May 1976

SOME EXAMPLES OF AI MECHANISMS FOR GOAL SEEKING, PLANNING, AND REASONING

by

Nils J. Nilsson

Artificial Intelligence Center
Stanford Research Institute

Technical Note 130

Invited Paper
XXI International Congress of Psychology
Symposium No. 16
"Natural and Artificial Intelligence"
Paris, France July 19 - 23, 1976

SOME EXAMPLES OF AI MECHANISMS FOR GOAL SEEKING, PLANNING, AND REASONING

by

Nils J. Nilsson

ABSTRACT

Psychological models of intelligent behavior have increasingly used concepts from computer science, especially Artificial Intelligence (AI). The use of semantic networks as memory models is a familiar example. In this paper we review some perhaps lesser known AI ideas that might also have utility in psychological modeling.

We are interested here in mechanisms for goal-seeking behavior, for planning sequences of actions, and for common-sense reasoning. We illustrate these mechanisms by describing five fictitious robot systems. These are simple enough so that the AI techniques involved can be clearly exposed. Original sources are cited for more details and for examples in which the techniques were used for solving more complex problems.

I. INTRODUCTION

A. The Use of AI Concepts in Psychology

There is a long tradition in psychology of constructing models to help explain human and animal behavior. Some of these models have been successful at predicting behavior under certain specified conditions; others have been at least of metaphorical value in helping us to understand the possible mechanisms of intelligence.

For the past twenty years or so, some of the most provocative models of aspects of intelligent behavior have been developed using ideas borrowed from computer science and especially from Artificial Intelligence (AI). Probably the most well-known example of psychological modeling using computer-science concepts involves "semantic network" models (Quillian, 1968; Anderson and Bower, 1974; Norman, et al., 1975) for the organization of long-term memory and for language comprehension. Refining these models is now almost a subspecialty of cognitive psychology, and the subject needs no further elaboration here. Equally familiar is the work of Newell and Simon, 1972, who describe problem-solving models based on "heuristic search" and "productions"--computer science concepts that they helped develop.

Other potentially useful ideas abound in AI, and some of these might not be as familiar to psychologists as are semantic networks and the work of Newell and Simon. Unfortunately, there is no written catalog of these ideas, and the AI literature is becoming somewhat formidable and inaccessible to nonspecialists. It is the goal of this paper to take a constellation of four or five of these ideas and to describe them in clear terms so that their merit (or lack of it) for psychological modeling purposes can be examined.

B. Organization of the Paper

I shall be talking about some principles and methods, developed by computer scientists and AI researchers, for goal-seeking behavior, problem solving, planning, and reasoning. To base this exposition on the original papers and to describe the automatic problem solving systems presented there would be lengthy and, I think, confusing. Instead, I shall

take the liberty of describing some fictitious systems suitably scaled down so that their principles of operation are laid bare. In doing so, I risk arguments with the authors of these ideas, for I might have oversimplified beyond their tolerance. My defense is that this paper should be considered as a primer; readers who are intrigued by any of the ideas should next consult the referenced sources.

II. FIVE ROBOTS

A. Goal Seeking: MARK

Fundamental to much behavior are mechanisms for pursuing a goal, setting up and pursuing subsidiary goals, and recognizing when goals have been met. Some proposed mechanisms are illustrated below in a robot called MARK.

MARK is a robot for finding buried treasure. He has a hierarchy of routines that elicit behavior, and he has a model of the world he inhabits. His model of the world at any instant of time is a set of statements such as: "I am at position A," "I have a shovel," "I have a map," "B is a treasure site," and so on. We shall represent these statements by expressions like: AT(A), HAVE(SHOVEL), HAVE(MAP), and TREASURESITE(B). These expressions are called "propositional" statements and use formalisms borrowed from the predicate calculus. Each such statement can be true or false. If it is true, and if MARK "knows" it is true, it is contained in his model. If MARK knows it is false, then its negation is contained in his model. For example, if MARK knows that he doesn't have a shovel, then his model will contain $\sim$HAVE(SHOVEL). (Sets of propositional statements have often been used by cognitive psychologists for short- and long-term memory models. Semantic networks can be thought of as structurally richer variants of propositional models.)

MARK's model keeps him up to date about the status of his world and his condition in it. We assume that MARK's perceptual processes directly modify his model, so that if he is looking at buried treasure, his "buried-treasure" detector adds INVIEW(TREASURE) to the model.

The model also serves as a central "bulletin board" for MARK's behavior routines. Chains of actions appropriate to MARK's circumstances

are triggered by the contents of the model, and these actions in turn change the model in ways sufficient to trigger appropriate subsequent behavior. The behavior routines are organized in a formalism called "Markov tables." (These tables are similar to Markov algorithms. See, for example, Galler and Perlis, 1970.)

Since MARK is a treasure-seeking robot, he is under the control of a top-level action routine called "findtreasure." This routine is described in Markov table form below:

findtreasure

Condition	Action
HAVE(TREASURE)	stop
INVIEW(TREASURE)	pickup(TREASURE)
TREASURESITE(x)	excavate(x)
HAVE(MAP)	read(MAP)
T	shopfor(MAP)

MARK's behavior is controlled as follows: A "behavior interpreter," with access to the Markov table and to the model, checks in turn each of the propositions listed under the "condition" column of the Markov table. The interpreter starts at the top condition and, working down the list, looks for the first one found to be contained in the model. It then "executes" the corresponding action (or sequence of actions) listed opposite this condition and, after finishing the execution, returns again to the top of the table to continue checking conditions. For example, if MARK does not already have the treasure, but it is in view, MARK will pick it up. (Unfortunately for MARK, as soon as he has a treasure, the "stop" action is executed causing his behavior to cease.)

One of the conditions in findtreasure, namely, TREASURESITE(x), contains a variable x. This condition is satisfied by the model if the model contains a proposition of the form TREASURESITE(x), for example TREASURESITE(A), where A is a definite place. In this case the variable x is "bound" to the definite place found in the model, and the bound form of the action, for example excavate(A), is executed.

Each of the actions in a Markov table may itself be a routine described by a Markov table, or it may be a primitive action that is "hard-wired" into MARK. In interpreting findtreasure, we are guaranteed eventually of finding a condition satisfied by the model, because the last condition T (for True) is by definition always satisfied.

Some of MARK's actions in the findtreasure table are themselves Markov tables. These are shown in Figure 1. In "excavate(x)," note the following points: If INVIEW(TREASURE) is contained in the model, the action is "return." This action merely ceases activity in the excavate table and returns activity to the process that called for the excavate action. If DUG(A), for any A, is satisfied in the model [after the interpreter "falls through" INVIEW(TREASURE)], then TREASURESITE(A) in the model is deleted, and control returns to the calling process.

excavate(x)

Condition	Action
INVIEW(TREASURE)	return
DUG(x)	delete: TREASURESITE(x), return
HAVE(SHOVEL)	godig(x)
T	shopfor(SHOVEL)

shopfor(x)

Condition	Action
HAVE(x)	return
AT[store(x)]	buy(x)
T	goto[store(x)]

godig(x)

Condition	Action
AT(x)	dig, assert: DUG(x), return
T	goto(x)

buy(x)

Condition	Action
HAVE(MONEY)	purchase(x), assert: HAVE(x), return
T	stop

Figure 1 MARKOV TABLES FOR MARK'S SUBSIDIARY ACTIONS

In the Markov table for shopfor(x), note the construct "store(x)." Here, "store" is a primitive function that produces for any item the store that sells it. Thus, for example, store(SHOVEL) might yield SEARS, and store(MAP) might yield BRENTANOS. (Sears and Brentanos are American stores that sell, among other things, shovels and maps, respectively.) When the condition AT[store(x)] is met in the table, we assume that store(x) has been replaced by its value. Note that in godig(x), if action "dig" is executed, the action "assert: DUG(x)" follows. This assertion adds an instance of DUG(x) to the model. The "stop" action in buy(x) causes the whole complex of behavior to cease.

