Modifying Previously-Used Plans to Fit New Situations

Roy M. Turner
School of Information and Computer Science
Georgia Institute of Technology

Abstract

Re-using plans that were created for one situation to solve a new problem is often more efficient than creating a new plan from scratch (e.g., [Fikes *et al*, 1972] and [Carbonell, 1986]. However, a plan that was created for one problem may not exactly fit a new situation; in that case, it will have to be modified. There are two major problems with re-using plans: (1) deciding whether to modify a plan, use it as is, or discard it; and (2) modifying the plan efficiently. Our solution to these problems is to store information with plan preconditions to guide the planner during plan application. Our approach is novel in two ways. First, we have identified a type of precondition, called a *flexible precondition*, that has information associated with it that helps the planner decide whether or not to modify the plan should the precondition be violated. Second, our preconditions contain information (derived from past experience using the plan) that provides heuristics for changing the plan so that the offending precondition is either no longer violated or no longer necessary. By using this approach, our planner can quickly determine whether or not to modify a plan, then efficiently perform the modification. Our work is implemented in the Consumer-Advisor System (CAS) [Kolodner and Cullingford, 1986; Turner, 1986; Turner, in press], a common-sense advice-giving program.

1.0 Introduction

When a problem is presented to a planner, it has one of two choices: it can attempt to formulate a new plan to solve the problem by combining operators from its repertoire; or it can recall and use a plan that was formulated to solve a problem in the past.

A difficulty with re-using a plan, however, is that the problem for which the plan was originally constructed may be only similar to the current problem and not identical. Some goals in the new problem may not be met by the expected results of the plan, for example, or one or more of the plan's preconditions may be violated. In this case, the planner can do one of several things. It can discard the plan immediately. This is unattractive, however, since the plan represents previous planning experience that should not be wasted. The planner can perform additional planning in order to satisfy the violated preconditions, the traditional approach to precondition violations. This can be arbitrarily hard, however, and makes sense only when it is easier than modifying the plan. The third alternative available to a planner is to use the plan "as is". Unless the planner can predict the outcome of doing this, however, a good solution is unlikely; and predicting the outcome usually involves simulation of the plan, which is costly in terms of time. A better solution than all of these, especially when the recalled plan almost fits the current problem, is to modify the plan.

There are two major problems to be dealt with in plan modification: (1) deciding whether to modify the plan at all, or instead to use it "as is" or to discard it; and (2) modifying the plan efficiently. Our solution to both of these problems is to store planning information with a plan's preconditions. This approach is novel in two ways. First, not all of our preconditions are criteria that *must* be satisfied in order for a plan to be applied. Some preconditions, called *flexible preconditions*, have information associated with them that allows the planner to determine what the outcome of applying the plan would be should the violated precondition be ignored. Second, all of our preconditions have directives stored with them that provide heuristics to the planner to guide it during plan modification.

Our approach is implemented as part of the Consumer-Advisor System (CAS) [Kolodner and Cullingford, 1986; Turner, 1986; Turner, in press], a common-sense advice-giving program whose domain is consumer products. In this paper, we first give a brief overview of CAS, then discuss our method of plan modification.

2.0 The Consumer-Advisor System

CAS is a common-sense advice-giving program. It gives advice about acquiring consumer products, such as furniture and bookshelves. Its primary problem-solving strategy is plan instantiation: it recalls a previous plan, then attempts to fit it to the current problem.

CAS uses several concurrent processes to perform its tasks. A *dynamic memory* [Schank, 1982; Kolodner, 1984] constantly tries to remember plans and other information pertinent to the current problem situation; when it is reminded of something, it notifies the planner. The planner is a separate process that can opportunistically use information from remindings to influence its problem-solving behavior. If the reminding is of a previously-used plan, the planner attempts to apply it to the current problem, if possible. If the plan is a close fit, but the current situation violates some of its preconditions, the planner uses information stored with the plan's preconditions in an attempt to modify the plan to fit the situation.

