STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving[1]

Richard E. Fikes, Nils J. Nilsson
Stanford Research Institute, Menlo Park, California

Recommended by B. Raphael

Presented at the 2nd IJCAI, Imperial College, London, England, September 1-3, 1971.

ABSTRACT

We describe a new problem solver called STRIPS that attempts to find a sequence of operators in a space of world models to transform a given initial world model into a model in which a given goal formula can be proven to be true. STRIPS represents a world model as an arbitrary collection of first-order predicate calculus formulas and is designed to work with models consisting of large numbers of formulas. It employs a resolution theorem prover to answer questions of particular models and uses means-ends analysis to guide it to the desired goal-satisfying model.

DESCRIPTIVE TERMS

Problem solving, theorem proving, robot planning, heuristic search.

CONTENT

1. INTRODUCTION

This paper describes a new problem-solving program called STRIPS (STanford Research Institute Problem Solver). An initial version of the program has been implemented in LISP on a PDP-10 and is being used in conjunction with robot research at SRI. STRIPS is a member of 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 that 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 stated

[1] The research reported herein was sponsored by the Advanced Research Projects Agency and the National Aeronautics and Space Administration under Contract NAS12-2221.

goal condition.

This framework for problem solving has been central to much of the research in artificial intelligence [1]. Our primary interest here is in the class of problems faced by a robot in re-arranging objects and in navigating, i.e., problems that require quite complex and general world models compared to those needed in the solution of puzzles and games. In puzzles and games, a simple matrix or list structure is usually adequate to represent a state of the problem. The world model for a robot problem solver, however, must 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. In STRIPS, a world model is represented by a set of well-formed formulas (wffs) of the first-order predicate calculus.

Operators are the basic elements from which a solution is built. For robot problems, each operator corresponds to an *action routine*[2] whose execution causes a robot to take certain actions. For example, we might have a routine that causes it to go through a doorway, a routine that causes it to push a box, and perhaps dozens of others.

Green [4] implemented a problem-solving system that depended exclusively on formal theorem-proving methods to search for the appropriate sequence of operators. While Green's formulation represented a significant step in the development of problem-solvers, it suffered some serious disadvantages connected with the “frame problem”[3] that prevented it from solving nontrivial problems.

In STRIPS, we surmount these difficulties by separating entirely the processes of theorem proving from those of searching through a space of world models. This separation allows us to employ separate strategies for these two activities and thereby improve the overall performance of the system. Theorem-proving methods are used only *within* a given world model to answer questions about it concerning which operators are applicable and whether or not goals have been satisfied. For searching through the space of world models, STRIPS uses a GPS-like means-end analysis strategy [6]. This combination of means-ends analysis and formal theorem-proving methods allows objects (world models) much more complex and general than any of those used in GPS and provides more powerful search heuristics than those found in theorem-proving programs.

We proceed by describing the operation of STRIPS in terms of the conventions used to represent the search space for a problem and the search methods used to find a solution. We then discuss the details of implementation and present some examples.

2. THE OPERATION OF STRIPS

2.1. The Problem Space

The problem space for STRIPS is defined by the initial world model, the set of available operators and their effects on world models, and the goal statement.

As already mentioned, STRIPS represents a world model by a set of well-formed formulas (wffs). For example, to describe a world model in which the robot is at location

[2] The reader should keep in mind the distinction between an *operator* and its associated *action routine*. Execution of action 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 action routines will produce a desired state of the world. (See the papers by Munson [2] and Fikes[3] for discussions of the relationships between STRIPS and the robot executive and monitoring functions.)

[3] Space does not allow a full discussion of the frame problem; for a thorough treatment, see [5].

LOC_A and boxes *B* and *C* are at locations *LOC_B* and *LOC_C* we would include the following wffs:

$$\mathrm{atr}(\mathrm{LOC\_A}),\ \mathrm{at}(\mathrm{B}, \mathrm{LOC\_B}),\ \mathrm{at}(\mathrm{C}, \mathrm{LOC\_C}).$$

We might also include the wff:

$$(\forall u \forall x \forall y)\{[\mathrm{at}(u,x) \wedge (x \neq y)] \Rightarrow \sim \mathrm{at}(u,y)\}$$

to state the general rule that an object in one place is not in a different place. Using first-order predicate calculus wffs, we can represent quite complex world models and can use existing theorem-proving programs to answer questions about a model.

The available operators are grouped into families called schemata. Consider for example the operator *goto* for moving the robot from one point on the floor to another. Here there is really a distinct operator for each different pair of points, but it is convenient to group all of these into a family *goto (m,n)* parameterized by the initial position[4] *m* and the final position *n*. We say that *goto (m,n)* is an *operator schema* whose members are obtained by substituting specific constants for the *parameters* *m* and *n*. In STRIPS, when an operator is applied to a world model, specific constants will already have been chosen for the operator parameters.

Each operator is defined by an operator description consisting of two main parts: a description of the effects of the operator, and the conditions under which the operator is applicable. The effects of an operator are simply defined by a list of wffs that must be added to the model and a list of wffs that are no longer true and therefore must be deleted. We shall discuss the process of calculating these effects in more detail later. It is convenient to state the applicability condition, or *precondition*, for an operator schema as a *wff schema*. To determine whether or not there is an instance of an operator schema applicable to a world model, we must be able to prove that there is an instance of the corresponding wff schema that logically follows from the model.

For example, consider the question of applying instances of the operator subschema *goto (m, LOC_B)* to a world model containing the wff *atr(LOC_A)*, where *LOC_A* and *LOC_B* are constants. If the precondition wff schema of *goto (m,n)* is *atr(m)*, then we find that the instance *atr(LOC_A)* can be proved from the world model. Thus, an applicable instance of *goto(m, LOC_B)* is *goto(LOC_A, LOC_B)*.