Besides the five Markov tables, MARK has a number of primitive actions that we will not break down further. The action pickup(TREASURE) causes MARK to have the treasure and adds HAVE(TREASURE) to the model. The action read(MAP) causes MARK to know a treasure site by adding a specific statement of the form TREASURESITE(x) to the model. The action "dig," digs a hole wherever MARK happens to be. If "dig" uncovers the buried treasure, it adds INVIEW(TREASURE) to the model. The action goto(x) causes MARK to go to the place x, and after finishing, adds AT(x) to the model and deletes any other AT( ) from the model. The action purchase(x) causes MARK to have x.

Thus, here is a relatively simple system consisting of a propositional model, some Markov tables and an interpreter, and some primitive actions and functions. It is capable of rather complex goal-seeking behavior. The reader is encouraged to work through various action sequences by hand for some hypothetical models to gain a feeling for the capabilities of the system.

The primary advantage of MARK's goal-seeking mechanism is flexibility. It can operate in a variety of different situations, some close to the goal of treasure possession, others distant. The Markov tables evaluate the situation and specify the appropriate action. If, during operation, certain actions fail to achieve their usual effects, subsequent actions will nevertheless be based on whatever situation exists.

The system does have several liabilities. One is that while a subsidiary action is being executed, key conditions in superordinate action tables might become fortuitously satisfied, thus rendering either futile or inappropriate the continuation of the subordinate action.

Also the system is capable of getting into infinite loops. Returning to the top condition in a table after executing an action is rather blind and sometimes wasteful even if foolproof.

Several elaborations can be added to mitigate these drawbacks. A large-scale system of Markov tables, with a few added features, was constructed to control the SRI robot, SHAKEY. (See Raphael, et al., 1971.) Markov tables are known in computer science as a form of "production system." It can be shown that suitably elaborated systems of Markov tables are "universal" in the sense that they can produce any describable behavior.

Specialized production systems have previously been employed in psychological models. (See Newell and Simon, 1972, and Newell, 1973.) Systems of Markov tables bear some resemblance to stimulus-response models in psychology. (See, for example, Estes, 1959.) Markov tables are more complex because the "stimulus" can be any statement occuring in the propositional model--even special "control statements" contrived to force certain behavior sequences. Also the actions are arranged in a hierarchy that gives the full power of subroutine calls. (Recursion, an action invoking another version of itself, is also permitted.)

B. Planning: MAY

To a person accustomed to thinking through the consequences of actions, MARK's behavior may seem a bit impulsive. True, if the Markov tables are cleverly organized, each triggered action may be appropriate and without bad effects. But a more cautious and untrusting robot might want to construct for each goal a proposed chain of actions and to "simulate" this chain before actually performing any of the actions. In short he may want to "plan."

I will now describe a robot called MAY. MAY is also interested in treasure finding, but her activities are confined to planning how to obtain the treasure. The chief problem in planning is creating a sequence of actions that will achieve the goal.

In order to construct plans, MAY has, in addition to a propositional model, a model of each action she might want to perform. These models of actions, or "m-actions," are programs that do not actually perform

actions in MAY's world, but they do change MAY's propositional model in ways similar to how it would be changed if it were tracking changes actually made in the world.

Thus the m-actions are used to "simulate" actions. When a chain of m-actions is found that changes the propositional model in a desirable way [causing it to contain HAVE(TREASURE), for example], then this chain constitutes a "plan" that refers to actual actions that MAY could execute to produce a desired effect. (We assume that each m-action corresponds to an actual executable action that changes the world in a manner modeled by the m-action.)

We note that the planning process makes changes to the propositional model, so that it no longer corresponds to MAY's actual world but to some hypothetical world that MAY is considering. In order to consider several competing hypothetical worlds, MAY will construct copies of her propositional model differing only by the effects of the different actions she is considering. (We assume that MAY also keeps an intact copy of a propositional model to keep track of her world as it actually is.)

Just as real actions can be executed successfully only if certain preliminary conditions are met, MAY's m-actions also have "preconditions." One of the important problems in constructing a plan is to make sure that the preconditions of each m-action are satisfied at the time it is applied. MAY attempts to solve this problem by "reasoning backwards" from her main goal. (Reasoning backwards is similar to the process of "means-ends" analysis discussed by Newell and Simon, 1972.)

The mechanism that permits MAY to reason backwards involves a set of "reducer programs." Each reducer program is advertised to know how to achieve a certain effect. (That is, it knows how to add a certain type of proposition to the model.) It achieves this effect by using one of the m-actions after having first set up and solved the subgoals of achieving the m-action's preconditions. These subgoals are "reduced" to others by appropriate reducers until a chain of m-actions is produced that achieves the main goal. This process depends for its success on an assumption that the reducers specify subgoals that are in some sense simpler (or closer to the present situation) than are the goals being reduced.

Let us consider an example reducer, called "r-findtreasure," for finding a treasure. The program for this reducer is shown below:

r-findtreasure [to-achieve: HAVE(TREASURE)]
achieve: INVIEW(TREASURE)
apply: m-pickup(TREASURE)

The notation of the program is explained as follows: The expression [to-achieve: HAVE(TREASURE)] after the name of the program describes the program's purpose, namely, to cause the model to contain the expression HAVE(TREASURE). Since MAY reasons backwards, this purpose statement is very important; it is the clue that tells her that if her goal is to have treasure, she should consider applying the reducer r-findtreasure. The body of this reducer consists of an "achieve" statement and an "apply" statement. The achieve statement sets up a subsidiary goal for MAY, namely, the goal of getting the expression INVIEW(TREASURE) into the model. Once this precondition is achieved, the m-action, m-pickup(TREASURE), can be "applied." The effect of m-pickup(TREASURE) is to add the statement HAVE(TREASURE) to MAY's model. Thus we see that this reducer merely converts the problem of achieving its goal into a problem of achieving a subsidiary goal.

MAY has a full complement of reducers. In particular there is one that advertises itself as being able to achieve INVIEW(TREASURE) so that MAY can continue reasoning backwards until she can actually begin applying some m-actions. In running reducers, whenever an "achieve" statement is encountered that is already satisfied by the model, it can be passed over and the next statement can be considered.

Reasoning backwards, sometimes called problem reduction, top-down, or goal-directed reasoning, is a key strategy in AI systems. Reducers of the kind we are discussing here have proved to be valuable formalisms for this type of reasoning.*

*We could easily introduce other types in addition to the "to-achieve" reducers. Some of these would be of interest if MAY's world (and model) could change spontaneously due, perhaps, to the effects of other agents. Thus we might also have reducers describing how "to escape" from an existing undesirable situation, how "to protect" an existing desirable situation from being spoiled as a side effect of another action, how "to avoid" creating an undesirable situation, how "to await" the spontaneous occurrence of a desirable situation, how "to endure" an undesirable situation that is about to end spontaneously, how "to maintain" an existing desirable situation that would otherwise end spontaneously and how "to prevent" an undesirable situation from occuring spontaneously. (See Fikes, Hart, and Nilsson, 1972a, for a more complete discussion of these other possibilities.)

It is likely that MAY would have more than one reducer to achieve a particular goal. In such a circumstance, MAY faces a choice about which one to use. The choice may be crucial because one may specify subgoals that turn out to be difficult or impossible to achieve, while others may specify trivial subgoals.

Let us consider an example. MAY also has a reducer called r-shopfor(x) that can be used to achieve HAVE(x). The program is as follows:

r-shopfor(x) [to-achieve: HAVE(x)]

condition: PURCHASABLE(x)
condition: HAVE(MONEY)
achieve: AT[store(x)]
apply: m-purchase(x)

In this program we use a "condition" statement. The condition statement checks to see if the proposition listed after it is contained in the propositional model. If not, a "failure report" is generated. (I'll mention what happens to this failure report later.) If the condition is satisfied in the model, the next statement is considered. Condition statements are used instead of achieve statements for those propositions that must be true in order to apply the reducer. Here, if x is not PURCHASABLE, the reducer r-shopfor cannot be applied at all.

The last statement in r-shopfor(x) changes the model by applying m-purchase(x). This m-action adds HAVE(x) to the model.

Before listing some of MAY's other reducers, let us consider what happens if we start MAY's planning activities by running the simple program: achieve: HAVE(TREASURE). Assuming that HAVE(TREASURE) is not already in the propositional model, there are two reducers MAY could use: r-shopfor, with x bound to TREASURE, and r-findtreasure. Lacking absolute fore-knowledge of which would be best, MAY creates a symbolic structure, called a planning tree, to keep track of both possibilities. The tree is shown in Figure 2.

