

The Role of Plans in Automated Consultation

Michael R. Genesereth
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Mass. 02139

Abstract: Consultation is a method widely used in computer centers for helping people to use unfamiliar computer systems or languages. Unfortunately, consultants are scarce, expensive, and often unavailable when needed. This paper describes the implementation of an automated consultant for MACSYMA, called the Advisor.

The Advisor's implementation is based on the assumption that the MACSYMA novice behaves in accordance with a standard heuristic problem solving algorithm, called MUSER. This assumption is supported by a body of data on the problem solving behavior of MACSYMA users. In solving his problem, the user implicitly generates a goal-subgoal graph, called a "plan". The plan is a direct proof that the commands used actually achieve the goal (in terms of the user's beliefs), and the problem solving algorithm constitutes a grammar for such plans.

The key to the consultation process is the analysis of the user's plan. First, the plan is reconstructed by a combined process of dialogue and automatic plan recognition, and then the underlying beliefs are checked for errors. Because of the MUSER model, the Advisor is able to diagnose not only standard errors but also more general misconceptions.

1. Introduction

Consider a person trying to solve a problem with a computer system he does not fully understand, and assume that he has encountered a difficulty due to his lack of knowledge of that system. For example, he might be unaware of the capabilities available or not know the names of its commands, or he might get an unexpected result he can't explain. The simplest way for him to acquire just the information he needs is to ask a consultant for help. Then, armed with the consultant's advice, he may surmount the difficulty and solve the problem. Consultation is a method widely used in computer centers as well as in domains like business, law, and medicine; and, as computer technology becomes more pervasive and computer systems more complex, the need for consultation will grow. Unfortunately, human consultants are scarce, expensive, and often unavailable when needed. This paper describes the implementation of an automated consultant to fill this need.

The type of difficulty described above can be called a *resource usage difficulty* to distinguish it from a difficulty inhering in the problem or a difficulty in learning about the resource without a specific problem in mind. A resource usage difficulty is one which is encountered in trying to solve a problem using a resource and due to the problem

solver's lack of knowledge of that resource. The consultant's task is to remedy the deficiency in the user's knowledge underlying a resource usage difficulty.

The hardest resource usage difficulty to handle is a violated expectation which the user can't explain. In other cases, the deficiency in the user's knowledge is implicit in his question, and the answers can simply be looked up in a data base (e.g. "how do I invert a matrix?" or "what are the arguments to MAP?"). In the case of an unexplainable violated expectation, even the user is unaware of the nature of his "misconception". The consultant's first step, therefore, is to identify this misconception, after which he can generate his advice.

The method described here for the identification of the user's misconception is based on the assumption that the user's actions are "rational", i.e. that he has some "plan" for achieving his goal. This plan explains why he chose the commands he did in terms of his beliefs about the input-output specifications of those commands. The key to the identification of the user's misconception is the reconstruction and debugging of this plan.

The plan reconstruction procedure is based on the assumption that the user of a complex resource behaves (up to ordering of strategies) in accordance with a stand-

ard heuristic problem solving procedure, called MUSER. MUSER is a largely domain independent problem solver with the ability to invoke domain dependent methods when necessary. In solving a problem the user implicitly generates a graph of goals and subgoals, in accordance with this algorithm. MUSER is guaranteed to produce only correct solutions (given a correct data base), and so this goal-subgoal graph constitutes a direct proof that the user's commands achieve his goal (in terms of his beliefs). In the consultant described here, the notion of the user's plan (as proof) is identified with the goal-subgoal graph of his solution.

With this view, plan reconstruction is a complex "parsing" problem, in which the user's actions are the "sentences" and in which the MUSER algorithm is the "grammar". The plan reconstruction process consists of heuristic suggestion of partial parsings together with forward and backward symbolic evaluation of the problem solving procedure to filter these suggestions. Some of the partial parsings are suggested by the patterns in a library of frequently recurring errors. Others are suggested by correct facts about MACSYMA and the structure of the MUSER model. Although the recognition of standard errors plays a large part in effective consultation, it is possible to reconstruct plans containing misconceptions using just the MUSER model and correct facts about MACSYMA.

The method of consultation is exemplified by an automated consultant for MACSYMA novices, called the Advisor. It is a program distinct from MACSYMA, with its own separate data base and expertise. The Advisor accepts a description of a resource usage difficulty from its user and generates advice tailored to his need. Eventually, the Advisor should be able to converse in English; however, at present, all communication is conducted in the Advisor's LISP-like data base language.

