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

October 1970

STRIPS: A NEW APPROACH TO THE APPLICATION OF THEOREM PROVING TO PROBLEM SOLVING

by

Nils J. Nilsson
Richard E. Fikes

Artificial Intelligence Group

Technical Note 43

SRI Project 8259

This research is sponsored by the Advanced Research Projects Agency and the National Aeronautics and Space Administration under Contract NAS 12-2221.

I INTRODUCTION

A. Overview of STRIPS

This note describes a new problem-solving program called STRIPS (STanford Research Institute Problem Solver). The program is now being implemented in LISP on a PDP-10 to be used in conjunction with robot research at SRI. Even though the implementation of STRIPS is not yet complete, it seems to us important to discuss some of its planned features so that they can be compared with other on-going work in this area.

STRIPS belongs to the class of problem solvers that search a space of "world models" to find one in which a given goal is achieved. For any world model, we assume there exists a set of applicable operators each of which transforms the world model to some other world model. The task of the problem solver is to find some composition of operators that transforms a given initial world model into one that satisfies some particular goal condition.

This framework for problem solving, discussed at length by Nilsson,[1*] has been central to much of the research in Artificial Intelligence. A wide variety of different kinds of problems can be posed in this framework.[†] Our primary interest here is in the class of

[*] References are listed at the end of this technical note.

[†] It is true that many problems do not require search and that specialized programs can be written to solve them. Our view is that these special programs belong to the class of available operators and that a search-based approach can be used to discover how these and other operators can be chained together to solve even more difficult problems.

problems faced by a robot in rearranging objects and in navigating. The robot problems we have in mind are of the sort that require quite complex and general world models compared to those needed in the solution of puzzles and games. Usually in puzzles and games, a simple matrix or list structure is adequate to represent a state of the problem. The world model for a robot problem solver, however, needs to include a large number of facts and relations dealing with the position of the robot and the positions and attributes of various objects, open spaces, and boundaries.

Thus, the first question facing the designer of a robot problem solver is how to represent the world model. A convenient answer is to let the world model take the form of statements in some sort of general logical formalism. For STRIPS we have chosen the first-order, predicate calculus mainly because of the existence of computer programs for finding proofs in this system. Initially, STRIPS will use the QA3 theorem-proving system[2] as its primary deductive mechanism.

Goals (and subgoals) for STRIPS will be stated as first-order predicate calculus wffs (well formed formulas). For example, the task "push a box to place b" might be stated as the wff $(\exists u)[BOX(u) \wedge AT(u,b)]$, where the predicates have the obvious interpretation. The task of the system is to find a sequence of operators that will produce a world model in which the goal can be shown to be true. The QA3 theorem prover will be used to determine whether or not a wff corresponding to a goal or subgoal is a theorem in a given world model.

Although theorem-proving methods will play an important role in STRIPS, they will not be used as the primary search mechanism. A graph of world models (actually a tree) will be generated by a search process that can best be described as GPS-like (Ernst and Newell[3]). Thus it is fair to say that STRIPS is a combination of GPS and formal theorem-proving methods. This combination allows objects (world models) that can be much more complex and general than any of those used in previously implemented versions of GPS. This use of world models consisting of sets of logical statements causes some special problems that are now the subject of much research in Artificial Intelligence. In the next and following sections we will describe some of these problems and the particular solutions to them that STRIPS employs.

B. The Frame Problem

When sets of logical statements are used as world models, we must have some deductive mechanism that allows us to tell whether or not a given model satisfies the goal or satisfies the applicability conditions of various operators. Green[4] implemented a problem-solving system based on a theorem prover using the resolution principle. In his system, Green expressed the results of operators as logical statements. Thus, for example, to describe an operator goto(x,y) whose effect is to move a robot from any place x to any other place y, Green would use the wff

$$(\forall x,y,s)[ATR(x,s) \Rightarrow ATR(y,goto'(x,y,s))] \quad ,$$

where ATR is a predicate describing the robot's position. Here, each predicate has a state term that names the world model to which the predicate applies. Our wff above states that for all places x and y and for all states s, if the robot is at x in state s then the robot will be at y in the state goto$'$(x,y,s) resulting from applying the goto operator to state s.

With Green's formulation, any problem can be posed as a theorem to be proved. The theorem will have an existentially quantified state term, s. For example, the problem of pushing a box B to place b can be stated as the wff

$$(\exists s)\ AT(B,b,s) \quad .$$

If a constructive proof procedure is used, an instance of the state proved to exist can be extracted from the proof (Green,[2] Luckham and Nilsson[5]). This instance, in the form of a composition of operator functions acting on the initial state, then serves as a solution to the problem.

Green's formulation has all the appeal (and limitations) of any general-purpose problem solver and represents a significant step in the development of these systems. It does, however, suffer from some serious disadvantages that our present system attempts to overcome. One difficulty is caused by the fact that Green's system combines two essentially different kinds of searches into a single search for a proof of the theorem representing the goal. One of these searches is in a space of world models; this search proceeds by applying operators to these models to produce new models. The second type of search concerns finding a proof that a given world model satisfies the goal theorem or the applicability conditions of a given operator. Searches of this type proceed by applying rules of inference to wffs within a world model.

When these two kinds of searches are combined in the largely syntactically guided proof-finding mechanism of a general theorem prover, the result is gross inefficiency. Furthermore, it is much more difficult to apply any available semantic information in the combined search process.

The second drawback of Green's system is even more serious. The system must explicitly describe, by special axioms, those relations _not_ affected by each of the operators. For example, since typically the positions of objects do not change when a robot moves, we must include the statement

$$(\forall u,x,y,z,s)[\mathrm{OBJECT}(u,s) \wedge \mathrm{AT}(u,x,s) \Rightarrow \mathrm{AT}(u,x,\mathrm{goto}'(y,z,s)] \quad .$$

Thus, after every application of _goto_ in the search for a solution, we may need to prove that a given object B remains in the same position in the new state if the position of B is important to the completion of the solution.