It is important to distinguish between the parameters appearing in wff schemata and ordinary existentially and universally quantified variables that may also appear. Certain modifications must be made to theorem-proving programs to enable them to handle wff schemata; these are discussed later.

Goal statements are also represented by wffs. For example, the task *Get Boxes B and C to Location LOC_A* might be stated as the wff:

$$\mathrm{at}(\mathrm{B}, \mathrm{LOC\_A}) \wedge \mathrm{at}(\mathrm{C}, \mathrm{LOC\_A})$$

To summarize, the problem space for STRIPS is defined by three entities:

(1) An initial world model, which is a set of wffs describing the present state of the

[4] The parameters *m* and *n* are each really vector-valued, but we avoid vector notation here for simplicity. In general, we denote constants by capital letters near the beginning of the alphabet (A, B, C, …), parameters by letters in the middle of the alphabet *(m, n, …)*, and quantified variables by letters near the end of the alphabet *(x, y, z)*.

world.

(2) A set of operators, including a description of their effects and their precondition wff schemata.

(3) A goal condition stated as a wff.

The problem is solved when STRIPS produces a world model that satisfies the goal wff.

2.2. The Search Strategy

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 (say in breadth-first fashion) until a model was produced in which the goal formula was a theorem. 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 have adopted 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 [6]. 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. In this section, we review this basic GPS search strategy as employed by STRIPS.

STRIPS begins by employing a theorem prover to attempt to prove that the goal wff G_0 follows from the set M_0 of wffs describing the initial world model. If G_0 does follow from M_0, the task is trivially solved in the initial model. Otherwise, the theorem prover will fail to find a proof. In this case, the uncompleted proof is taken to be the "difference" between M_0 and G_0. Next, operators that might be relevant to "reducing" this difference are sought. These are the operators whose effects on world models would enable the proof to be continued. In determining relevance, the parameters of the operators may be partially or fully instantiated. The corresponding instantiated precondition wff schemata (of the relevant operators) are then taken to be new subgoals.

Consider the trivially simple example in which the task is for the robot to go to location *LOC_B*. The goal wff is thus *atr(LOC_B)*, and unless the robot is already at location *LOC_B*, the initial proof attempt will be unsuccessful. Now, certainly the instance *goto(m, LOC_B)* of the operator *goto(m, n)* is relevant to reducing the difference because its effect would allow the proof to be continued (in this case, completed). Accordingly, the corresponding precondition wff schema, say *atr(m)*, is used as a subgoal.

STRIPS works on a subgoal using the same technique. Suppose the precondition wff schema *G* is selected as the first subgoal to be worked on. STRIPS again uses a theorem prover in an attempt to find instances of *G* that follow from the initial world model M_0. Here again, there are two possibilities. If no proof can be found, STRIPS uses the incomplete proof as a difference, and sets up (sub) subgoals corresponding to their precondition wffs. If STRIPS does find an instance of *G* that follows from M_0, then the corresponding operator instance is used to transform M_0 into a new world model M_1. In our previous simple example, the subgoal wff schema *G* was *atr(m)*. If the initial model contains the wff *atr(LOC_A)*, then an instance of *G* – namely *atr(LOC_A)* – can be proved from M_0. In this case, the corresponding operator instance *goto(LOC_A, LOC_B)* is applied to M_0 to produce the new model M_1. STRIPS then continues by attempting to prove G_0 from M_1. In our example, G_0 trivially follows from M_1 and we are through. However, if no proof could be found, subgoals for this problem would be set up and the process would continue.

The hierarchy of goal, subgoals, and models generated by the search process is represented by a *search tree*. Each node of the search tree has the form (<world model>, <goal list>), and represents the problem of trying to achieve the sub-goals on the goal list (in order) from the indicated world model.

An example of such a search tree is shown in Fig.1. The top node (M_0, (G_0)) represents the main task of achieving goal G_0 from world model M_0.

