The Role of Plans in Intelligent Teaching Systems.
Michael R. Genesereth,
Nov 1980

STANFORD HEURISTIC PROGRAMMING PROJECT
Memo HPP-80-4

November 1980

DEPARTMENT OF COMPUTER SCIENCE
Report No. STAN-CS-80-842

The Role of Plans in Intelligent Teaching Systems

by

Michael R. Genesereth

DEPARTMENT OF COMPUTER SCIENCE
School of Humanities and Sciences
STANFORD UNIVERSITY

1. Introduction

One of the k ys to effective remedial instruction is information about the student's beliefs and misconceptions. This information can often be obtained by analyzing the student's efforts in solving problems that require knowledge of the subject matter. In some areas, it is possible to design diagnostic tests that can pinpoint a student's misconception directly from his answers. For example, in their work on the Buggy system, Burton and Brown developed a methodology for automatically generating diagnostic tests that discover the underlying problems responsible for a student's errors in subtraction. However, in subject areas where the student chooses his own problem or when there are several ways of solving a problem, it is helpful to consider not only the student's final answer but also his steps in producing it and his rationale for taking those steps. This paper is concerned with the process of automatically analyzing a student's work to infer his rationale and the use of this information in providing remediation tailored to the student.

1.1 The Student's Plan

The student's rationale is important because it is the link between his beliefs about the subject area and his solution. In the exercise of elementary arithmetic skills like addition and subtraction, this link is direct. The sequence of steps is fixed in advance, and the student simply executes the steps*. If there is an error it can be localized by following the student's steps and comparing his actions with those of the correct algorithm. In more complex domains like algebra, geometry, and calculus, there is no fixed procedure that solves all problems. Consequently, the student must piece one together for each new problem, and generally there are multiple ways this can be done. Identifying a misconception in this type of situation is more difficult and requires an understanding of why the student chose the steps he did.

{Footnote: This assumes that the student already has command of some procedure, possibly an incorrect one. In actual practice, the student might revise his procedure while trying to apply it to a novel problem.}

There is a crucial distinction here between the student's steps in solving a problem and his mental operations in selecting those steps. The steps of the solution manipulate the objects of the problem (e.g. mathematical expressions and equations in algebra, drawings and proofs in geometry); the mental operations take goals and beliefs as "inputs" and produce step sequences as "outputs". For example, given a polynomial to solve, a student must first recall an appropriate method (like using the quadratic formula) or invent one he hasn't seen before (say factoring); he must check the method's preconditions (like integer coefficients) and figure out how to satisfy them (e.g. multiplying through by a constant); and only then can he begin to carry out the steps.

Insofar as a trace of the student's mental operations explains why he chose the steps he did, it constitutes a rationale for the solution. If the operations are guaranteed to produce correct solutions (given correct facts about the actions and the problem area), then the trace constitutes a direct proof that the solution is correct. If the methods are merely plausible, then the trace constitutes only a plausible argument but still captures the rationale. In what follows the word *plan* will be used to designate a student's solution, the relevant beliefs, and a trace of his mental operations in using these beliefs to produce the solution.

One advantage of recognizing the distinction between the steps of a solution and the mental operations that produce it is that it suggests what it means for a solution to "make sense". Even when a sequence of steps is incorrect, we sometimes deem the solution "reasonable" in light of revealed misconceptions on the part of the student. We snap our fingers and exclaim "Ah, I see what he's doing" and sometimes "But he's wrong about x". This often occurs in grading problem sets and quizzes where one tries to understand the student's problem solving "logic" in order to

detect his misconceptions. One possible explanation for this is that we recognize the student's mental operations are correct and that the student's mistake is due to a misconception about the subject area.

As an illustration of this, consider the MACSYMA* interactions in figure 1. In each case the student was asked to solve the same problem, and each student used a different approach. The command lines (e.g. C6) are typed by the user; the display lines (e.g. D6) are MACSYMA's response. A command line may be terminated by a semicolon or a dollar sign; if the latter is used, the display is suppressed (e.g. line C7). Expressions are entered in linear form (e.g. (3-2*z)/z); colon is the assignment operator; and other commands are applied using functional notation (e.g. Factor(D6,X)). Parentheses delimit compound statements.

{Footnote: MACSYMA [Mathlab] is a large, interactive computer system designed to assist mathematicians, scientists, and engineers in performing symbolic manipulation of mathematical expressions. For the purposes of this paper, one need only realize that it has a large number of complex commands and options. One of the advantages of using computers as a subject area is that all of the student's actions are explicit (as "C lines"), whereas in other domains only the intermediate results are shown (the "D lines"). One of the difficulties of applying the techniques described here to other domains is that these intermediate actions must first be identified.}

Arthur's and Bertram's solutions are correct; and, even without knowledge of MACSYMA, one quickly sees how they got their answers. Arthur obtained his solution by factoring the expression and extracting one of the factors. Bertram expanded the expression, picked out the coefficients, and plugged them into the quadratic formula. Carleton's solution is similar to Arthur's but is incorrect because it is missing the preliminary expansion. The COEFF command does not return the coefficient of its argument in all cases, as one might suspect. What it does do is to loop over the terms of the argument collecting coefficients of the variable raised to the desired degree. Since there are no terms in D6 free of X, COEFF(D6, X, 0) returns 0. The reason for the name is that the command *does* compute the coefficient when the argument is expanded. Although the solution is incorrect, to many people it seems reasonable in light of this misconception.

1.2 Plan Recognition

MACSYMA experts are quick to diagnose Carleton's misconception, even those who haven't seen the problem before. The intermediate steps in solving the problem are important in making this determination, but an understanding of the student's plan is also essential in pinpointing the misconception. The reason is that, taken individually, the steps are correct; it's only their composition that's wrong. Note that it is not illegal to use the COEFF command on D6; in fact, it is often used on non-expanded polynomials in finding roots over polynomial domains. Its use in figure 1c is incorrect only because it was expected to produce the coefficient of D6 to plug into the quadratic formula in line C8.

The work of the artificial intelligence community on planning suggests that it is possible to characterize the mental operations a person uses in devising a sequence of actions to solve a problem. Given this characterization, the plan recognition process can be automated. The task is a difficult one, especially where there is a possibility that the solution is incorrect, as in Carleton's case. However, a number of researchers have already presented plan recognition algorithms, including [Genesereth 78, 81], [Miller and Goldstein], [Schmidt and Sridharan], and [Waters].

