STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

January 1976

KNOWLEDGE REPRESENTATION IN AUTOMATIC PLANNING SYSTEMS

by

Richard E. Fikes
Artificial Intelligence Center
Stanford Research Institute
Menlo Park, California 94025

Technical Note 119

The research reported in this paper was sponsored by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC04-75-C-0005 with the U.S. Army Research Office and supported by the National Aeronautics and Space Administration under Contract NASW-2086.

ABSTRACT

This paper is a tutorial on automatic planning systems with particular emphasis given to knowledge representation issues. The effectiveness of a planner depends, to a large extent, on its ability to make use of descriptions and expertise associated with particular task domains. Examples of such domain knowledge include action models, state description models, scenarios, and special purpose plan composition methods. Our discussion focuses on how such knowledge is represented in planning systems and on how various planning strategies make use of it.

I INTRODUCTION

A planning system is one that assists in the accomplishment of tasks by generating plans and then monitoring their execution. The active agent that actually carries out the plans might be another part of the system. For example, a mechanical manipulator or a human to which the system is providing instructions. A task is usually specified to such a system by describing an initial situation and a desired (goal) situation. The system is aware of a collection of actions that the active agent can perform and the plan it produces is a sequence of these actions that are expected to transform the initial situation into the desired situation. Example domains for which automatic planning systems have been designed include the directing of a mobile robot to do tasks that involve pushing boxes around in several rooms (Fikes and Nilsson, 1971),* a robot arm building towers of multiple-shaped blocks (Fahlman, 1974), and humans who are repairing electro mechanical equipment (Sacerdoti, 1975).

Taking a broader view, one can think of a planner as being a constructor of scenarios for whatever reasons they are needed. For example, Minsky (1974), Schank (1975), and others have described the use of stored "action frames" or "scripts" as guides to a system attempting to understand a story being input in natural language. The stored scenario indicates what typically happens in the situation being described in the story (such as dining at a restaurant or attending a child's birthday party). It aids the system by

*References are appended.

providing a set of expectations about the sequence of events and the relationships among the characters in the story. To make effective use of stored scenarios, a system needs the capability of modifying and combining them to form new scenarios that are appropriate for understanding any particular story; that is, the system needs a planning capability.

In this paper I will examine some of the characteristics of automatic planning systems by focusing on the knowledge representation issues that arise in their design. My intention is not to survey the entire field of automatic planning, but to provide a feeling for the general structure of such systems and for some of the techniques that have been and are being used in their design.

II STATE DESCRIPTION MODELS

Typically, a planner proceeds by hypothesizing sequences of actions for inclusion in a plan and testing each hypothesis by simulating the action sequence. The simulation produces a description of the situation that would be expected from the actions, and that situation is examined to determine if it satisfies the subgoals that the planner is trying to achieve. This process requires facilities for creating, updating, and querying state description models of the task domain (Fikes, 1975).

Most planning systems use state description models that can be characterized as collections of relational expressions, with each expression having a truth value associated with it. The expressions are typically represented as list structures with the first element of each list being a relation name -- or example, "(INROOM ROBOT R1)" or "(STATUS (CONNECTION PUMP PLATFORM) FASTENED)." The models provide the data base needed to determine for any given relationship in any given state whether the relationship is known true, known false, or unknown in that state.

Some form of retrieval mechanism is needed for this data base so that the planner can obtain information about these models. Retrieval functions rely on a basic access mechanism that in most planning systems associatively retrieves all relational expressions from the data base that match a given pattern (where a pattern is an expression containing unbound variables) and then examines the truth value of each retrieved expression with respect to the given state. For example, a retrieval using the pattern "(INROOM ROBOT ←X)" would be used to determine what room the robot is in.

Known truth values are usually not all explicitly stored in a model. Hence, a planner needs deductive machinery for querying state description models. Most planners are designed so that a user can provide derivation functions that compute implicit truth values when they are needed. These functions may embody formal theorem-proving strategies or may simply be statements of implicational rules derived from the semantics of the task domain. They extend each model in the sense that, from the calling program's point of view, the derived instances of a pattern can be made to be indistinguishable from instances actually found in the model.