This does not mean that the consultation technology described here is in any way restricted to MACSYMA. Consultation is useful in any situation where the user of a resource must do some problem solving with partial knowledge of the resource. The Advisor could equally well be applied to any complex, interactive computer system, such as a text editor, a document production program, or a data base query system.

2. An Example of a Resource Usage Difficulty

As a concrete example of a resource usage difficulty, con-

sider the MACSYMA interaction and consultation session shown in figure 1. 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 ";" or a "\$"; if the latter is used, the display line is suppressed (e.g. C7). Expressions are entered in linear form (e.g. $(3-2*Z)/Z$); ":" is the assignment operator; and other commands are applied using functional notation (e.g. `RATSIMP(D4)`). Parentheses delimit compound statements.

```
(C6) RATSIMP(D4);
(D6)      (-3X - 3)Z + X2 + 3X
(C7) (A:COEFF(D6,X,2),B:COEFF(D6,X,1),
      C:COEFF(D6,X,0))$
(C8) (-B+SQRT(B2-4*A*C))/(2*A);
(D8)      0
(C9) HELP($)
```

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

Advisor: Did you expect *COEFF* to return the coefficient of D6?

User: Yes, doesn't it?

Advisor: *COEFF(EXP,VAR,POW)* returns the correct coefficient of VAR^{POW} in *EXP* only if *EXP* is expanded with respect to *VAR*. Either expand first and then use *COEFF* or use *RATCOEF*.

User: Ok, thanks. Bye.

Fig. 1 - An example of MACSYMA consultation

In this example, the user's difficulty was due to his "misconception" that *COEFF* always returns the coefficient of its argument. This misconception gave rise to the erroneous use of *COEFF* in line C7 without a preliminary expansion of D6. However, the user didn't observe this "bug" until the zero appeared in line D8. Being unable to explain the violation of his expectations, the user then appealed to the on-line consultant for help.

One common suggestion for eliminating such difficulties is to switch the names *COEFF* and *RATCOEFF*. One reason this hasn't been done is that *COEFF* is much more efficient than *RATCOEFF*; by giving *COEFF* the more common name, the hope is that users will try the more efficient command first. Even if they were switched, not all problems would be eliminated. For example, a user who wants to solve a polynomial over a polynomial domain would need a version of *COEFF* with the present

semantics; seeing a command called COEFF he might believe that it does what he wants and would get the wrong answer. So long as there are commands with different interpretations depending on the type of problem the user is trying to solve, there will be violated expectations and a need for consultants.

3. The MUSER Model and Plans

In order to discover the misconception underlying the user's violated expectation, it is first necessary to know why he expected it. The user must have had in his mind some argument explaining how his particular sequence of commands was structured to achieve his expectation. For single commands, an adequate explanation is simply a belief about the command involved. For a sequence of several commands, the explanation requires an enumeration of the intrinsic properties of the individual commands together with an argument showing how they co-operate to achieve the expectation. A *plan* is a formalization of this notion of co-operation, essentially a formal proof that the command sequence results in the expectation.

This view of plans raises two questions. How does one characterize plans, i.e. is there any syntax for or constraint on plan structure? Second, how does the Advisor acquire knowledge of the user's plan and how is it debugged? The answer to the first question is given as a formal model for the problem solving behavior of the MACSYMA novice. The Advisor's techniques for plan reconstruction and debugging are described in the sections thereafter.

3.1 The MUSER Model

In solving problems MACSYMA novices do not randomly explore all possible combinations of commands. In fact, the analysis of more than a hundred problem solving sessions supports the conjecture that novices act in accordance with a standard heuristic problem solving procedure called MUSER. The differences between users stem from differences in their knowledge of MACSYMA and their ordering of strategies. The strategies themselves seem to be common.

The MACSYMA user typically has a mathematical problem he is trying to solve and approaches MACSYMA for its powerful abilities at algebraic manipulation. Thus, in any MACSYMA problem, there are two distinct domains involved, viz. mathematics and MACSYMA. In the MUSER model, knowledge about these domains is grouped into three categories (see figure 2). The "Task Environment" is a body of mathematical facts and procedures; the "Static Model of MACSYMA" contains the input-output specifications of MACSYMA's commands; and the "State Description" is a data base of facts about the current MACSYMA (such as variable bindings and function definitions). The MUSER problem solver is a domain independent procedure which accepts a problem statement in a domain independent formalism similar to predicate calculus and produces a sequence of MACSYMA commands that solves it. In doing so, it utilizes the facts stored in the data bases and implicitly generates a goal-subgoal graph, or "Plan".