Before describing our approach to plan modification, we present an example of CAS' problem-solving behavior to illustrate the process. The problem presented to CAS was the following:

> I want to buy some bookshelves for my study at home. I am a student, so I can't spend much. What kind of bookshelves should I buy?

We will begin at the point in CAS' problem solving when the memory returns a plan to build wooden objects, BUILD-WOODEN-OBJECTS (Figure 1); for a more complete example of CAS' behavior, see [Turner, in press].

The planner first notifies the user that it has been reminded (by the memory) of a plan for building wooden objects (Figure 1):

I am reminded of plan ⟨BUILD-WOODEN-OBJECT⟩, the plan "building wooden objects".

The planner next tries to determine, by matching goals of the problem with results of the plan, whether or not the plan is potentially applicable to the current problem:

Determining if plan is applicable...

Plan's results meet all of problem's major goals...plan is applicable.

The planner now examines the plan's preconditions in light of the current situation:

Examining preconditions of ⟨BUILD-WOODEN-OBJECT⟩...

There are several preconditions for the plan: the user should have carpentry skill, the user should know how to design the object to be built, and the user should know where to get the materials from which to build the object. CAS knows very little about the user at this point, other than that he is a student and can't spend much money. In order to determine if the preconditions are met, it must ask the user.[1]

"Do you have carpentry skill?"

The user responds negatively.

If "you do not have carpentry skill", then "object to be built will be of poor quality".

Is this acceptable?

Again, the user responds negatively. The planner then tries to modify the plan to eliminate the need for the precondition:

Modifying plan...

The planner uses the first piece of information stored with the precondition to attempt to modify the plan. This is a directive to replace the assembly step with a new step, getting someone else to assemble them for the user:

...trying directive "(:REPLACE ?ASSEMBLE ?USE-AGENT-ASSEMBLE)"...

The new step also has preconditions that must be satisfied:

"Do you know anyone who can perform plan "ASSEMBLE" for you?"

---

[1] The questions to the user are, at the moment and for the purposes of this example, canned text.

```
name: "building wooden objects"
actors: ?a1
props: ?object ?materials
goals: ?acquire
actions: ?design ?get-materials ?assemble
preconditions: ?pc-carpentry-skill ?pc-have-design-knowledge ?pc-know-sources-matl
results: ?have-object

————— Variables:—————

a1: isa ↑ANIMAL
object: isa ↑ARTIFACT
materials: isa ↑SET
    type: ↑POBJ  ;;physical objects
acquire: isa ↑AGOAL  ;;achievement goal [Schank and Abelson, 1977]
        actor: ?a1
        state: isa ↑POSSESSION
               actor: ?a1
               object: ?object
               mode: pos
design: isa ↑INSTANCE
       structure: ↑DESIGN-PLAN
       bindings: ((actor . a1) (object . object))
get-materials: isa ↑INSTANCE
              structure: ↑GET-OBJECTS-PLAN
              bindings: ((actor . a1) (objects . materials))
assemble: isa ↑INSTANCE
          structure: ↑ASSEMBLE
          bindings: ((actor . a1) (object . object)
                     (materials . materials))
pc-carpentry-skill: isa ↑PRECONDITION
                    state: ↑S-CARPENTRY-SKILL
                    reason: ?assemble
                    if-violated: ↑S-DECREASED-QUALITY1
                    fix: (or (:replace ?assemble ?use-agent-assemble)
                             (:add ?learn-carpentry (?assemble before))
                             (:replace ?assemble ?assemble-with-book)
                                          •
                                          •
                                          •
```

Figure 2: A portion of the plan BUILD-WOODEN-OBJECT. Symbols of the form "?name" are variables, referring to other slots in the frame; those of the form ↑name represent pointers to other frames.

If the user does, then the planner performs the replacement:

> **Replacing ASSEMBLE ("assemble object "BOOKSHELF"") with USE-AGENT-ASSEMBLE ("use agent to assemble "BOOKSHELF"").**

There are other preconditions to be satisfied. Assuming that the user knows where to get the materials for building bookshelves, but does not know how to design them, the completed plan would look like that shown in

Figure 2: find a design for bookshelves in a book, get the materials required, then have someone assemble the bookshelves.