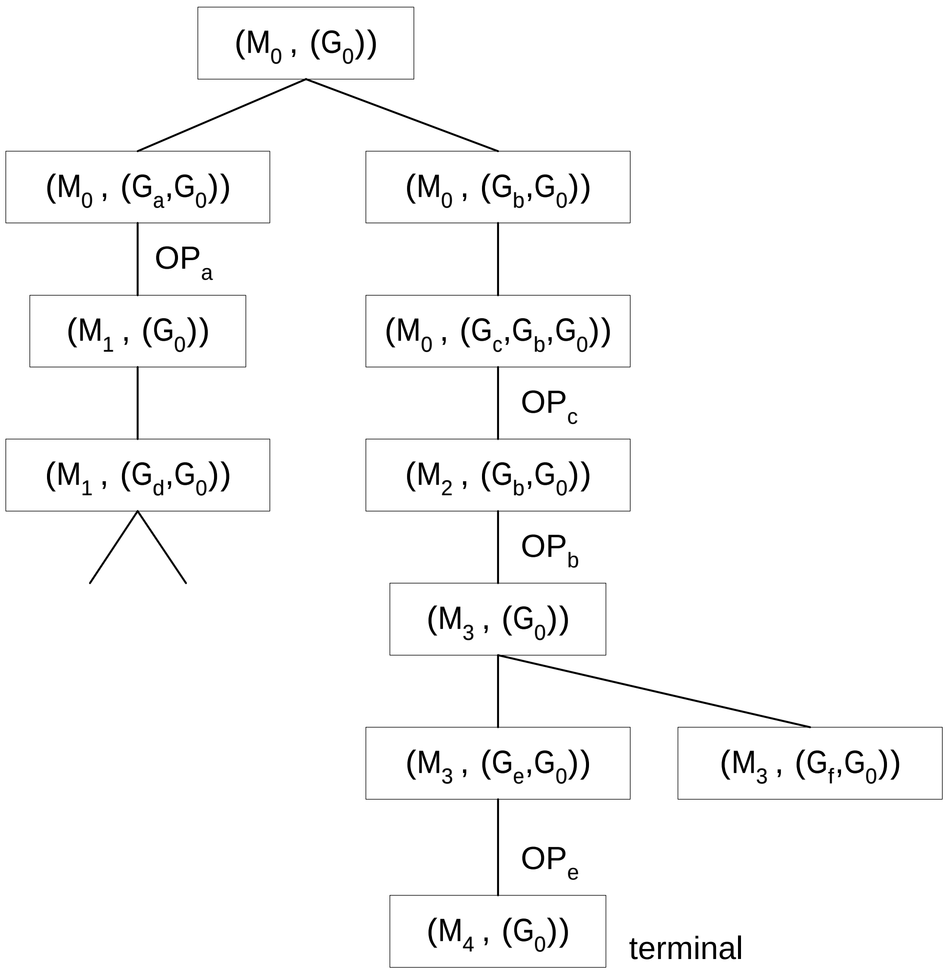


Fig. 1. A typical STRIPS search tree

In this case, two alternative subgoals G_a and G_b are set up. These are added to the front of the goal lists in the two successor nodes. Pursuing one of these subgoals, suppose that in the node (M_0, (G_a, G_0)), goal G_a is satisfied in M_0; the corresponding operator, say OP_a, is then applied to M_0 to yield M_1. Thus, along this branch, the problem is now to satisfy goal G_0 from M_1, and this problem is represented by the node (M_1, (G_0)). Along the other path, suppose G_c is set up as a subgoal for achieving G_b and thus the node (M_0, (G_c, G_b, G_0)) is created. Suppose G_c is satisfied in M_0 and thus OP_c is applied to M_0 yielding

M_2. Now STRIPS must still solve the subproblem G_b before attempting the main goal G_0. Thus, the result of applying OP_c is to replace M_0 by M_2 and to remove G_c from the goal list to produce the node (M_2, (G_b, G_0)).

This process continues until STRIPS produces the node (M_4, (G_0)). Here suppose G_0 can be proved directly from M_4 so that this node is terminal. The solution sequence of operators is thus (OP_c, OP_b, OP_e).

This example search tree indicates clearly that 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. This flexible search strategy embodied in STRIPS 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).