The nodes of the planning tree consist of two parts. The first part names the propositional model from which a plan is to be constructed.

In this case, let us call the initial model, M_0. The second names the reducer program that MAY would use at that node of the tree in order to generate the plan.

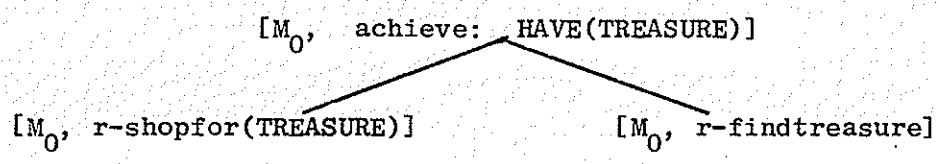


Figure 2 INITIAL VERSION OF MAY'S PLANNING TREE

We assume MAY has some way of selecting which tip node of the planning tree to try. The AI literature (see, for example, Nilsson, 1971) discusses several ways of proceeding, and for our present purposes we do not need to concern ourselves with tree-searching mechanisms. Suppose MAY selects the left-hand node and begins to execute r-shopfor(TREASURE). She quickly encounters a condition that fails: There is no PURCHASABLE(TREASURE) in M_0. When the failure is reported, MAY rules out the left-hand tip node and decides to work on the right-hand one. That is, she decides to use the reducer, r-findtreasure, still using M_0 as the model. (If r-findtreasure should fail, MAY's entire planning effort would itself fail, but fortunately, as we shall see, this branch of the tree will succeed.)

Before continuing with MAY's plan-generation task, let us introduce some additional reducers that MAY will need to complete her plan. These are shown in Figure 3. In r-excavate, the first condition statement, condition: TREASURESITE(x), has an unbound variable, x. When this statement is encountered, the model is searched for a proposition of this form. If one is found, say TREASURESITE(A), x is bound to the specific place, A, named in the proposition. There may be more than one possibility, for example TREASURESITE(B) may also occur in the model. If multiple possibilities exist, note is made of that fact in case the possibility chosen causes a failure later.

In r-excavate we also employ a "negative condition" statement, namely, negcondition: DUG(x). By this we mean that if DUG(x), with x

```
r-excavate [to-achieve:  INVIEW(TREASURE)]
     condition:  TREASURESITE(x)
     negcondition:  DUG(x)
     achieve:  HAVE(SHOVEL)
     achieve:  AT(x)
     apply:  m-dig

r-goto(x) [to-achieve:  AT(x)]
     apply:  m-go(x)
```

Figure 3 ADDITIONAL REDUCERS FOR MAY'S PLANNING

bound to the specific place obtained in the line above, does exist in the model, a failure is sent back to the previous choice point. In this case the failure would be caught by the line above, namely, condition: TREASURESITE(x), and a new value for x would be sought in the model. The variable x would be bound to this new value, and the negcondition tried again. If no new values of x can be found, condition: TREASURESITE(x) itself fails, and the failure is sent back to the program that called r-excavate.

After all condition tests and subgoals have been achieved in r-excavate, the model is changed by the application of m-dig. This m-action adds INVIEW(TREASURE) to the model. (MAY's planning system is optimistic and assumes that r-excavate will result in putting the treasure in view. As it happens, none of MAY's m-actions can put the proposition DUG(x) in the model. We are merely allowing for the possibility that versions of it may already be there.)

The other reducer contained in Figure 3, namely, r-goto, sets up no subgoals and tests no conditions. This easily-satisfied routine merely changes the propositional model by applying m-go(x). The m-action m-go(x) adds AT(x), for some value of x, to the propositional model after deleting any other propositions of the form AT(x).

Now we can continue with MAY's planning process. As can be seen in the planning tree in Figure 2, she executes the reducer r-findtreasure.

In so doing, she runs into the statement achieve: INVIEW(TREASURE). At this point, MAY's attention is shifted to the problem of achieving this subgoal. [She notes that after achieving it she must return to continue r-findtreasure, starting at apply: m-pickup(TREASURE).] Assume INVIEW(TREASURE) is not already in the propositional model. There is only one reducer advertised to put it there, namely, r-excavate. So a single successor node below [M_0, r-findtreasure] is set up in the planning tree as in Figure 4. This node contains M_0 to refer to the fact that we are still working from the initial propositional model. It also contains the "stack" of reducers, r-excavate, and 'r-findtreasure. The top routine in the stack, r-excavate, is the next to be run, and 'r-findtreasure refers to the continuation of r-findtreasure [namely, the statement apply: m-pickup(TREASURE)].

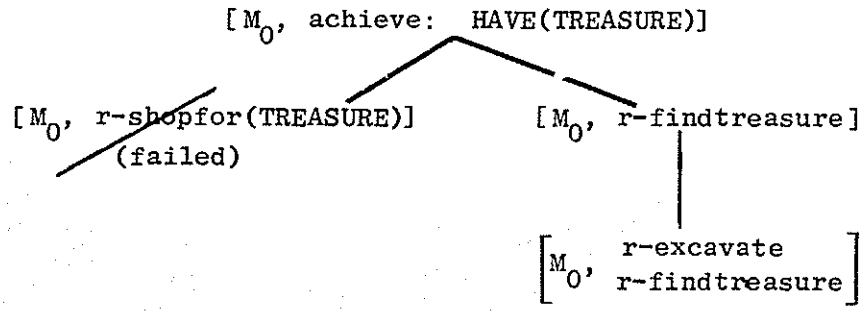


Figure 4 A STAGE OF MAY'S PLANNING TREE

Now MAY looks at her planning tree to decide which tip node to work on next. In this case there is only one, so MAY runs the routine r-excavate. In running this reducer, suppose MAY's model contains the proposition TREASURESITE(A) and does not contain DUG(A). Then, achieve: HAVE(SHOVEL) will be encountered, which, assuming no HAVE(SHOVEL) in the model, will cause additions to the planning tree. If we skip over some of the intermediate details of MAY's backward reasoning processes (assuming that the achieve statements call the appropriate reducers) we eventually produce the planning tree shown in Figure 5. In this figure, note that there are several reducers to be "continued" at the points where MAY's attention was shifted to work on subgoals. These are all indicated by the "'" notation.

At this stage, MAY selects the node at the bottom of Figure 5 and executes r-goto(SEARS). This reducer creates no subsidiary goals, but

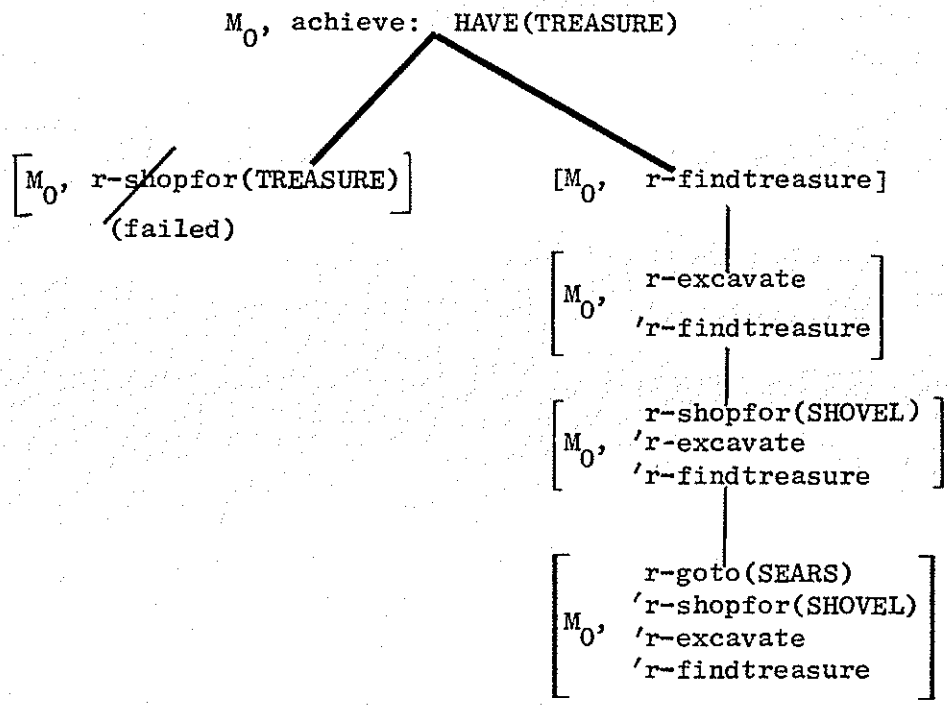


Figure 5 ANOTHER STAGE OF MAY'S PLANNING TREE

allows the direct application of m-go(SEARS). This m-action changes the propositional model, M_0, directly by substituting AT(SEARS) for MAY's position. Let us call this new model M_1. We represent this change in the planning tree by creating a new successor node:

$$\left[M_1,\ \begin{array}{l} \text{'r-shopfor(SHOVEL)} \\ \text{'r-excavate} \\ \text{'r-findtreasure} \end{array}\right]$$

MAY also keeps a list of the m-actions used so far. This list will ultimately constitute the finished plan. At this stage it is m-go(SEARS) .

