PLANNING AND PLAN RECOGNITION

Henry A. Kautz and Edwin P.D. Pednault

***Henry Kautz** and **Edwin Pednault** are members of technical staff at AT&T Bell Laboratories. Mr. Kautz is in the Artificial Intelligence Principles Department at Murray Hill, New Jersey. He has an A.B. from Cornell University, an M.S. from the University of Toronto, and a Ph.D. in computer science from the University of Rochester. He joined AT&T in 1987. Mr. Pednault is a member of technical staff in the Knowledge Systems Research Department at Holmdel, New Jersey. He has a B.Eng. in electrical engineering from McGill University, an M.S. in computer science from Stanford University, and a Ph.D. in electrical engineering, also from Stanford. He joined AT&T in 1986.*

This paper introduces the reader to the areas of automatic planning and plan recognition. The former concentrates on the goal-directed synthesis of plans from primitive actions, and has applications in robotics, experiment design, and other fields. The latter concentrates on methods for inferring the goals that underlie an agent's actions, in order to support more natural human/machine interfaces and to model human communication. The paper also provides an overview of some recent results obtained by the authors as part of their doctoral research. The authors are continuing research in these areas at AT&T Bell Laboratories.

Introduction

A central part of intelligent behavior is the ability to explicitly reason about action. What will occur if I try to pick up the bottom block in a stack of blocks? What sequence of machine-tool operations should I perform to manufacture a mechanical part? Why did Richard send me the following message:

```
From: richard @ vivace
Date: Thu, 10 Sep 87 18:12:18 EDT
To: kautz@bebop
Who knows how to use Gnu Emacs?
```

and what does he expect me to do next?

The fields of planning and plan recognition aim to formally describe and construct computer systems that can answer these questions. Both fields address issues of representing knowledge about actions and the plans created by the composition of actions. The former concentrates on the synthesis of plans that can be used to achieve specified goals; the latter, on the discovery of the plans, and thereby the implicit goals, that underlie an agent's explicit actions.

The first question above illustrates the need in both fields to be able to reason about the effects of actions and the contexts in which they are performed. If the stack of blocks is glued together, the result is that I'm holding the entire stack. If the blocks are loose, the result may be that I'm holding a single block, and the rest have probably spilled on the table.

The second question is an example of planning—i.e., "stringing" actions together to make a plan that achieves a desired goal (Panel 1). The actions in this case are machining operations, such as drilling and milling. The goal is to manufacture a specified part.

The third question is an example of a plan recognition problem, where one might reason as follows: "Richard must have some reason for asking who knows how to use Emacs (a text-editing program). Finding out who knows a piece of information is often the first step in a plan to find out that information itself. Therefore, it is a good bet that Richard wants to know how to use Emacs. Now, I don't know any users of Emacs, but I can respond to Richard's implicit goal by telling him where the Emacs documentation is kept."

This paper discusses techniques developed by the authors and others for formalizing and solving planning and plan recognition problems.

Applications of Planning. Research in automatic planning is generally concerned with ways of solving problems of the following form: Given

1. A set of goals
2. A set of allowable actions
3. A description of an initial state of affairs

find a sequence of actions that will bring about a state of affairs in which all of the desired goals are satisfied. Problems of this form were the first to be explored in automatic planning. We shall therefore refer to them as classical planning problems. While not all planning problems fit this mold, at some level almost all planning problems embody within them one or more classical problems. In this respect, the classical planning problems are among the most fundamental.

Many real-world problems may be formulated directly as classical planning problems. Thus, techniques for solving classical problems potentially have wide application. Deciding how to machine a mechanical part is one application that is currently receiving attention in the literature.[1] Several other applications have been or are being examined as well. These include:

1. Robot planning,[2-5] where the object is to find a sequence of arm/hand/body motions necessary to accomplish a desired task.
2. Automatic programming/microcoding,[3,4] where the object is to find a sequence of machine/microcode instructions (i.e., a straight-line program) for efficiently carrying out a specified computation.
3. Experiment design in molecular genetics,[6] where the

Panel 1. What Does AI Mean by Planning?

Work on planning and plan recognition in artificial intelligence is distinguished by the attempt to automatically generate and explore part of the space of plans, in order to find candidate plans that meet the desired criteria. This complements but differs significantly from the concerns of operations research and decision theory, which attempt to optimize a given plan, or choose between a small number of fully specified alternatives. While probability theory is relevant to plan recognition, nearly all work in that field again makes the assumption that the problem is solely one of choosing among fixed alternatives—not, as in AI, that of constructing the alternatives themselves.

A project manager thinks of "planning software" as programs that assist in organizing a large number of parallel tasks. The human makes the basic decisions about what should be performed and in what order; the computer notes conflicts and calculates temporal constraints. To someone working in artificial intelligence, a planning system should itself decide what steps to take in what order. This is a much harder problem, and at present most planning systems can only construct short, sequential plans. There is certainly great potential for research on systems that combine the planning technologies of the two fields.

object is to propose a sequence of laboratory steps for creating a strain of bacteria capable of manufacturing a desired protein.

4. Geological interpretation,[7] where the object is to hypothesize a sequence of geological events (i.e., sedimentation, uplifting, intrusion, etc.) that could account for an observed geological formation.
5. Natural-language generation,[8] where the object is to construct a sequence of utterances to satisfy certain communicative goals (i.e., informing, requesting, promising, etc.).

A general technique for solving such problems is outlined in this paper. This recent technique[9] unifies and generalizes ideas found in many of its predecessors.

Applications of Plan Recognition. Plan recognition problems occur when human or computerized agents must closely interact while lacking perfect knowledge of each other's intentions. The knowledge gained through plan recognition can improve coordination and reduce the unpleasant consequences of unforeseen conflicts.