The problem posed by the evident fact that operators affect certain relations and don't affect others is sometimes called the _frame problem_.[6,7] Since, typically, most of the wffs in a world model will not be affected by an operator application, our approach will be to name only those relations that are affected by an operator and to assume that the unnamed relations remain valid in the new world model. Since proving that certain relations are still satisfied in successor states is tedious, our convention can drastically decrease the search effort required.

Because we are adopting special conventions about what happens to the wffs in a world model when an operator is applied, we have chosen to take the process of operator application out of the formal deductive system entirely. In our approach, when an operator is applied to a world model, the computation of the new world model is done by a special extra-logical mechanism. Theorem-proving methods are used only <u>within</u> a given world model to answer questions about it concerning which operators are applicable and whether or not the goal has been satisfied. By separating the theorem proving that occurs within a world model from the search through the space of models we can employ separate strategies for these two activities and thereby improve the overall performance of the system.

II OPERATOR DESCRIPTIONS AND APPLICATIONS

The operators are the basic elements out of which a solution is built. For robot-like problems we can imagine that the operators correspond to routines or subprograms whose execution causes a robot to take certain actions. For example, we might have routines that cause the robot to turn and move, a routine that causes it to go through a doorway, a routine that causes it to push a box and perhaps dozens of others. When we discuss the application of problem-solving techniques to robot problems, the reader should keep in mind the distinction between an <u>operator</u> and its associated <u>routines</u>. Execution of routines actually causes the robot to take actions. Application of operators to world models occurs during the planning (i.e., problem solving) phase when an attempt is being made to find a sequence of operators whose associated routines will produce a desired state of the world. Since routines are programs, they can have parameters that are instantiated by constants when the routines are executed. The associated operators will also have parameters, but as we shall soon see, these can be left free at the time they are applied to a model.

In order to chain together a sequence of operators to achieve a given goal, the problem solver must have <u>descriptions</u> of the operators. The descriptions used by STRIPS consist of three major components:

(1) Name of the operator and its parameters,

(2) Preconditions, and

(3) Effects.

The first component consists merely of the name of the operator and the parameters taken by the operator. The second component is a formula in first-order logic. The operator is applicable in any world model in which the precondition formula is a theorem. For example, the operator push(u,x,y) which models the action of the robot pushing an object u from location x to location y might have as a precondition formula

$$(\exists x,u)[AT(u,x) \wedge ATR(x)] \quad .$$

The third component of an operator description defines the effects (on a set of wffs) of applying the operator. We shall discuss the process of computing effects in some detail since it plays a key role in STRIPS. When an operator is applied, certain wffs in the world model are no longer true (or at least we cannot be sure that they are true) and certain other wffs become true. Thus to compute one world model from another involves copying* the world model and in this copy deleting some of the wffs and

*In our implementation of STRIPS we employ various bookkeeping techniques to avoid copying; these will be described in a later section.

adding others. Let us deal first with the set of wffs that should be added as a result of an operator application.

The set of wffs to be added to a world model depends on the results of the routine modeled by the operator. These results are not completely specified until all of the parameters of the routine are instantiated by constants. For example, the operator goto(x,y) might model the robot moving from location x to location y for any two locations x and y. When this operator's routine is executed, the parameters x and y must be instantiated by constants. However, we have designed STRIPS so that an operator can be applied to a world model with any or all of the operator's parameters left uninstantiated. For example, suppose we apply the operator goto(a,x) to a world model in which the robot is at some location* a. If the parameter x is unspecified, so will be the resulting world model. We could say that the application of goto(a,x) creates a family or schema of world models parameterized by x. The power and efficiency of STRIPS is increased by searching in this space of world model families rather than in the larger space of individual world models.

If we are to gain this reduction in search space size, then we must be able to describe with a single set of predicate calculus wffs the world model family resulting from the application of an operator with free parameters. One way in which this can be done is to use a state term in each literal of each wff. Thus, the principal effect of applying the operator goto(a,x) to some world model s_o, say, is to add the wff

$$(\forall x)(\exists s)ATR(x,s)$$

*We shall adopt the convention of using letters near the beginning of the alphabet (a,b,c,etc.) to stand for constants and letters near the end of the alphabet (u,v,w,x,etc.) as variables.

which states that for all values of the parameter x, there exists a world model s in which the robot is at x. With expressions of this sort, a set of wffs can represent families of world models. Selecting specific values for the parameters selects specific members of the family.

Anticipating the use of a resolution-based theorem prover in STRIPS, we shall always express formulas in clause form.[1]
Then the formula above would be written

$$ATR(x,goto'(a,x,s_o))$$

where goto'(a,x,s_o) is a function of x replacing the existentially quantified state variable. The value of goto'(a,x,s_o), for any x, is that world model produced by applying the operator goto(a,x) to world model s_o. Recall that any variables (such as x in the formula above) occurring in a clause have implicit universal quantification.

The description of each operator used in STRIPS contains a list of those clauses to be added when computing a new world model. This list is called the _add list_.

The description of an operator also includes information about which clauses can no longer be guaranteed true and must therefore be deleted in constructing a new world model. For example, if the operator goto(a,y) is applied, we must delete any clause containing the atom* ATR(a). Each operator description contains a list of atoms, called the _delete list_, that is used to compute which clauses should be deleted. Our rule for creating a new world model is to delete any clauses containing atoms (negated or unnegated) that are instances of atoms on the delete list. We also delete any clauses containing atoms of which the atoms on

*An atom is a single predicate letter and its arguments.

on the delete list are instances. The application of these rules might sometimes delete some clauses unnecessarily, but we want to be guaranteed that the new world model will be consistent if the old one was.

When an operator description is written, it may not be possible to name explicitly all the atoms that should appear on the delete list. For example, it may be the case that a world model contains clauses that are derived from other clauses in the model. Thus from AT(B1,a) and from AT(B2,a+Δ) we might derive NEXTTO(B1,B2) and insert it into the model. Now, if one of the clauses on which the derived clause depends is deleted, then the derived clause must be deleted also.