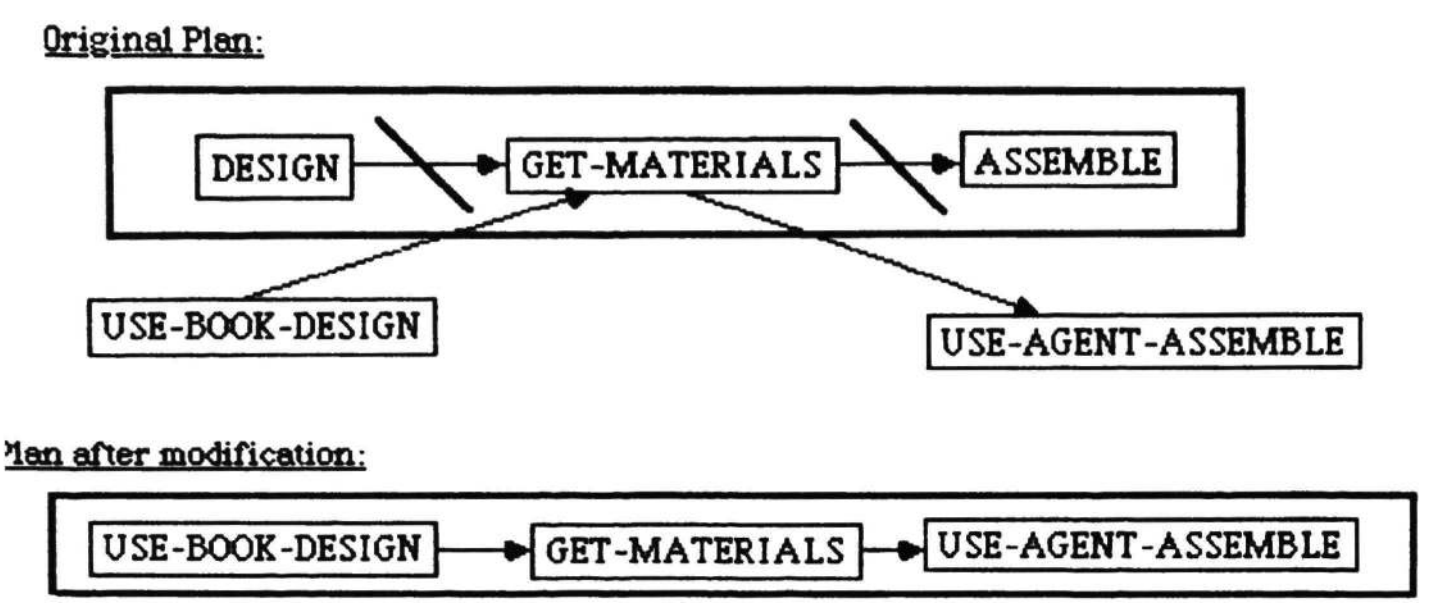

Figure 2: Example of plan modification.

In the remainder of this paper, we will discuss the knowledge CAS uses to produce this behavior and where that knowledge comes from.

3.0 Plan Structure and Origin

Before explaining the plan modification process, we need to explain the structure of plans. Recall that the plans that get modified were derived from experience. That is, small plan steps were put together in the past to successfully solve a problem. As in STRIPS [Fikes *et al*, 1972], those specific plans are generalised to create a plan skeleton.[2] It is these plan skeletons that are modified and instantiated to create a solution to a new problem. The plan skeletons (to be called plans from now on) contain plan steps (arranged hierarchically, since each step can itself be a plan) and plan preconditions. For example, Figure 1 shows a plan for building wooden objects that has three steps ("design", "get-materials", and "assemble") and three preconditions ("pc-have-carpentry-skill", "pc-have-design-knowledge", and "pc-know-sources-matl"). Each step in the plan is itself a plan, with steps and preconditions of its own. Preconditions from the steps are used, in part, to derive the preconditions of the entire plan (see [Turner, in press] for more details).

