linguistic Theory of Design. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 383a (Logo Memo 30a), February 1977.

Miller, M.L., & Goldstein, I.P. Parsing protocols using problem solving grammars. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 385 (Logo Memo 32), December 1976b.

Miller, M.L., & Goldstein I.P. SPADE: A grammar based editor for planning and debugging programs. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 386 (Logo Memo 33), December 1976c.

Miller, M.L., & Goldstein, I.P. PAZATN: A linguistic approach to automatic analysis of elementary programming protocols. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 388 (Logo Memo 35), December 1976e.

Rich, C., & Shrobe, H.E. Initial report on a LISP programmer's apprentice. Massachusetts Institute of Technology, AI-TR-354, December 1976.

Rubin, A. Hypothesis formation and evaluation in medical diagnosis. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 316, January 1975.

Sacerdoti, E. A structure for plans and behavior. Stanford Research Institute, Artificial Intelligence Center Technical Note 109, August 1975.

Sacerdoti, E. The nonlinear nature of plans. In Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, U.S.S.R., September 1975, pp. 206-218.

Self, J.A. Student models in computer-aided instruction. International Journal of Man-Machine Studies, 1974, 6, 261-276.

Stansfield, J.L., & Carr, B. The Wumpus Advisor (Draft). Massachusetts Institute of Technology, Artificial Intelligence Laboratory, forthcoming Memo, July 1976.

Sussman, G.J., & McDermott, D.V. From PLANNER to CONNIVER -- A Genetic Approach. Proceedings of Fall Joint Computer Conference, Montvale, New Jersey, American Federation of Information Processing Societies, 1972.

West, T. Diagnosing pupil errors: looking for patterns. The Arithmetic Teacher, November 1971.

Woods, W.A. Transition network grammars for natural language analysis. Communications of the Association For Computing Machinery, October 1970, 13(10), 591-606.

APPENDIX I

RED TEST

Student 1:

83	330	89	354
+106	+187	+132	+69
189	417	111	313

Explanation:

Student 2:

94	498	77	48
+115	+215	+26	+41
119	611	91	89

Explanation:

Student 3:

347	758	437	923
+139	+296	+284	+481
476	944	601	1404

Explanation:

Student 4:

109	98	98	35
+452	+105	+111	+64
501	103	209	99

Explanation:

Student 5:

352	784	1784	8
+18	+3080	+3080	+35
360	6364	7364	63

Explanation:

Student 6:

8372	6527	893	63
-657	-2394	-195	-47
6725	3233	608	16

Explanation:

Student 7:

913	5394	477	893
-76	-797	-284	-195
777	4497	101	718

Explanation:

Student 8:

394	77	935	126
-166	-53	-361	-117
248	24	774	29

Explanation:

Student 9:

48	394	57	239
-15	-166	-23	-95
43	340	60	124

Explanation:

Student 10:

305	987	340	9280
-108	-320	-56	-6090
107	667	290	3090

Explanation:

112

Appendix 2

List of all responses to the question:

What do you think you learned from this experience?

I see from this system that you learn from your mistakes. In a certain operation there are so many mistakes that you can make. When you learn what the mistakes are you learn to do the operation correctly.

That children's errors can be a way of diagnosing the way the child learns material. Also it raises questions about the way a child is tested both standardized and informally.

A student's errors and/or misunderstanding of a concept may have not been due to carelessness but rather involved a complex and logical thought process.

I learned that it is necessary to try many different types of examples to be sure that a child really understands. Different types of difficulties arise with different problems.

Trying to beat the machine can be challenging. Feedback is extremely important in trying to determine the error. It's difficult for me to describe the error but the machine doesn't care as long as I can prove my point through examples.

Although it's hard to tell from these pre and post tests, in the middle is learned a great deal about the complexity of student's errors. I know that young students can get these preconceived notions about how to do things and it's very hard to find a pattern to their errors but there is and I believe that BUGGY convinced me of [it].

That if you study the errors long enough you can eventually come up with a reasonable solution as to why the [error] is occurring.

Through looking carefully at children's math errors it is sometimes possible to discover a pattern to them. This pattern will tell you an area or a concept the child does not understand.