Again let us skip over some intermediate details and continue the planning process until the planning tree shown in Figure 6 is produced. The bottom node has an empty (indicated by "nil") stack of reducers to run, and M_5 contains the goal statement HAVE(TREASURE), so planning is now complete. The final list of m-actions is the plan: [m-go(SEARS), m-purchase (SHOVEL), m-go(A), m-dig, m-pickup(TREASURE)]. Each of these routines corresponds to an action in MAY's action repertoire that can now be executed to achieve her main goal.

A glance at MAY's planning tree shows that her plan generation was rather direct (without many failures) in this simple example. A more powerful planning system would have a greater variety of reducers--sometimes

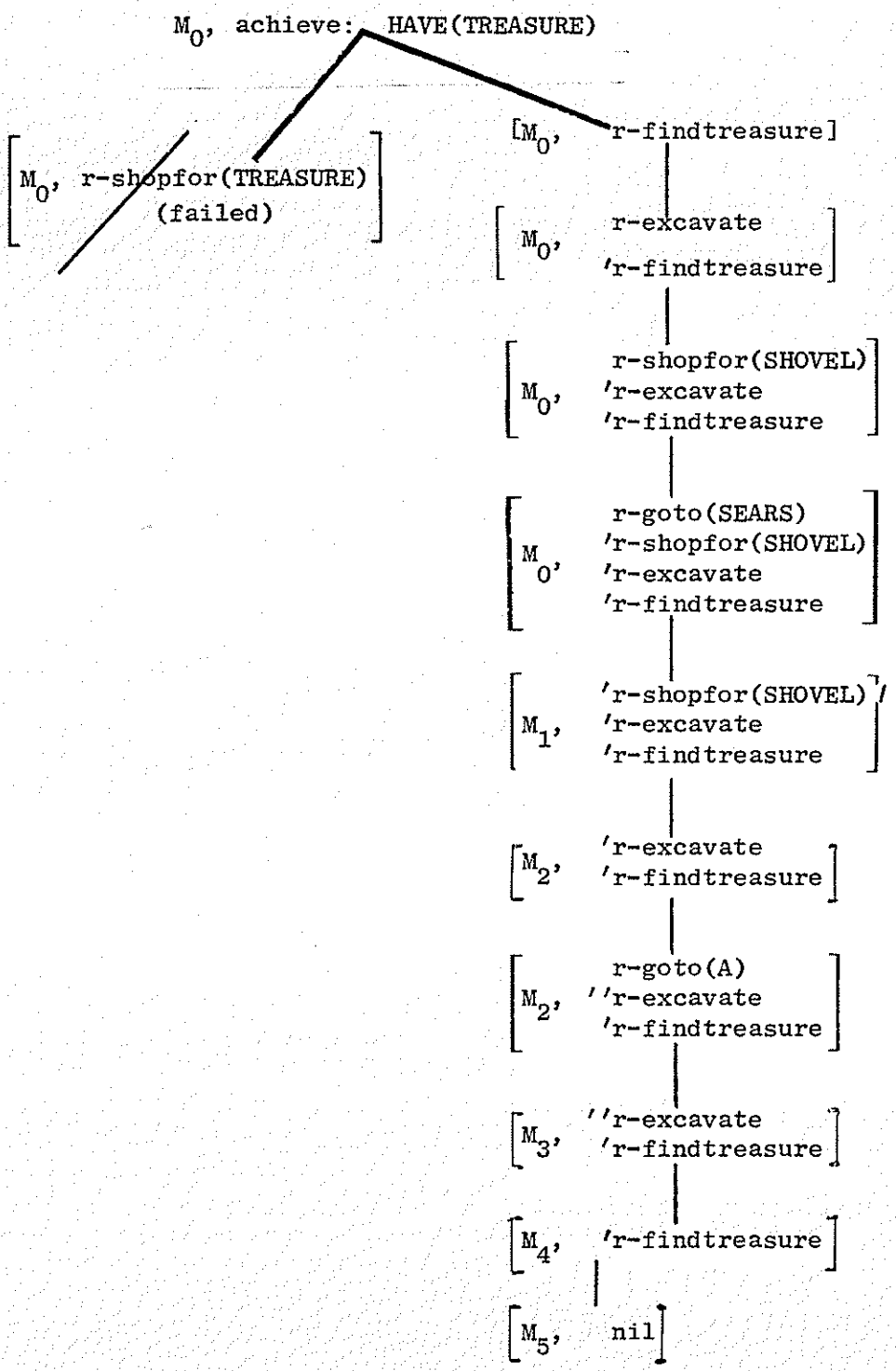


Figure 6 MAY'S FINAL PLANNING TREE

several different ones for accomplishing the same goal. The resulting planning tree would have had more branches and there would be a need to make use of better mechanisms to select best tip nodes. In fact the greatest limitations to using AI techniques like this for generating plans is the fact

that planning trees usually have too many branches to permit finding long plans. The so-called "combinatorial explosion" of alternatives finally dulls the power of this simple process.

The mechanisms used by MAY are simplifications of a cluster of more elaborate AI ideas. In particular, I have limited the discussion to simple "straight-line" plans--ones without loops and branches. A planning system called STRIPS (Fikes and Nilsson, 1971) that generated plans for the SRI robot, SHAKEY (Hart, et al., 1972) is quite similar in spirit to MAY. MAY's reducers can also be thought of as simplified versions of the "consequent theorems" of Hewitt, 1972. (Consequent theorems allowed arbitrary programming operations in addition to condition tests, subgoal generation, and model changes.) MAY's planning tree and ability to shift attention from one tip node to another closely parallel the mechanisms of STRIPS and the built-in features of the planning language QA4 (Rulifson, 1972). (Neither STRIPS nor QA4 literally made multiple copies of complete propositional planning models. Each had efficient ways of representing different models merely by noting the differences among them.)

MAY illustrates rather forcefully the advantages of reasoning backwards. Her only forward steps, namely m-actions, are embedded in reducers, and therefore they are applied only in the service of achieving specific goals. There are other occasions, however, when a forward-reasoning step is appropriate. For example, suppose MAY had been frustrated in her planning to find buried treasure because of lack of money to purchase a map listing treasure sites. Now if someone told MAY that place C is a place visited by Sir Francis Drake, then we might expect that she should jump (forward) to the conclusion that it is also a treasure site. That is, should if ever: VISITED-BY-SIR-F-DRAKE(x) suddenly appear in her propositional model, we might think it appropriate that TREASURESITE(x) should be immediately added.

Let us suppose that an augmented MAY possesses such watchful programs, that we shall call "demons," for adding (or deleting) model propositions. Demons are a means of forward reasoning just as reducers are a means of backward reasoning. The form of MAY's demons is illustrated by the following example:

d1(x) [if-ever: VISITED-BY-SIR-F-DRAKE(x)]
assert: TREASURESITE(x)

Demons are similar to productions and Markov tables in that they trigger actions based on the contents of the propositional model. In general the events triggered by demons can be arbitrary. They might be m-actions, actions in the real world or planning efforts. The following examples illustrate some possibilities:

d2(x) [if-ever: TREASURESITE(x)]
apply: m-go(x)
achieve: INVIEW(TREASURE)
d3(x) [if-ever: HAVE(MONEY)]
achieve: HAVE(TREASURE)

Skillful use of both demons and reducers by designers of robots like MAY can result in quite sophisticated and highly competent systems.