In their work on computer-aided instruction, Brown and Burton pioneered the use of the expert comparison technique in diagnosing student errors. In SOPHIE-II [Brown, Burton, and Bell], WEST [Burton & Brown], and BUGGY [Brown & Burton], misconceptions were inferred by comparing the student's solutions with those of an automated expert or an expert with various bugs inserted. However, in none of these systems were the student's answers or actions used in ordering the search. Plan recognition is a way of using information about the student's actions in dealing with the combinatorics in domains where the number of reasonable solutions and bugs is too large for the expert difference technique to work effectively.

```
(C6)  X↑2-Z*(X+X*Y-Y*Z);
(D6)                    X^2 - Z (X + X Y - Y Z)

(C7)  FACTOR(D6, X);
(D7)                          (X - Z) (X - Y Z)

(C8)  PART(D7, 1, 2);
(D8)                                  Z
```

Figure 1a - Arthur's solution by factoring

```
(C6)  X↑2-Z*(X+X*Y-Y*Z);
(D6)                    X^2 - Z (X + X Y - Y Z)

(C7)  EXPAND(D6);
(D7)                    X^2 - X Z - X Y Z + Y Z^2

(C8)  (A:COEFF(D7, X, 2), B:COEFF(D7, X, 1), C:COEFF(D7, X, 0))$

(C9)  (-B+SQRT(B↑2-4*A*C))/(2*A);
(D9)                                  Z
```

Figure 1b - Bertram's solution using the quadratic formula

```
(C6)  X↑2-Z*(X+X*Y-Y*Z);
(D6)                    X^2 - Z (X + X Y - Y Z)

(C7)  (A:COEFF(D6, X, 2), B:COEFF(D6, X, 1), C:COEFF(D6, X, 0))$

(C8)  (-B+SQRT(B↑2-4*A*C))/(2*A);
(D8)                                  0
```

Figure 1c - Carleton's incorrect solution

1.3 Plan Use

In addition to helping pinpoint the student's misconception, studying his plan is advantageous in that it enables the tutor to offer remediation in the context of the student's problem and his approach to solving it. Consider, for example, the consultation session in figure 2. The "Advisor" in this case is a consultant for MACSYMA, and the user is the novice responsible for the session in figure 1c. After accepting a statement of the user's goal and complaint, the consultant looks at his work, suspects a misconception about coeff, and confirms it with a question. He then corrects the misconception and offers an alternative in the form of the ratcoeff command. This alternative fits nicely into the user's plan, and its importance is emphasized by its use in context.

Consultant: Speak up!

User: I was trying to solve D6 for X, and I got 0.

Consultant: Did you expect Coeff to return the coefficient of D6?

User: Yes, doesn't it?

Consultant: Coeff(exp, var, pow) returns the correct coefficient of var^{pow} in exp only if exp is expanded with respect to var. Perhaps you should use Ratcoeff.

User: Ok, thanks. Bye.

Figure 2 - An example of MACSYMA Consultation

This paper discusses some approaches to automating the process of plan recognition and using plans to detect misconceptions and provide remediation in context. Fundamental to this endeavor is a formal representation for plans. Section 2 offers a precise definition for the notion of a plan as a "dependency graph" relating a student's actions to his beliefs about the problem area and the actions involved via the problem solving methods he used in piecing together his solution. The recognition of plans in this representation is discussed in section 3, and the confirmation of tentative plans is discussed in section 4. Section 5 indicates how confirmed plans can be used in providing advice tailored to the student. The final section sums up the key points of the paper and indicates directions for future work.

At least one tutorial program has been written in which plan recognition is used to determine student misconceptions. The MACSYMA Advisor [Genesereth 78] is an automated consultant for MACSYMA. It is a program separate from MACSYMA, with its own own data base and expertise. The Advisor accepts a description of a violated expectation from its user, tries to reconstruct his plan, and if successful generates advice tailored to his need. For example, the Advisor is capable of conducting the consultation shown in figure 3. Although the Advisor was developed as a MACSYMA consultant, this does not mean that the techniques are in any way restricted to MACSYMA, only that they have not yet been tested in other subject areas.

2. Plans

Intuitively, a plan is a program of action, or sequence of steps, designed to achieve a desired goal. In this paper, the notion of plans is extended to include an argument for how the steps achieve the goal in terms of the planner's beliefs about the problem area and the characteristics of the actions involved. In imputing a plan to a student's actions, there is an underlying assumption that each of those actions is in some way intended to contribute to the overall goal. For sure, student's often exhibit purposeful exploration not directed toward solving any particular problem, but for simplicity such possibilities are ignored here.

The part of a plan explaining how the sequence of actions achieves its goal can be thought of as a logical argument in which the goal is the conclusion and the actions and relevant facts about the

problem area are the premises. The view taken here is that a trace of the goals and subgoals, methods and assumptions that a problem solver uses in generating a sequence of actions is an appropriate structure for such an argument. This idea of using a computational trace as an argument for the correctness of a solution is similar to the explanation facility of MYCIN [Shortliffe] and the dependency relations in EL [Stallman & Sussman].

If one of the user's beliefs about a problem area and the available actions are incorrect, then the resulting action sequence may not achieve its goal, even though all of the problem solving methods are correct. The MACSYMA session in figure 1c is an example. The user's strategy was flawless; it was his misconception about coeff that led to the difficulty. Once that was corrected, he successfully solved the problem, using the same plan except with coeff replaced by ratcoeff.

In order to make this notion of plans precise, the following discussion will concentrate on a particular problem solver called MUSER (for MACSYMA user), designed to mimic the behavior of novice users of MACSYMA. MUSER takes as argument a goal specified in a formal, predicate calculus-like language and uses facts about MACSYMA and mathematics to produce a sequence of MACSYMA commands that computes the desired result. In doing so, it uses a variety of problem solving methods that either achieve the goal directly or generate appropriate subgoals. The goal language and problem solving methods were chosen after analysis of several dozen scenarios of MACSYMA usage and include many domain-independent as well as domain-specific techniques. Only a few of these techniques are shown here.

2.1 Goals

In MUSER goals take the form of abstract operations in which implementational details are left unspecified. There are several different types of goals, and each has a corresponding set of parameters. For example, the goal **(obtain (coefficient g6 x 2))** means to obtain an expression for the coefficient of x^2 in g6, either to print it out or pass as argument to some MACSYMA command. Note that this can be done either by finding an already computed expression (stored, say, as the value of some variable) or by computing it anew. Either implementation is satisfactory so long as it computes the desired expression. **(Achieve (value (a 2) 6))** means to modify the environment so that the value of the array element **(a 2)** is 6. As with **obtain**, one can do this in a variety of ways, e.g. by creating a new array and assigning it to **a** or by using the array assignment command. Similarly, there are goals for testing facts, e.g. **(test (value (a 2) 6))**, and proving them without further testing, e.g. **(prove (value (a 2) 6))**.