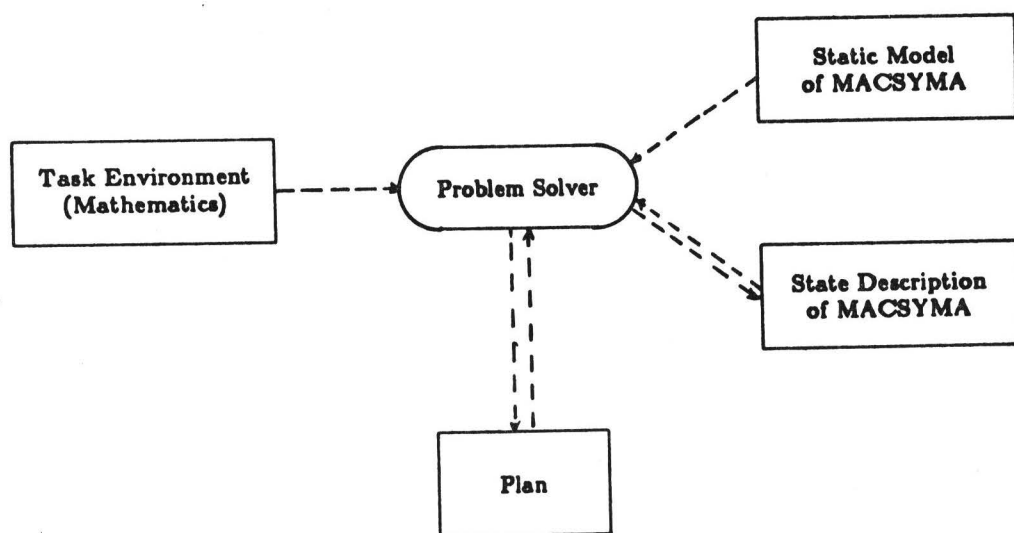


Fig. 2 - MUSER's data structures

The MUSER problem solver is guaranteed to produce only correct solutions (given a correct data base), and so this goal-subgoal graph constitutes a direct proof that the commands achieve the goal (in terms of the facts in the data bases). Any errors in a solution must derive from incorrect entries in one of the data bases. A resource usage difficulty often arises due to an error in the user's static model of MACSYMA, and the purpose of consultation is to correct such errors.

The MUSER problem solver is represented as a "parameterized procedural net", or PPN. A PPN is a hierarchical network of actions at varying levels of detail in which each action node has associated with it sets of input and output objects and sets of "prerequisites" and "postrequisites" (conditions which must be true for the action to succeed and those which become true after its execution). In the procedural nets described by Sacerdoti [Sacerdoti], each action node is described in terms of specific objects in the world being modeled. In a parameterized procedural net, the flow of data into and out of an action is described in terms of the inputs and outputs of other actions without naming any specific real world objects. Thus, a PPN is a kind of partial program. Except for its hierarchical character and the explicit representation of prerequisites and postrequisites, this formalism is similar to the dataflow graphs described by Dennis [Dennis].

Figure 3 shows a very small fragment of the MUSER PPN. The square boxes represent problem solving methods at varying levels of detail. Methods at different levels are connected by vertical "supergoal-subgoal" links. Links with crossbars denote conjunctive subgoals; those without, denote alternative subgoals. The rounded boxes represent data base assertions. In order to simplify the presentation and emphasize their hierarchical character, PPNs are drawn here as trees without control or dataflow links and with various problem solving methods duplicated (e.g. "Obtain an object"). In looking at such abbreviated diagrams, the reader should bear in mind that the action nodes are actually interconnected through their inputs and outputs and through their prerequisites and postrequisites.

Each problem solving method in MUSER satisfies a particular type of goal by a particular method or a choice of methods. For example, the goal of "Obtain an object" is to "obtain" the object specified as its argument (either to print it out or pass it as argument to a function). In

MACSYMA one can do this either by finding an already computed expression (the "Find a variable" method) or by constructing one anew (the "Construct an object" method).