A major area of application involves advanced natural-language interfaces for databases and expert systems. One of the first research projects in this area[10] modeled a guide in a train station that could infer a user's goal from a fragmentary input ("The 8:15 to New York?") or from an indirect question ("Do you know where the Toronto train leaves?"). Once the goal was known, a planning module "fleshed out" the user's plan, and provided helpful information or warnings of possible problems. While this project performed a very sophisticated analysis of an extremely limited range of inputs, commercial systems are now being marketed which "skim" databases of newspaper stories and other simple texts in order to extract the stereotypical plans of the desired agents.[11]

A main stream of research in computational linguistics analyzes human language in terms of the communication plans and speech acts[12] that make up a dialog. This very active area[13,14] of theoretical research promises to show how the plans and goals that structure a discourse are recovered and used in language comprehension. Difficult problems such as word-sense ambiguity and the reference of pronouns can often be resolved by this approach. While these projects aim to perform a very sophisticated analysis of an extremely limited range of inputs, commercial systems are now being developed and promoted which perform a "shallow" (and possibly incorrect) interpretation of a broader range.

Not all applications of plan recognition involve language. The MACSYMA Advisor[15] helped users of that powerful symbolic-algebra program by creating a representation of the user's intended but "buggy" plan, and then suggesting repairs. The input to the system was the series of formulas originally input to MACSYMA. Another nonlinguistic application is an "operating system advisor" that analyzes a transcript of a user's session at a terminal.[16] The resulting user model includes information about the user's goals, and can serve as input to a "do what I mean" (DWIM) facility.

Three approaches to solving plan recognition problems are examined in this paper, with particular attention given to a recent technique called abductive classification.[17]

Planning

Mathematical Model. To program a computer to solve a problem, one must first construct a mathematical abstraction of the problem that reflects all of the features relevant to finding a solution. Planning problems are no exception.

In the standard mathematical abstraction of the classical planning problems, the world is presumed to be in one of a potentially infinite number of states. The effect of an action is to cause the world to make a transition from one state to another. An action is therefore modeled as a set of current-state/next-state pairs specifying what the effects of the action would be in each state of the world. In a planning problem, we are told that the world is initially in one of a set of possible initial states and we are asked to find a sequence of actions that will leave the world in one of a set of acceptable goal states.

Figure 1 illustrates this model. Notice that for the problem shown, the only solution is to perform action A followed by action B, since this is the only sequence of actions for which there is a path from each of the possible initial states to one of the acceptable goal states. The sequence A followed by C, on the other hand, is not a solution. Even though this sequence leads to a goal state if we start in state S_3, we would not end up in a goal state if we were to start in state S_2. Notice also that not all actions can be performed in every state; for example, action B can be performed in state S_6 but not in state S_3. An example in everyday life would be the action of walking through a doorway, which can be performed only when the door is open. Actions can also have unpredictable effects and, hence, multiple outcomes. For example, performing action C in state S_4 has two potential outcomes (S_5 and S_7), but it is impossible to predict beforehand which will occur. This would correspond, for instance, to flipping a coin.

Problem Representation. From Figure 1, it is evident that classical planning problems are closely related to path-finding problems in graph theory. However, unlike typical path-finding problems, the number of states involved in a typical planning problem is either infinite or at least so large that it is impractical to enumerate them all. For example, it has been shown[9] that the problem of landing a spacecraft on the moon may be cast as a classical planning problem. For this problem, the number of states is uncountably infinite. Clearly, the standard graph-theoretic path-finding techniques are inappropriate in such instances.

Since the numbers involved usually prevent us from dealing with states and state transitions explicitly, we must instead deal with them implicitly through language. By constructing suitable (formal) languages for describing facts about states and actions, problems that involve an infinity of states can be formulated and solved.

In most planning systems, formulas of first-order logic[18] are used to describe facts about states (i.e., goals, initial conditions, etc.). Each formula represents a set of states—namely, the set of states in which the formula is true. To illustrate this use of first-order logic, consider the following problem.

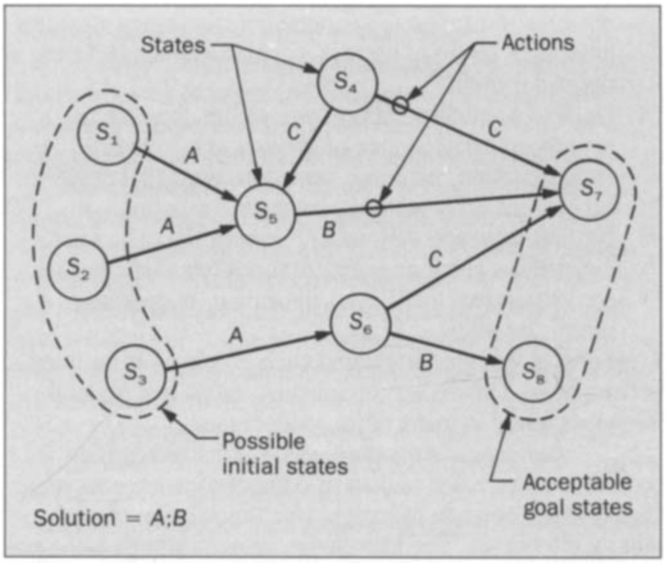

Figure 1. A sample planning problem expressed as a state-transition graph.

Suppose that we have a world that consists of three objects—a briefcase, a dictionary, and a paycheck—each of which may be situated in one of two locations: the home or the office. Actions are available for putting objects in the briefcase and for taking objects out, as well as for carrying the briefcase between the two locations. Initially, the briefcase, the dictionary, and the paycheck are at home; the paycheck is in the briefcase, but the dictionary is not. The goal is to have the briefcase and the dictionary at the office, and the paycheck at home.

To refer to the objects and locations, we will use five constant symbols, B, D, P, H, and O, corresponding, respectively, to the briefcase, dictionary, paycheck, home, and office. In addition, two relation symbols, `At` and `In`,

Panel 2. Goals and Initial Conditions

(For the briefcase problem as expressed in first-order logic)

Goals:

$\mathtt{At}(B,O) \wedge \mathtt{At}(D,O) \wedge \mathtt{At}(P,H)$

Initial conditions:

1. $c_1 \neq c_2$ for all distinct pairs $c_1, c_2 \in B,D,P,H,O$
2. $\forall x\,(x = B \vee x = D \vee x = P \vee x = H \vee x = O)$
3. $\forall x,y\,(\mathtt{At}(x,y) \leftrightarrow [(x = B \vee x = D \vee x = P) \wedge y = H])$
4. $\forall x\,(\mathtt{In}(x) \leftrightarrow x = P)$

Symbol	Meaning	Symbol	Meaning
B	briefcase	$\wedge$	and
D	dictionary	$\vee$	or
P	paycheck	$\neg$	not
H	home	$\rightarrow$	implies
O	office	$\leftrightarrow$	if and only if
`At`	at location	$\forall$	for all
`In`	in briefcase	$\exists$	there exists

will be used to refer to the relations that may exist between the objects and locations. `At` is a binary relation such that $\mathtt{At}(x,y)$ is true if and only if object x is at location y. `In` is a unary relation such that $\mathtt{In}(x)$ is true if and only if object x is in the briefcase.