Many of MUSER's goal types exist merely for expressiveness and efficiency and can be subsumed by others. For example, the goal **(find (coefficient g6 x 2))** means to obtain an expression for the coefficient of x^2 in g6 that has already been computed, whereas **(construct (coefficient g6 x 2))** means to compute it anew. **(Transform a p)** means to convert the expression **a** into a mathematically equivalent expression that satisfies **p**. **(Achieve-by-round (all (i) (if (< i 10) (value (a i) 0))))** means to set the first 10 elements of the array **a** to 0 by iteration.

2.2 Planning Methods

The planning methods in MUSER are internal operations that add implementational detail to evolving plans. Each method has associated with it sets of input and output objects and a procedural definition. Each methods satisfies a particular type of goal and is annotated with prerequisites and postrequisites (conditions that must be true in order for the method to succeed and those which become true after it is executed). Some planning methods produce complete plans to achieve their goals; some merely generate subgoals; others are mixed in their effects.

The **find-a-variable** method satisfies a goal of the form **(obtain g)** and returns it as value. The procedural definition of **find-a-variable** is shown in figure 3. It uses the subroutine **fetch** to search its data base for a variable v such that val(v)=g. (The "?" prefix here designates the symbol that follows it as a pattern variable; the "," indicates that the value of the variable f is to be plugged into the pattern.) If it succeeds in finding an appropriate variable, it passes it as argument to the MACSYMA evaluator.

```
(defun find-a-variable (g)
  (prog (v)
    (setq v (fetch '(val ?v ,g)))
    (action 'meval v)))


(defun valuate (f a)
  (prog (p c g)
    (setq (p c) (fetch '(all x y (if (?p x) (= (,f x) (?c x))))))
    (setq g (obtain a))
    (if (not (fetch '(,p ,a))) (setq g (transform g p)))
        (action c g)))


(defun convert (a p)
  (prog (c)
    (setq c (fetch '(all x (and (= (?c x) x) (,p (?c x))))))
    (action c a)))


(defun solve-by-formula (x v)
  (prog (a b c)
    (setq a (obtain '(coefficient ,x ,v 2))
          b (obtain '(coefficient ,x ,v 1))
          c (obtain '(coefficient ,x ,v 0)))
    (action 'pquad a b c)))


(defun solve-by-factoring (x v)
   (obtain '(first ,(obtain '(factors ,x ,v)))))
```

Figure 3 - Some of MUSER's problem solving methods

The fetch method used here is MUSER's data base access function. It searches the data base for an assertion of the form specified in its argument and returns as value an alist of bindings for the existential variables in the pattern. The action method is MUSER's way of adding actions to its plan. It takes as arguments the name of a command and descriptions of its inputs and outputs. In the case of find-a-variable, the command is meval, the input is a variable, the output is the goal; and, when the plan is executed, the variable will be evaluated to obtain the desired expression.

The alternative to retrieving a pre-computed object is to construct a new one. The goal of the valuate method is to produce an object defined as the value of some function f applied to an argument a. The strategy is to search the data base for a command that computes f, given preconditions on the argument. The definition for valuate is shown in figure 3. The outputs of the call to fetch are the command c and the preconditions p. Valuate then obtains the argument a, transforms it into an expression that satisfies p, and then calls c. The extension to multiple argument functions is straightforward.

The convert method is intended to satisfy the goal (transform a p), i.e. to produce an expression equivalent to a that satisfies p. To do so, it searches the data base for a MACSYMA command c that preserves mathematical equivalence but achieves p. If it succeeds, it calls c on a and returns the answer. The definition is shown in figure 3.

In addition to the domain-semidependent methods shown here, Muser also uses domain-specific procedures. As examples, consider the procedures use-quadratic-formula and solve-by-factoring shown in figure 3. Each has the same goal but they differ drastically in their strategies.

2.3 Plans

Plans are dependency graphs that explain how a student's actions are supposed to achieve his goals in terms of his beliefs about the problem area and the actions involved. Each goal is related to its subgoals and the actions used to achieve it by some problem solving method; and each method is annotated with its inputs and outputs. In particular, all calls to the fetch subroutine are annotated with the data base assertions they retrieved.

Figure 4 shows a partial plan for transforming the expression called g6 to expanded form. Each box represents a problem solving method, with its goal written in the upper half and its name in the lower. The inputs and outputs are indicated to the left and right. In the convert method, MUSER looks in its database for a subroutine that returns an expanded expression equivalent to its input and finds the assertion (all x (and (= (expand x) x) (expanded (expand x)))). Fetch returns expand, and it is added to the sequence of steps to be performed. In this plan the problem solver's expand action is explained in terms of his goal (transform g6 expanded) and his underlying belief about expand.

Figure 5 shows a plan for computing the positive root of g6 using the quadratic formula. As defined in figure 3, the use-quadratic-formula method first computes the coefficients of the argument and then plugs them into the formula. These subgoals are indicated by the obtain boxes connected to the use-quadratic-formula box. For simplicity only the subplan for obtaining one of the coefficients is shown. The problem solver's choice in this case is to use the valuate method. The assertion attached to the fetch box indicates that coeff does the job so long as the first argument is expanded. In order to obtain g6 in expanded form, valuate creates the subgoals (obtain g6) and (transform g6 expanded). In satisfying the obtain subgoal, the find-a-variable method discovers that the variable d6 has g6 as value and starts off the plan with an evaluation of d6. In trying to transform g6 to expanded form, convert finds that the expand command returns an expanded version mathematically equivalent to its argument. Hence, expand is called on the value of d6, and the result ge is passed as argument to coeff along with the results of obtaining x and 1. Finally, the coefficients are fed into the quadratic formula to obtain the root g8.