As a plan gets used several times, information about preconditions gets updated (e.g., the planner may learn that the precondition of having carpentry skill can be ignored if the user of the plan doesn't care about the quality of the object being built) and new, more specific plans are derived. Plans are indexed such that during plan retrieval the most specific applicable plan can be found. In addition, information about the modifications to the plan which yielded new plans is generalised and stored with the old plan's preconditions. This information can be used in the future to modify the plan for new problems.

The following sections discuss the content and use of the information associated with plan preconditions.

[2] Explaining this process is beyond the scope of this paper. See [Turner, in press] for more details.

4.0 Plan Modification Directives

When a planner decides to modify a plan, it must determine the best way make the plan fit the current problem. According to our model, knowledge associated with preconditions of the plan is used to modify it for the new situation.

CAS' preconditions have information, called *directives*, associated with them that provide suggestions about how to modify the plan. For example, the precondition PC-CARPENTRY-SKILL of the plan BUILD-WOODEN-OBJECT (Figure 1) contains three heuristics: (1) replace the step "assemble" with a plan to use an agent (i.e., a plan to have someone else assemble the object); (2) add a plan to learn carpentry before the step to assemble the object; and (3) replace the assembly step with a plan to assemble the object using information from a book. The slot "fix" holds this information.[3] When a particular precondition is violated, "fix" heuristics associated with it are used by the planner as it modifies the plan.

Preconditions, when violated, can require strategic changes to a plan, i.e., changes that affect the plan's overall strategy. Such changes result in a plan that no longer requires the violated precondition, or one in which the precondition is achieved as part of the plan. An example of this can be seen in Figure 1. The precondition PC-CARPENTRY-SKILL calls for (via one of its directives) adding a step to learn some carpentry skills before applying the existing "assemble" step of the plan. Since the precondition has as its source the "assemble" step, it will be not be violated by the new plan (i.e., the user will have carpentry skill by the time he applies the "assemble" step)—indeed, it will no longer be needed as a precondition of the new plan and can be discarded. Violated preconditions can also require only tactical, or local, changes to a plan, usually replacement or modication of one step in the plan. The precondition PC-CARPENTRY-SKILL also contains an example of this: one of its directives calls for the replacement of the "assemble" step by a new plan that will use another agent to assemble the object.

We have identified three types of strategic directives and two types of tactical directives for changing a plan.

4.1 Strategic Plan Modification

A *strategic* change to a plan is one that affects the overall strategy used by the plan. Examples of this are adding a step, deleting a step, or re-ordering steps.

Adding a step to a plan. Suppose the plan BUILD-WOODEN-OBJECT is to be used by someone who doesn't know anything about carpentry; a precondition is violated. One thing that can be done is to have the user learn carpentry, then run the plan. This is not a very good idea, however, since learning all of carpentry can take a long time; the user only needs a small subset of that information to run the plan. A better approach is to add a step to the plan to learn what is needed just before the information is actually needed; that is, just before the "assemble" step of the plan. Suppose the planner decides, based on recursive planning, that this is a good modification to the plan. This information can be added to the precondition in the form of the following directive:

(:add *new-step side old-step*)

which directs the planner to add a new step described by *new-step* to the plan before or after (depending on the value of *side*) the old plan step described by *old-step*. When solving a later problem in which it was reminded of this plan and the same precondition was violated, the planner would not have to modify the plan from scratch, but rather could use this directive to quickly patch the plan for the new situation.

Deleting a step from the plan. Another strategic directive specifies that in order to eliminate the need for a precondition, a step should be deleted from the plan. This is specified by the following:

(:delete *old-step*)

where *old-step* describes a step currently called for by the plan. An example of a case in which this would be useful is the following. Suppose a plan for making chili con carne has a precondition that none of the people who will eat the chili are vegetarians. To modify the plan in response to a violation of this precondition, the step of adding meat can be eliminated from the plan. This can be represented by a :delete directive.

---

[3] This information can, in principle, come from two experiential sources (though so far in CAS it is built in). The first source, mentioned above, is previous modifications of the plan. The second source is observations of others using the plan. A program that could watch a person and learn from its observations (or that could be told about a person's behavior) could use an episode of a person modifying a plan in response to a precondition violation to learn information to add to the plan.