In the "Find a variable" method, MUSER searches its data base to find a variable with the goal as value and then evaluates it. In the terms used in figure 3, it looks a variable v such that $Val(v) = g$, where g is the specified goal.

The "Fetch" method is MUSER's data base access function. It searches the data base for a fact of the form specified in the assertion attached to it. (This includes more than exact matches. "Fetch" employs a fast but limited inference algorithm [Genesereth 1977] to do property inheritances and set intersections as well.) In this case, the goal g would be supplied as an input to "Fetch" and the variable v would be returned as output.

The "Action" method is MUSER's way of executing a command in MACSYMA. Given a command and a set of objects as inputs, "Action" will cause MACSYMA to call the command with the objects as arguments and will return an internal representation of the answer as its output. In the case of "Find a variable", the command is MACSYMA's evaluator MEVAL.

The alternative to obtaining a pre-computed object is to construct a new one. The goal of the "Construct an object" method is to produce an object defined as $f(a)$. The strategy is to search the data base for a command c which computes f , given preconditions p on the argument and general prerequisite q , as shown in figure 3. The outputs of "Fetch" are the command c , the preconditions p , and the prerequisite q . Having found such a command, the method then obtains the argument a in a form that satisfies p , achieves q , and executes the command. The extension to multiple argument functions is straightforward.

The arguments of a function are obtained by the "Obtain arguments" method. If a particular argument must satisfy a precondition, MUSER uses the "Obtain and Convert" method. If not, it simply calls "Obtain" recursively.

The strategy of "Obtain and Convert" is to obtain the argument and then transform it to satisfy the precondition. In doing so, it searches for a suitable transformation, achieves its prerequisites, and applies the transformation, as shown.