C. Hierarchical Planning: MATT

In the last section I spoke of the combinatorial explosion that limits the power of MAY's planning system. Here I shall describe a "hierarchical planner" that impedes the combinatorial explosion a bit and thus has greatly increased powers.

In attempting to compose a plan, MAY reasoned backward one step at a time until an m-action was found that allowed an immediate change to the propositional model. At each step, MAY was faced with possible choices, so long plans might require the investigation of large numbers of alternatives. Some of her reducers accomplished goals that could be regarded as unimportant details compared with the goals achieved by more important reducers. Consideration of details could well be postponed until the important "high-level" components of the plan are in place. For example, when an architect designs a house, he does not allow a concern about where to purchase nails to frustrate the completion of his preliminary sketches.

MATT is a robot who also makes plans to find buried treasure. He reasons backwards, as did MAY, except that his reasoning proceeds in hierarchically organized levels. The hierarchy is determined by the way

in which his reducers are defined. These reducers distinguish between those subgoals that must be achieved at the "same level" as that of the m-action in the reducer and those that can be postponed until a rough plan at this level has been completed. The process is best explained by referring to an example.

MATT's reducers are written slightly differently than are MAY's. Again we start with r-findtreasure:

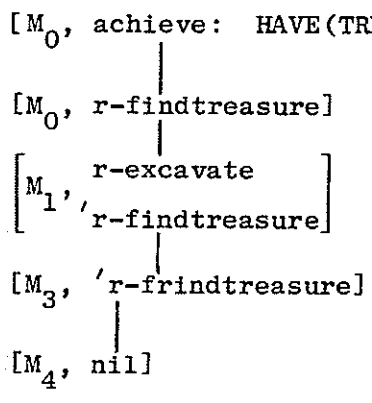
```
r-findtreasure [to-achieve:  HAVE(TREASURE)]
     p-achieve:  HAVE(SHOVEL)
     achieve:  INVIEW(TREASURE)
     apply:  m-pickup(TREASURE)
```

The statement "p-achieve: HAVE(SHOVEL)" means that planning for this subgoal is a detail to be postponed until later. For the moment, we assume the existence of some sequence of m-actions, temporarily denoted by p-achieve: HAVE(SHOVEL), that successfully changes the model by adding HAVE(SHOVEL) to it. Thus, a p-achieve statement optimistically adds its proposition to the model, but sets up the need to generate a more detailed plan later for accomplishing its assumed result. The rest of MATT's reducers are shown in Figure 7.

```
r-shopfor(x)  [to-achieve:  HAVE(x)]

     condition:   PURCHASABLE(x)
     condition:   HAVE(MONEY)
     p-achieve: [AT store(x)]
     apply:  m-purchase(x)


r-excavate [to-achieve:  INVIEW(TREASURE)]

     condition:  TREASURESITE(x)
     negcondition:  DUG(x)
     p-achieve:  AT(x)
     apply:  m-dig


r-goto(x)  [to-achieve:  AT(x)]

     apply:  m-go(x)
```

Figure 7 ADDITIONAL REDUCERS FOR MATT'S HIERARCHICAL PLANNING

Let us trace through a level of MATT's planning activity evoked by executing the statement "achieve: HAVE(TREASURE)." For economy of explanation, let's ignore the blind alley that results from considering r-shopfor(TREASURE) and instead invoke r-findtreasure directly. This reducer first assumes that HAVE(SHOVEL) can be achieved at a lower level of detail and then runs into the statement "achieve: INVIEW(TREASURE)." The reducer r-excavate is called, and this reducer in turn assumes that a treasure site A can be visited. After applying the appropriate m-actions, the first level of the plan is completed. The planning tree generated in completing this first level is shown in Figure 8.



Notes

In M_1 we add HAVE(SHOVEL) as an assumed result of a p-achieve.

In M_2 (not shown here) we add AT(A) as an assumed result of a p-achieve.

In M_3 we add INVIEW(TREASURE) as a result of an m-action.

In M_4 we add HAVE(TREASURE) as a result of an m-action.

Figure 8 PLANNING TREE FOR FIRST LEVEL OF MATT'S HIERARCHICAL PLAN

In this planning tree, new models M_1, M_2, M_3, and M_4 are produced as a result of applying m-actions and of assuming the results of p-achieves. The plan generated by the process at this stage is:
[p-achieve: HAVE(SHOVEL), p-achieve: AT(A), m-dig, m-pickup(TREASURE)].
In this plan there are two p-achieve statements that now need to be planned to a deeper level. These are converted to achieve statements and the planning process is called to generate plans for each of them. The planning trees for each of these are shown in Figure 9.

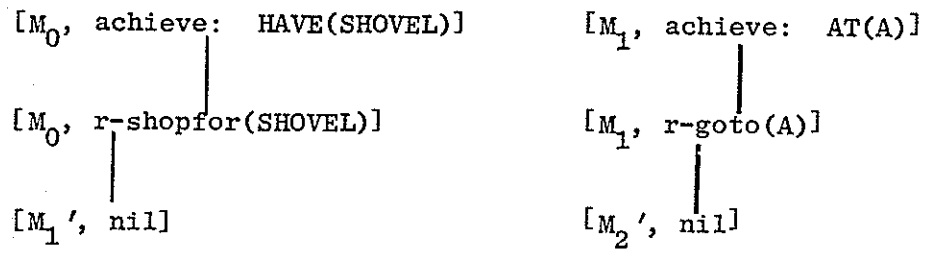


Figure 9 PLANNING TREES PRODUCED AT A LOWER LEVEL OF MATT'S PLANNING

In Figure 9, new models M_1' and M_2' appear. M_1' is the result of assuming AT(SEARS) and applying m-purchase(SHOVEL). Thus it differs from M_1 by possessing the additional "detail" AT(SEARS). M_2' and M_2 are identical; they each contain AT(A).

The plans produced by each of these subsidiary planning efforts, together with the higher level plan generated earlier, can be represented in another tree structure called a "plan net." We show the plan net in Figure 10.

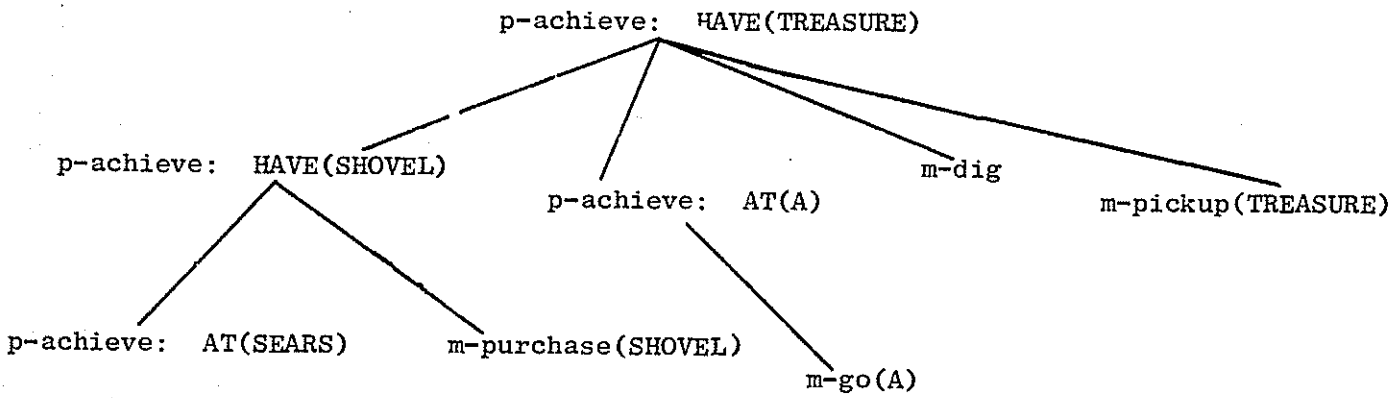


Figure 10 THE PLAN NET FOR THE FIRST TWO LEVELS OF MATT'S PLANNING

In the plan net, each p-achieve node has as descendants a sequence of nodes that represent a plan for the p-achieve node. The net of Figure 10 indicates that the plan must be carried to yet another level of detail since the net still contains a p-achieve node at a tip. The next stage of planning merely adds the node m-go(SEARS) as the single successor of the node p-achieve AT(SEARS). The final detailed plan is obtained by collecting the tips of the planning tree, in left-to-right order, into a list.