Re-ordering steps in a plan. The third type of strategic directive calls for changing the order of steps in a plan. An example of when this is appropriate is the following. Suppose, for example, that instead of having one step for designing the object to be built, our plan for building wooden objects had two steps: choose the wood and then select the configuration. It makes sense for these steps to be performed in this order if aesthetics are more important than the function of the object, since the choice of the wood constrains the configuration (e.g., thin cherry would need a great deal of structural support). However, if function is more important than aesthetics, the steps should be run in the reverse order: choose an optimal design, then select a wood that can be used in the design. One way these alternatives can be represented in a single plan is to have a precondition whose criteria for satisfaction is: "aesthetic considerations are more important than functional considerations". Associated with this precondition would be a directive to reverse the order of the two steps should the condition be violated.

4.2 Tactical Plan Modification

Tactical plan modification involves changing a single step in a plan without affecting the overall strategy of the plan. Examples of this are replacing one step with another step (that doesn't impact the remaining steps in the plan), and changing a step within a step of the plan.

Replacing a step. We saw an example of this type of tactical plan modification in our example above, when the planner substituted the plan of getting someone else to assemble the bookshelves for the step of having the user assemble them himself. The directive to specify step replacement looks like:

(:replace *old-step new-step*)

where *old-step* is a description of the step to be replaced, and *new-step* describes a plan or an action to use as the replacement. Although replacement could be accomplished by doing an addition and then a deletion, we use a single directive to specify the change. This simplifies the process of modifying the plan (i.e., only one directive, not two, needs to be followed), and is conceptually much clearer.

Modifying a step. Preconditions of a plan can also direct an internal change to be made to one of the plan's steps. For example, suppose the "get-materials" step in the plan for building a wooden object has a step of driving from store to store. If the user doesn't have a car, this step must be changed to substitute an alternate form of transportation, but the rest of the step, and the other steps in the plan, should be left the same. The form of a directive to specify this would be:

(:modify *step directive*)

where *step* is the step to be modified and *directive* is a directive, either strategic or tactical, decribing the modification to be made.

4.3 Specifying More than One Change

A violated precondition can specify more than one change to be made to a plan in order to eliminate the violation. This is done by directives that are boolean ("and" and "or" only) combinations of other directives. For example, a directive to either replace the assembly step with a plan for getting a neighbor to help or adding a step to learn carpentry would look like:

```
(:or
    (:replace ⟨description of assembly step⟩
              ⟨description of the replacement⟩)
    (:add ⟨description of learning carpentry⟩
          :before
          ⟨description of assembly step⟩)
)
```

An example of a compound directive can be seen in Figure 1.

5.0 Absolute and Flexible Preconditions

Researchers working on problem solving have traditionally treated preconditions as conditions that must be met before using a plan or doing a plan step. Some conditions on a plan or plan step, however, affect the degree of success of the plan rather than its absolute success. Lack of carpentry skill, for example, will preclude the complete success of the BUILD-WOODEN-OBJECT plan, but will not prevent it being applied with limited success—the quality of the object built will not be as good. We distinguish between preconditions (called *absolute preconditions*) that must be met and those (called *flexible preconditions*) that need to be met only for perfect results. CAS uses each differently during plan modification.

Flexible preconditions [4] have information associated with them that predicts the likely outcome should they be violated and the plan applied anyway. An example of a flexible precondition can be seen in Figure 1. PC-CARPENTRY-SKILL states that the user should have carpentry skill in order to apply the plan. If he doesn't, the plan can still be applied, but the result will not be as nice. Flexible preconditions allow the planner to make judgements about whether to modify a plan to fit the current problem or to ignore the violated precondition and apply the plan "as is". Absolute preconditions, on the other hand, have no such information, and no such decisions need be made about them. If they are violated, the plan simply cannot be used as is. Using it requires modifications that either make the precondition unnecessary or achieve it.