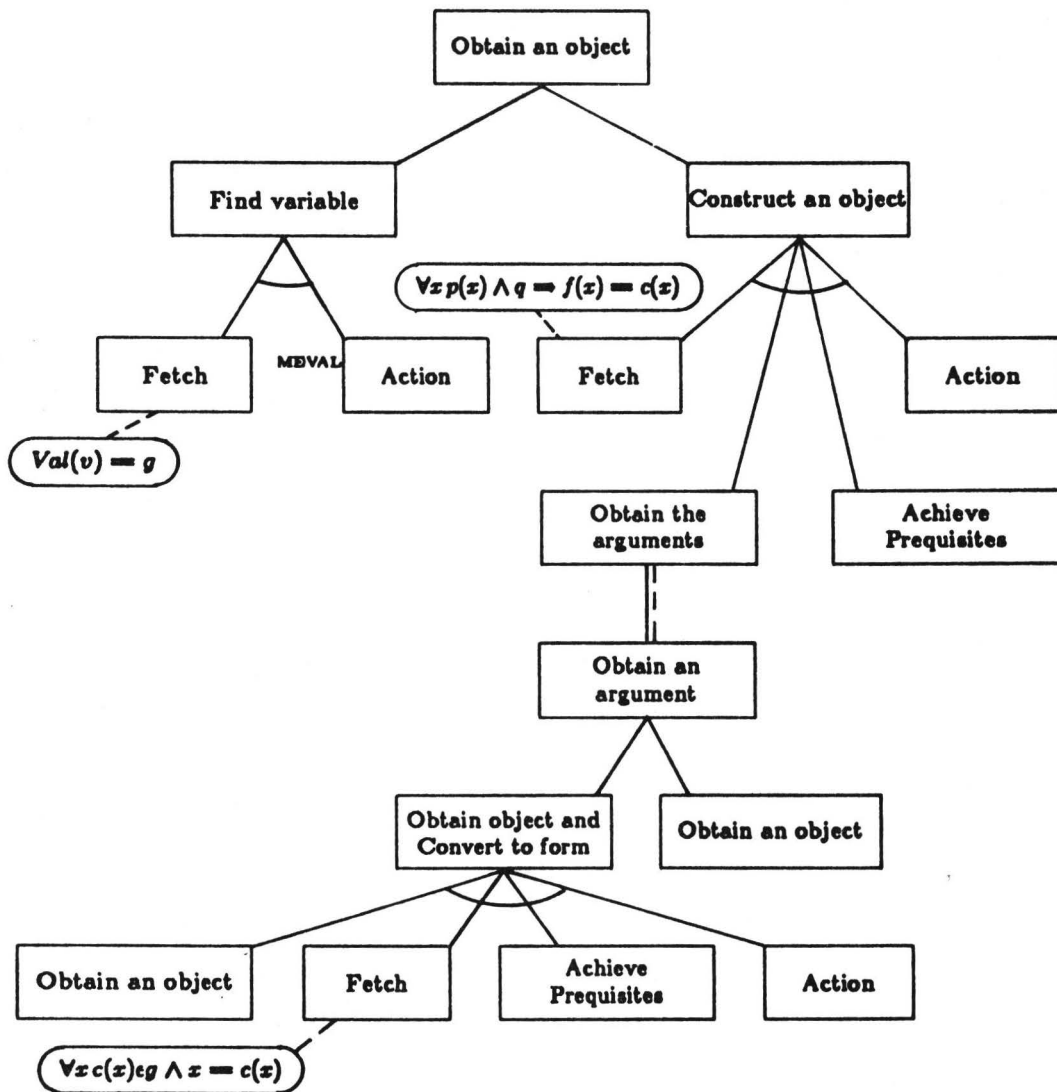


Fig. 3 - A fragment of the MUSER Model

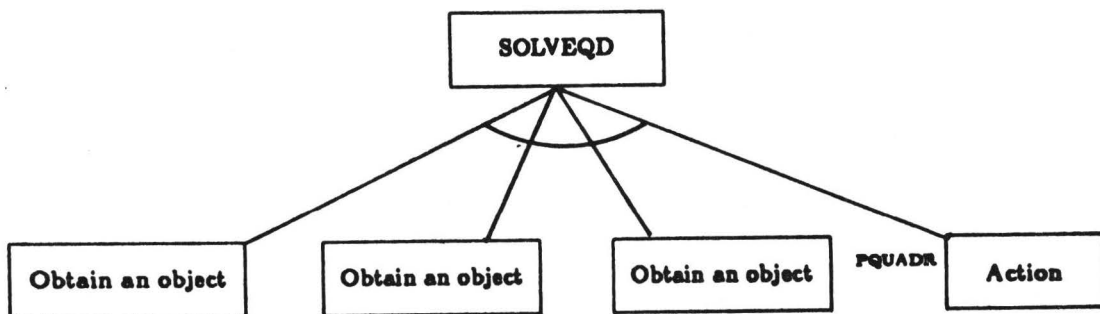


Fig. 4 - The domain dependent method SOLVEQD

The full MUSER model is considerably larger than the small fragment shown here (see [Gencsereh 1978]). In addition, MUSER has the ability to invoke domain dependent procedures when appropriate. As MUSER generates each subgoal in solving a problem, it searches its data base for any domain dependent methods with a matching goal. If such a procedure is found, it is used just as if it were a method in the MUSER model. As an example, consider the procedure SOLVEQD shown in figure 4. SOLVEQD computes the coefficients of the quadratic and then plugs the results into the quadratic formula. If MUSER needed to obtain the root of a quadratic, it would find SOLVEQD and invoke it with the corresponding "Obtain an object" method as supergoal.

3.2 Plans

A plan is the result of executing a parameterized procedural net with particular bindings for the parameters. More formally, a plan is a PPN in which all network objects are bound to real world objects, all parent-child cycles have been eliminated by copying, and all disjunctions have been resolved. Additionally, a plan includes pointers to all data base assertions accessed in constructing it.

Figure 5 shows a plan for computing the coefficient C1G6 of X in an expression G6. The data base documents that the COEFF command computes the coefficient of an expression so long as it is expanded. In trying to obtain the argument in expanded form, MUSER 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. Note that this plan is completely correct, unlike the user's plan in figure 1.

4. Overview of the Consultant

In handling a violated expectation, the Advisor's first step is to reconstruct the user's plan. When the Advisor is called, it has a data the relevant sequence of MACSYMA commands; and, if the user has not already supplied it, the Advisor obtains from him a statement of his overall goal. The commands are assumed to represent the fringe of a plan in which the root is the user's goal. The purpose of plan reconstruction is to fill in the intervening structure.