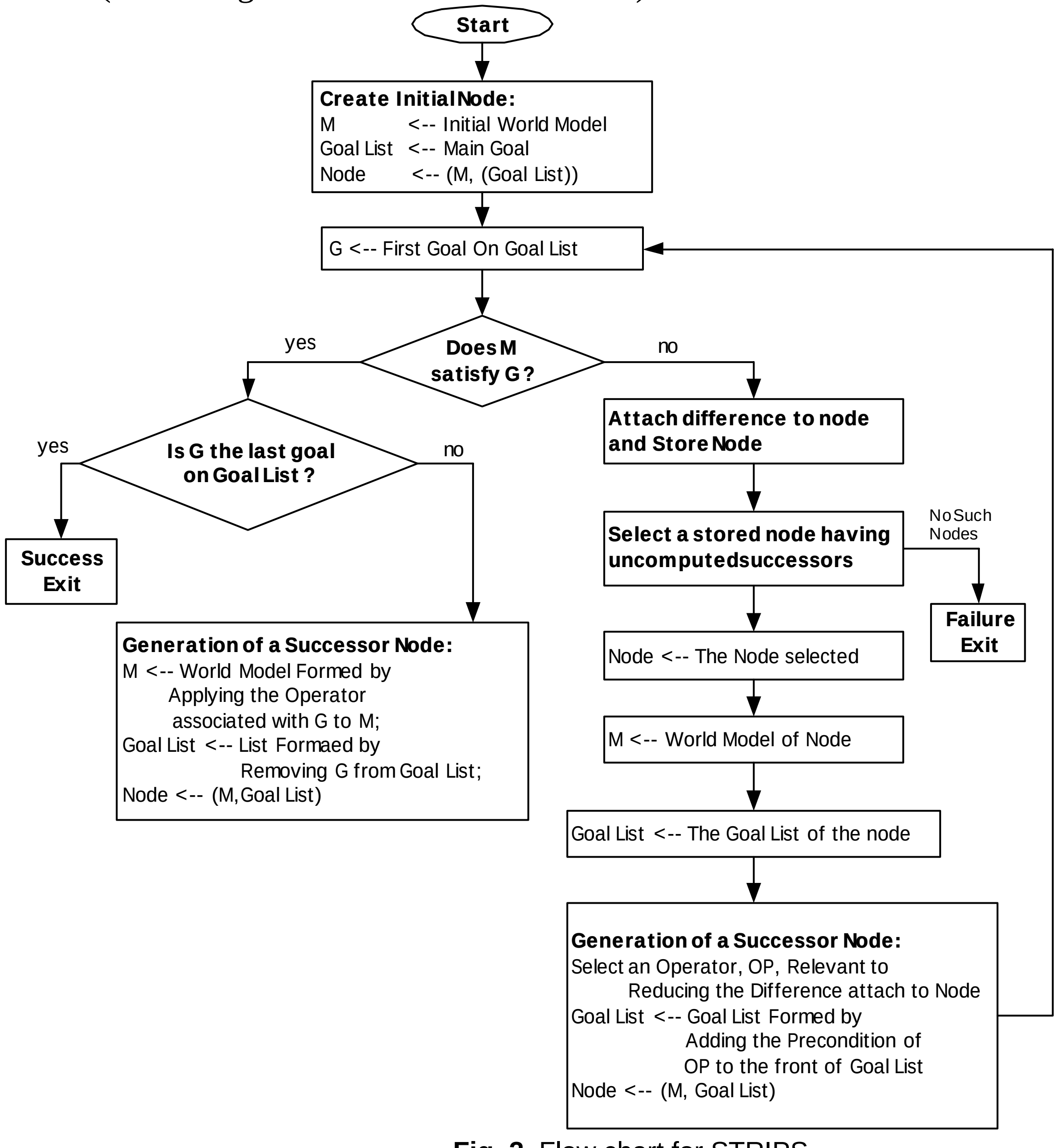

Start
Create Initial Node:
M <-- Initial World Model
Goal List <-- Main Goal
Node <-- (M, (Goal List))
G <-- First Goal On Goal List
Does M satisfy G ?
yes
no
Is G the last goal on Goal List ?
yes
no
Success Exit
Generation of a Successor Node:
M <-- World Model Formed by Applying the Operator associated with G to M;
Goal List <-- List Formaed by Removing G from Goal List;
Node <-- (M,Goal List)
Attach difference to node and Store Node
Select a stored node having uncomputed successors
No Such Nodes
Failure Exit
Node <-- The Node selected
M <-- World Model of Node
Goal List <-- The Goal List of the node
Generation of a Successor Node:
Select an Operator, OP, Relevant to Reducing the Difference attach to Node
Goal List <-- Goal List Formed by Adding the Precondition of OP to the front of Goal List
Node <-- (M, Goal List)

Fig. 2. Flow chart for STRIPS

Whenever STRIPS generates a successor node, it immediately tests to see if the first goal on the goal list is satisfied in the new node's model. If so, the corresponding operator is applied, generating a new successor node; if not, the difference (i.e., the uncompleted proof) is stored with the node. Except for those successor nodes generated as a result of applying operators, the process of successor generation is as follows: STRIPS selects a node and uses the difference stored with the node to select a relevant operator. It uses the precondition of this operator to generate a new successor. (If all of the node's successors have already been generated, STRIPS selects some other node still having uncompleted successors.) A flowchart summarizing the STRIPS search process is shown in Fig. 2.

STRIPS has a heuristic mechanism to select nodes with uncompleted successors to work on next. For this purpose we use an evaluation function that takes into account such factors as the number of remaining goals on the goal list, the number and types of predicates in the remaining goal formulas, and the complexity of the difference attached to the node.

3. IMPLEMENTATION

3.1. Theorem-Proving with Parameters

In this section, we discuss the more important details of our implementation of STRIPS; we begin by describing the automatic theorem-proving component.

STRIPS uses the resolution theorem-prover QA3.5 [7] when attempting to prove goal and sub-goal wffs. We assume that the reader is familiar with resolution proof techniques for the predicate calculus [1]. These techniques must be extended to handle the parameters occurring in wff schemas; we discuss these extensions next.

The general situation is that we have some goal wff schema $G(p)$, say, that is to be proved from a set M of clauses where p is a set of schema parameters. Following the general strategy of resolution theorem provers, we attempt to prove the inconsistency of the set $\{M \cup \sim G(p)\}$. That is, we attempt to find an instance p' of p for which $\{M \cup \sim G(p')\}$ is inconsistent.

We have been able to use the standard unification algorithm of the resolution method to compute the appropriate instances of schema variables during the search for a proof. This algorithm has the advantage that it finds the most general instances of parameters needed to effect unification. To use the unification algorithm we must specify how it is to treat parameters. The following substitution types are allowable components of the output of the modified unification algorithm:

- *Terms that can be substituted for a variable:* variables, constants, parameters, and functional terms not containing the variable.
- *Terms that can be substituted for a parameter:* constants, parameters, and functional terms not containing Skolem functions, variables, or the parameter.

The fact that the same parameter may have multiple occurrences in a set of clauses demands another modification to the theorem prover. Suppose two clauses C_1 and C_2 resolve to form clause C and that in the process some term t is substituted for parameter p. Then we must make sure that p is replaced by t in all of the clauses that are descendants of C.

3.2. Operator Descriptions and Applications

We have already mentioned that to define an operator, we must state the preconditions under which it is applicable and its effects on a world model schema. Preconditions are stated as wff schemata. For example, suppose *G(p)* is the operator precondition schema of an operator *O(p)*, *p* is a set of parameters, and *M* is a world model. Then if *p'* is a constant instance of *p* for which $\{M \cup \sim G(p')\}$ is contradictory, then STRIPS can apply operator *O(p')* to world model *M*.

We next need a way to state the effects of operator application on world models. These effects are simply described by two lists. On the *delete list* we specify those clauses in the original model that might no longer be true in the new model. On the *add list* are those clauses that might not have been true in the original model but are true in the new model.

For example, consider an operator *push(k, m, n)* for pushing object *k* from *m* to *n*. Such an operator might be described as follows:

push(k, m, n)
precondition = atr(m) $\wedge$ at(k, m)
delete list = {atr(m), at(k, m)}
add list = {atr(n), at(k, n)}.

The parameters of an operator schema are instantiated by constants at the time of operator application. Some instantiations are made while deciding what instances of an operator schema are relevant to reducing a difference, and the rest are made while deciding what instances of an operator are applicable in a given world model. Thus, when the add and delete lists are used to create new world models, all parameters occurring in them will have been replaced by constants.