Using these symbols, the goals and initial conditions of the briefcase problem may be written as shown in Panel 2. Thus, the acceptable goal states are precisely those states in which the briefcase is at the office (i.e., $\mathtt{At}(B,O)$), the dictionary is at the office (i.e., $\mathtt{At}(D,O)$), and the paycheck is at home (i.e., $\mathtt{At}(P,H)$). The possible initial states are those states in which:

1. B, D, P, H, and O are mutually unequal and, hence, represent distinct entities.
2. B, D, P, H, and O are the only entities in existence.
3. The only entities that have locations are B, D and P, and they are all at H.
4. The only entity in the briefcase is P.

To describe actions and their effects, separate languages are often used. Most planning systems employ variations on the STRIPS operator language developed by Fikes and Nilsson.[2] However, as discussed by Lifschitz[19] and Chapman,[20] STRIPS and STRIPS-based languages have many inherent limitations having to do with semantics and expressive power. A recent language developed by Pednault,[9] on the other hand, overcomes these limitations to a large extent.

In this language, called ADL (Action Description Language), an action is described in two parts. The first part is a list of formulas that define the preconditions of execution of the action. These preconditions define the set of states in which the action can be performed. For exam-

Panel 3. Actions for the Briefcase Problem
(As expressed in ADL)

```
PutIn(x)     ;;;Put object x in the briefcase

   Precond: x ≠ B, ∃ l [At(x,l) ∧ At(B,l)]
       Add: In(x)

TakeOut(x)   ;;; Remove object x from the
                 briefcase

    Delete: In(x)

MovB(l)      ;;; Move the briefcase to location l

       Add: At(B,l)
            At(z,l) for all z such that In (z)

    Delete, At(B,m) for all m such that m ≠ l
            At(z,m) for all z,m) such that m≠l ∧ In(z)
```

ple, a precondition to walking through a doorway is for the door to be open. The second part of an action description is a set of transformation rules that specify how the state of the world changes when the action is performed. Given that formulas of first-order logic are used to describe states, each state must correspond semantically to an algebraic structure.[18] An algebraic structure defines a set of objects that exist in the world, along with a complete enumeration of the relations that hold among the objects. It is these relations that are referred to in formulas (e.g., `At` and `In`). The effect of jumping from one state to another amounts to making changes in the values of the relations. Such changes are specified in ADL schemas as sets of tuples to be added to and deleted from each relation.

The ADL schemas defining the actions for the briefcase problem are shown in Panel 3. The effect of `PutIn`(x) is to cause object x to be placed in the briefcase if it is not already there. Since the briefcase cannot be placed inside itself, `PutIn`(x) has as one of its preconditions that x cannot be the briefcase (i.e., $x \neq B$). It also has the precondition that x must be at the same location as the briefcase (i.e., $\exists\, l\, [\mathtt{At}(x,l) \wedge \mathtt{At}(B,l)]$), since otherwise it would be impossible to put x in the briefcase. The effect of `TakeOut`(x) is to cause object x to be removed from the briefcase. If x is not in the briefcase, the action has no effect. `MovB`(l) causes the briefcase and everything in it to change from their current location to location l. If the briefcase and its contents are already at location l, `MovB`(l) has no effect.

As a notational convenience, a relation that is not explicitly modified is presumed to remain unchanged. Thus, `PutIn`(x) and `TakeOut`(x) do not affect the `At` relation, and `MovB`(l) does not affect the `In` relation. In this way, one need only specify what changes take place when an action is performed without having to enumerate everything that does not change (compare Reference 21).

Plan Synthesis. Historically, most research aimed at solving planning problems has been highly experimental in nature, the emphasis being to demonstrate ideas through computer programs. Recently, though, the mathematical foundations of plan synthesis have been explored by Chapman[20] and Pednault.[9] Chapman's analysis addresses the STRIPS framework and thus excludes actions whose effects depend on the context in which they are performed. Such actions include the `MovB`(l) action of Panel 3 and the action of picking up the bottom block in a stack of blocks discussed in the introduction. Pednault's analysis greatly extends Chapman's results to include such actions and to allow for highly complex goals and incomplete knowledge of the initial conditions.

The synthesis technique developed by Pednault is based on two properties of classical planning problems. The first is that the state of the world can change only when an action is performed. Consequently, if a condition is true at one point during the execution of a plan but not at an earlier point, then at some point in between an action was performed that made it true. The other property is that plans must be finite. This implies that if some condition is true at one point during the execution of a plan but not at an earlier point, then not only must there have been an action in between that caused it to become true, but there must have been a last such action in the interval. These properties thus give rise to the following theorem:

Theorem 1. A condition ϕ is true at a point p during the execution of a plan if and only if one of the following holds:

1. An action is executed prior to point p that causes ϕ to become true and ϕ remains true thereafter until at least point p.
2. ϕ is true in the initial state and remains true thereafter until at least point p.

The second clause of the theorem comes about by noticing that if some condition is true at a point p during the executing of a plan and is not false at some previous point, then the condition must be true at all points prior to p.

The preceding theorem is used as the basis for an incremental planning technique in which actions are inserted into a plan as needed and where needed to achieve the desired goals. With this technique, one begins with the empty plan (i.e., the plan containing no actions) and adds actions until a solution is obtained. At each stage in the process, there is some current plan. This plan is analyzed to identify those goals and preconditions that are not yet satisfied and to determine what additional actions are needed to bring them about. The appropriate actions are then inserted, producing a new current plan and initiating a new cycle of analysis and modification. This process of repeatedly analyzing and modifying the current plan continues until all goals and preconditions have been satisfied. In situations where there are multiple ways of causing a particular goal or precondition to become true, the analysis produces a set of alternative modifications of the current plan. In this case, one of the alternatives must be selected and then carried through. Since some ways of effecting one goal or precondition may make it impossible to achieve another, not all alternatives necessarily lead to solutions. Thus, it may be necessary to explore a number of alternatives before a solution is found.