We deal with this problem by defining a set of primitive predicates (e.g., AT, ATR, BOX) and relating all other predicates to this primitive set. In particular, we require the delete list of an operator description to indicate all the atoms containing primitive predicates which should be deleted when the operator is applied. Also, we require that any non-primitive clause in the world model have associated with it those primitive clauses on which its validity depends. (A primitive clause is one which contains only primitive predicates.) For example, the clause NEXTO(B1,B2) would have associated with it the clauses AT(B1,a) and AT(B2,a+Δ).

By using these conventions we can be assured that primitive clauses will be correctly deleted during operator applications, and that the validity of nonprimitive clauses can be determined whenever they are to be used in a deduction by checking to see if all of the primitive clauses on which the nonprimitive clause depends are still in the world model.

In the next section, we shall describe the search process for STRIPS and also present a specific example in which the process of operator application is examined in detail.

III THE OPERATION OF STRIPS

A. Computing Differences and Relevant Operators

In a very simple problem-solving system we might first apply all of the applicable operators to the initial world model to create a set of successor models. We would continue to apply all applicable operators to these successors and to their descendants until a model was produced in which the goal formula was a theorem. Checking to see which operators are applicable and to see if the goal formula is a theorem are theorem-proving tasks that could be accomplished by a deductive system such as QA3. However, since we envision uses in which the number of operators applicable to any given world model might be quite large, such a simple system would generate an undesirably large tree of world models and would thus be impractical.

Instead we would like to use the GPS strategy of extracting "differences" between the present world model and the goal and of identifying operators that are "relevant" to reducing these differences. Once a relevant operator has been determined, we attempt to solve the subproblem of producing a world model to which it is applicable. If such a model is found then we apply the relevant operator and reconsider the original goal in the resulting model.

When an operator is found to be relevant, it is not known where it will occur in the completed plan; that is, it may be applicable to the initial model and therefore be the first operator applied, its effects may imply the goal so that it is the last operator applied, or it may be some intermediate step toward the goal. Because of this flexibility, the STRIPS search strategy combines many of the advantages of both forward search (from the initial model toward the goal) and backward search (from the goal toward the initial model).

Two key steps in this strategy involve computing differences and finding operators relevant to reducing these differences. One of the novel features of our system is that it uses a theorem prover as an aid in these steps. The following description of these processes assumes that the reader is familiar with the terminology of resolution-based theorem-proving systems.

Suppose we have a world model consisting of a set, S, of clauses, and that we have a goal formula whose <u>negation</u> is represented by the set, G, of clauses. The difference-computing mechanism attempts to find a contradiction for the set $S \cup G$ using a resolution theorem prover such as QA3. (The theorem prover would likely use, at least, the set-of-support strategy with G the set receiving support.) If a contradiction is found, then the "difference" is nil and STRIPS would conclude that the goal is satisfied in S.

Our interest at the moment though is in the case in which QA3 <u>cannot</u> find a contradiction after investing some prespecified amount of effort. Let R be the set consisting of the clauses in G and the resolvents produced by QA3 which are descendants of G. Any set of clauses D in

R can be taken as a "difference" between S and the goal in the sense that if a world model were found in which a clause in D could be contradicted, then it is likely* that the proof of the goal could be completed in that model.

STRIPS creates differences by heuristically selecting subsets of R, each of which acts as a difference. The selection process considers such factors as the number of literals in a clause, at what level in the proof tree a clause was generated, and whether or not a clause has any descendants in the proof tree.

The quest for relevant operators proceeds in two steps. In the first step an ordered list of candidate operators is created for each difference set. The selection of operators for this list is based on a simple comparison of the clauses in the difference set with the add lists in the operator descriptions. For example, if a difference set contained a clause having in it the robot position predicate ATR, then the operator goto would be considered a candidate operator for that difference.

The second step in finding an operator relevant to a given difference set involves employing QA3 to determine if clauses on the add list of a candidate operator can be used to "resolve away" (i.e., continue the proof of) any of the clauses in the difference set. If, in fact, QA3 can produce new resolvents which are descendants of the add list clauses, then the candidate operator (properly instantiated) is considered to be a relevant operator for the difference set.

*That is, a proof could be completed if this new model still allows a deduction of this clause in D.

To complete the operator-relevance test STRIPS must determine which instances of the operator are relevant. For example, if the difference set consists of the unit clauses -ATR(a) and -ATR(b), then goto(x,y) is a relevant operator only when y is instantiated by a or b. Each new resolvent which is a descendant of the operator's add list clauses is used to form a relevant instance of the operator by applying to the operator's parameters the same instantiations that were made during the production of the resolvent. Hence the consideration of one candidate operator may produce several relevant operator instances.

One of the important effects of the difference-reduction process is that it usually produces specific instances for the operator parameters. Furthermore, these instances are likely to be those occurring in the final solution, thus helping to narrow the search process. So, although STRIPS has the ability to consider operators with uninstantiated parameters, it also has a strong tendency toward instantiating these parameters with what it considers to be the most relevant constants.

B. The STRIPS Executive

STRIPS begins by attempting to form differences between the initial world model, s_o, and the main goal (as described in the previous section). If no differences are found, then the problem is trivially solved. If differences are found, then STRIPS computes a set of operators relevant to reducing those differences.

Suppose, for example, that STRIPS finds two instantiated operators, OP_1 and OP_2, relevant to reducing the differences between s_o and the main goal. Let the (instantiated) precondition formulas for these operators be denoted by PC_1 and PC_2, respectively. Thus STRIPS has found two ways to work on the main problem:

(1) Produce a world model to which OP_1 is applicable, apply OP_1, and then produce a world model in which the main goal is satisfied, or

(2) Produce a world model to which OP_2 is applicable, apply OP_2, and then produce a world model in which the main goal is satisfied.

STRIPS represents such solution alternatives as nodes on a search tree. The tree for our example can be represented as follows:

$$\begin{tikzcd}
 & (S_o,(G_o)) \arrow[dl, no head] \arrow[dr, no head] & \\
(S_o,(G_1,G_o)) & & (S_o,(G_2,G_o))
\end{tikzcd}$$,

where G_o,G_1, and G_2 are sets of clauses corresponding to the negations of the main theorem, PC_1 and PC_2, respectively.