III ACTION MODELS

Typically, a planner has models available of the actions that can occur as steps in a plan. For example, the actions available to a mobile robot might include "Go to location (x,y)," "Turn X degrees right," and so forth. For these models to be useful to a planner, they must include certain descriptive information about the actions. In particular, the planner needs to know under what conditions an action can be carried out. These are the action's "preconditions" or "applicability conditions." They provide a source of subgoals during the planning process and allow the planner to ensure that an executable plan is being produced. In addition, the planner needs a description of the transformations that execution of an action is expected to make. This information allows the planner to simulate hypothetical action sequences and thereby investigate their results. Finally, the planner needs to know what kinds of tasks or subgoals each action is useful for achieving. This information indicates to the planner which actions to consider putting into the plan being constructed.

The action models that are available to a planner as potential plan steps usually represent an entire class of actions by containing slots that must be filled by specific values. For example, a robot action for moving a box might include slots for the name of the box, the starting location of the move, and the final location of the move; hence, the action can be used to move any box from any location to any other location. An action model will specify constraints on the values that can fill its slots, such as type requirements, and for the box example, perhaps a restriction that the two locations be in the same room. A significant part of the planning activity is to determine for these slots the values that satisfy the model's constraints and that produce an instance of the model useful in the new plan. Hence, the planning process entails both selecting and "instantiating" a sequence of action models that will achieve the desired goal.

A. Assertional Action Models

Action models are often represented as data structures containing lists of parameters, preconditions, and effects. The parameters are the "slots" that the planner must instantiate when using the model, and they serve as the model's local variables. The preconditions are an explicit list of the relationships that must be true in a state before the action can be executed. The effects indicate the state changes that the action is expected to make. Each element of the effects list indicates a relationship and the truth value that the relationship is to have in the state produced by the action; all other relationships are assumed to remain unaffected by the action. For example, the model for a "Go to location (x,y)" robot action might have parameter list (x,y), preconditions that require location (x,y) to be in the same room as the robot, and an effects list that denies the old location of the robot, deletes (i.e., assigns truth value "unknown") the old orientation of the robot, and asserts (x,y) as the new location of the robot.

This action model representation lends itself to a means-ends analysis method for composing plans, as follows. A goal state is described by a collection of relationships that must be true in the desired state. The planner proceeds by checking to see which of the goal relationships are not yet true in the current state. When such a relationship is found, a scan is made of the action models to find one whose effects list indicates that some instance of it will assert the goal relationship. When such an action is found, it is taken to be "relevant" to achieving the goal, and the planner attempts to include it in the plan. An action can be included in a plan only if a state exists in which its preconditions are true. If they are true in the current state, the action is simulated to produce a new current state, and the goal is reconsidered as before. If the action's preconditions are not true in the current state, the achievement of those preconditions becomes for the planner a new goal to which the same method can be applied. Hence, the basic algorithm entails matching a goal to an action capable of achieving some portion of it and then attempting to make that action applicable.

When more than one relevant action is found for any particular goal, or when a goal is a disjunction, a "choice point" occurs in the algorithm. The occurrence of such choice points requires the planner to impose a search executive on top of the basic algorithm to decide at each step which subgoal to consider next.

B. Procedural Action Models

The planning algorithm described above is essentially the one used in the STRIPS system developed in 1969. One of the major ways in which the effectiveness of planning systems has been improved since then is by augmenting this basic algorithm with facilities that accept planning expertise specific to the particular domain in which tasks are to be given. That is, in any given domain there will be planning strategies and heuristics that a planner designed specifically for that domain would employ -- for example, route finding algorithms, or tower building strategies. To a large extent, the effectiveness of a planning system can be measured in terms of its ability to accept and use such expertise.

One way in which this capability has been achieved is by representing action models in a procedural form rather than as a data structure (for example, see Derksen, Rulifson, and Waldinger, 1972).