CAS recognises a precondition as flexible if it has knowledge associated with it about the results of its violation (held in an "if-violated" slot). This information is used to decide whether or not to ignore the precondition's violation. Knowledge about precondition violation takes the form of a state that will result if the plan is applied and the precondition is violated. The knowledge associated with the precondition PC-CARPENTRY-SKILL (Figure 1), for example, indicates that if the person applying the plan lacks carpentry skill, the quality of the object built will be poor.

The information in the "if-violated" slot, like all precondition information, can be derived from past experience using the plan, though so far in CAS we have built it into our plans. When a plan is first formulated, its preconditions are derived from the preconditions of its steps and initially have no information indicating the effect of violating them. As the plan is used and its preconditions are violated, however, information is added to the preconditions describing the result of their violation. This information can come from simulating the plan, or it can come from the planner deciding to ignore the precondition, then recording the effects.

Although not yet completely implemented in CAS, flexible preconditions serve several purposes. Information in the "if-violated" slot of a precondition can be used in combination with a user model to decide when to attempt to modify a precondition and when to ignore a violation entirely. This will allow CAS to predict the outcome of applying a faulty plan without needing to simulate it. Knowledge associated with flexible preconditions also allow a planner to select and apply the best plan from a set of less than optimal plans when that is all that is available; with traditional (absolute) preconditions, no plan could be chosen.

6.0 Using Preconditions in Plan Modification

When CAS' planner is presented with a plan, it examines the expected results and compares them to the goals of the current problem. If the results satisfy all of the major goals in the problem, then the plan is deemed applicable, and the preconditions of the plan are checked to determine if there are any violations.

If a flexible precondition is violated, the planner tries to determine if the violation can be ignored. At the moment, the way it does this is to ask the user if the result of violating the precondition is acceptable. Another way to do this is to use a model of the user to help it decide whether or not a result is acceptable. If the result of ignoring the violation is acceptable, then the planner considers the next precondition; otherwise, it attempts to modify the plan, using any directives associated with the precondition. If an absolute precondition is violated, the planner assumes that the plan cannot be run if the violation occurs and immediately attempts to modify the plan.

Once the planner decides to modify the plan because of a precondition violation, it examines the directives stored with the precondition. The directives are not currently stored in any particular order. If there is only one directive, it is applied. If there are more than one, they are tried in order until one is found that is acceptable. A directive is acceptable if the change it suggests is possible and acceptable to the user.

---

[4] Called "relative preconditions" previously [Turner, 1986].

7.0 Related Work

Re-using plans is not a new idea; for example, STRIPS [Fikes *et al*, 1972] stored plans it created in structures called MACROPS for later use. Nor is the idea of plan modification itself new. MACROPS were modified by variable substitution—a process Carbonell [1983] calls "transformational analogy"—and pieces of the MACROP could be used if the entire thing was not needed. This type of modification is rather trivial and not very flexible. For instance, if a MACROP or piece of a MACROP was selected to solve a new problem, the structure was applied "as is",[5] with no changes to the structure allowed.

Carbonell [1986] proposed another form of reasoning by analogy to a previous problem, called *derivational analogy*. This work is closely related to our own, as we both rely on previous reasoning to guide plan modification. Yet there are important differences. Carbonell's approach involves stepping through the reasoning that was done in creating a plan for a previous situation, noticing problems with respect to the current problem, and fixing them. Our approach, however, relies on compiled information in the preconditions to immediately alert the planner to potential difficulty applying the plan. The planner can then evaluate the severity of the problem—again using information stored with the preconditions—and modify the plan. This is a more efficient approach, since we are not simulating planning that was done previously. Another difference between our planner and Carbonell's is that ours is hierarchical, and his is not.

Hammond [1984] proposes using information stored with *thematic organization packets*, or TOPs [Schank, 1982], to patch faulty plans. TOPs represent planning information at a high level of abstraction. For example, a TOP might suggest that in order to avoid a violation of a goal by some plan step, that plan step should be replaced or changed; however, additional reasoning would need to be done to decide how to replace or change the step. We use TOPs in CAS, too, but have found that it is advantageous to store information about modifying a specific plan with the plan itself. This allows the specific information, i.e., that which pertains specifically to the plan, to be accessible immediately upon recalling the plan, without performing any additional reasoning.