In general, each node of the search tree has the form (⟨world model⟩,⟨goal list⟩). The subgoal being considered for solution at each node is the first goal on that node's goal list. The last goal on each list is the negation of the main goal, and each subgoal is the negation of the preconditions of an operator. Hence, each subgoal in a goal list represents an attempt to apply an operator which is relevant to achieving the next goal in the goal list.

Whenever a new node, $(s_i,(G_m,G_{m-1},\ldots,G_1,G_o))$, is constructed and added to the search tree as a descendant of some existing node, the new node is tested for goal satisfaction. This test is performed by QA3 which looks for a contradiction to $s_i \cup G_m$.

If a contradiction is found and m is o (i.e., the node has the form $(s_i,(G_o)))$, then the main goal is satisfied in s_i and the problem is solved. If a contradiction is found and m is not o, then G_m is the negation of a precondition formula for an operator that is applicable in s_i. STRIPS produces a new world model, s_i', by applying to s_i the operator corresponding to G_m. The node is changed to $(s_i'(G_{m-1},\ldots,G_1,G_o))$ and the test for goal satisfaction is performed on it again. This process of changing the node continues until a goal is encountered which is not satisfied or until the problem is solved.

If no contradiction is found in the goal satisfaction test, QA3 will return a set R of clauses consisting of the clauses in G_m and resolvents that are descendants of clauses in G_m. This set of resolvents is attached to the node and is used for generating successors to the node.

The process for generating the successors of a node $(s_i,(G_m,G_{m-1},\ldots,G_1,G_o))$ with R attached involves forming difference sets $\{D_i\}$ from R and finding operator instances relevant to reducing these differences (as described in the previous section). For each operator instance found to be relevant, a new offspring node is created. This new node is formed with the same world model and goal list as its parent node. The goal of finding a world model in which the relevant operator instance can be applied is added to the new node. This is done by creating the appropriate instance of the operator's preconditions and adding the negation of the instantiated preconditions to the beginning of the new node's goal list.

Since the number of operators relevant to reducing sets of differences might be rather large in some cases, it is possible that a given node in the search tree might have a large number of successors. Even before the successors are generated, though, we can order them according to the heuristic merit of the operators and difference sets used to generate them. The process of computing a successor node can be rather lengthy, and for this reason STRIPS actually computes only that single next successor judged to be best. STRIPS adds this successor node to the search tree, performs a goal-satisfaction test on it, and then selects another node from the set of nodes which still have uncomputed successors. STRIPS must therefore associate with each node the sets of differences and candidate operators it has already used in creating successors.

STRIPS will have a heuristic mechanism to select nodes with uncomputed successors to work on next. For this purpose we will use an evaluation function that takes into account such factors as the number and types of literals in the remaining goal formulas, the number of remaining goals, and the number and types of literals in the difference sets.

A simple flowchart of the STRIPS executive is shown in Figure 1.