I learned that there could be more to a child's mistakes other than carelessness. Working with children with special needs I have encountered many such problems, yet never stopped to analyze what could be a systematic problem -- for this I thank you.

Children do have problems and they are very difficult to spot especially when a number of different operations are used to come to an answer. I've learned to be more aware of how these children reach these "answers" and to help them to correct them; first by knowing how they arrived at the answer.

Although many arithmetic errors may be careless, there may also be a pattern that the kid is locked into. If you pick up on a pattern you can test the child to see if he/she conforms to it and work on it from there.

The types of analysis necessary to "debug" student errors on the test (paper/pencil) seems more difficult than with the computer. But that doesn't make any sense. The "analysis" ought to be the same. Perhaps the computer motivated my analytical ability.

I found that I have looked closer at the problems, looking for a relationship between the set after working with BUGGY.

How to perceive problems, that don't look too consistent, a little easier. How to have a good time with a computer. (I've only played tic-tac-toe at the Science Museum, and have always wanted to do more). Machines can be tempermental (when pestered by a large number of students?)

I learned and was exposed to the many different types of problems children might have. I never realized the many different ways a child could devise his own system to do a problem. I am now aware of problems that could arise and I'm sure this will help me [in] my future career as a teacher.

How to more effectively detect "problems" students have with place value.

That you can find causes of a child's problem without the child's work in front of you. In looking for the "bug", up and down aren't the only possibilities, also diagonally. I suppose horizontally also. How specific the problem might be -- only works in one situation.

I have learned several new possible errors students may make in computation. I have also learned somewhat how to diagnose these errors, i.e. what to look for, and how specific errors can be.

I think I learned more about computers and how to use them. Also I learned about diagnosing math difficulties. It makes me aware of problems that children have and they sometimes think logically, not carelessly as sometimes teachers think they do.

I learned that computers are very complicated pieces of machinery. If one isn't experienced with the mechanisms, then problems could result. That computers can be an asset to the classroom is not doubted, but I think many problems can result. They can add much to a classroom until they start breaking down.

That there are many problems that you can diagnose about a child by looking at his homework.

If a child has repeatedly made [the] same mistakes, it is more easily identified if the teacher has an opportunity to try and make [the] same mistakes. This method can be solved at least quicker than...

Computers are concise. Information can be gathered and stored for reference.

Tuned in to picking up malfunctions in simple addition and subtraction which seemed to be realistic problems.

Appendix 3

List of all responses to the question:

What is your reaction to BUGGY?

I think it would be a fantastic resource for a school with a lot of money to spend.

Too early to tell. But the potential seems stupendous. I enjoyed it and see it as a powerful future tool.

I like it.

Working with a partner is good for being forced to explain (defend) your theory [as long as partner requires that]. Useful tool for those with pretty good number ability. What about those who don't have good feeling for numbers?

Good!!! Forces one to get very specific answer to the problem. You can be slightly wrong and then, rather moving way off base in your second theory as to the problem, you pinpoint/modify your first (assuming it's almost right). Bad. It's too much fun and I wasn't being very professional in mu usage (though under different situation I might).

I think this system is fantastic. It's a wonderful way to expose people (who are involved with children) to the problems children will probably have. It might be especially useful with special learning needs children:

It's great! When will it be in my "price" range?

As for the game itself, it would have been continued for another 3 or 4 hours.

I think it's an excellent device for trying to diagnose some of the difficulties found in mathematics. For a teacher the time element -- having the machine diagnosis would be more practical.

It's a nice toy.

The Bug is great. Makes you stop and think.

I enjoyed the BUGGY experience extensively. Solving or determining errors was much easier on the computer -- and fun too!

I enjoyed working with BUGGY but when it breaks down it is very frustrating. This might be difficult for children to understand that problems with computers do arise. Also it may be complicated for younger children to understand how to use it. High school students may enjoy it though.

I think BUGGY would be a definite "plus" in the classroom but right now I feel there are too many "bugs" with BUGGY. Too many times did BUGGY go crazy. I find it amazing though that a machine can help one detect problems. It sure is a better way than the present.