Theorem 1 suggests two ways of modifying a plan to bring about a goal: one is to insert an action that causes the goal to become true; the other is to set up the appropriate conditions to prevent the goal from becoming false. There is, however, a third option: since plans are built incrementally, the action that causes a goal to become true in the final solution may already appear in the current plan. Therefore, another way of achieving a goal would be to establish the appropriate conditions to enable an existing action in the plan to cause the goal to become true. The conditions that enable actions to achieve goals are called causation preconditions, while the conditions that enable actions to preserve the truth of goals are called preservation preconditions. These conditions ensure that actions are performed in contexts conducive to producing their desired effects and are a key component of the planning

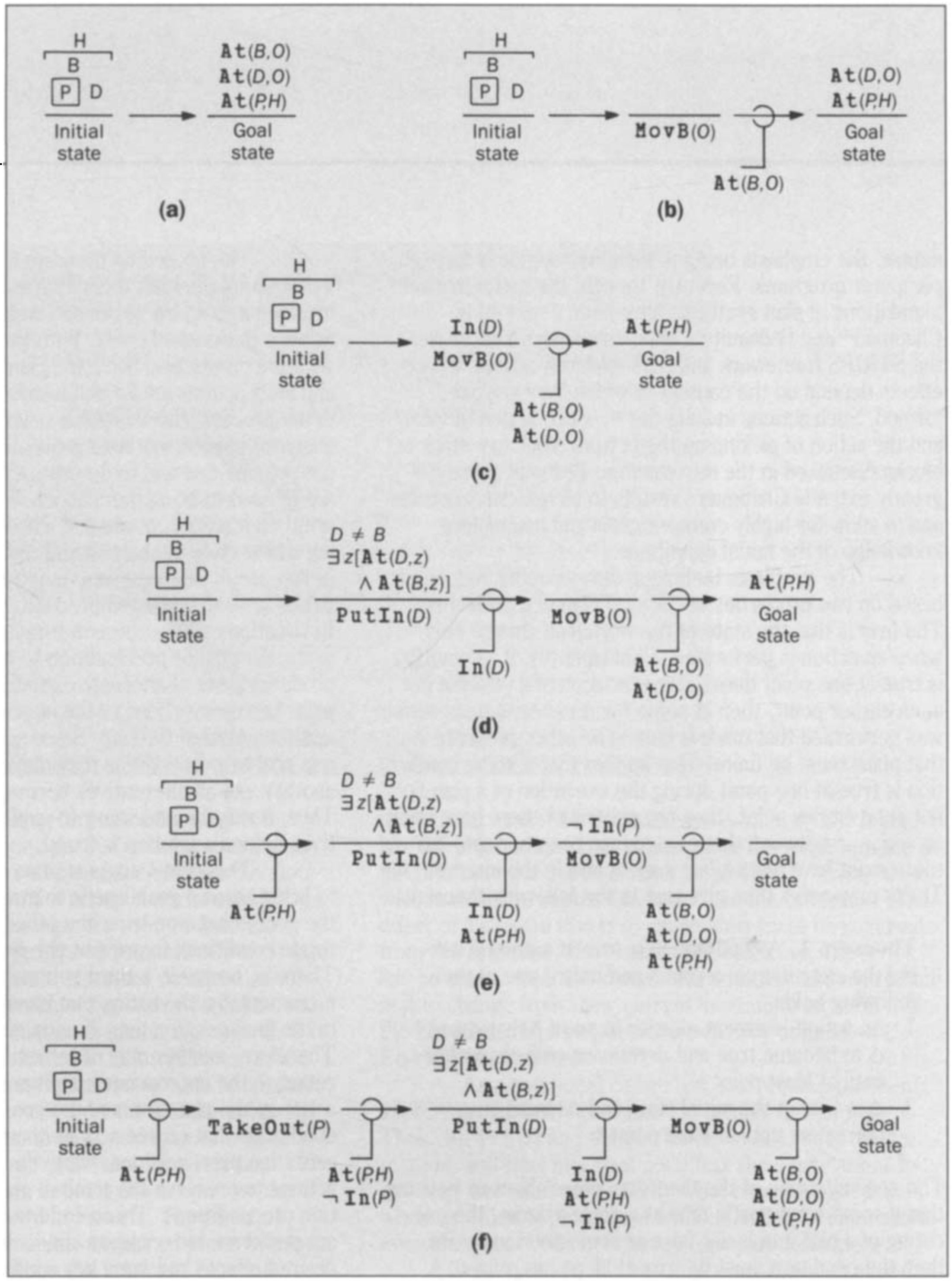

Figure 2. Steps in the solution of the brief-case problem.

technique outlined here. Methods have been developed to construct causation and preservation preconditions automatically from ADL schemas.

The three ways of modifying a plan may be illustrated by solving the briefcase problem introduced earlier. We begin the planning process with the plan shown in Figure 2a. In the diagram, the initial state is depicted graphically while the goals are simply listed. The arrow from the initial state to the goal state indicates that the initial state precedes the goal state in time. Note that no actions yet appear in the plan.

Let us first consider the goal of having the briefcase at work. Since the briefcase is not initially at the office, it must be brought there. The plan is therefore modified by inserting the action `MovB`(O) (Figure 2b). `MovB`(O) is to be the last action that achieves `At`(B,O), so `At`(B,O) is removed from the list of goals and a record is made that `At`(B,O) is to remain true between the `MovB`(O) action and the goal state.

If we now consider the second goal, which is to have the dictionary at the office, we notice that it too is unsatisfied in the initial state. However, since we have already decided to bring the briefcase to work, the dictionary would be brought along as well if it happened to be in the briefcase at the time. This can be inferred from the `MovB`(l) schema in Panel 3 by setting $l = O$ and matching z with D in the second add-clause. The plan is therefore modified by introducing `In`(D) as a subgoal to `MovB`(O) to allow this existing action to achieve `At`(D,O) in addition to `At`(B,O) (Figure 2c). Technically speaking, `In`(D) is a causation precondition that enables `MovB`(O) to achieve `At`(D,O).