It should be obvious that the postponement of planning of details greatly increases efficiency when there are several reducers to accomplish the same effects. Postponing details reduces the branchiness of the planning tree and allows plans that are going to fail for high-level reasons to fail more quickly so that successful ones can be investigated sooner. The price MATT must pay for this enhanced efficiency is that sequences of low-level plans might not match up properly. That is, the unanticipated detailed effects on the model of the low-level plans might cause higher-level plans to be invalid. MATT may sometimes generate plans that paint him into a corner!

Such difficulties can usually be dealt with, however, because after all, MATT is only planning, not performing. Plans produced by MATT can be examined and criticized by other processes that look specifically for such conflicts and defects.

My description of MATT has attempted to explain just a small part of a much larger hierarchical planning and execution system, called NOAH, developed by Sacerdoti, 1975. NOAH generated hierarchical plans for assembling and disassembling mechanical equipment. (An air compressor was used as the chief experimental example.) NOAH had other important mechanisms, too. It could "criticize" its plans to spot and eliminate various conflicts and inefficiencies in them. It could generate plans in which no unnecessary commitments were made about the order in which some of the component actions were to be performed. Lastly, it had mechanisms for supervising the execution of the plans it generated, and it could replan if the execution got into unexpected difficulties.

In my descriptions of MAY and MATT I have ignored the problem of actually executing the plans that are produced. One would hope that there would be some way to combine the action abilities of MARK with the planning abilities of MAY or MATT. Sacerdoti's NOAH system (Sacerdoti, 1975) did combine planning and execution as did the STRIPS system for controlling the SRI robot SHAKEY (Fikes, Hart, and Nilsson, 1972b; Hart, et al., 1972). Both of these planning/execution systems introduced important ideas, but a completely satisfactory synthesis is probably still to be achieved.

D. Deductions: MAX

The logical powers of the previous robots were limited to checking for the presence of a given proposition in a propositional model. Their most complex feat required them to look for specific instances of a general proposition containing a variable. One could hope to expand these powers so that both the model and the condition to be tested could contain general logical formulas with quantified variables.

MAX is a robot that can perform some of these feats of deduction. MAX reasons about a static world. He does not perform actions nor does he even make plans to do so. He nevertheless is interested in buried treasure and can make various deductions about some propositions having to do with treasure hunting.

MAX reasons backwards also. He has reducers that replace propositions that he is trying to prove, with (it is hoped) easier propositions. He uses reducers until he obtains propositions that are contained in the model. (Each reducer embodies some particular chain of reasoning using inference rules of logic.)

MAX is interested in knowing whether or not SEARS sells items that his friends need for treasure hunts, namely, maps and shovels. One of his reducers can be written as follows:

r1(x) [to-deduce: SELLS(x,MAPS)]
    deduce: LARGE(x)
    deduce: HAS(x,STATIONERY-DEPT)
    assert: SELLS(x,MAPS)

This reducer is able to add the proposition SELLS(x,MAPS) to the propositional model if it first can show that the model contains LARGE(x) and HAS(x,STATIONERY-DEPT). The "assert statement" adds a proposition to the model. The "deduce" statements set up subgoal deductions. The reducer r1(x) makes use of the following syllogism:

from: LARGE(x) AND HAS(x, STATIONERY-DEPT)
    and
    LARGE(x) AND HAS(x,STATIONERY-DEPT) $\supset$ SELLS(x,MAPS)

we can infer:
    SELLS(x,MAPS)

Because there may be multiple reducers that can be used to deduce the same statement, again there is a need for some selection mechanism. Since we are not here considering different hypothetical worlds resulting from different reducers, we can use a single model that benefits from all of the assertions made.

Figure 11 shows some additional reducers used by MAX. Now let us assume that MAX's initial model, M_0, contains only the statements SELLS(SEARS,PENCILS) and HAS(SEARS,PARKINGLOT). If MAX is given the task deduce: SELLS(SEARS,MAPS), its "deduction tree" might look something like that shown in Figure 12. The end result of this deduction process is to add the desired statement, SELLS(SEARS,MAPS) to the model.

```
r5(x)  [to-deduce:  LARGE(x)]
        deduce:  HAS(x,PARKINGLOT)
        assert:  LARGE(x)

r6(x)  [to-deduce:  HAS(x,HARDWARE-DEPT)]
        deduce:  SELLS(x,NAILS)
        assert:  HAS(x,HARDWARE-DEPT)

r2(x)  [to deduce:  SELLS(x,SHOVELS)]
        deduce:  LARGE(x)
        deduce:  HAS(x,HARDWARE-DEPT)
        assert:  SELLS(x,SHOVELS)

r3(x)  [to deduce:  HAS(x,STATIONERY-DEPT)]
        deduce:  SELLS(x,PENCILS)
        assert:  HAS(x,STATIONERY-DEPT)

r4(x)  [to-deduce:  HAS(x,STATIONERY-DEPT)]
        deduce:  SELLS(x,BOOKS)
        assert:  HAS(x,STATIONERY-DEPT)
```

Figure 11 ADDITIONAL REDUCERS USED BY MAX

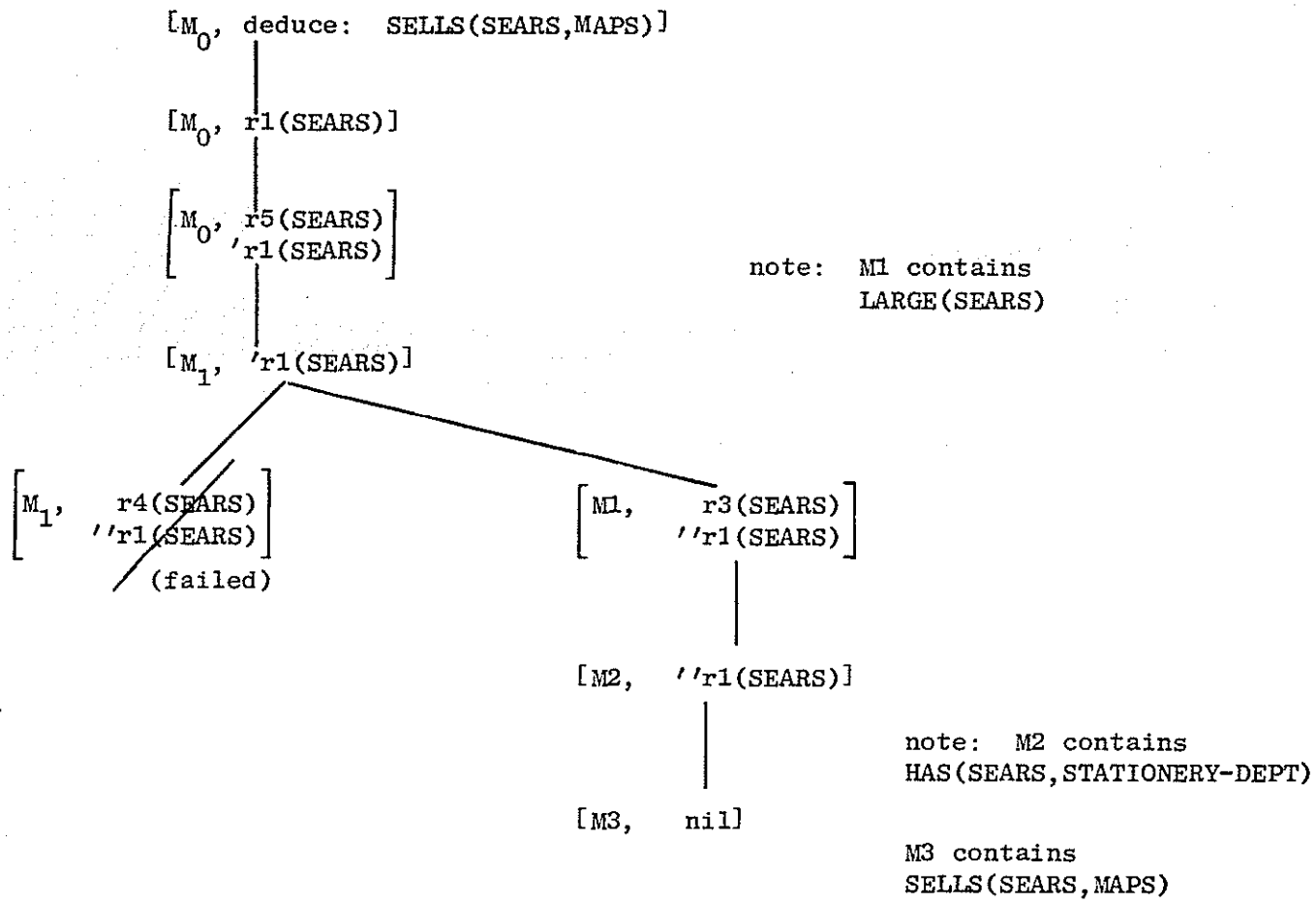


Figure 12 A DEDUCTION TREE GENERATED BY MAX

The deductive mechanisms of MAX are similar to those of several programs written in the newer AI languages such as QLISP (Sacerdoti, et al., 1976). Programs have been written to prove theorems (Waldinger and Levitt, 1974) and to make deductions about electromechanical equipment (Fikes, 1975).