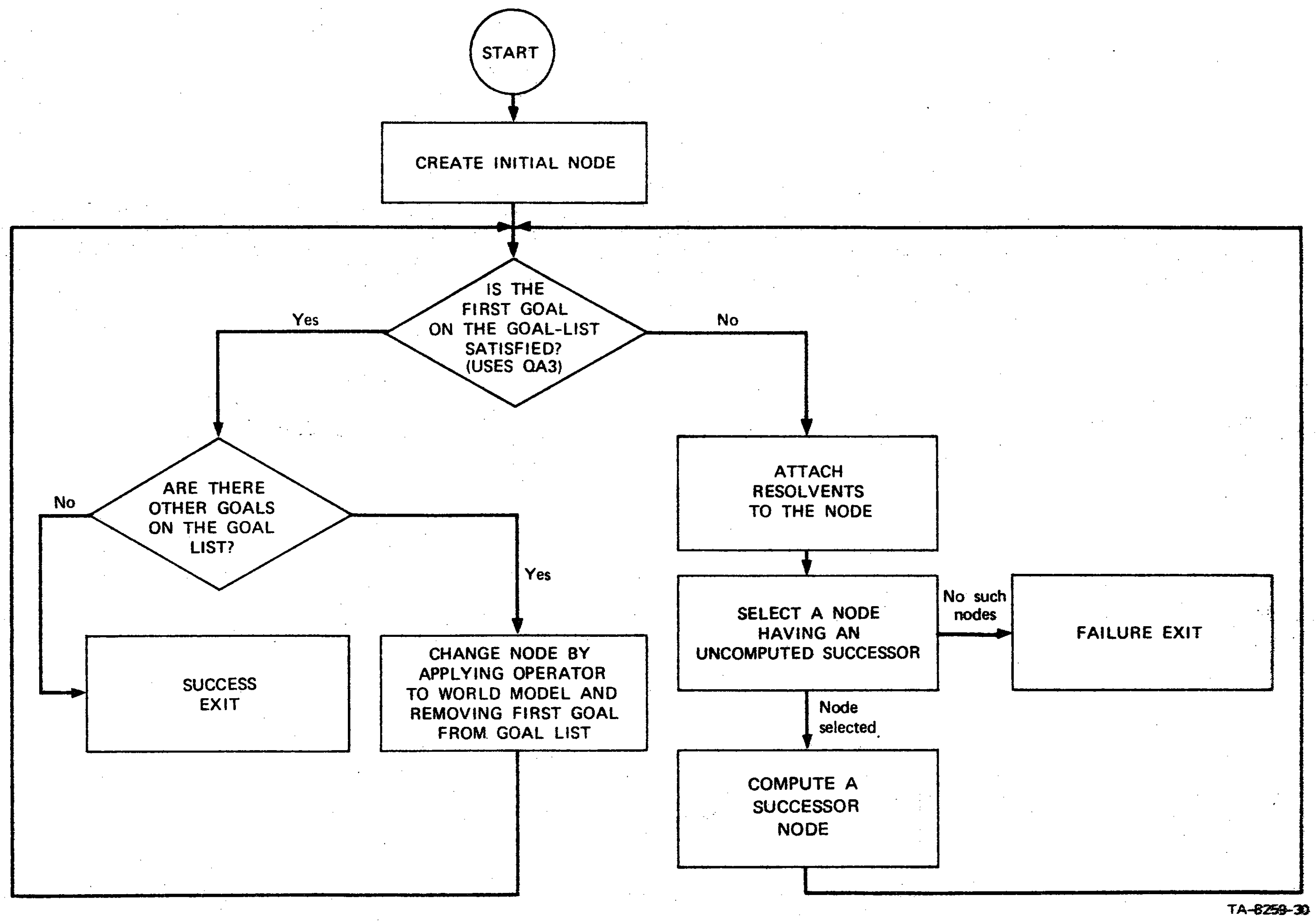

START
CREATE INITIAL NODE
IS THE FIRST GOAL ON THE GOAL-LIST SATISFIED? (USES QA3)
Yes
No
ARE THERE OTHER GOALS ON THE GOAL LIST?
No
Yes
SUCCESS EXIT
CHANGE NODE BY APPLYING OPERATOR TO WORLD MODEL AND REMOVING FIRST GOAL FROM GOAL LIST
ATTACH RESOLVENTS TO THE NODE
SELECT A NODE HAVING AN UNCOMPUTED SUCCESSOR
No such nodes
FAILURE EXIT
Node selected
COMPUTE A SUCCESSOR NODE
TA-8259-30

FIGURE 1 FLOWCHART FOR THE STRIPS EXECUTIVE

C. An Example

An understanding of how STRIPS works is aided by tracing through a simple example. Consider the configuration shown in Figure 2 consisting of two objects B and C and a robot R at places b, c, and a, respectively. The problem given to STRIPS is to achieve a configuration in which object B is at place k and in which object C is not at place c.

The existentially quantified theorem representing this problem can be written

$$(\exists s)[AT(B,k,s) \wedge \sim AT(C,c,s)] \quad .$$

If we can find an instance of s (in terms of a composition of operator applications) that satisfies this theorem, then we will have solved the problem. The negation of the theorem is

$$G_o: \sim AT(B,k,s) \vee AT(C,c,s) \quad .$$

Let us suppose that STRIPS is to compose a solution using the two operators goto and push. These operators can be described as follows:

1. push(u,x,y): Robot pushes object u from place x to place y.

Precondition formula:

$$(\exists u,x,s)[AT(u,x,s) \wedge ATR(x,s)]$$

Negated precondition formula:

$$\sim AT(u,x,s) \vee \sim ATR(x,s)$$

Delete list:

$$AT(u,x,s)$$

$$ATR(x,s)$$

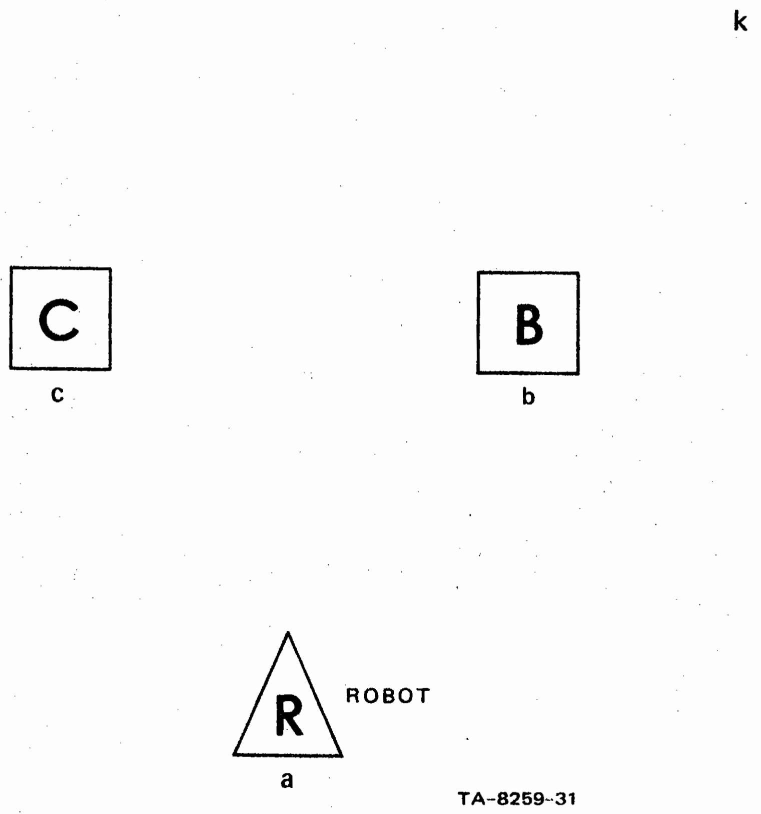


FIGURE 2 CONFIGURATION OF OBJECTS AND ROBOT FOR EXAMPLE PROBLEM

Add list:

$$AT(u,y,push'(u,x,y,s^*))$$

$$ATR(y,push'(u,x,y,s^*))$$

where s^* is the state to which the operator is applied.

2. goto(x,y): Robot goes from place x to place y.

Precondition formula:

$$(\exists x,s)ATR(x,s)$$

Negated precondition formula:

$$\sim ATR(x,s)$$

Delete list:

$$ATR(x,s)$$

Add list:

$$ATR(y,goto'(x,y,s^*))$$

The initial configuration can be described by the following world model:

$$s_o:\ ATR(a,s_o)$$
$$AT(B,b,s_o)$$
$$AT(C,c,s_o) \quad .$$

In addition, we have a universal formula, true in all world models, that states if an object is in one place, then it is not in a different place:

$$F:\quad (\forall u,x,y,s)[AT(u,x,s) \wedge (x \neq y) \Rightarrow \sim AT(u,y,s)] \quad .$$

The clause form of this formula is

$$F':\quad \sim AT(u,x,s) \vee (x=y) \vee \sim AT(u,y,s) \quad .$$

We assume that F' is adjoined to all world models.

STRIPS first constructs the node N_o, consisting of the list $(s_o,(G_o))$, as the root of the problem-solving tree and tests it for a solution by attempting to find a contradiction for the set $s_o \cup \{G_o\}$. No contradiction is found but some resolvents can be obtained; among them are two resolvents of G_o and F':

$$R_1: \ \sim AT(B,k,s) \vee (c=y) \vee \sim AT(C,y,s)$$

and

$$R_2: \ \sim AT(B,k,s) \vee (x=c) \vee \sim AT(C,x,s) \quad .$$

Additional resolvents can be produced also, but these happen all to be tautologies and can thus be eliminated.[†] A sophisticated system would detect that R_1 and R_2 are identical, so let us suppose that R_1 is the only resolvent attached to N_o.