The subgoal of having the dictionary in the briefcase is not true initially; therefore, we must insert the `PutIn`(D) action to place it there (Figure 2d). Note that the action of placing the dictionary in the briefcase has as preconditions that $D \neq B$ and that the dictionary and the briefcase be at the same location, which already happen to be true.

After making these modifications, we are left with only one more goal to consider, which is to have the paycheck at home. Since the paycheck is initially at home, one way to achieve this goal is to prevent it from becoming false. This is accomplished by setting up the appropriate conditions to enable all the intervening actions to preserve the fact that the paycheck is at home. Putting the dictionary in the briefcase does not affect the location of the paycheck, so no special enabling conditions are necessary for this action. However, bringing the briefcase to the office will cause the paycheck to go along as well if it happens to be in the briefcase at the time. Therefore, to prevent the paycheck from changing locations, the formula $\neg$`In`(P) is introduced as a subgoal to the move-briefcase action (Figure 2e). Technically speaking, $\neg$`In`(P) is a preservation precondition that enables `MovB`(O) to preserve `At`(P,H).

Since the paycheck is initially in the briefcase, it must be removed by introducing a `TakeOut`(P) action. If we choose to remove the paycheck from the briefcase before inserting the dictionary, then our final plan will be to remove the paycheck from the briefcase, put the dictionary in the briefcase, and bring the briefcase to the office (Figure 2f).

It can be shown that all solutions to any classical planning problem can be constructed in the manner illustrated above—i.e., by a combination of inserting new actions to achieve goals, establishing the appropriate conditions to enable existing actions to achieve goals, and establishing the appropriate conditions to prevent goals from becoming false. Hence, the technique is both general and complete. It is also important to point out that the technique does not require goals to be addressed in a particular order as do other techniques, e.g., those in References 2, 3, and 22. This avoids a tremendous amount of wasted effort trying different orderings until an appropriate one is found.

Managing the Search Space. As discussed earlier, at any point in the planning process there may be several modifications one can make to the current plan. Exploring these alternatives can be quite expensive. If b is the average number of alternatives encountered at each step and n is the number of modification steps necessary to arrive at a solution, one might potentially have to explore on the order of b^n alternative plans before a solution is found. If b and n are too large, the problem under consideration becomes computationally intractable. Clearly, it is advantageous to find ways of minimizing b and n.

One way to accomplish this is to employ a least-commitment search strategy in which certain decisions are delayed as long as possible until a more educated assessment can be made as to the correct decision. This has the effect of reducing the average number of alternatives that will be considered. With the planning technique described above, as with most planning techniques, one can delay the exact choice of which action to insert in a plan and the exact point of insertion.

To defer a choice among actions, variables called formal objects are introduced as arguments to parameterized actions. For example, suppose that to perform a particular task, a certain object A must be moved to make way for another object B. Instead of deciding immediately where to move A, a variable can be used as a place-holder to defer this choice. Depending on the other goals we wish to achieve, it may be advantageous to move A to a particular spot, perhaps to act as a support for other objects. By introducing a variable, we can first determine what must be done to achieve our other goals and then use this information to decide upon the best place to move A. Such use of variables was first proposed by Sussman[22] and has been incorporated into virtually all subsequent planning systems.

To defer a choice of where to insert an action, a plan can be represented as a partial ordering of actions instead of a total ordering as shown in Figure 2. For example, in the solution to the briefcase problem, it really does not matter whether we put the dictionary in the briefcase before removing the paycheck or after. This indifference is conveyed by the following partial order:

$$\begin{tikzcd}
& \texttt{PutIn}(D) \arrow[dr] & & \\
\text{Initial state} \arrow[ur] \arrow[dr] & & \texttt{MovB}(O) \arrow[r] & \text{Goal state} \\
& \texttt{TakeOut}(P) \arrow[ur] & &
\end{tikzcd}$$

This plan states that `PutIn`(D) and `TakeOut`(P) must both be performed before `MovB`(O); however, the ordering between `PutIn`(D) and `TakeOut`(P) is left unspecified. Similarly, partial orders enable us to insert an action into a plan without having to specify the ordering of that action with respect to *all* other actions. Such nonlinear plans were first introduced by Sacerdoti[23] and are viewed as being essential to efficient planning.

Another way of reducing the size of the search space is to employ a hierarchical planning strategy. The idea here is to first plan at a coarse level of detail and then successively refine the plan to finer levels of detail. For example, consider the problem of getting to Chicago. The initial plan might be to take an airplane, as opposed to driving or taking a train. The next refinement adds the step of getting from one's current location to the airport. This subproblem might be solved by devising a plan to drive to the park-and-ride lot and catch a shuttle bus. When it comes time to execute this plan, it will be "fleshed out" to the smallest detail, including the acts necessary to start and steer the car, and so forth. Hierarchical planning is essentially top-down design. It was first introduced to planning by Sacerdoti.[24] A recent analysis by Korf[25] shows that, while the plans constructed by this method are rarely optimal, complex problems are virtually impossible to solve without employing a hierarchical approach.

Plan Recognition

The previous section defined a planning problem by a set of goals, a set of allowable actions, and an initial state. The statement of a plan recognition problem

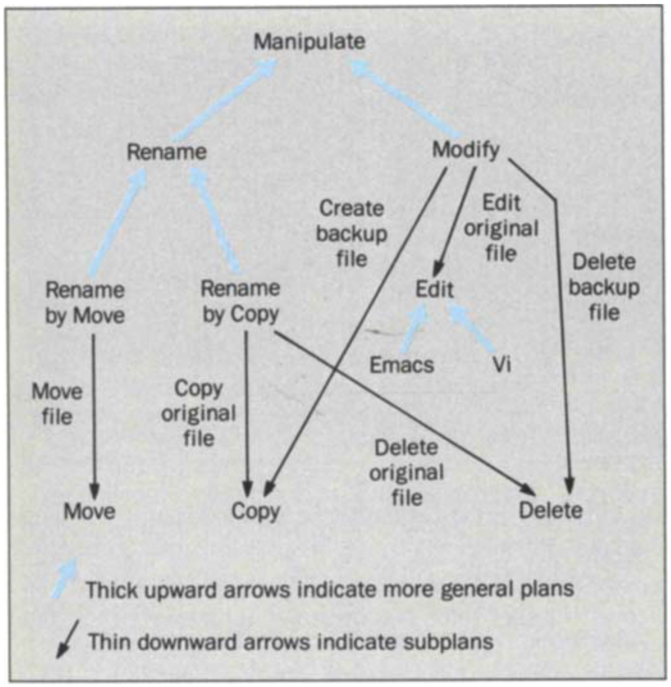

Figure 3. Simplified operating system plan hierarchy.

includes a similar but not identical set of factors:

1. A set of allowable actions
2. A set of observed actions
3. Descriptions of the contexts in which the observed actions were performed.