The Advisor's plan reconstruction procedure has three parts. The first step is to convert the user's actions into a

single-level dataflow graph. The second step is mechanical plan recognition. In principle, the plan could be obtained by asking the user to describe in detail how he intended to achieve his goal. However, human consultants are able to bypass most and sometimes all such questioning, and it would be preferable if the Advisor could do as well. Furthermore, in practice the user often won't know what terms to use in describing his intentions, or the description might be too tedious for his patience to bear. Therefore, to be effective, a consultant must reconstruct as much of the user's plan as possible with as little questioning as possible. The third phase of plan acquisition is user interrogation. When the Advisor's plan recognition procedure fails to produce a complete plan, the Advisor tells the user what it has figured out and asks for help.

The plan recognition procedure is a heuristic "parsing" procedure that searches the partially reconstructed plan for plan fragments (from its "Plan Library") or error fragments (from its "Error Library"). Since the user's plan is assumed to be an instantiation of the MUSER model, the Advisor also uses the MUSER model as data in inferring further structure. All possible parsings are simultaneously explored until a single plan is found that ties the user's goal to his dataflow graph. The first such "parsing" of the graph to be found is tentatively adopted by the Advisor as the correct plan. If at some later stage the Advisor finds that its tentative plan does not correspond with the user's, the Advisor eliminates it from consideration and searches for another parsing. In some cases, the recognizer can rule out potential plans by checking the component data base assertions against its model of the user's beliefs. If the tentative plan includes a misconception the user is known not to possess, the plan is rejected and another sought. In the episode shown in figure 1, the Advisor was able to reconstruct the plan shown in figure 6. In comparing this plan to the one shown as figure 5, note the different data base assertion about the command COEFF and the resulting use of the "Obtain" method instead of "Obtain and Convert".

Once the Advisor has a version of the user's plan, it tries to identify the misconception. The first step in doing this is to acquire a suspicion of what might be wrong. Suspicion can be aroused either by recognition of some standard error in the user's plan or by a general model debugging process. Once a suspicion is aroused, the Advisor confirms that it is a misconception by asking the user whether or not he believes it. In the example, the Advisor found that