Such a procedure has the action's parameters as its input parameters and begins by querying the current state description model to determine if the action's preconditions are true in that state; if they are not, it calls the planner's executive to create a plan to achieve the preconditions. When the preconditions have been achieved, the action model proceeds to simulate the execution of the action by creating a new state description model and making the appropriate assertions, deletions, and denials in it. Finally, the procedure adds the action that it represents to the end of the plan being constructed.

As with assertional action models, a planner using procedural action models must be able to deal with choice points that occur when multiple actions or multiple instantiations of actions are relevant. This requirement implies the need for some form of co-routine or backtracking facilities that allow alternative program control paths to be considered in a pseudo parallel manner under the direction of the planner's executive.

A procedural language provides more expressive power for representing action models than do the data structures we described above. For example, a procedure can describe effects of an action that are conditional on properties of the state in which the action is executed. The most important aspect of this expressive power in terms of the effectiveness of the planning system is that procedural action models can contain strategy and heuristic information about how to achieve the action's preconditions. For example, if an actions's preconditions are a set of relationships, all of which must be achieved (that is, a conjunction), the action model can indicate the order in which they should be achieved by making a sequence of calls to the planning executive with each call indicating the next relationship to be achieved. In effect, the action model procedure can contain a "subplanner" specifically designed to guide the production of plans that achieve the action's preconditions.

Procedural action models suffer from the standard difficulty that their "knowledge" is hidden from the system in the code. That is, the system cannot reason about the preconditions or effects of an action if all that is available is the procedure. One can deal with this problem by providing declarative information about the action in addition to the procedural action model. For example, the planner needs to know what relationships an action is useful for achieving so that it can determine when to consider including the action in a plan. Hence, a procedural action model needs essentially to have a list of "primary effects" associated with it. This declarative information will be an abstraction or an approximation of the effects that the procedure will produce in any given situation, and can be thought of as a model of the procedure that itself is a model of a class of actions.

IV PROCEDURAL SUBPLANNERS

Once a planning system has the facilities for using procedural action models, an obvious and powerful generalization becomes possible. A procedural action model can be considered to be a "subplanner" specifically designed to produce plans for achieving the primary effects of an action. We can generalize this concept by removing the restriction that such subplanners must be models of an action, and can consider writing subplanners for any specific class of tasks. For example, in a robot domain, a route finding algorithm could be written that would determine a sequence of rooms for the robot to travel through to get from one room into another. The route finder could employ any appropriate search algorithm for determining a suitable route, making use of geometric information, previously computed routes, and so forth. Such an algorithm could be represented as a subplanner designed to make the relationship "(INROOM ROBOT ←X)" true.

Often the primary role of a subplanner is to decompose a task by producing a sequence of subgoals to be achieved. The subplanner produces the actual plan by calling the planning executive with each of the subgoals in turn. For example, the route finder may determine only a sequence of rooms through which the robot is to travel. It would call the planning executive to determine the detailed plan for the movements from room to adjacent room for each step along the route.

Writing subplanners is an easy and natural way of including domain-specific plan generation knowledge in a general-purpose planning system, and it provides the system with the power that is needed to make such a system effective and useful.

V HIERARCHICAL PLANNING

The robot route finding example suggests another important theme in automatic planning systems -- namely, "hierarchical planning." We have been assuming that the planner is given a single set of actions and that all plans must be composed of elements from that set. Hence, if the actions for the robot domain are at the level of detail of "Move forward X meters" and "Turn right Y degrees" a route consisting of the sequence of rooms that the robot is to go through is not an acceptable plan. It is a "meta-plan" that can be made into a plan by determining the detailed moves and turns that will take the robot from each room to the next. The question arises as to when and if the planner should be concerned with details such as determining the actual moves and turns for the route. One might argue that it is just as easy to include a "Go to adjacent room X" action in the action set or, for that matter, a "Go to room X" action that would incorporate the route finding algorithm.