The most general goal of a plan recognition problem is to find an explanation of the observed actions in terms of their occurrence in one or more plans that can be attributed to the actor. If only these factors were considered, then every plan recognition problem would have a very uninteresting solution: the plan consisting solely of the observed actions. But in general only certain plans are acceptable explanations. Such a plan relates a *typical* pattern of events to a goal that one may reasonably expect to arise in the current context.

In a classical planning problem, one can logically deduce that a proposed solution achieves the given goal. In plan recognition, however, one tries to infer a plausible plan. This process is not purely deductive, but involves the use of heuristic assumptions and libraries of common plans. One must also determine at what point explanation stops, that is, when one says, "The agent acted in such a manner simply because he so desired." Such backward reasoning from effect to cause is called "abduction" and is much harder to formally characterize than pure deduction. Most work on plan recognition has been empirical and experimental; the work by Kautz[17] described below is the first to provide an abstract mathematical characterization of the process.

Consider the electronic message described at the beginning of the paper. An inadequate explanatory plan would be that Richard sent the message simply because he wanted me to receive it. A better solution could be the one described, where I view his act as a step in the plan to learn how to use Emacs. But suppose I know that Richard in fact wrote Emacs; in this context, the proposed plan to find out how to use Emacs is very unlikely, because its goal already holds. A better explanation might be, for example, a plan to find tutors to train new users.

Other factors may also help determine what constitutes an explanatory plan: for example, one may devise a way to measure the complexity of a plan, and prefer the simplest explanations. A crucial heuristic is to minimize the number of unrelated actions, that is, prefer plans in which the observed actions contribute toward achieving the same overall goal, rather than toward unrelated goals.

Thus a plan recognition problem may include any or all of the following:

1. A set of expected goals
2. A library of typical plans
3. A preference ordering over the space of plans.

The Plan Library. The plan library imposes a hierarchical structure on the space of plans. We generalize the concept to permit a plan to be recursively constructed from smaller subplans, and identify actions with plans of length one. Furthermore, one plan may be taken to be a specific way to perform another more general plan.

The semantic net[26] in Figure 3 illustrates the plan/subplan and specialization relationships between some of the plans useful to the user of a computer operating system shell. We see, for example, that one way to `Manipulate` a file is to `Modify` it, which requires three substeps: make a backup `Copy` of the original, `Edit` the original, and then `Delete` the backup. The `Edit` plan can itself be performed in two ways: one can invoke `Emacs` or invoke `Vi`.

Such a library, or plan taxonomy, provides much of the control information used in the plan recognition process. Three main approaches have been taken to describing this process itself. These are search, parsing, and abductive classification.

Search. Plan recognition can be directly formulated as a search in plan space, starting with a plan fragment describing the observation, and one or more goals which are expected to arise in the context. For example, the train station help system of Allen[10] input a description of a patron's utterance (such as the cryptic question, "The Montréal train?"), and considered two expected goals: the patron was expected either to travel on a train, or to meet someone arriving on a train. The system searches for a plan which connects the observed action to one of the expected goals. (For a discussion of language plans, see Panel 4.)

Hypothesized plans are modified by inserting actions which are enabled by the observed action, or help achieve the expected goal. If a plan appears which may be a subplan of another, then the latter plan may also be inserted. A number of heuristics are used to guide this search. Plans are disfavored if they contain actions whose effects are already true or whose preconditions are not satisfied at the time they occur. The incremental nature of the search favors shorter plans.

The result of the search is a single explanatory

Panel 4. Language Plans

In order to understand how and why language is used, one may go beyond the syntactic analysis of language, and view speech as a kind of planned, rational behavior. The preconditions and effect of a speech act describe the mental state of the speaker and hearer. For example, in order to inform someone of some fact, the speaker must know the fact; afterward, the hearer should know the fact. This speech act could be represented in ADL as follows:

```
Inform(speaker, hearer, fact)
  Precond: Know(speaker,fact)
     Add: Know(hearer,fact)
```

Other speech acts include interrogatives (or requests to inform), commands, and performatives (nonlinguistic actions that are performed by language, such as a judge declaring a defendant guilty). A series of speech acts makes up a discourse plan. For example, a plan to discover whether some proposition *P* holds may include the performance of two speech acts: (1) the speaker requests the hearer to either inform the speaker that *P* is true or inform the speaker that *P* is false, and (2) the hearer carries out that request.

The plan-based analysis reveals important uniformities of language. For example, an agent normally performs an action only if the agent desires the effect to be true. Thus the hearer of an inform can deduce that the speaker wants the hearer to know the fact—and this desire must itself be explainable in terms of further goals of the speaker. In this way a simple informative act can transmit a request to achieve some higher-level goal. For instance, the utterance, "It's cold in this room," can be understood as an attempt by the speaker to make the hearer warm the room.