Alterman [1986] has developed an adaptive planner that can use both specific and general plans to solve a new problem. If the specific plan fails, then the failing step is generalized to find a representative category of action, then that is specialized to find a new step. Each time a step fails in a plan, this must be done. The difference between this approach and our own is that we avoid the generalization and specialization procedure by storing compiled knowledge about previous plan modifications with the preconditions. Instead of looking for an alternative step, CAS can immediately use past experience to substitute a step that worked before.

Schank and Abelson [1977] have divided the preconditions of their *planboxes* into three types: *controllable*, *uncontrollable*, and *mediating*. Controllable preconditions are those satisfiable by operators a planner knows about. Uncontrollable preconditions are those the planner doesn't know how to satisfy. Mediating preconditions are those the planner can satisfy by using other planboxes from the *persuade package*. This was a good start at categorizing preconditions by how they can be satisfied. However, the problem with this approach is that it is too static for use by a planner that learns plans over time. Preconditions that are uncontrollable now may be controllable at some future time, and preconditions that are satisfied by plans similar to those in the persuade package may at some later time be satisfiable by other plans. Their approach also was not concerned with plan modification.

8.0 Conclusion

Re-using plans often involves modifying them to fit a new situation. Plan modification is made easier by using information gained from past experience using the plan. Since this information usually relates to how the plan was modified in response to one or more precondition violations, CAS takes the approach of storing the information with the preconditions themselves. The planner can easily access this information when it needs it during the process of modifying the plan.

There are two important features of our approach. First, some preconditions are *flexible preconditions*. These contain information about the likely result of applying the plan if they are violated. This information allows a planner to quickly decide whether it should attempt to modify the plan or simply ignore the offending precondition. Since relative preconditions do not necessarily have to be satisfied in order for a plan to be applied,

---

[5] Unless there was a precondition violation, in which case the structure was applied after additional planning to meet the preconditions.

they also have the advantage, in principle, of allowing a planner to choose the best plan from among several plans with violated preconditions, if that is all that is available.

The second feature of our approach is that preconditions contain information that can be used by the planner as heuristics during plan modification. This information is compiled from experience using the plan, and takes the form of planning directives—suggestions to the planner based on what has worked before. By making use of these directives, the planner can use previous experience to allow it to efficiently modify a plan.

Bibliography

Alterman, R. (1986). "An Adaptive Planner," in *Proceedings of the Fifth National Conference on Artificial Intelligence* (AAAI-86), Morgan Kaufmann Publishers, Inc., Los Altos, California, pp. 65–69.

Carbonell, J.G. (1983). "Learning by Analogy: Formulating and Generalizing Plans from Past Experience," in *Machine Learning: An Artificial Intelligence Approach*, eds. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Tioga Publishing Company, Palo Alto, California.

Carbonell, J.G. (1986). "Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition," in *Machine Learning: An Artificial Intelligence Approach, Volume II*, ed. Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, Morgan Kaufman Publishers, Inc., Los Altos, California.

Fikes, R.E., Hart, P.E., and Nilsson, N.J. (1972). "Learning and Executing Generalized Robot Plans," *Artificial Intelligence* **3** pp. 251-288.

Hammond, C. (1984). "Indexing and Causality: The organization of plans and strategies in memory," Yale Technical Report YALEU/CSD/RR #351.

Kolodner, J.L. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey.

Kolodner, J.L., and Cullingford, R.E. (1986). "Towards a Memory Architecture that Supports Reminding," *Proceedings of the Eighth Annual Conference on Cognitive Science*, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey.

Schank, R.C. (1982). *Dynamic Memory*, Cambridge University Press, New York.

Schank, R.C., and Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Turner, R.M. (1986). "A Derivational Approach to Plan Refinement for Advice-Giving," *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 858-862.

Turner, R.M. (in press). *Issues in the Design of Advisory Systems: the Consumer-Advisor System*, Technical Report #GIT-ICS-87/??, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.