(We can make certain modifications to STRIPS to allow it to apply operators with uninstantiated parameters. These applications will produce world model schemata. This generalization complicates somewhat the simple add and delete-list rules for computing new world models and needs further study.)

For certain operators it is convenient to be able merely to specify *the form* of clauses to be deleted. For example, one of the effects of a robot *goto*-operator must be to delete information about the direction that the robot was originally facing even though such information might not have been represented by one of the parameters of the operator. In this case we would include the atom *facing($)* on the delete list of *goto* with the convention that any atom of the form *facing($)*, regardless of the value of *$*, would be deleted.

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 also be deleted.

We deal with this problem by defining a set of primitive predicates (e.g., *at*, *atr*) 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 that should be deleted when the operator is applied. Also, we require that any nonprimitive 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 *nextto(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 used in a deduction by checking to see if all of the primitive clauses on which the non-primitive clause depends are still in the world model.

3.3. Computing Differences and Relevant Operators

STRIPS uses the GPS strategy of attempting to apply those operators that are relevant to reducing a difference between a world model and a goal or subgoal. We use the theorem prover as a key part of this mechanism.

Suppose we have just created a new node in the search tree represented by $(M, (G_i, G_{i-1}, \ldots, G_0))$. The theorem prover is called to attempt to find a contradiction for the set $\{M \cup \sim G_i\}$. If one can be found, the operator whose precondition was G_i is applied to M and the process continues.

Here, though, we are interested in the case in which no contradiction is obtained after investing some prespecified amount of theorem-proving effort. The uncompleted proof *P* is represented by the set of clauses that form the negation of the goal wff, plus all of their descendants (if any), less any clauses eliminated by editing strategies (such as subsumption and predicate evaluation). We take *P* to be the difference between *M* and G_i and attach *P* to the node.[5]

Later, in attempting to compute a successor to this node with incomplete proof *P* attached, we first must select a relevant operator. The quest for relevant operators proceeds in two steps. In the first step an ordered list of candidate operators is created. The selection of candidate operators is based on a simple comparison of the predicates in the difference clauses with those on the add lists of the operator descriptions. For example, if the difference contained a clause having in it the negation of a position predicate *at*, then the operator *push* would be considered as a candidate for this difference.

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

Note that the consideration of one candidate operator schema may produce several relevant operator instances. For example, if the difference set contains the unit clauses *~atr(LOC_A)* and *~atr(LOC_B)*, then there are two relevant instances of *goto(m, n)*, namely *goto(m,LOC_A)* and *goto(m,LOC_B)*. Each new resolvent that 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 substitutions that were made during the

[5] If P is very large we can heuristically select some part of P as the difference.

production of the resolvent.

3.4. 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 assuming 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 with which we are currently concerned. For these problems most of the wffs in a model describe rooms, walls, doors, and objects, or specify general properties of 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 QA3.5 has been modified to consider only clauses from the memory structure that are marked as visible. Hence, we can "define" a particular world model for QA3.5 by marking that model's clauses visible and all other clauses invisible. When clauses are entered into the initial world model, they are all marked as visible. Clauses that are not changed 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 that are no longer present in the model being defined. The second list, additions, names all those clauses in the model being defined that 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 QA3.5, 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 QA3.5 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 that 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 add list.

3.5. An Example

Tracing through the main points of a simple example helps to illustrate the various mechanisms in STRIPS. Suppose we want a robot to gather together three objects and that the initial world model is given by:

M_0 = {atr(LOC_A), at(BOX1, LOC_B), at(BOX2,LOC_C), at(BOX3,LOC_D)}

The goal wff describing this task is

$$G_0 = (\exists x)\ [at(BOX1,x) \wedge (at(BOX2,x) \wedge at(BOX3,x)]$$

Its negated form is,

$$\sim G_0 = \sim at(BOX1, x) \vee \sim at(BOX2, x) \vee \sim at(BOX3, x)$$

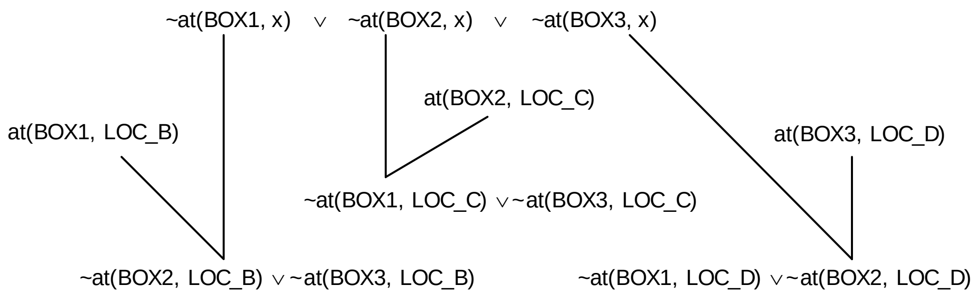
(In $\sim G_0$, the term *x* is a universally quantified variable.)
(**Futher, we shall denote: precondition as pre, delete list as del, add list as add.**)

We admit the following operators:

push (k, m, n) – robot pushes object k from place m to place n.
pre = at(k, m) $\wedge$ atr(m), ~ at(k, m) $\vee$ ~ atr(m)
del = atr(m), at(k, m)
add = at(k, n), atr(n)

goto(m, n) – robot goes from place m to place n.
pre = atr(m) $\vee$ ~ atr(m)
del = atr(m)
add = atr(n).