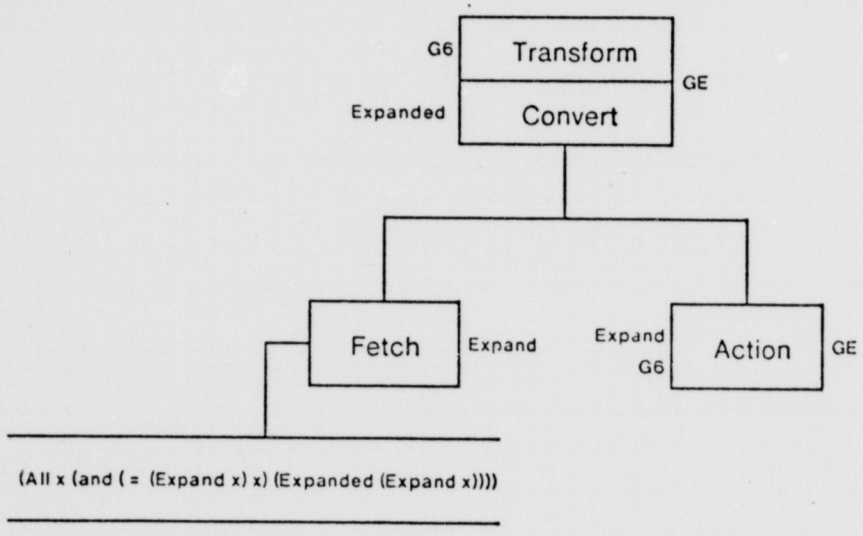

Figure 4 - Plan to get G6 into expanded form

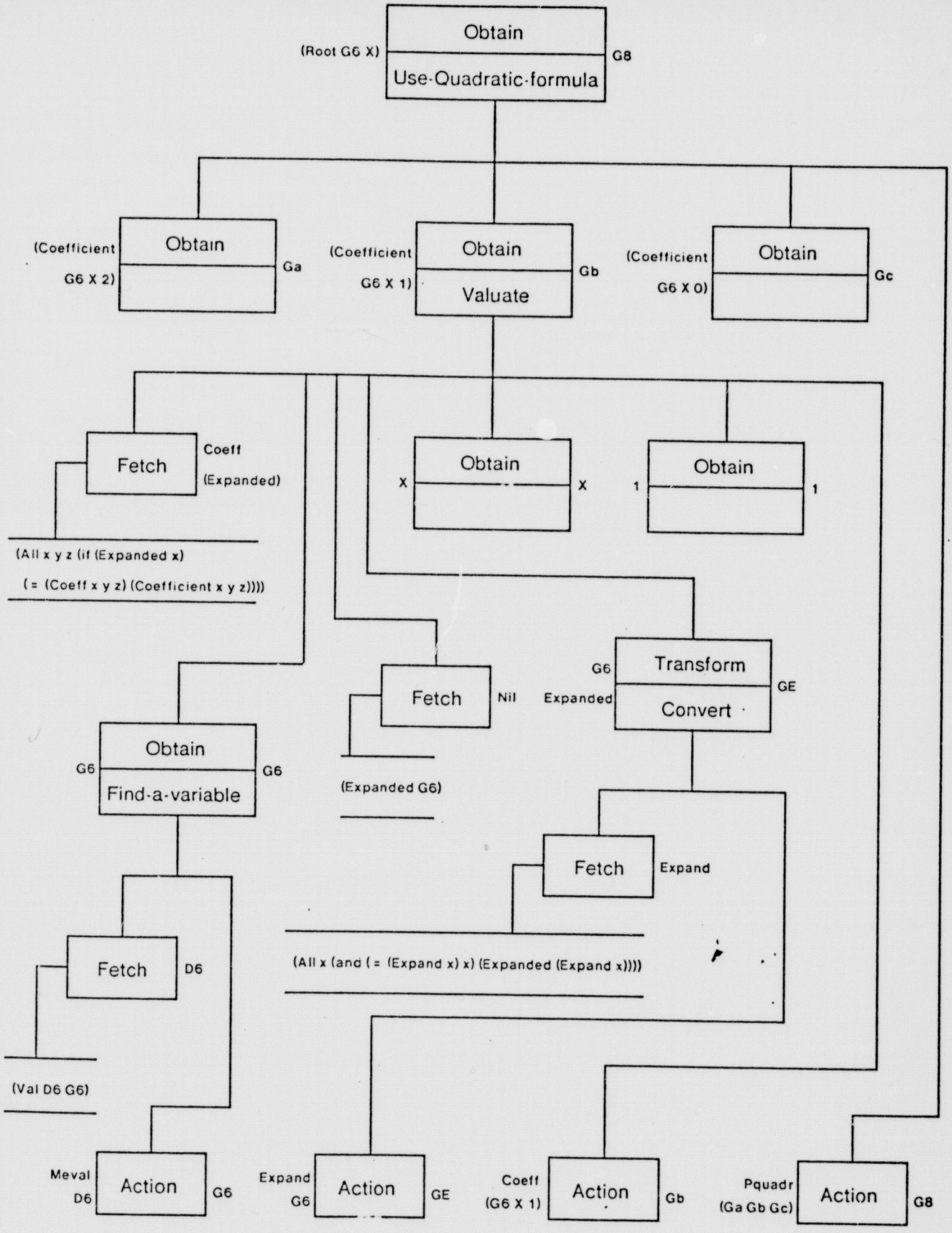

Figure 5 - Plan to compute the root of G6

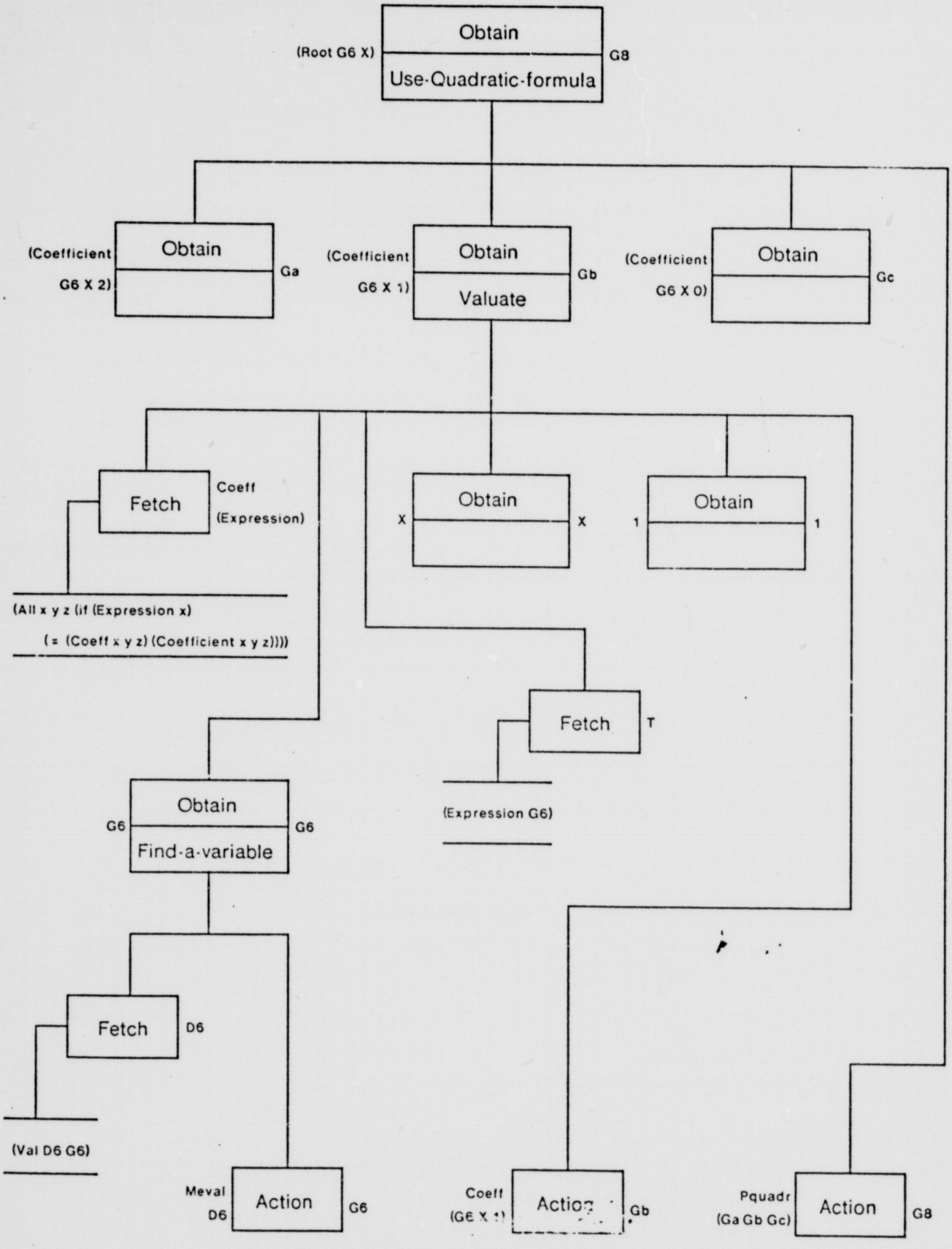

Figure 6 - Erroneous plan to compute the root of G6

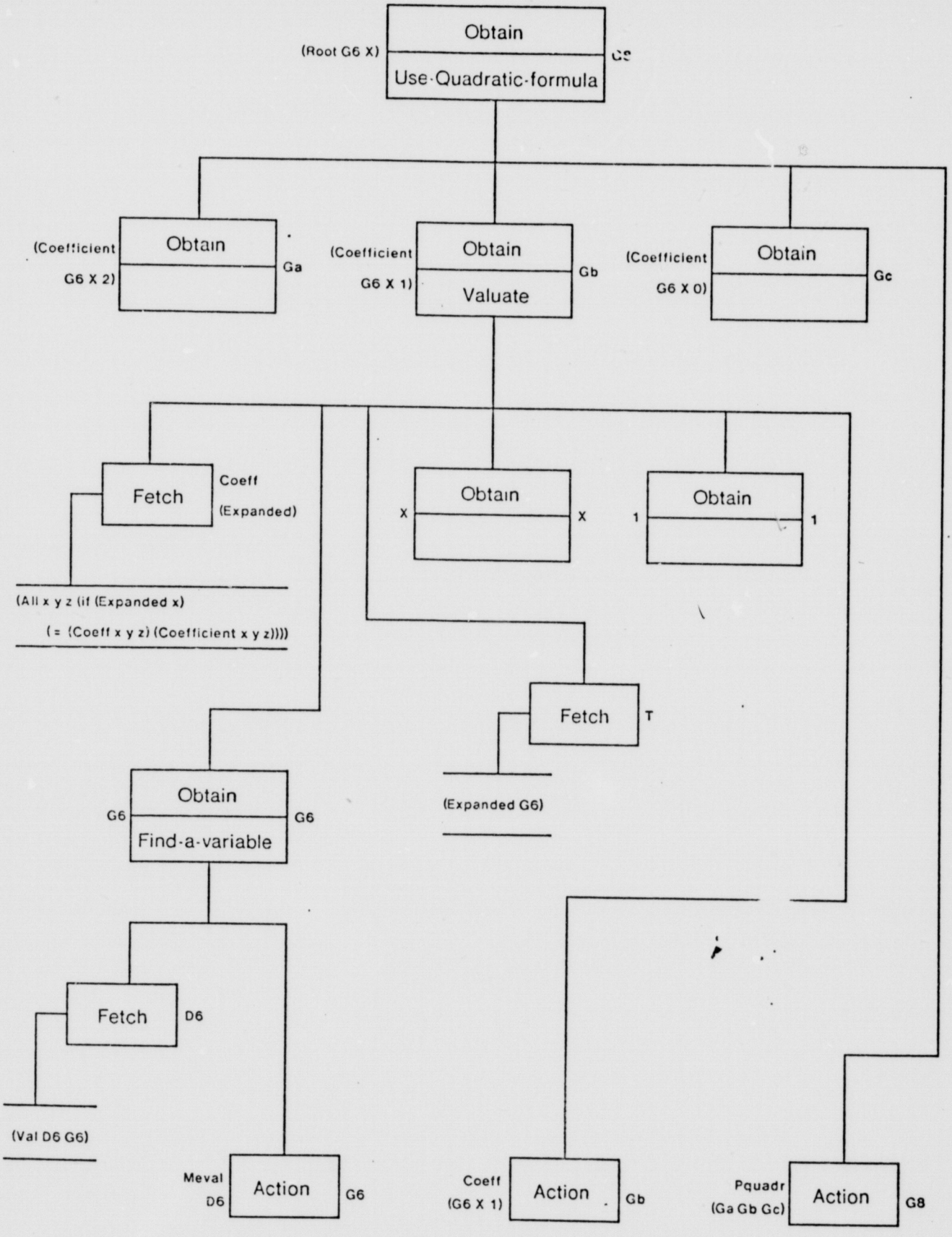

Figure 7 - Another erroneous plan to compute the root of G6

Figure 6 shows a plan for the user's effort in figure 1c. The problem in this case is that the user believes coeff's argument need only be a mathematical expression in order for coeff to compute its coefficient. This erroneous belief is reflected in the predicate expression being returned by fetch, which g6 trivially satisfies; and, consequently, the transform subgoal is not created (see the definition of valuate in figure 3). Note that the structure of the plan here is correct, and so the action sequence "makes sense"; it's the premise that's wrong. If the premise were correct, the plan would constitute a flawless proof that g8 is the root of g6.

Figure 7 shows another plan to account for the session shown in figure 1c. In this case, the assertion about coeff correctly lists expanded as a precondition, and g6 is obtained as before. However, the succeeding fetch call indicates that the user believes g6 to be expanded; and, therefore, the transform subgoal is not created. Because g6 is visible to the user, this mistake is less likely than the one in figure 6; but, when a user nests function calls, the intermediate expressions are not visible and such mistakes often arise.

3. Plan Recognition

Plan recognition is the inverse of planning. A planner uses facts about its problem area and the available actions to produce a plan that achieves its goal. A plan recognizer starts with a sequence of actions, reconstructs the problem solver's plan, and thereby infers the underlying beliefs about the problem area and the actions involved.

As discussed in the last section, a plan is essentially a proof that a sequence of actions achieves its goal (in terms of the problem solver's beliefs). So, in a sense, plan recognition is similar to program verification. One difference is that in the case of plan recognition the problem solving methods that produced the sequence are known and are treated as rules of inference. More importantly, program verification is concerned with proving the behavior of a program for a class of possible inputs (e.g. showing that a program solves all quadratics). By contrast, most problem solving plans are designed with particular inputs in mind (e.g. a sequence of steps that solves a particular quadratic).