Next STRIPS selects a node (N_o is now the only one available) and begins to generate successors. First it selects a difference set D_1 from the set of resolvents attached to N_o. In this case it sets $D_1 = \{R_1\}$. Then STRIPS composes a list L of candidate operators for reducing D_1. Here L would consist of the single element <u>push</u>.

Next STRIPS attempts to reduce D_1 using clauses on the add list of <u>push</u>. Again using theorem-proving methods we obtain two resolvents from D_1 and $AT(u,y,push'(u,x,y,s^*))$:

$$\sim AT(B,k,push'(C,x,y,s^*)) \vee (c=y)$$

and

$$\sim AT(C,y,push'(B,x,k,s^*)) \vee (c=y) \quad .$$

[†]We are assuming a set-of-support strategy with the initial support set consisting only of the negated theorem.

Assuming that these resolutions represent acceptable reductions in the difference, we extract the state terms of the resolvents to yield appropriate instances of the relevant operator. This gives us:

OP_1: <u>push</u>(C,x,y)

and OP_2: <u>push</u>(B,x,k) .

Next, we construct the negated versions of the precondition formulas for OP_1 and OP_2:

G_1: $\sim AT(C,x,s) \vee \sim ATR(x,s)$

and G_2: $\sim AT(B,x,s) \vee \sim ATR(x,s)$.

These formulas are then used to construct two successor nodes

N_1: $(s_o,(G_1,G_o))$

and N_2: $(s_o,(G_2,G_o))$.

These nodes would be immediately tested for solutions. For brevity, let us consider just N_1. In testing for a solution STRIPS attempts to find a contradiction for $s_o \cup G_1$.

Again no contradiction is found, but the following resolvents are obtained:

R_3: $\sim ATR(c,s_o)$ from G_1 and $AT(C,c,s_o)$

and R_4: $\sim AT(C,a,s_o)$ from G_1 and $ATR(a,s_o)$.

Although these clauses represent differences between s_o and G_1, we do not insist that these differences be reduced in s_o. We would accept a reduction occurring in any world model, so STRIPS rewrites the clauses as:

R_3': $\sim ATR(c,s)$

and R_4': $\sim AT(C,a,s)$.

These clauses refer to preconditions for pushing object C. To contradict R_3' the robot must be at c; to contradict R_4' object C must be at a. Suppose our system recognizes that an attempt to contradict R_4' is circular and attaches just the set $\{R_3'\}$ to node N_1.

Next STRIPS selects a node for consideration. Suppose it selects N_1. In generating successors, it sets the difference set, D_2, to $\{R_3'\}$.

The list of operators useful for reducing D_2 consists only of goto. STRIPS now attempts to perform resolutions between the clauses on the add list of goto and D_2. The clause in D_2 resolves with ATR(y,goto'(x,y,s^*)) to yield nil, and answer extraction produces the instance substituted for the state term, namely

$$s = \text{goto}'(x,c,s^*) \quad .$$

Thus STRIPS identifies the following instance of goto:

OP_3: goto(x,c) .

The associated negated precondition is

G_3: $\sim$AT(R,x,s) .

STRIPS then constructs the successor node

N_3: $(s_o,(G_3,G_1,G_o))$

and immediately attempts to find a contradiction for $s_o \cup G_3$. Here a contradiction is obtained, and answer extraction yields the state term:

$$\text{goto}'(a,c,s_o) \quad .$$

Thus STRIPS applies goto(a,c) to s_o to yield

s_1: ATR(c,goto'(a,c,s_o))

AT(B,b,goto'(a,c,s_o))

AT(C,c,goto'(a,c,s_o)) .

Node N_3 is then changed to

N_4: $(s_1,(G_1,G_o))$

and STRIPS immediately checks for a contradiction for $s_1 \cup G_1$. Again a contradiction is found; answer extraction produces the following instances for x and s:

$$x = c$$

and
$$s = goto'(a,c,s_o) \quad .$$

Thus STRIPS applies the following instance of OP_1:

$$\underline{push}(C,c,y) \quad .$$

The result is the world model family s_2 consisting of the following clauses:

$$\begin{aligned} s_2: \quad & ATR(y,push'(C,c,y,goto'(a,c,s_o))) \\ & AT(B,b,push'(C,c,y,goto'(a,c,s_o))) \\ & AT(C,y,push'(C,c,y,goto'(a,c,s_o))) \quad . \end{aligned}$$

Note that this application of the operator <u>push</u> involved an uninstantiated parameter, y.

Node N_4 is then changed to

N_5: $(s_2,(G_o))$

and STRIPS checks for a contradiction for $s_2 \cup G_o$. In doing so it produces the following tree of resolutions:

$$\begin{tikzcd}
\sim AT(B,k,s) \vee AT(C,c,s) \arrow[dr, dash] & & & \sim AT(u,x,s) \vee (x=y) \vee \sim AT(u,y,s) \arrow[dll, dash] \\
& \sim AT(B,k,s) \vee (c=y) \vee \sim AT(C,y,s) \arrow[ddr, dash] & & \\
& & & AT(C,y,push'(C,c,y,goto'(a,c,s_o))) \arrow[dl, dash] \\
& & \sim AT(B,k,push'(C,c,y,goto'(a,c,s_o))) \vee (c=y) \quad . &
\end{tikzcd}$$