In some instances enhanced efficiency is achieved by using demons to generate propositions that follow directly from other propositions already in the model. Thus, besides his reducers, MAX could have demons to do forward reasoning. In fact, the information in each of his reducers could also be written in demon form. The decision about whether to implement a logical inference rule as a reducer or as a demon (or both) is largely a question of efficiency. Are the required deductions made more quickly and directly by backward or by forward reasoning? The right mix depends largely on the domain being reasoned about. If there are few goals and many "axioms," reducers are likely to be more efficient. If there are few axioms and many possible goals, demons might be preferred.

MAX's example deduction involved only a simple goal expression not containing any quantified variables. More complex expressions can be handled by similar mechanisms. (See, for example, Fikes, 1975.)

Processes of deduction are as susceptible to the combinatorial explosion as are those of planning. It would probably be worthwhile to incorporate hierarchical techniques in deduction (similar to those used by MATT) to help make deductions more efficiently. One could consider augmenting MAX's reducers by using "p-deduce" statements in connection with subgoals that can be regarded as mere details. (I am unaware of any AI systems that use p-deduce statements.)

E. Inference: MAGGIE

The propositional models that we have used so far had only limited abilities for expressing uncertainty. If a proposition occurred in a robot's model, the robot was completely certain about it; if a proposition was absent, the robot was completely uncertain about it. Intermediate degrees of certainty were not represented.

If we examine the reducer programs used by MAX, we note that each incorporates a logical rule of the form: if ⟨various subgoal expressions can be deduced⟩, then ⟨a conclusion can be asserted⟩. This "if...then" rule is certain: it is not hedged by "maybes."

Yet in many situations (even in MAX's map-buying example) it would be more realistic to use deduction rules whose conclusions were uncertain (to a degree) even if the premises were certain. A parking lot might only suggest a large store, not necessarily imply one, for example.

MAGGIE is a robot that can deal with these kinds of uncertainties. MAGGIE represents both her propositional knowledge and her knowledge about how to make inferences in a structure called an "inference net." For each of the reducer programs used by MAX, there is an analogous "inference rule" in MAGGIE's inference net. The rules correspond to lines connecting nodes in the net. Each rule has a premise node or nodes, a conclusion node and a pair of numbers representing the strength of the rule. Each node in the net corresponds to a propositional statement to which MAGGIE attaches a probability (or degree of truth) varying between 0 (certainly false) to 1 (certainly true).

MAGGIE's inference net is shown in Figure 13. To each node there is attached a number representing MAGGIE's subjective view of the probability of the corresponding proposition. Thus, for example, MAGGIE is more sure that SEARS sells shovels than she is that it sells maps. We use a triangular symbol to indicate that two propositions combine as premises for a single conclusion.

Each rule has a pair of numbers to denote its strength. The first number in the pair indicates how sufficient the premise is to support the conclusion. Large positive values indicate a high degree of sufficiency; large negative values indicate that the premise is strongly sufficient for supporting the negation of the conclusion. The second number in the pair indicates how necessary the premise is for supporting the conclusion. Large negative values indicate a high degree of necessity; large positive values indicate that the premise is strongly necessary for supporting the negation of the conclusion. For both numbers, values near zero indicate weak connections between the premises and the conclusion. Thus, it is quite necessary for Sears to sell pencils for MAGGIE to believe that Sears has a stationery department, and it is strongly sufficient that Sears sells books to make that same conclusion.

The rule strengths are used by MAGGIE whenever it is appropriate for her to update her probabilities. For example, if someone were to tell MAGGIE (convincingly) that Sears had a parking lot, then MAGGIE would use the (6,-1) strength measure to update the probability of LARGE(SEARS) from 0.7 to 0.8, perhaps. This change would then in turn propagate to affect MAGGIE's belief in SELLS(SEARS,MAPS) and SELLS(SEARS,SHOVELS). The updating procedure uses the rule strength numbers and the new probabilities of the premises to calculate the new probability of the conclusion. (A full discussion of a suggested probability updating procedure and its relation to probability theory is contained in Duda, Hart, and Nilsson, 1976. These details are not needed for our present discussion.)

MAGGIE's inference net also tells her how to become more certain of any proposition: She should simply become more certain about some of its possible sets of premises. Note the close parallel between these inference net operations and MAX's reducers and demons. Working backward

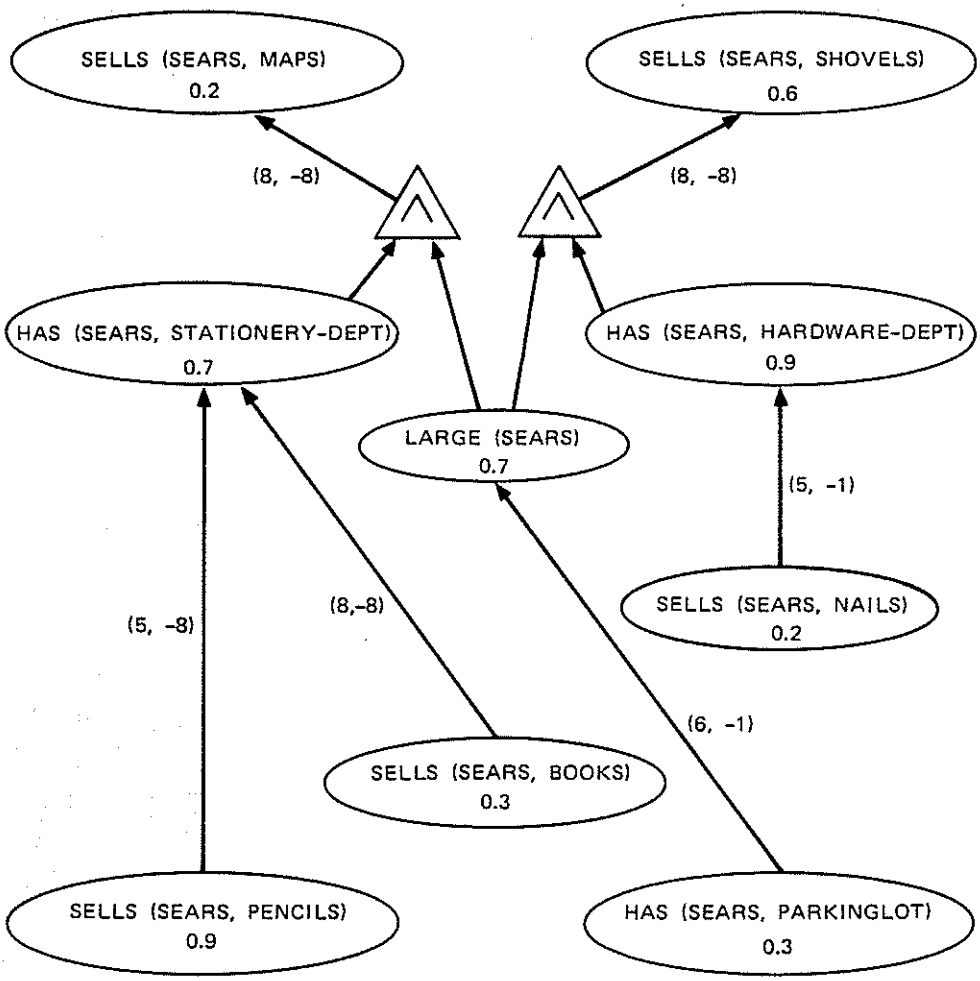


FIGURE 13 MAGGIE'S INFERENCE NET

through the net from conclusions to premises that might support them is analogous to using reducer programs. Propagating new probability values forward from premises to conclusions is analogous to using demon programs.