3.1 Approaches to Plan Recognition

Plan recognition can also be viewed as a parsing problem in which the student's actions are the "words" and his problem solving methods are the "rules of grammar". This is an overly simplistic view, since the dataflow amongst the commands must also be taken into account. However, the basic strategies of top-down and bottom-up parsing are relevant.

In the top-down approach, the plan recognizer acts as a planner, using the problem solving methods to generate action sequences that solve the problem. Fetch requests are answered from a correct data base about the problem area or in accordance with a plausible misconception, as discussed in the next subsection. The process halts when a sequence of actions is found that matches the student's.

In the bottom-up approach, the plan recognizer uses the problem solving methods as data in guessing possible goals for each of the user's actions. The process then repeats using these goals as data. Since an action may suggest more than one method and goal, a search space of possible parsings is generated; and the plan recognizer explores this space until it finds a parsing that connects the student's actions to his overall goal.

In comparing the top-down and bottom-up approaches, the key issue is how drastically the search space branches. The top-down approach works well in problem areas where the number of possible solutions and the cost of obtaining each solution are not large. In many other areas, however, it is more difficult to solve a problem than to recognize a solution once presented. The bottom-up approach is ideal for such areas. In MACSYMA, for example, there are often many ways to solve a problem, whereas each command has only a small number of mathematical uses. What's more, generating each solution often entails considerable search. Consequently, it is usually easier to recognize plans in bottom-up fashion than to generate solutions. In general programming

languages, like LISP, it is usually possible to recognize the programming intent of each operation, but it is much more difficult to guess the intent in the programmer's application area. For example, in a LISP program it's easy to see that car is used to get the first element of a list or that a prog implements a search loop. It's much more difficult to guess that getting the first element of a list is equivalent to getting the real part of a complex number or the numerator of a rational number or that a search loop is a prime number program. The number of possible interpretations is simply too large.

3.2 Hypothesizing Misconceptions

Both of these approaches are complicated by the likelihood of misconceptions on the part of the student. Not only must the plan recognizer fill in the problem solving methods connecting the student's action to his goal, but it must also guess the student's beliefs and misconceptions.

One solution is for the recognizer to assume the student's beliefs are correct expect for one or more hypothesized misconceptions. The program can then try to reconstruct the student's plan for each possible misconception or combination of misconceptions until it succeeds in finding a plan for the student's actions. The work of Brown and Burton on West tutor [Burton & Brown] and the Buggy system [Brown and Burton] exemplifies this approach. Although their programs do not use the student's intermediate work and do not reconstruct the student's plan, they do infer misconceptions from the student's final answers, largely by a "generate and test" approach. Given correct models of arithmetic expertise, both systems try to explain the student's error by proposing possible faults and checking for each whether it predicts the student's answers. Buggy's analysis is particularly sophisticated in that it can use a bank of test results, both correct and incorrect, to discriminate amongst possible faults.

A bottom-up version of this approach can be implemented by expanding the set of problem solving methods to include buggy methods as well as correct ones, with one buggy method per misconception. Such buggy methods could be constructed by copying the appropriate correct methods and modifying their fetch calls so that they return values that reflect misconceptions. (In compiler terminology , this is called "constant folding".) In recognizing such buggy methods, the plan recognizer would also be inferring the corresponding misconception. The disadvantage of the approach is the large number of buggy methods that must be constructed in advance.

In point of fact, it is possible to build a program that recognizes plans with no hints whatsoever about the student's beliefs and misconceptions, so long as he uses a known set of problem solving methods. The MACSYMA Advisor, for example, is able to infer beliefs and misconceptions directly from the user's goal and actions and its corpus of problem solving methods. The Advisor also uses a set of buggy methods as described above but only for efficiency in recognizing common misconceptions.

The Advisor's plan recognition procedure is a hybrid top-down, bottom-up method. Problem solving methods are suggested in bottom-up fashion on the basis of the user's actions. The identity, inputs, and outputs of each action place constraints on the variables in the suggested method. To utilize these constraints, the method is symbolically evaluated in the forward and backward directions to propagate the information to the inputs and outputs of the method and to generate subgoal expectations. A variety of heuristics are used for associating expected goals with recognized methods. The inference of the student's beliefs (and misconceptions) comes when the propagated constraints on a method's variables reach any calls to fetch and thereby constrain the contents of the student's data base.

As an example of its operation, consider how the Advisor would recognize the plans in figures 6 and 7. When the plan recognition process begins, it has as data the relevant sequence of MACSYMA commands (the **action** boxes) and a statement of the overall goal. The Advisor's first step is to determine which problem solving methods could have dictated the student's observed actions. At this point every method that contains a call to **action** is a potential caller of every **action** box. Information about the inputs and outputs of each action are propagated by symbolic execution to the inputs and outputs of the suggested methods. The Advisor then tries to pair up expected subgoals with the problem solving methods that have been suggested, on the basis of the methods'

goal types, inputs, and outputs. For example, the overall goal **(obtain (root g6 x))** pairs up with the suggested method **(solve-by-quadratic (root g6 x))**. Similarly, the expected subgoal **(obtain g6)** is paired with the suggested method **(find-a-variable g6)**, and the goal **(obtain (coefficient g6 x 0))** is paired with **(valuate (? g6 x 0))**, where the ? signifies that the function being computed is unknown prior to the pairing. Since these pairings account for all of the student's actions and any implementation of a **transform** subgoal entails additional actions, it can be concluded that no **transform** subgoal was created, i.e. the student believed that g6 satisfied the precondition **p** and so the **fetch** condition in line 5 of **valuate** succeeded. At this point the plan is complete except for the binding of the variable **p**. In order to guess the identity of **p**, the Advisor could just generate all possible predicates (like **expanded**, **univariate**, **expression**, etc.) and any one would make a legal plan since there isn't enough information in the user's actions to discriminate amongst the possibilities. However, since this is potentially a very large set, the program starts with 2 special cases: the correct precondition (i.e. **expanded**) and the most general possibility (i.e. **expression**). These two possibilities lead to the plans shown in figures 6 and 7. Only if these fail to be confirmed (see next section) would the Advisor enumerate the other possibilities.

4. Plan Confirmation and Misconception Detection

The goal of plan recognition, as described in the last section, is to generate plausible plans that explain how a given action sequence achieves its goal. In the event of ambiguities, remedial action cannot be undertaken until one or another of the possibilities is confirmed.

One obvious way of pruning the space of possibilities is to use knowledge about the individual student's beliefs and misconceptions. If it is known that the student understands particular facts, then there is little chance that his plan is based on contradictory facts, and any plan that mentions such misconceptions can be eliminated. Knowledge of this sort might be gleaned from his performance on other problems or from previous interactions with the tutor.