How does one decide what level of detail to include in a plan? We can gain insight into that question by considering the purposes for creating a plan. The primary purpose of planning is to determine a suitable method for accomplishing a task and to predict the consequences of applying that method. Finding a suitable method typically means achieving some acceptable degree of certainty that the method can be applied, that it will succeed in accomplishing the task, and that it is less expensive than other feasible methods with respect to some resource such as time or energy. The planner's predictive capability allows the system to answer questions about applying the method without having to expend the resources and deal with the consequences of the actual execution. Planning is necessary at a particular level of detail, then, only to the extent that some question exists as to what method to apply at that level or that information is needed about the execution of the method before the execution occurs.

If we can write a "Go to room X" action program that can provide the necessary degree of certainty that a route exists and can provide a sufficiently accurate estimate of the cost of moving the robot through the route, there may be no need to plan the route and the individual moves until it is time to execute the plan. On the other hand, if there is uncertainty about the route, such as whether doors are locked or whether clear pathways exist through rooms, it may be necessary to plan the details of the route to remove the uncertainty.

Even if a "meta-action" such as "Go to room X" cannot answer all the questions that arise before plan execution, there are often advantages to delaying the planning of the meta-action's details until it is certain that such details will be included in the final plan. For example, there is no need to plan the details of how to get the robot into some particular room until it is clear that the trip to that room will be part of the plan.

When plans are being constructed for humans to carry out, such as for the SRI Computer Based Consultant system (CBC) (Hart, 1975), which assists in the repair of equipment, the use of meta-actions becomes even more valuable. In general, the people receiving instructions from the system will have varying levels of expertise and understanding of the steps in the plan. The system can attempt to match its level of instruction to this variability. For example, the CBC might tell a trained mechanic simply to "Replace the pump," whereas it might tell a novice to "Remove the 4 mounting bolts at the base of the pump using a 3/8-inch open-end wrench." If the person does not understand an instruction, the system can call the meta-action's subplanner and respond by providing the more detailed instructions that the subplanner produces.

For a planner to include meta-actions in plans, it must be provided with "meta-action models" containing the same information and having the same form as the action models. That is, the planner must be able to achieve the preconditions of the meta-action, simulate its execution, and match it to tasks for which it is useful. Given such a model, a planner has the option of including a meta-action in a plan without calling its subplanner to determine the detailed plan steps that it represents.

This approach to planning has been explored by Sacerdoti in two systems (Sacerdoti, 1974, 1975) and found to provide significant advantages during both the generation and the execution of plans. In these systems, plans tend to grow in a top-down "breadth-first" manner, in that typically a complete plan will be constructed using high-level meta-actions before the detailed steps of any of the meta-actions are determined. Since a meta-action's subplanner may itself produce a plan containing meta-actions, a multiple-level plan hierarchy is formed, and the planner can make independent decisions for each meta-action relative to whether its subplanner should be called.

VI ASSERTIONAL DOMAIN KNOWLEDGE

Another form in which domain-specific knowledge can be represented in a planning system is scenarios, or generalized plans. That is, any given domain will typically include many methods for accomplishing tasks for which the order and the identity of the steps do not vary for different situations. Such methods provide a planner with the skeleton for an entire subplan and the task of composing plans using such scenarios is largely one of finding the appropriate scenario and then producing a suitable instantiation of it.

A. Scenarios

We have found explicit scenarios useful in a management support system currently being developed at SRI (Fikes and Pease, 1975). In this work, we wish to allow a manager to specify to the system various operational procedures that he uses in his organization. The system can then act as an administrative assistant by planning and monitoring the execution of these procedures at the manager's request. The planning activity in this system is primarily one of instantiation. However, the instantiation process in this case is nontrivial because it entails scheduling the individual steps of the plan and ensuring the availability of resources and personnel at specific times.

The example domain being considered for this system is that of managing the logistics of operations on board a Navy aircraft carrier. One operation of interest in that domain is the flying of training missions. Such missions entail steps such as preflight and postflight maintenance of the aircraft, fueling the aircraft, briefing and debriefing the pilot and launching and recovering the aircraft. These steps and the order in which they must occur do not vary from mission to mission; hence they can be included in a scenario that is part of the definition of a mission.

A scenario in this system has many "terminals" with unassigned values, and the planner's basic task is to find an acceptable set of values for them. Most of these slots specify the start and the end time of some step in the plan or the quantity and identity of some resource that will be used by the plan.