BUGGY makes one look at each problem carefully and detect exactly what a child cannot do or cannot comprehend without formal testing.

As far as BUGGY is concerned, I had a very good time "playing" with BUGGY. It was quicker and somehow easier than pencil and paper. It took less concentration and was definitely more efficient. Can this be used as a strictly diagnostic tool? If so, I think that BUGGY is great.

He's a trip! Seriously, he's fine if you can master him in case he decides to break down.

I think BUGGY is a good idea and would like to hear more about it.

It's a program that should be further researched and has excellent potential.

Great experience in beginning to play with computers -- exercised problem focussing without frustrating a child with inadequate preparation.

I think that BUGGY could be used to sharpen a teacher's awareness of different difficulties with addition and subtraction. It might be fun for the kids to play such a game together.

Appendix 4

This appendix presents answers and descriptions for some of the subtraction bugs for the problem:

```
15300
-9522
-----
 5778
```

95778: When borrowing from a column which has a 1 on top, the student treats the 1 as if it were a 10.

27998: When borrowing is necessary, instead of subtracting 1 from the top digit of the next column, the student adds 1 to it.

24822: The student adds instead of subtracts.

16888: When the student needs to borrow, he adds 10 to the top digit of the current column without subtracting 1 from the top digit of the next column.

15778: The student borrows correctly except he doesn't take 1 from the top digits that are over blanks.

14822: The student adds without carrying instead of subtracts.

14378: The student subtracts the smaller digit in a column from the larger digit regardless of which is on top. and No matter what other bugs the student may have, he performs the units column correctly even if it requires borrowing.

14222: The student subtracts the smaller digit in each column from the larger regardless of which is on top. The exception is when 10 is in the left-most columns of the top number; in this case 10 is treated like a single digit.

14222: The student subtracts the smaller digit in a column from the larger digit regardless of which is on top.

14200: The student subtracts the smaller digit in each column from the larger digit regardless of which is on top. The exception is when the top digit is 0, in which case a 0 is written as the answer for that column, i.e. 0-N=0.

10022: The student doesn't know how to borrow. If the top digit in a column is 0, the student writes the bottom digit in the answer (i.e. 0-N=N). If the top digit is smaller than the bottom digit, then 0 is written in the answer.

10000: The student writes a 0 in any column in which borrowing is needed.

8748: The student gets 6 and 9 mixed up when decoding (reading) the digits in the problem, misreading 6 for 9, and 9 for 6.

7998: When borrowing from a column, the student borrows from the larger digit disregarding whether it is the top or the bottom digit.

6888: The student will only borrow from a column in which the top digit is larger. In the columns he skips (where the top digit is smaller) he automatically adds 10 to the top digit.

6822: The student borrows from the next column to the left which has a larger top digit. Any intervening columns have 10 added to their top digit. The exception is when 0 is on top in which case the student writes the bottom number in the answer (e.g. 0-N=N).

5878: When borrowing from a column whose top digit is 0, the student writes 9, but does not continue borrowing from the column to the left of the 0.

5822: Whenever the top digit in a column is 0, the student writes the bottom digit in the answer, i.e. 0-N=N.

5800: Whenever the top digit in a column is 0, the student writes 0 in the answer, i.e. 0-N=0.

5798: When borrowing from a column with 0 on top, the student borrows from the bottom digit instead of the 0 on top. In all other cases the student borrows correctly.

5788: The student forgets to change 10 to 9 after borrowing into a column whose top digit is 0.

5688: When the student needs to borrow from a column whose top digit is 0, he skips that column and borrows from the next one.

5678: Once the student needs to borrow from a column, he continues to borrow into every column whether he needs to or not.

5372: When faced with borrowing, the student decrements the next column correctly, but instead of adding ten to the top digit of the current column, he simply subtracts the smaller digit from the larger digit even though the smaller digit is on top.

4822: The student adds instead of subtracts, but when carrying he subtracts the carry from the top digit of the next column instead of adding it.

4222: The student subtracts the smaller digit in a column from the larger digit regardless of which is on top.
and The student stops working the problem as soon as the bottom number runs out.