Where the available information on the student is insufficient to confirm a single plan, an alternative is to acquire the necessary information by interrogating the student about his plan. The Teiresias system [Davis] illustrates how a derivation tree (= possible plan) can be presented to a person piecemeal, allowing him to confirm each step or, where a discrepancy is noted, to suggest an alternative. If the student agrees to all steps, then the plan must be a copy of his own. If not, then alternative plans that differ at the disputed step can be presented until he either agrees or no alternatives remain. One difficulty with this approach is that it requires that the student already be familiar with the system's formal vocabulary of problem solving methods. A second difficulty is that it can be very tedious when the size of the possible plans or the number of alternatives is very large. The possibility of having the student explain his own plan awaits further progress in natural language understanding.

An alternative that eliminates the need for the student to be acquainted with the system's problem solving vocabulary is to restrict the questioning to his beliefs about the problem area, i.e. to ask about the premises attached to each plan only. Of course, even if he agrees to all of the premises, this doesn't guarantee that he agrees to the plan. However, assuming there is some misconception amongst the assertions, it will be discovered by these methods, and appropriate remedial action can be taken.

The advantage of plans as defined here is that the underlying assumptions are explicit and, therefore, it is easy to detect misconceptions. One simply compares each of the assumptions with the correct facts and notes the discrepancies. For example, in figure 6, all of the premises are correct except for the one about **coeff**. In figure 7, all are correct except for the one stating that g6 is expanded. In larger plans than these, a misconception sometimes leads to false conclusions, which in turn are taken as premises later in the plan and lead to other false conclusion, etc. The result is that incorrect plans often contain more than one erroneous premise. However, in such cases, the underlying misconceptions are usually distinguishable because they have no dependency links connecting them to previous assertions.

Once the underlying misconception has been determined for each plan, there still remains the task of choosing a plan to ask about first. Two criteria are helpful in this regard.

The first criterion is parsimony, the assumption that the student has only one underlying misconception (or as few as possible). When choosing amongst possible plans, the MACSYMA Advisor, for example, asks about those plans with the fewest misconceptions and moves on to buggier plans only if it fails to get a confirmation. As an example, consider the MACSYMA s..ion in figure 1c. One explanation of the student's behavior, in addition to the plans shown in figures 6 and 7, is that the student believes that, in order for **coeff** to work, the first argument must be, say, univariate (i.e. it must contain only one variable), and he believes that g6 satisfies this condition. Of course, both beliefs are incorrect, and neither has dependency links to other assertions in the plan, i.e. there are two independent misconceptions. Consequently, the Advisor would consider this possibility only after it had eliminated all of the plans with only single misconceptions.

A second criterion is that the misconception be plausible in light of the student's learning characteristics and experience. For example, in MACSYMA people often choose commands on the basis of their names; thus it is more likely that a novice would believe that **coeff** computes an expression's coefficient than **ratsimp**. Or, if a student had always observed that **coeff** computes the coefficient (because it had only ever been called on expanded expressions), then the misconception of figure 6 is likely. The body of work on learning models has been steadily growing and may some day be of use in evaluating the relative likelihood of misconceptions (see [Burton & Brown], [vanLehn and Brown], and [Matz]). However, at the moment, most of that work is inapplicable because it depends on knowledge of the student's experience, and this is generally unavailable to programs like the Advisor.

5. Advice Generation

Once a plan has been confirmed and the misconception identified, remedial action can be undertaken. The misconception can be corrected, and the tutor can take the opportunity to offer the student an alternative. Since the information is provided in the context of a specific problem of interest to the student, the chances are good that it will be understood and remembered.

The best way to handle a detected misconception is pedagogical issue of some debate. One can simply tell the student what's wrong or help him discover it for himself socratic dialogue. The Advisor uses the former approach. For example in figure 3, it tells the student how the **coeff** command really works. In general, it supplies the correct information about a command whenever it discovers the student has a misconception about it.

Unfortunately, in the MACSYMA world, as in other problem areas, relaying the correct information about the wrong operation doesn't always lead the student to the right one. By providing this additional information, the tutor can help the student solve the problem and at the same time further its pedagogical aims. Of course, problems often have multiple solutions, and it isn't obvious which alternative to suggest.

One possibility is for the tutor to present the solution that is best or simplest according to an expert. For example, in the MACSYMA problem of figure 1c, The Advisor might have suggested that the student use the **factor** command to compute the roots of the quadratic in a single step. Alternatively, it could have taken advantage of the situation to present material the student hasn't yet seen. Instead of telling him about **ratcoeff** or **factor**, it could have introduced MACSYMA's **part** commands, which select parts of expressions, and could have shown him how to put them together in a program that solves quadratics.

Both of these criteria are independent of the student's approach to solving the problem. While both have advantages, a good tutor often prefers to tailor his advice to the student's problem solving strategy instead of presenting an entirely different approach. Another major advantage of plan recognition is that it provide the tutor with a record of the student's strategy to use in formulating remedial action. In response to a misconception, the MACSYMA Adv..or, for example, searches its own (correct) data base with the same **fetch** pattern that the student used in retrieving the erroneous belief; and, if the search succeeds, it conveys the answer to the student. In the consultation example of figure 3, this resulted in the offer of **ratcoeff** as a substitute for **coeff**. With this information, the student was still able to use the quadratic formula approach to solving the problem.

6. *Conclusions*

The MACSYMA Advisor was implemented as a test of the plan recognition approach to automated consultation. The methods to include in MUSER were determined after the analysis of several dozen sessions of novice MACSYMA use, and they account for all the non-exploratory behavior in the data in the sense that the humans' solutions can be generated from the MUSER methods. Working with MUSER as its student model, the plan recognition program was able to reconstruct the student's plan with very few questions. The results must be viewed with caution, however, since only three different problems were represented in the data.

6.1 Reactive Environments

The Advisor's chief goal is to pinpoint and correct misconceptions on the part of the student. Misconceptions are often promoted by environments that do not provide immediate feedback after each action. This feedback could be of two types, viz. purely local feedback on the results of one's actions and more global feedback that takes into acccount the student's goals.

MACSYMA is a good example of an environment that provides local feedback. In the example of figure 1c, the user could have traced the problem to **Coeff** by displaying the value of each command rather than grouping several commands on a single line and suppressing the display. However, had the interaction been longer and the expressions larger, the feedback would probably have been less useful. If each step had to be checked by hand, what would have been the point of using MACSYMA? In this case only global feedback sensitive to the user's goal would have provided the necessary diagnosis. The same is true of Bertram's case, where each step produced reasonable looking results. In fact, his solutions (15 and 2) were simpler than the correct solutions. The point here is not to prove that local feedback is without value, but only to suggest that sensitivity to the student's plan is sometimes necessary.