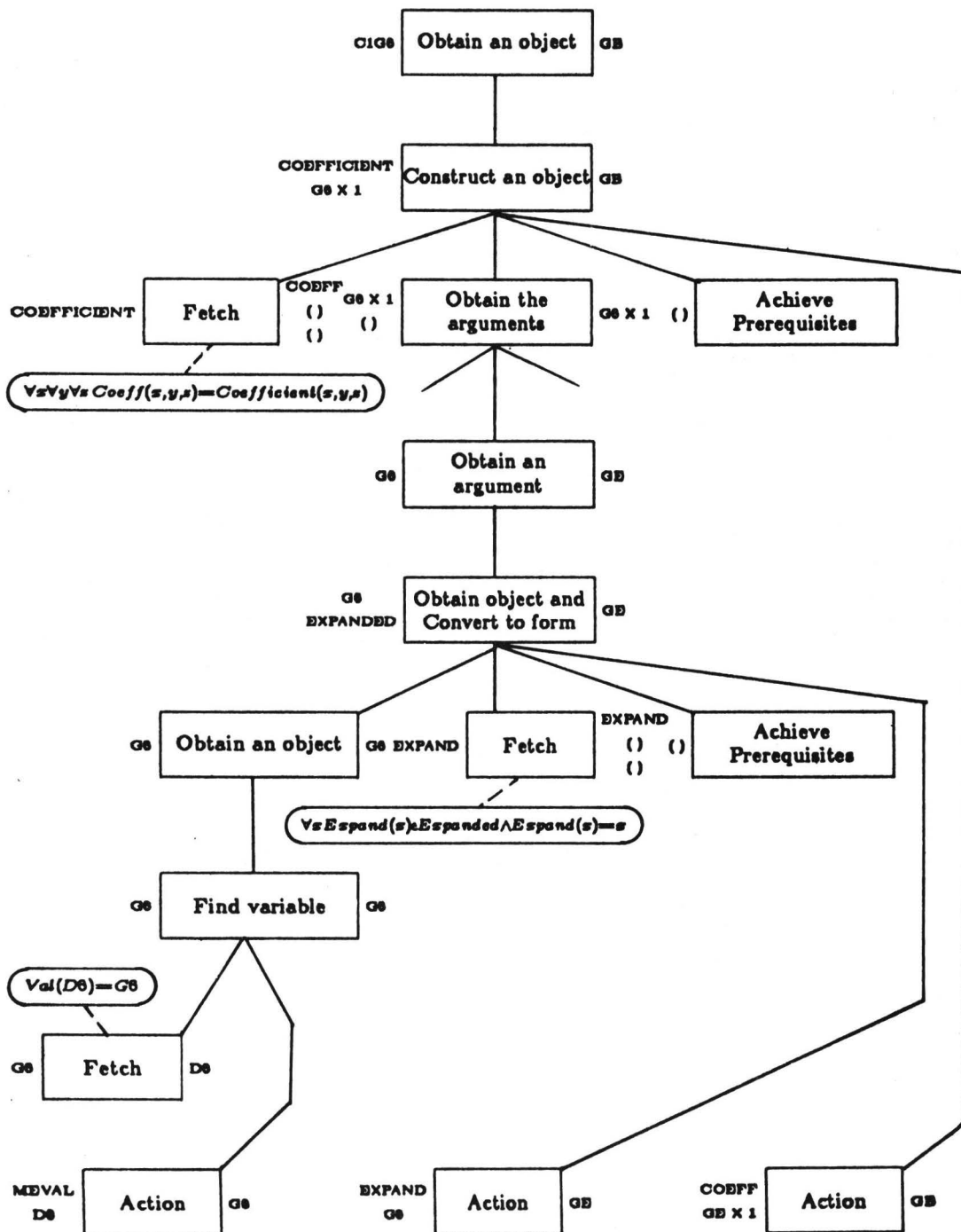


Fig. 5 - Plan to compute the coefficient of an expression

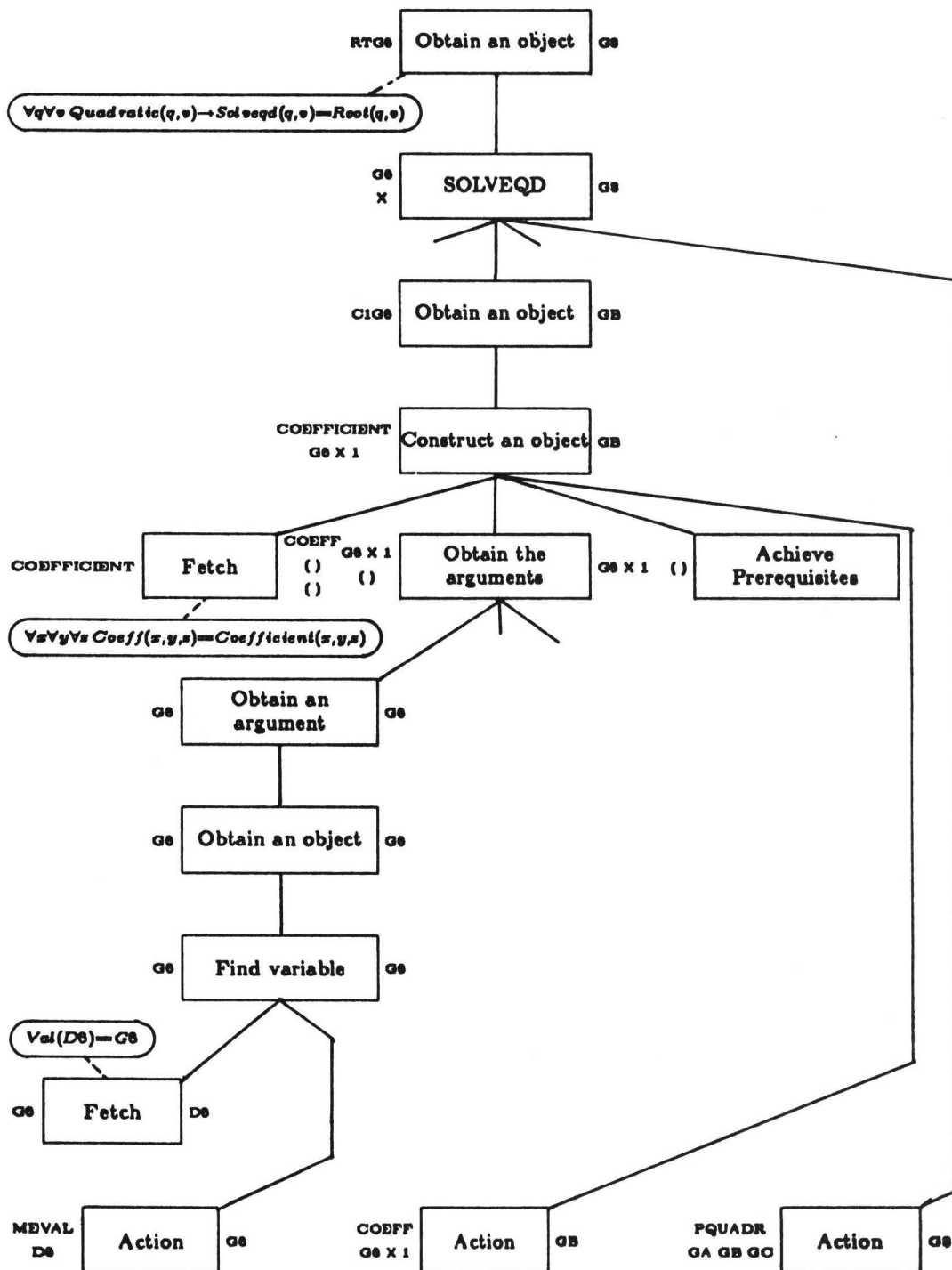


Fig. 6 - The plan for the MACSYMA session shown in Fig. 1

the assertion about COEFF in the reconstructed plan was incorrect and then asked the user whether he believed it.

Once a misconception has been identified, the Advisor corrects it and generates advice. In the current implementation all standard errors have scripts associated with them. When such an error is detected and confirmed, the script is recited to the user. These scripts are prepared by the Advisor's programmer and contain all the necessary directions. In other cases, the Advisor follows a more general procedure.

In this more general procedure, correcting the misconception is easy; the user is simply told that his misconception is wrong. If there is a corresponding correct fact about the command or variable involved, this information is then supplied. In this example, the user was first told the correct information about COEFF.

Unfortunately, simply relating the correct information about the wrong command doesn't always help the user find the right command or option. Misconception correction is intended to improve the user's knowledge about the command or variable used. Helping him to useful information to achieve the goal involved is the purpose of the "misconception alternative" step of advice generation. In order to find an alternative, the Advisor executes (on its own data base) the same "Fetch" call with which the user retrieved his misconception (as indicated in the plan). If it succeeds, the information is conveyed to the user, as in the example. If it fails, no further action is taken. In either case the Advisor considers its work done and the episode ends.

5. Summary