Following the flow chart of Fig. 2, STRIPS first creates the initial node *(M_0 , (G_0))* and attempts to find a contradiction to *{M $\cup$ ~G_0}*. This attempt is unsuccessful; suppose the incomplete proof is:


~at(BOX1, x) ∨ ~at(BOX2, x) ∨ ~at(BOX3, x)
at(BOX2, LOC_C)
at(BOX1, LOC_B)
at(BOX3, LOC_D)
~at(BOX1, LOC_C) ∨~at(BOX3, LOC_C)
~at(BOX2, LOC_B) ∨~at(BOX3, LOC_B)
~at(BOX1, LOC_D) ∨~at(BOX2, LOC_D)

We attach this incomplete proof to the node and then select the node to have a successor computed.

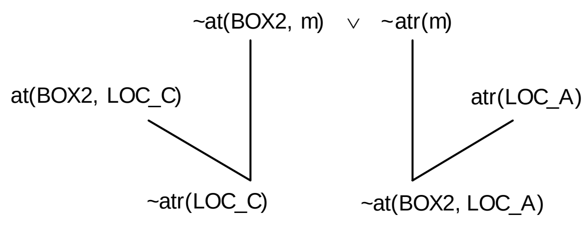
The only candidate operator is *push(k, m, n)*. Using the add list clause *at(k, n)*, we can continue the uncompleted proof in one of several ways depending on the substitutions made for *k* and *n*. Each of these substitutions produces a relevant instance of *push*. One of these is:

$$OP_1 = push(BOX2, m, LOC\_B)$$

given by the substitutions *BOX2* for *k* and *LOC_B* for *n*. Its associated precondition (in negated form) is:

$$\sim G_1 = \sim at(BOX2, m) \vee \sim atr(m).$$

Suppose **OP_1**, is selected and used to create a successor node. (Later in the search process another Successor using one of the other relevant instances of *push* might be computed if our original selection did not lead to a solution.) Selecting **OP_1** leads to the computation of the successor node **$(M_0, (G_1, G_0))$**. STRIPS next attempts to find a contradiction for **$\{M_0 \cup \sim G_1\}$**. The uncompleted proof (difference) attached to the node contains:



When this node is later selected to have a successor computed, one of the candidate operators is *goto(m, n)*. The relevant instance is determined to be:

$$OP_2 = \text{goto}(m, c) \text{ with (negated) precondition } \sim G_2 = \text{atr}(m).$$

This relevant operator results in the successor node **$(M_0, (G_2, G_1, G_0))$**.

Next STRIPS determines that **$\{M_0 \cup \sim G_2\}$** is contradictory with *m = LOC_A*. Thus, STRIPS applies the operator *goto(LOC_A, LOC_C)* to **M_0** to yield:

$$M_1=\{\text{atr(LOC\_C), at(BOX1,LOC\_B), at(BOX2,LOC\_C), at(BOX3,LOC\_D)}\}.$$

The successor node is **$(M_1,(G_1,G_0))$**. Immediately, STRIPS determines that **$\{M_1 \cup \sim G_0\}$** is contradictory with *m = LOC_C*. Thus, STRIPS applies the operator *push(BOX2, LOC_C, LOC_B)* to yield:

$$M_1=\{\text{atr(LOC\_B), at(BOX1,LOC\_B), at(BOX2,LOC\_B), at(BOX3,LOC\_D)}\}.$$

The resulting successor node is **$(M_2, (G_0))$**, and thus STRIPS reconsiders the original problem but now beginning with world model **M_2**. The rest of the solution proceeds in similar fashion.

Our implementation of STRIPS easily produces the solution:

(goto(LOC_A,LOC_C), push(BOX2,LOC_C,LOC_B), goto(LOC_B,LOC_D), push(BOX3, LOC_D, LOC_B)).

(Incidentally, Green's theorem-proving problem-solver [4] has not been able to obtain a solution to this version of the 3-Boxes problem. It did solve a simpler version of the problem designed to require only two operator applications.)

4. EXAMPLE PROBLEMS SOLVED BY STRIPS

STRIPS has been designed to be a general-purpose problem solver for robot tasks, and thus must be able to work with a variety of operators and with a world model containing a large number of facts and relations. This section describes its performance on three different tasks.

The initial world model for all three tasks consists of a corridor with four rooms and doorways (see Fig. 3).

Initially, the robot is in *ROOM1* at location *LOC_E*. Also in *ROOM1* are three boxes and a lightswitch: *BOX1* at location *LOC_A*, *BOX2* at location *LOC_B*, and *BOX3* at location *LOC_C*; and a lightswitch, *LIGHTSWITCH1* at location *LOC_D*. The lightswitch is high on a wall out of normal reach of the robot.

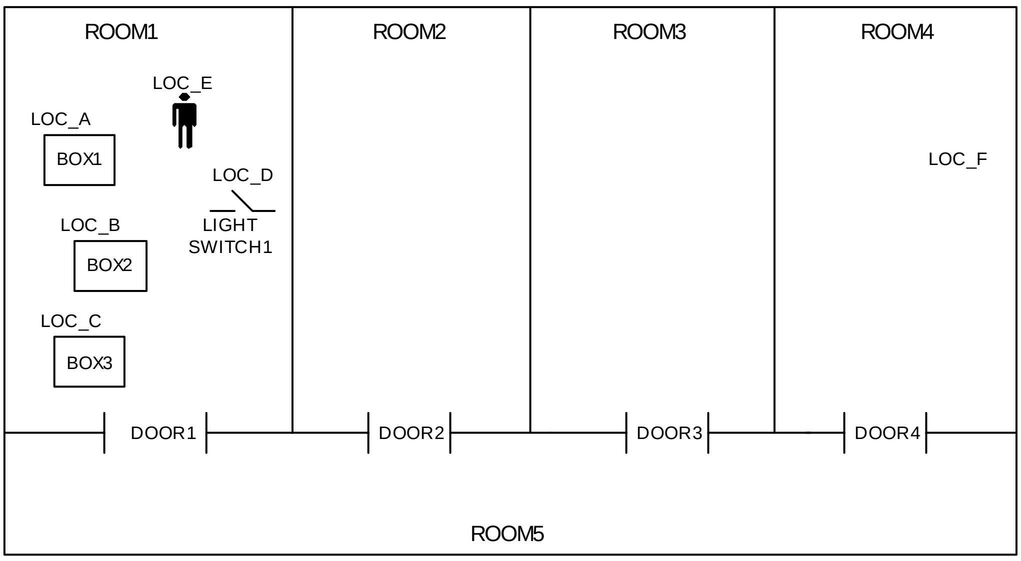


Fig. 3. Room plan for the robot tasks

Formally, the **initial world model** is described by the list of axioms:

(∀x∀y∀z)[connects(x,y,z) => connects(x,z,y)],
connects(DOOR1,ROOM1,ROOM5),
connects(DOOR2,ROOM2,ROOM5),
connects(DOOR3,ROOM3,ROOM5),
connects(DOOR4,ROOM4,ROOM5),
locinroom(LOC_F,ROOM4),
at(BOXl,LOC_A), at(BOX2,LOC_B),
at(BOX3,LOC_C),
at(LIGHTSWITCH1,LOC_D),
atrobot(LOC_E),
type(BOX1,BOX), type(BOX2,BOX),
type(BOX3,BOX),
type(D1,DOOR), type(D2,DOOR),
type(D3,DOOR), type(D4,DOOR),
type(LIGHTSW1TCH1,LIGHTSWITCH),
inroom(BOX1,ROOM1),
inroom(BOX2,ROOM1),
inroom(BOX3,ROOM1),
inroom(ROBOT,ROOM1),
inroom(LIGHTSWITCH1,ROOM1),
pushable(BOX1), pushable(BOX2),
pushable(BOX3),
onfloor,
status(LIGHTSWITCH1,OFF)

At the input of STRIPS the finite set of operators is given.

For convenience we define two *goto* operators, *goto1* and *goto2*. The operator *goto1(m)* takes the robot to any coordinate location *m* in the same room as the robot. The operator *goto2(m)* takes the robot next to any item *m* (e.g., lightswitch, door, or box) in the same room as the robot. The operator *pushto(m,n)* pushes any pushable object *m* next to any item *n* (e.g., lightswitch, door or box) in the same room as the robot. Additionally, we have operators for turning on lightswitches, going through doorways, and climbing on and off boxes.

The precise formulation of the preconditions and the effects of these **operators**:

goto1(m) – robot goes to coordinate location m

pre = (onfloor) $\wedge$ ($\exists$x)[inroom(ROBOT,x) $\wedge$ locinroom(m,x)]
del = {atrobot($), nextto(ROBOT,$)}
add = {atrobot(m)}

goto2(m) – robot goes next to item m.
pre = (onfloor) $\wedge$ [($\exists$x)[inroom(ROBOT,x) $\wedge$ inroom(m,x)] $\vee$
($\exists$x)($\exists$y) [inroom(ROBOT,x) $\wedge$ connects(m,x,y)]]
del = {atrobot($), nextto(ROBOT,$)}
add = {nextto(ROBOT,m)}

pushto(m,n) – robot pushes object m next to item n
pre = pushable(m) $\wedge$ onfloor $\wedge$ nextto(ROBOT,m) $\wedge$
(($\exists$x)[inroom(m,x) $\wedge$ inroom(n,x)] $\vee$ ($\exists$x,$\exists$y)[inroom(m,x) $\wedge$ connects(n,x,y)])
del = {atrobot($), nextto(ROBOT,$), nextto($,m), at(m$), nextto (m$)}
add = {nextto(m,n), nextto(n,m), nextto(ROBOT,m)}

turnonlight(m) – robot turns on lightswitch m.
pre = (($\exists$n)[type(n,BOX) $\wedge$ on(ROBOT,n) $\wedge$ nexto(n,m)]) $\wedge$ type(m,LIGHTSWITCH)
del = {status(m,OFF)}
add = {status(m,ON)}

climbonbox(m) – robot climbs up on box m.
pre = onfloor $\wedge$ type(m,BOX) $\wedge$ nextto(ROBOT,m)
del = {atrobot($), onfloor}
add = {on(ROBOT,m)}

climboffbox(m) – robot climbs off box m.
pre = type(m,BOX) $\wedge$ on(ROBOT,m)
del = {on(ROBOT,m)}
add = {onfloor}

gothrudoor (k,l,m) – robot goes through door k from room l into room m.
pre = nextto(ROBOT,k) $\wedge$ connects(k,l,m) $\wedge$ inroom(ROBOT,l) $\wedge$ onfloor
del = {atrobot($), nextto(ROBOT,$), inroom(ROBOT,$)}
add = {inroom(ROBOT,m)}

The first robot task is to turn on the lightswitch. The robot can solve this problem by going to one of the three boxes, pushing it to the lightswitch, climbing on the box[6] and turning on the lightswitch.

The second task is to push the three boxes in *ROOM1* together. This task is a more realistic elaboration of the three-box problem used as an example in the last section.

The third task is for the robot to go to a designated location, *LOC_F*, in *ROOM4*.

The **goal wffs** for the three tasks and the **solutions** obtained by STRIPS are following:

[6] This task is a robot version of the so-called "Monkey and Bananas" problem. STRIPS can solve the problem even though the current SRI robot is incapable of climbing boxes and turning on lightswitches.