Let us suppose that MAGGIE ultimately gets new information about its world by talking on the telephone. She calls up Sears to ask if they sell maps, but has the misfortune of speaking with a new and not well-informed clerk. The clerk thinks Sears might sell maps, but isn't sure. MAGGIE revises the probability of SELLS(SEARS,MAPS) from 0.2 to 0.4. Using her inference net structure MAGGIE asks the clerk if Sears has a stationery department, updates probabilities, and so on until she has sufficient supporting evidence for her main question.

MAGGIE uses simple versions of techniques just now being explored by AI research. Her knowledge base of imprecise inference rules is similar to that of a large medical diagnosis system called MYCIN (Shortliffe, 1976). MYCIN uses inference rules that were obtained by interviewing physicians; each has rule strength values also supplied by the physicians. Questions involving how to search backwards through an inference net are just now being approached. But even though this work is preliminary, its goals are important. Much human reasoning seems to involve the ability to deal with uncertain evidence using nondefinite inference rules.

III. DISCUSSION

The AI mechanisms I have described have been used in a variety of applications. Their utility extends beyond the domain of goal-seeking actions, planning, and common-sense reasoning illustrated by our five robots. Similar mechanisms have been used in AI programs for visual scene analysis (Winston, 1972; Garvey, 1976), language understanding (Winograd, 1972), and the automatic generation of computer programs (Manna and Waldinger, 1975). Nor does this class of mechanisms exhaust the set of AI concepts that might be of use for constructing psychological models. Our brief vignettes of AI mechanisms should be regarded as just a sample. Similar ones could probably be written to illustrate other interesting AI concepts involved in language understanding (parsing, augmented transition networks, semantic networks) and perception (structures for describing scenes, scene-analysis methods.)

Semantic networks, in particular, are of increasing interest to psychologists. They can be regarded as a type of propositional model possessing somewhat richer interconnections than do most computer implementations of predicate calculus models. (Specifically, the "pointers" between predicate symbols and argument symbols are bidirectional in semantic networks.) Many of the early shortcomings of semantic networks have been overcome. Hendrix's partitioned semantic networks (Hendrix, 1975) can easily represent quantified propositions, hypothetical worlds, modal propositions and other constructs.

In view of the enhanced power of semantic networks, it is interesting to speculate about what changes would be necessary to our five robots if each used a semantic network propositional model instead of a standard predicate calculus propositional model. We would certainly have to extend our notion of reducers and demons so that they responded to and generated "pieces" of semantic network rather than sentences of the predicate calculus. To the extent that such semantic networks were also useful in building natural language understanding (and generating) systems, our robots would then have the added potential for conversation.

With or without semantic networks, the AI concepts discussed in this paper would appear to be useful building blocks in psychological models of both stimulus-driven and goal-seeking behavior involving action, planning, and reasoning. Perhaps a useful first domain would involve modeling nonhuman animals. An ideal choice to model might be some higher animal in which language and super-reasoning abilities would not have to be accounted for but one possessing modest planning abilities. Such a modeling effort would, to be sure, contrast somewhat with the current directions being pursued by cognitive psychologists. It is not being suggested here as an alternative to their work but rather as an addition. Also, just as the modern cognitive psychologists have elucidated certain ideas useful to AI scientists, we might hope that renewed efforts in animal modeling would help to clarify certain conceptual impediments involving coordinated planning and plan execution.

Acknowledgments

The author would like to thank the following people for reading an early draft of this paper and for making many helpful suggestions: Richard Duda, Richard Fikes, Peter Hart, Earl Sacerdoti, Georgia Sutherland, and Jay Tenenbaum.

REFERENCES

Anderson, J. A., and G. H. Bower, Human Associative Memory, John Wiley and Sons, New York (1974).

Duda, R. O., P. E. Hart, and N. J. Nilsson, "Subjective Bayesian Methods for Rule-Based Inference Systems," SRI Artificial Intelligence Center Technical Note 124, also in Proc. 1976 National Computer Conference (1976).

Estes, W. K., "The Statistical Approach to Learning Theory," in S. Koch (ed.), Psychology: A Study of a Science, Vol. 2, McGraw-Hill, New York (1959).

Fikes, R. E., "Deductive Retrieval Mechanisms for State Description Models," Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Vol. 1, pp. 99-106, Tbilisi, Georgia, USSR (September 1975).

Fikes, R. E., P. E. Hart, and N. J. Nilsson, "Some New Directions in Robot Problem Solving," B. Meltzer and D. Michie (eds.), Machine Intelligence, Vol. 7, Edinburgh University Press, Edinburgh (1972).

Fikes, R. E., P. E. Hart, and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," Artificial Intelligence, Vol. 3, No. 4, pp. 251-288 (Winter, 1972).

Fikes, R. E., and N. J. Nilsson, "STRIPS: A New Approach to the Applications of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2, pp. 189-208 (1971).

Galler, B. A., and A. J. Perlis, A View of Programming Languages, "Markov Algorithms," pp. 3-11, Addison-Wesley Publishing Co., Inc. (1970).

Garvey, T., "Perceptual Strategies for Purposive Vision," SRI Artificial Intelligence Center Technical Note 117 (1976).

Hart, P. E., et al., Artificial Intelligence-Research and Applications, Stanford Research Institute Annual Technical Report on Project 1530, Advanced Research Projects Agency, Contract DAHC04-72-C-0008 (December, 1972).

Hendrix, G., "Expanding the Utility of Semantic Networks Through Partitioning," Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Vol. 1, pp. 115-121, Tbilisi, Georgia, USSR (September, 1975).

Hewitt, C., Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot, MIT Artificial Intelligence Laboratory, Report No. AI-TR-258 (1972).

Manna, Z., and R. Waldinger, "Knowledge and Reasoning in Program Synthesis," Artificial Intelligence, Vol. 6, No. 2, pp. 175-208 (Summer, 1975).

Newell, A., "Production Systems: Models of Control Structures," in W. G. Chase (ed.), Visual Information Processing, Academic Press, New York (1973).

Newell, A., and H. Simon, Human Problem Solving, Prentice Hall, New York (1972).

Nilsson, N. J., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York (1971).

Norman, D. A., D. E. Rumelhart, and the LNR Research Group, Explorations in Cognition, W. H. Freeman and Co., San Francisco (1975).

Quillian, M. R., "Semantic Memory," in M. Minsky (ed.), Semantic Information Processing, M.I.T. Press, Cambridge, Massachusetts (1968).

Raphael, B., et al., Research and Applications--Artificial Intelligence, Stanford Research Institute Final Report on Project 8973, Advanced Research Projects Agency, Contract NASW-2164 (December, 1971).

Rulifson, J. F., J. A. Derksen, and R. J. Waldinger, QA4, a Procedural Calculus for Intuitive Reasoning, SRI Artificial Intelligence Center Technical Note 73 (1972).

Sacerdoti, E. D., A Structure for Plans and Behavior, SRI Artificial Intelligence Center Technical Note 109 (August, 1975).

Sacerdoti, E., et al., "QLISP: A Language for the Interactive Development of Complex Systems," SRI Artificial Intelligence Center Technical Note 120 (March 1976).

Shortliffe, E. H., Computer-Based Medical Consultations: MYCIN, Elsevier, New York (1976).

Waldinger, R. J., and K. N. Levitt, "Reasoning About Programs," Artificial Intelligence, Vol. 5, No. 3, pp. 235-316 (Fall, 1974).

Winograd, T., Understanding Natural Language, Academic Press, New York (1972).

Winston, P. H., "The MIT Robot," Machine Intelligence, Vol. 7, pp. 431-463, B. Meltzer and D. Michie (eds.), Edinburgh University Press, Edinburgh (1972).