{Footnote: In point of fact, Macsyma users often don't take advantage of the available feedback and, when they do, frequently are still unable to identify the problem. Also, feedback tends to encourage lazy experimentation rather than careful thought. For example, in simplifying expressions novices blindly try all the automatic simplifiers instead of discriminating amongst them to determine the most suitable.}

The Sophie-II system [Brown, Burton, and Bell] was an attempt to create a *reactive environment* of this sort, in which the system supplied feedback not only on the results of the student's measurements and replacements but also on their appropriateness to the diagnostic goal. Sophie made this evaluation by comparing the student's efforts to the actions of an expert diagnostician. The plan recognition approach is an attempt to generalize this activity to domains where the number of reasonable solutions is too large for the expert comparison technique to work effectively.

6.2 Complexity in Computer Systems

The complexity of theories in natural science very much reflects the complexity of the phenomena being modeled. For artifacts like computer systems, however, there is a prevalent belief that complexity can be eliminated by better design. If so, why not forget automated user aids and concentrate on making the systems easier to use? The problem is that there is a tension between ease of learning and ease of use once learned. Even if it were true that complexity were independent of "power", this conflict would remain.

MACSYMA is a good example. Many of its commands admit an ambiguity of interpretation on different problem areas. Some users think of **Coeff** as returning the full coefficient of its argument and are surprised to find that it doesn't. The suggestion is then made to rename **Ratcoeff**, which gets the full coefficient, to be **Coeff** and rename **Coeff** to be something innocuous like **Syncoeff**. Unfortunately there are users who are interested in obtaining roots of polynomials over polynomial domain, for which they need the syntactic version. Seeing **Coeff**, they would assume it is what they want and get the wrong answer. More importantly, the implementors of MACSYMA are loathe to perform the swap because more people would use the computationally expensive **Ratcoeff** even when the argument is already expanded. Another example is the existence in MACSYMA of several

commands with hidden prerequisites and side-effects (that rob the system of its local feedback). Any suggestions to pass extra arguments or eliminate all non-explicit side-effects have met great resistance from the experienced users. As a result new users develop misconceptions, and plan-sensitive consultants become necessary.

In general, a consultant like the Advisor is necessary whenever one is faced with a problem solving situation in a domain one does not fully understand. The lack of knowledge may be incidental, as it is when the domain or device is fairly simple but time constraints make it impossible for the user to learn all that is necessary (e.g. using a calculator or oscilloscope). Or it may be essential, as when the domain is inherently complex (as in MACSYMA or algebra or natural science).

6.3 Summary

The basic idea in this paper is that plans can be viewed as dependency graphs that explain how a set of actions achieves its goal in terms of the problem solver's beliefs about the actions and the problem area. The planning methods used in creating the action sequence can be thought of as rules of inference in these graphs, and so the graphs very much resemble computation traces. A number of plan recognition routines have been implemented that can reconstruct plans, even those based on misconceptions, directly from action sequences. The importance of plan recognition to research in intelligent teaching systems is that it provides a way to interpret a student's intermediate steps in problem solving. The resulting plans can be used to identify the student's misconceptions and to offer remediation in context. Much work needs to be done in building better planning models and making plan recognition procedures sensitive to the learning characteristics of the student; but, because of their demonstrated ability to recognize plans, programs like the Advisor suggest the feasibility of the approach.

References

Brown, J. S.; Burton, R. R.; Bell, A.: "Reactive Learning Environment for Computer Assisted Instruction", BBN Report No. 3314, Bolt Beranek and Newman Inc., Oct. 1976.

Brown, J. S.; Burton, R. R.: "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills", BBN Report No. 3669, Bolt Beranek and Newman Inc., Dec. 1977.

Brown, J. S. & vanLehn, K.: "Repair Theory -- A Generative Theory of Bugs in Procedural Skills", Xerox Palo Alto Research Centers, August 1980.

Burton, R. R.; Brown, J. S.: "A Tutoring and Student Modeling Paradigm for Gaming Environments", SIGCSE 8, 1, Feb. 1976, pp 236-246.

Carr, B.; Goldstein, I. P.: "Overlays: A Theory of Modeling for Computer Aided Instruction", Memo 406, M.I.T. Artioficial Intelligence Laboratory.

Cheatham, T. E. & Townley, J.: "Symbolic Execution of Programs -- A Look at Loop Analysis", Proc. of the Symposium on Symbolic and Algebraic Manipulation, August 1976, pp 90-96.

DeKleer, J.: "Causal and Teleological Reasoning in Circuit Recognition", Ph.D. thesis, M.I.T. Artificial Intelligence Laboratory, Sept. 1979.

Genesereth, M. R.: "Automated Consultation for Complex Computer Systems", Ph.D. thesis, Harvard University, Nov. 1978.

Genesereth, M. R.: "Plan Recognition", Memo HPP-81-10, Stanford University, 1981.

Goldberg, A.: "Computer-Assisted Instruction: The Application of Theorem-Proving to Adaptive Response Analysis", Report No. 203, Institute for Mathematical Studies in the Social Sciences, Stanford University, May 1973.

Hewitt, C.: "Towards a Programming Apprentice", Trans. on Software Engineering, SE-1, 1, IEEE March 1975, pp 26-45.

Mathlab Group: "MACSYMA Reference Manual", M.I.T. Laboratory for Computer Science, Dec. 1977.

Miller, M.; Goldstein, I. P.: "Parsing Protocols Using Problem Solving Grammars", Memo 385, M.I.T. Artificial Intelligence Laboratory, Dec. 1976.

Schmidt, C. F.; Sridharan, N. S.; Goodson, J. L.: "The Plan Recognition Problem", *Artificial Intelligence* Vol. 11, No. 1, 2, Aug. 1978, pp 45-83.

Shortliffe, E. H.: "MYCIN: A Rule-Based Computer Program for Advising Physicians Regarding Anitmicrobial Therapy Selection", STAN-CS-74-465, Stanford University, Oct. 1974.

Sleeman, D. H.: "A System Which Allows Students to Explore Algorithms", Proc. of the 5th International Joint Conference on Artificial Intelligence, Aug. 1977.

Stallman, R. M. & Sussman, G. J.: "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", *Artificial Intelligence* 9, 1977, pp 135-196.

Stansfield, J. L.; Carr, B.; Goldstein, I. P.: "WUMPUS Advisor I", Memo 381, M.I.T. Artificial Intelligence Laboratory, Oct. 1976.

Waters, R. C.: "A Method for Automatically Analyzing the Logical Structure of Programs", Ph.D. thesis, M.I.T. Artificial Intelligence Laboratory, Aug. 1978.