The clause at the root produces one of the resolvents to be attached to N_5, namely

$$R_5: \sim AT(B,k,s) \vee (c=y) \quad .$$

Suppose STRIPS selects N_5 next and begins generating successors based on a difference $D_3 = \{R_5\}$. The operator list for this difference consists solely of <u>push</u>, and the relevant instance of <u>push</u> is found to be

$$OP_4: \ push(B,x,k) \quad .$$

Its (negated) precondition is

$$G_4: \sim AT(B,x,s) \vee \sim ATR(x,s) \quad .$$

A successor node to N_5 is then

$$N_6: \ (s_2,(G_4,G_o)) \quad .$$

STRIPS then finds a contradiction between s_2 and G_4, and extracts

$$s = push'(C,c,b,goto'(a,c,s_o))$$

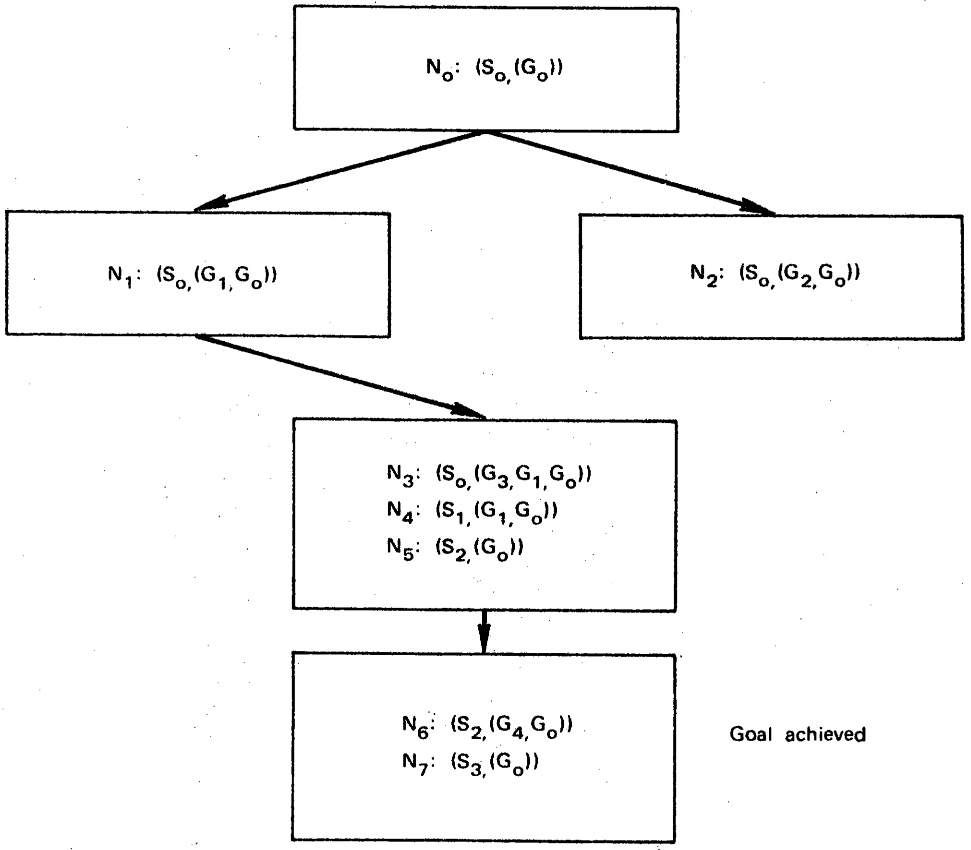
and $x = b$. Therefore, it applies push(B,b,k) to an instance of s_2 (with $y = b$) to yield

$$\begin{aligned} s_3: \ & ATR(k,push'(B,b,k,push'(C,c,b,goto'(a,c,s_o)))) \\ & AT(B,k,push'(B,b,k,push'(C,c,b,goto'(a,c,s_o)))) \\ & AT(C,b,push'(B,b,k,push'(C,c,b,goto'(a,c,s_o)))) \quad . \end{aligned}$$

Node N_6 is then changed to node

$$N_7: \ (s_3,(G_o)) \quad .$$

STRIPS can find a contradiction between s_3 and G_o [assuming that the equality predicate $(b=c)$ can be evaluated to be false] and exits successfully. The successful plan is embodied in the state term for s_3. We show the solution path in the STRIPS problem-solving tree in Figure 3.


N_o: $(S_o, (G_o))$
N_1: $(S_o, (G_1, G_o))$
N_2: $(S_o, (G_2, G_o))$
N_3: $(S_o, (G_3, G_1, G_o))$
N_4: $(S_1, (G_1, G_o))$
N_5: $(S_2, (G_o))$
N_6: $(S_2, (G_4, G_o))$
N_7: $(S_3, (G_o))$
Goal achieved

TA-8259-32

FIGURE 3 SEARCH TREE FOR EXAMPLE PROBLEM

D. Efficient Representation of World Models

A primary design issue in the implementation of a system such as STRIPS is how to satisfy the storage requirements of a search tree in which each node may contain a different world model. We would like to use STRIPS in a robot or question-answering environment where the initial world model may consist of hundreds of wffs. For such applications it is infeasible to recopy completely a world model each time a new model is produced by application of an operator.

We have dealt with this problem in STRIPS by first making the assumption that most of the wffs in a problem's initial world model will not be changed by the application of operators. This is certainly true for the class of robot problems we are currently concerned with. For these problems most of the wffs in a model describe rooms, walls, doors, and objects, or specify general properties of the world which are true in all models. The only wffs that might be changed in this robot environment are the ones that describe the status of the robot and any objects which it manipulates.

Given this assumption, we have implemented the following scheme for handling multiple world models. All the wffs for all world models are stored in a common memory structure. Associated with each wff (i.e., clause) is a visibility flag, and QA3 has been modified to consider only clauses from the memory structure which are marked visible. Hence, we can "define" a particular world model for QA3 by marking that model's clauses visible and all other clauses invisible. When clauses are entered into the initial world model they are marked visible and given a variable as a state term. Clauses not changed will remain visible throughout STRIPS' search for a solution.

Each world model produced by STRIPS is defined by two clause lists. The first list, DELETIONS, names all those clauses from the initial world model which are no longer present in the model being defined. The second list, ADDITIONS, names all those clauses in the model being defined which are not also in the initial model. These lists represent the changes in the initial model needed to form the model being defined, and our assumption implies they will contain only a small number of clauses.

To specify a given world model to QA3, STRIPS marks visible the clauses on the model's ADDITIONS list and marks invisible the clauses on the model's DELETIONS list. When the call to QA3 is completed, the visibility markings of these clauses are returned to their previous settings.