We include at each terminal of a scenario a set of constraints on the value to be assigned there. The most common constraints are those that are derived from the temporal partial ordering of the steps; (for example, fueling of the aircraft must occur after the preflight maintenance and before the launch). Default values are included at the terminals, usually expressed as functions of other slot values in the scenario. For example, the default value for the start of preflight maintenance is expressed as the time of launch minus some constant. These default values allow the planner to make feasibility estimates and to make scheduling decisions before all of the constraints from the other schedulers have been determined.

Also included at each terminal is a specification of what function can be called to determine a value for the terminal. These are typically calls on planners and schedulers that are responsible for other operational functions in the organization. On board a carrier these would be aircraft maintenance, pilot assignment, aircraft assignment, the flight deck, and so forth. Hence, the planner that is using such a scenario engages in a negotiation dialogue with the other planners in an attempt to reach agreement on an acceptable schedule.

B. Generalized Plans

Another source of assertional domain knowledge is from previously produced plans. One would like the plans produced by a planner to be retained as new scenarios so that the system could be considered to be acquiring new expertise in the task domain.

In some of our earlier work with the STRIPS planning system, we attempted to develop such a learning mechanism (Fikes, Hart, and Nilsson, 1972). The challenge in such an effort is to have the system derive and store with the new scenario the information required for its use by other parts of the system, including the planner. This information includes all the information that is provided by action models -- namely, a set of preconditions for the plan, a description of the effects of the plan's execution and indications of the classes of tasks the plan is relevant to achieving. In addition, for the new scenario to be useful in situations other than just reoccurrences of the one for which it was originally constructed, it must somehow be generalized so that, like the other scenarios, there are slots at the scenario's "terminals" that can be filled with values suited to any of a class of situations and goals.

The basic goal of the STRIPS learning work was to generalize and save plans so that they could be used as single steps in future plans. Two aspects of this work are relevant to this discussion: the use of "kernels" and the manner in which plans were generalized.

In the STRIPS system, a stored plan was called a MACROP (for MACRo OPerator). A MACROP's preconditions were directly available as a side effect of the determination of kernels preceding each step of the plan. The kernel preceding Step i of a plan is a partial description of the situation that the planner expects to exist after execution of the first i-1 steps of the plan. In particular, it specifies a set of relationships that must be true if the remainder of the plan is to succeed. Hence, in effect, the kernel for each step is the set of preconditions for the remainder of the plan. The kernels for a plan can be computed in a direct manner by considering what relationships were made true by each step in the plan and by knowing what relationships from the state description models were needed to ensure the achievement of each action's preconditions and of the final goal.

A plan's kernels provide valuable information for many uses of the plan and therefore are an important part of the information stored with the plan. First, the kernel preceding the initial step of the plan specifies the preconditions for the plan. Second, the kernels specify the tests an execution monitor should make after each action to determine whether the plan is proceeding as expected during execution. Third, the kernels are useful for modifying or extending the plan to accommodate a new situation or new task goals.

That is, instead of forming a completely new plan to accomplish some task goal, the planner can be given the option of modifying an existing scenario by adding or changing some of its steps. The basic information that is needed to allow such modification is contained in the kernels, since each kernel indicates what must remain true between any two adjacent steps in a plan. Hence, the planner can add any number of steps between Step i-1 and Step i of an existing plan as long as the kernel between those two steps is not violated.

Providing a capability for modifying an existing plan is an important way of improving the power of a planning system. It allows the planner to consider adding steps anywhere in a plan (as apposed to only at the end) to achieve a goal (Waldinger, 1975). For example, if a plan has been constructed to move a robot from Room A into Room B, and then the planner considers a goal of transporting some object from Room A into Room B, the existing plan can be modified so that the robot transports the object during the first trip rather than making a second trip.