1. Turn on the lightswitch

Goal = status(LIGHTSWITCH1,ON)
Plan = (goto2(BOX1),climbonbox(BOX1),climboffbox(BOX1),
pushto(BOX1,LIGHTSWITCH1),climbonbox(BOX1),
turnonlight(LIGHTSWITCH1))

2. Push three boxes together

Goal = nextto(BOX1,BOX2) ∧ nextto(BOX2,BOX3)
Plan= (goto2(BOX2), pushto(BOX2,BOX1), goto2(BOX3), pushto (BOX3,BOX2))

3. Go to a location in another room

Goal = atrobot(LOC_F)
Plan = (goto2(DOOR1), gothrudoor(DOOR1,ROOM1,ROOM5),
goto2(DOOR4), gothrudoor(DOOR4,ROOM5,ROOM4), goto1(LOC_F))

Some performance figures for these solutions are shown in Table.

Table. Performance of STRIPS on Three Tasks

	Time taken (in seconds)		Number of nodes		Number of operator applications	
	Total	Theorem-proving	On solution path	In search tree	On solution path	In search tree
Turn on the lightswitch	113.1	83.0	13	21	6	6
Push three boxes together	66.0	49.6	9	9	4	4
Go to a location in another room	123.0	104.9	11	12	5	5

In Table, the figures in the "Time Taken" column represent the CPU time (excluding garbage collection) used by STRIPS in finding a solution. Although some parts of our program are compiled, most of the time is spent running interpretive code; hence, we do not attach much importance to these times. We note that in all cases most of the time is spent doing theorem proving (in QA3.5).

The next columns of Table indicate the number of nodes generated and the number of operator applications both in the search tree and along the solution path. (Recall from Fig. 2 that some successor nodes do not correspond to operator applications.) We see from these figures that the general search heuristics built into STRIPS provide a highly directed search toward the goal. These heuristics presently give the search a large "depth-first" component, and for this reason STRIPS obtains an interesting but non-optimal solution to the "turn on the light-switch" problem.

5. FUTURE PLANS AND PROBLEMS

The current implementation of STRIPS 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 mention some of these briefly.

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 node to work on next. Currently, we use an evaluation function that incorporates such factors- as the number and the estimated difficulty of the remaining subgoals, the cost of the operators applied so far, and the complexity of the current difference. 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 the 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.

We are also interested in getting STRIPS to “learn” by having it define new operators for itself on the basis of previous problem solutions. These new operators could then be used to solve even more difficult problems. It would be important to be able to generalize to parameters any constants appearing in a new operator; otherwise, the new operator would not be general enough to warrant saving. One approach [10] that appears promising is to modify STRIPS so that it solves every problem presented to it in terms of generalized parameters rather than in terms of constants appearing in the specific problem statements. Hewitt [11] discusses a related process that he calls “procedural abstraction”. He suggests that, from a few instances of a procedure, a general version can sometimes be synthesized.

This type of learning provides part of our rationale for working on automatic problem solvers such as STRIPS. Some researchers have questioned the value of systems for automatically chaining together operators into higher-level procedures that themselves could have been “hand-coded” quite easily in the first place. Their viewpoint seems to be that a robot system should be provided a priori with a repertoire of all of the operators and procedures that it will ever need.

We agree that it is desirable to provide a priori a large number of specialized operators, but such a repertoire will nevertheless be finite. To accomplish tasks just outside the boundary of a priori abilities requires a process for chaining together existing operators into more complex ones. We are interested in a system whose operator repertoire can “grow” in this fashion.

Clearly one must not give such a system a problem too far away from the boundary of known abilities, because the combinatorics of search will then make a solution unlikely. However, a truly “intelligent” system ought always to be able to solve slightly more difficult problems than any it has solved before.

ACKNOWLEDGEMENT

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.

The research reported herein was sponsored by the Advanced Research Projects Agency and the National Aeronautics and Space Administration under Contract NAS12-

2221.

REFERENCES

1. Nilsson, N. J. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Book Company, New York, New York, 1971.

2. Munson, J. H. Robot planning, execution, and monitoring in an uncertain environment. *Proc. 2nd Int'l. Joint Conf. Artificial Intelligence*, London, England (September 13**,** 1971).

3. Fikes, R. E. Monitored execution of robot plans produced by STRIPS. *Proc. IFIP 71*, Ljubljana, Yugoslavia (August 1971).

4. Green, C. Application of theorem proving to problem solving. *Proc. Int'l. Joint Conf. Artificial Intelligence*, Washington, D.C. (May 1969).

5. Raphael, B. The frame problem in problem-solving systems. *Proc. Adv. Study Inst. on Artificial Intelligence and Heuristic Programming*, Menaggio, Italy (August 1970).

6. Ernst, G. and Newell, A. *GPS: A Case Study in Generality and Problem Solving.* ACM Monograph Series. Academic Press, New York, New York, 1969.

7. Garvey, T. and Kling, R. User's guide to QA3.5 Question-Answering System. Stanford Research Institute Artificial Intelligence Group Technical Note 15, Menlo Park, California (December 1969).

8. Waldinger, R. and Lee, R. PROW: A step toward automatic program writing. *Proc. Int'l. Conf. Artificial Intelligence*, Washington, D.C. (May 1969).

9. Manna, Z. and Waldinger, R. Towards automatic program synthesis. *Comm. ACM. 14*, No. 3 (March 1971).

10. Hart, P. E. and Nilsson, N. J. The construction of generalized plans as an approach toward learning. Stanford Research Institute Artificial Intelligence Group Memo, Menlo Park, California (5 April 1971).

11. Hewitt, C. PLANNER: A language for Manipulating models and proving theorems in a robot. Artificial Intelligence Memo No. 168 (Revised), Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts (August 1970).