The Advisor is implemented in MACLISP and consists of about 50 pages of very compact code, including the data base manipulation routines and the MUSER problem solver but not counting the data base about MACSYMA. (At the moment the data base includes information about a few dozen commands only.) As mentioned earlier, there is no natural language interface. Preliminary testing of the system by hand translating consultation sessions into the Advisor's internal language shows that the major shortcomings are its lack of knowledge of the task environment (mathematics) and the weakness of the MUSER model. A full description of the implementation and the theory on which it is based can be found in [Genesereth 1978].

The important contributions of this research are (1) its emphasis on the need for consultation in any sufficiently complex domain and (2) its consultation technique. In general, a consultant 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 very complex and the user can't possibly learn everything (e.g. MACSYMA).

The implementation described here is based on the belief that a good consultant must possess not only substantial knowledge of its material but also a good model of its user's knowledge. The key point of the paper is that plan reconstruction and analysis is a good way to gain this user model. The consultant described here utilizes a formal domain-independent problem solving model as a grammar for plan reconstruction. Although the Advisor also uses an error library, the plan grammar allows it to diagnose bugs it has not previously encountered. This notion of a person's "plan" for solving a problem is central to consultation and should prove useful for man-machine interaction in general.

References

- Dennis, J.: "First Version of a Data Flow Procedure Language", Memo 61, M.I.T. Lab. for Comp. Science, 1975.
- Genesereth, M.R.: "Automated Consultation for Complex Computer Systems", Ph.D. thesis, Harvard University, Sept. 1978.
- Genesereth, M.R.: "A Fast Inference Algorithm Based on the Constrained Propagation of Marks in a Semantic Network", Mathlab memo 5b, M.I.T. Lab. for Computer Science, Dec. 1976.
- Mathlab Group: "MACSYMA Reference Manual", Lab. Computer Sci., M.I.T., Dec. 1977.
- Sacerdoti, E.D.: "The Nonlinear Nature of Plans", TN 101, AI Group, Stanford Research Institute, 1975.