Planning has also been successfully employed in text generation systems, where the system has the goal that the hearer or reader comes to know all the material to be conveyed.[8]

Panel 5. Logical Form of Operating System Plan Hierarchy (Partial)

1. Every `RenameByCopy` is a way of performing `Rename`.

$$\forall\, plan, old, new[\mathtt{RenameByCopy}(plan, old, new) \rightarrow \mathtt{Rename}(plan, old, new)$$

2. Every `RenameByCopy` has two steps, `Copying` the file and `Deleting` it.

$$\forall\, plan, old, new\ [\mathtt{RenameByCopy}(plan, old, new) \rightarrow \exists\, \mathrm{plan}_1, plan_2,\ \mathtt{Copy}(plan_1, old,\ new) \wedge \mathtt{Delete}\ (plan_1, old)]$$

3. Every `Modify` has three steps, `Copying` the file to back it up, `Editing` the file, and `Deleting` the backup.

$$\forall\, plan, old\ [\mathtt{Modify}(plan, old) \rightarrow \\ \exists\, new, plan_1, plan_2, plan_3, \mathtt{Copy}(\ plan_1, old, new) \wedge \mathtt{Edit}(plan_2, old) \wedge \mathtt{Delete}(plan_3, new)]$$

plan. This straightforward formulation has the drawback of finding only a single solution, even if several solutions are equally likely, and has trouble in dealing with multiple observations.

Plan Parsing. If one considers only the hierarchical structure of the plan library, and is dealing with a continuous sequence of discrete observations, it is natural to formulate plan recognition as a parsing problem. Most of the plan recognition systems applied to discourse use this formulation, and perform a bottom-up parse of the speaker's speech acts. A problem arises when there is ambiguity—for example, the utterance "Do you know where the Emacs manuals are?" could either be part of a literal Yes/No question exchange, or a request to find out where the manuals are. The parser can choose the interpretation which fits into an expected plan, or simply wait for further input.

Recent work at AT&T Bell Laboratories by Litman[13] and Hirschberg and Litman[27] extends the parsing paradigm to handle interruptive subdialogs and uses speech intonation to help reduce ambiguity.

Abductive Classification. The desire to deal with multiple, concurrent plans and observations in a formally satisfying manner has recently led to formulation of plan recognition in terms of abductive classification. The term "abduction" refers to the process of inference to the best explanation, while "classification" refers to use of the taxonomic plan library. The formal characterization of the plan recognition process provides a sound basis for the construction and justification of particular recognition algorithms.

The library specifies logical constraints on the structure of plans. If one plan is a way of performing another, then the former is a sufficient condition for the latter; logically, the former implies the latter. Similarly, a plan is a sufficient condition for each of its steps. Panel 5 shows the logical form of part of the operating-system plan hierarchy illustrated earlier. We have extended the notation for plans introduced earlier in the paper by including an initial parameter for the name of the particular instance of a plan. (This is not a feature of the ADL language.) For example, a particular observed instance of the user deleting the file "a.out" might be given the name D_1, and the act recorded by the formula $\mathtt{Delete}(D_1, a.out)$.

In order to use the taxonomy for recognition, several assumptions must be made. The first is that the taxonomy is complete. This justifies the abductive inference from the observation of a subplan to exactly the disjunction of all plans in the taxonomy which contain that subplan. Plans that do not appear as a step of any other plan are assumed to be "self-justifying." The particular application determines which plans are self-justifying, and do not stand in need of further explanation. Kautz[17] shows that "circumscription,"[28] an extension to ordinary logic which has generated much interest in the AI community, can be used to to automatically generate the necessary completeness assumptions.

The second assumption is that a set of observed actions can be explained by postulating as few unrelated (self-justifying) plans as possible. This is a version of "Occam's razor," which states that one should not multiply entities unnecessarily. Note that it is only an assumption: if it is logically necessary to allow for, say, two unrelated plans, one makes that assumption, rather than the stronger assumption that there is only one self-justifying plan in execution.

The second assumption defines a partial order over the space of explanations. Explanations containing n self-justifying subplans are preferred over those containing m such subplans, for all $m > n$. The observer is justified in inferring any statement which holds true for the entire class of minimal explanations. In logical terms, from the assumptions, observation, and taxonomy, the observer deduces a description of the set of most likely explanatory plans. The following simplified example from the domain of an operating-system advisor illustrates this approach.

Suppose the plan recognition system observes each action the user performs. During a session the user types the following commands.

1. `% copy foo bar`
2. `% copy jack sprat`
3. `% delete foo`

The system should recognize two concurrent plans. The first is to rename the file *foo* to *bar*. The second is to either rename or modify the file *jack*. Let's examine how these inferences could be made.

Call the first instance of copying a file C_1. The completeness assumption for `Copy` lets the system infer that C_1 is either the first step of an instance of `RenameByCopy` or the file backup step of an instance of `Modify`. Call this self-justifying plan S_1, described by the following disjunctive formula:

$$\texttt{RenameByCopy}(S_1, foo, bar) \lor \texttt{Modify}(S_1, foo)$$

From the second `Copy`, one can infer a similar disjunctively described plan S_2.

$$\texttt{RenameByCopy}(S_2, jack, sprat) \lor \texttt{Modify}(S_2, jack)$$

It is inconsistent to suppose that S_1 and S_2 are identical, so the strongest simplicity assumption one can make is that there are two unrelated plans executing simultaneously.

The completeness assumption for `Delete` states that every instance of that act is a subplan of either `RenameByCopy` or `Modify`. Therefore observation (3), `Delete`(C_3, foo), is explained by a hypothetical plan S_3, which is disjunctively described as either renaming *foo* or modifying a file for which *foo* is a backup. The variable x in the following formula stands for the other file (not yet determined) to which *foo* is related:

$$\exists x\ \texttt{RenameByCopy}(S_3, foo, x) \lor \texttt{Modify}(S_3, x)$$

The simplicity assumption attempts to force S_3 to be the same as S_1 or S_2. `Delete`(C_3, foo) cannot be grouped with either of the earlier observations as the step in modifying a file where the backup copy of the file is deleted (after successfully editing the original); no intervening `Edit` was observed, and furthermore, the file names mismatch. Furthermore, S_2 and S_3 cannot describe the same plan to `Rename` a file, because the file names *jack* and *foo* don't match. Thus one can conclude that observations (1) and (3) are part of `RenameByCopy`(S_1, foo), and (2) is part of either `RenameByCopy`$(S_2, jack, sprat)$ or `Modify`$(S_2, jack)$. At this point the plan

plan recognizer could trigger an "advice giver" to tell the user:

```
*** A more efficient way to rename a file
is to type

*** % move oldname newname
```

Although the recognition process has been described in logical terms, an implementation of this theory does not have to manipulate formulas as strings of symbols. One efficient technique is to dynamically construct a graph to explain each observation, chaining from each observation to the most-general self-justifying plan, and then to minimize the number of unrelated plans by graph matching. The characterization of the theory as a preference ordering over the plan space can be used to prove correctness of the graph-based algorithm.