When an operator is applied to a world model, the DELETIONS list of the new world model is a copy of the DELETIONS list of the old model plus any clauses from the initial model which are deleted by the operator. The ADDITIONS list of the new model consists of the clauses from the old model's ADDITIONS list as transformed by the operator plus the clauses from the operator's <u>add list</u>.

To illustrate this implementation design we list below the way in which the world models described in the example of the previous section are represented:

s_o: ATR(a,s)
AT(B,b,s)
AT(C,c,s)

s_1: DELETIONS: ATR(a,s)

ADDITIONS: ATR(c,goto′(a,c,s_o))

s_2: DELETIONS: ATR(a,s)
AT(C,c,s)

ADDITIONS: ATR(y,push′(C,c,y,goto′(a,c,s_o)))
AT(C,y,push′(C,c,y,goto′(a,c,s_o)))

s_3: DELETIONS: ATR(a,s)
AT(C,c,s)
AT(B,b,s)

ADDITIONS: ATR(k,push′(B,b,k,push′(C,c,b,goto′(a,c,s_o))))
AT(B,k,push′(B,b,k,push′(C,c,b,goto′(a,c,s_o))))
AT(C,c,push′(B,b,k,push′(C,c,b,goto′(a,c,s_o))))

IV FUTURE PLANS AND PROBLEMS

The implementation of STRIPS now being completed can be extended in several directions. These extensions will be the subject of much of our problem-solving research activities in the immediate future. We shall conclude this note by briefly mentioning some of these.

We have seen that STRIPS constructs a problem-solving tree whose nodes represent subproblems. In a problem-solving process of this sort, there must be a mechanism to decide which subproblem to work on next. We have already mentioned some of the factors that might be incorporated in an evaluation function by which subproblems can be ordered according to heuristic merit. We expect to devote a good deal of effort to devising and experimenting with various evaluation functions and other ordering techniques.

Another area for future research concerns synthesis of more complex procedures than those consisting of simple linear sequences of operators. Specifically we want to be able to generate procedures involving iteration (or recursion) and conditional branching. In short, we would like STRIPS to be able to generate computer programs. Several researchers[4,8,9] have already considered the problem of automatic program synthesis and we expect to be able to use some of their ideas in STRIPS.

Our implementation of STRIPS is designed to facilitate the definition of new operators by the user. Thus the problem-solving power of STRIPS can gradually increase as its store of operators grows.

An idea that may prove useful in robot applications concerns defining and using operators to which there correspond no execution routines. That is, STRIPS may be allowed to generate a plan containing one or more operators that are fictitious. This technique essentially permits STRIPS to assume that certain subproblems have solutions without actually knowing how these solutions are to be achieved in terms of existing robot routines. When the robot system attempts to execute a fictitious operator, the subproblem it represents must first be solved (perhaps by STRIPS). (In human problem solving, this strategy is employed when we say: "I won't worry about that [sub] problem until I get to it.")

We are also interested in getting STRIPS to define new operators for itself based on previous problem solutions. One reasonable possibility is that after a problem represented by $(S_o,(G_o))$ is solved, STRIPS could automatically generate a fictitious operator to represent the solution. It would be important to try to generalize any constants appearing in G_o; these would then be represented by parameters in the fictitious operator. The structure of the actual solution would also have to be examined in order to extract a precondition formula, _delete list_, and _add list_ for the fictitious operator.

A more ambitious undertaking would be an attempt to synthesize automatically a robot execution routine corresponding to the new operator. Of course, this routine would be composed from a sequence of the existing routines corresponding to the individual existing operators used in the problem solution. The major difficulty concerns generalizing constants to parameters so that the new routine is general enough to merit saving. Hewitt[10] discusses a related problem that he calls "procedural abstraction." He suggests that from a few instances of a procedure, a general version can sometimes be synthesized. We expect that our generalization problem will be aided by an analysis of the structure of the preconditions and effects of the individual operators used in the problem solution.

ACKNOWLEDGMENT

The development of the ideas embodied in STRIPS has been the result of the combined efforts of the present authors, Bertram Raphael, Thomas Garvey, John Munson, and Richard Waldinger, all members of the Artificial Intelligence Group at SRI.

REFERENCES

1. N. J. Nilsson, Problem-Solving Methods in Artificial Intelligence (McGraw-Hill Book Company, New York, to appear in April 1971).

2. C. Green, "Theorem Proving by Resolution as a Basis for Question-Answering Systems," in Machine Intelligence 4, B. Meltzer and D. Michie (Eds.), pp. 183-205 (American Elsevier Publishing Co., Inc., New York, 1969).

3. G. Ernst and A. Newell, GPS: A Case Study in Generality and Problem Solving, ACM Monograph Series (Academic Press, 1969).

4. C. Green, "Application of Theorem Proving to Problem Solving," Proc. Intl. Joint Conf. on Artificial Intelligence, Washington, D.C. (May 1969).

5. D. Luckham and N. Nilsson, "Extracting Information from Resolution Proof Trees," Artificial Intelligence (to appear).

6. J. McCarthy and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in Machine Intelligence 4, B. Meltzer and D. Michie (Eds.), pp. 463-502 (American Elsevier Publishing Co., Inc., New York, 1969).

7. B. Raphael, "The Frame Problem in Problem-Solving Systems," Proc. Adv. Study Inst. on Artificial Intelligence and Heuristic Programming, Menaggio, Italy (August 1970).

8. R. Waldinger and R. Lee, "PROW: A Step Toward Automatic Program Writing," Proc. Intl. Joint Conf. on Artificial Intelligence, Washington, D.C. (May 1969).

9. Z. Manna and R. Waldinger, "Towards Automatic Program Synthesis," Artificial Intelligence Group Technical Note 34, Stanford Research Institute, Menlo Park, California (July 1970).

10. C. Hewitt, "Planner: A Language for Manipulating Models and Proving Theorems in a Robot," Artificial Intelligence Memo No. 168 (Revised), Massachusetts Institute of Technology, Project MAC, Cambridge, Massachusetts (August 1970).