This modification capability is also useful when combining two scenarios to achieve a conjunctive goal (Sacerdoti, 1975). For example, in the equipment repair domain, two scenarios may be conjoined each of which includes removal and replacement of the same subassembly. The planner must be able to recognize this redundancy and to combine the scenarios so that the subassembly is removed only once. Finally, a plan modification capability is useful for "debugging" a plan (Sussman, 1973). Often, stored scenarios and the methods found in subplanners are heuristic in nature and do not always produce bug-free plans. Such bugs are typically discovered during the planner's simulation of the plan. When such a discovery is made, the planner can employ its plan modification facilities to remove the bug.

The second comment about the STRIPS learning work relates to the generalization that was done on the plans before they were stored. The goal was to "unbind" the slots in each of the plan's steps so that, whenever possible, they became unvalued slots in the scenario for the entire plan. The logical structure of the plan imposes restrictions on this unbinding process, so that some pairs of slots are required to take on the same value (they become a single slot in the new scenario) and others must retain their binding (they lose their status as slots in the new scenario). For example, if one step of the plan causes the robot to go to a door and the next step causes the robot to go through a door, the door in those two steps must be the same door.

This generalization process essentially parameterizes the plan and thereby provides it with enough generality to make it worthwhile to save. For example, consider a plan that takes a robot from Room R1 through Door D1 into Room R2 and then has the robot bring Box B1 from Room R2 back into Room R1 through Door D1. That plan would be generalized so that it would take the robot from any room into any adjacent room through any connecting door and then take any box in the room into any adjacent room through any connecting door. The generalization removes all bindings to the particular rooms, doors, and box and allows the room into which the box is taken to be different from the one in which the robot was initially located.

We see, then, that another way of expanding the power and effectiveness of a planning system is to provide it with facilities for instantiating and modifying plans that have been previously produced, generalized and stored.

VII SUMMARY

A planner is needed when a system does not have a prestored method for accomplishing a particular task. The planner's role is to combine the methods that are available to the system in order to produce a new "method" that will accomplish the task. The methods being composed may consist of single actions or of multiaction scenarios and the composition process may include instantiating and modifying the existing methods to match the particular situation. In addition to basic means-ends analysis and simulation facilities for composing prestored actions and scenarios, a planner typically has available a collection of subplanners that embody domain-dependent expertise and are designed to be applied to specialized classes of tasks.

REFERENCES

Derksen, J., J. F. Rulifson, and R. J. Waldinger, "The QA4 Language Applied to Robot Planning," Proc. Fall Joint Computer Conference, pp. 1181-1192 (1972).

Fahlman, S. E. , "A Planning System for Robot Construction Tasks," Artificial Intelligence, Vol. 5, No. 1, pp. 1-49 (1974)

Fikes, R. E., "Deductive Retrieval Mechanisms for State Description Models," Proc. Fourth International Joint Conference Artificial Intelligence, Tbilisi, USSR (1975)

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

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

Fikes, R. E., M. C. Pease, "An Interactive Management Support System for Planning, Control, and Analysis," Technical Report 12, Computer Science Group, SRI, Menlo Park, CA (1975).

Hart, P. E., "Progress on a Computer-Based Consultant," Proc. Fourth International Joint Conference Artificial Intelligence, Tbilisi, USSR (1975).

Minsky, M., "A Framework for Representing Knowledge," Memo No. 306, MIT Artificial Intelligence, Cambridge, MA (1974).

Sacerdoti, E. D., "Planning in a Hierarchy of Abstraction Spaces," Artificial Intelligence, Vol. 5, No. 2, pp. 115-135 (1974).

Sacerdoti, E. D., "A Structure for Plans and Behavior," Technical Report 109, Artificial Intelligence Center, SRI, Menlo Park, CA (1975).

Schank, R. C., "Using Knowledge to Understand," Proc. Theoretical Issues in Natural Language Processing Workshop, MIT, Cambridge, MA (1975).

Sussman, G. J., "A Computational Model of Skill Acquisition," Tech. Report AI-TR-297, MIT Artificial Intelligence Laboratory, Cambridge, MA (1973).

Waldinger, R., "Achieving Several Goals Simultaneously," Tech. Note 107, Artificial Intelligence Center, SRI, Menlo Park, CA (1975).