Abductive classification handles general temporal constraints between subplans, so that, for example, one subplan may be constrained to run concurrently with another, or run in sequence. The taxonomic hierarchy can be arbitrarily deep, and a single plan may be taken to be a way of performing several other plans. Independent work by Reggia, Nau, and Wang[29] presents a similar system as a general model of diagnostic reasoning.

Current Work

A system is currently being implemented to test and refine the planning technique discussed in this paper, and to extend the mathematical analysis upon which the technique is based. The planner is to be domain-independent in that its design does not assume a particular application. However, the design does permit the planner to be tailored and tuned to a specific domain. The planner will be tested, at least initially, on robot planning problems.

The theory of abductive classification is currently being applied in the construction of an interactive plan recognition and tracking system. One application under consideration would be integrated with an electronic mail system and recognize and track the plans underlying the flow of messages to and from a worker in a computerized office. Another possible application would be a system to help coordinate the plans of a number of different agents working on a joint project, such as a large piece of software.

References

1. C. C. Hayes, *Planning in the Machining Domain: Using Goal Interactions to Guide Search*, masters thesis, Mellon College of Science, Carnegie Mellon University, April 1987.
2. R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2, 1971, pp. 189-208.
3. R. Waldinger, "Achieving Several Goals Simultaneously," in *Machine Intelligence 8*, E. Elcock and D. Michie, eds., Ellis Horwood, Edinburgh, Scotland, 1977, pp. 94-136.
4. D. H. D. Warren, "Generating Conditional Plans and Programs," *Proceedings*, Summer Conference, Society for the Study of Artificial Intelligence and the Simulation of Behaviour, July 1976, University of Edinburgh, Edinburgh, Scotland, pp. 344-354.
5. T. Lozano-Perez et al., "Handey: A Robot System that Recognizes, Plans, and Manipulates," *Proceedings*, 1987 IEEE Intl. Conf. on Robotics and Automation, Raleigh, N. C., March 1987, pp. 843-849.
6. M. Stefik, "Planning With Constraints (MOLGEN: Part 1)," *Artificial Intelligence*, Vol. 16, No. 2, May 1981, pp. 111-140.
7. R. Simmons and R. Davis, "Generate, Test, and Debug: Combining Associational Rules and Causal Models," *Proceedings*, 1987 International Joint Conference on Artificial Intelligence, Milan, pp. 1071-1078.
8. D. E. Appelt, *Planning Natural-Language Utterances to Satisfy Multiple Goals*, Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, Calif., December 1981.
9. E. P. D. Pednault, *Toward a Mathematical Theory of Plan Synthesis*, Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, Calif., December 1986.
10. J. F. Allen, "Recognizing Intentions from Natural Language Utterances," in *Computational Models of Discourse*, M. Brady, ed., MIT Press, Cambridge, Mass., 1983.
11. G. DeJong, "Skimming Stories in Real Time: An Experiment in Integrated Understanding," Technical Report 158, Yale Univer-

sity, Department of Computer Science, New Haven, Conn., 1979.
12. J. R. Searle, *Speech Acts*, Cambridge University Press, Cambridge, England, 1979.
13. D. J. Litman and J. F. Allen, "A Plan Recognition Model for Subdialogues in Conversation," *Cognitive Science*, Vol. 11, 1987, pp. 163-200.
14. B. J. Grosz and C. L. Sidner, "Attention, Intentions, and the Structure of Discourse," *Computational Linguistics*, Vol. 12, No. 3, July 1986.
15. M. R. Genesereth, "The Role of Plans in Automated Consultation," *Proceedings*, 1979 International Joint Conference on Artificial Intelligence, Tokyo.
16. K. Huff and V. Lesser, "Knowledge-Based Command Understanding: an Example for the Software Development Environment," Technical Report 82-6, Computer and Information Sciences, University of Massachusetts at Amherst, Mass., 1982.
17. H. A. Kautz, "A Formal Theory of Plan Recognition," Technical Report 215, Department of Computer Science, University of Rochester, Rochester, N. Y., May 1987.
18. J. R. Shoenfield, *Mathematical Logic*, Addison-Wesley, Reading, Mass., 1967.
19. V. Lifschitz, "On the Semantics of STRIPS," in *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, M. P. Georgeff and A. L. Lansky, eds., Morgan Kaufmann, Los Altos, Calif., 1987.
20. D. Chapman, "Planning for Conjunctive Goals," *Artificial Intelligence*, Vol. 32, No. 3, July 1987, pp. 333-377.
21. F. M. Brown, ed., *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, Morgan Kaufmann, Los Altos, Calif., 1987.
22. G. J. Sussman, "A Computational Model of Skill Acquisition," Technical Report AI TR-197, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., August 1973.
23. E. D. Sacerdoti, *A Structure for Plans and Behavior*, Elsevier, New York, 1977.
24. E. D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, Vol. 5, No. 2, Summer 1974, pp. 115-135.
25. R. E. Korf, "Planning as Search: a Quantitative Approach," *Artificial Intelligence*, Vol. 33, No. 1, September 1987, pp. 65-88.
26. R. Brachman, "The Basics of Knowledge Representation and Reasoning," *AT&T Technical Journal*, Vol. 67, No. 1, January/February 1988, pp. 7-24.
27. J. Hirschberg and D. Litman, "Now Let's Talk about Now," *Proceedings*, 25th Meeting of the Association for Computational Linguistics, Stanford University, Stanford, California, July 1987, pp. 163-171.
28. J. McCarthy, "Applications of Circumscription to Formalizing Common Sense Knowledge," *Artificial Intelligence*, Vol. 28, 1986, pp. 89-116.
29. J. Reggia, D. S. Nau, and P. Y. Wang, "Diagnostic Expert Systems Based on a Set Covering Model," *International Journal of Man-Machine Studies*, Vol. 19, 1983, pp. 437-460.

(Manuscript received October 8, 1987)