Learning and Executing Generalized Robot Plans[1]

Richard E. Fikes, Peter E. Hart and Nils J. Nilsson

Stanford Research Institute, Menlo Park, California 94025

ABSTRACT

In this paper we describe some major new additions to the STRIPS robot problem-solving system. The first addition is a process for generalizing a plan produced by STRIPS so that problem-specific constants appearing in the plan are replaced by problem-independent parameters.

The generalized plan, stored in a convenient format called a triangle table, *has two important functions. The more obvious function is as a single macro action that can be used by STRIPS—either in whole or in part—during the solution of a subsequent problem. Perhaps less obviously, the generalized plan also plays a central part in the process that monitors the real-world execution of a plan, and allows the robot to react "intelligently" to unexpected consequences of actions.*

We conclude with a discussion of experiments with the system on several example problems.

1. Introduction

In this paper we describe a system of computer programs for controlling a mobile robot. This system can conceive and execute plans enabling the robot to accomplish certain tasks such as pushing boxes from one room to another in a simple but real environment. Although these sorts of tasks are commonly thought to demand little skill or intelligence, they pose important conceptual problems and can require quite complex planning and execution strategies.

In previous papers, we described two important components of our robot system, namely, STRIPS [1] and PLANEX [2]. When a task statement is given to the robot, STRIPS produces a plan consisting of a sequence of preprogrammed actions, and PLANEX supervises the execution of this sequence to accomplish the task. In this paper we present a major new addition to the original capabilities of STRIPS and PLANEX that enables the system to generalize and then save a solution to a particular problem. This generalization capability is used in two ways. The more obvious use of a generalized plan is as a "macro action" that can be used as a single component of a new plan to solve a new problem. When used in this fashion, generalization becomes a powerful form of learning that can reduce the planning time for similar tasks as well as allow the formation of much longer plans, previously beyond the combinatoric capabilities of STRIPS.

The second use of generalized plans involves the supervision or monitoring of plan execution. Often, a real-world robot must reexecute a portion of its plan because of some failure that occurred during the first attempt at execution. At such a time, the system has more flexibility if it is not restricted to repeating identically the unsuccessful portion of the plan, but instead can reexecute the offending actions with different arguments.

Before getting into details (and defining just what we mean by *generalize*), we present in outline form a scenario that illustrates some of the capabilities of the system. Suppose we give a robot the task "Close window WIND1 and turn off light LITE1."[2] To accomplish this, let us say that the robot decides to push box BOX1 to window WIND1, climb BOX1 in order to close the window, and then proceed to turn off light LITE1. First, the system generalizes this specific plan to produce a plan that can, under certain specified conditions, close an arbitrary window (not just WIND1) and turn off an arbitrary light. Next, the system applies the appropriate version of this generalized plan to the specific problem at hand, namely, "close WIND1 and turn off LITE1." While executing the appropriate version, let us suppose that the robot fails to push BOX1 to the window because, say, it discovers another box is already under the window. The PLANEX supervisor will recognize that this new box will serve the purpose that BOX1 was to serve, and the plan execution will proceed.

Now let us suppose that, after finishing the first task, the robot is given a new problem, "Close window WIND5 and lock door DOOR1." The system is capable of recognizing that a portion of the old generalized plan can help solve the new task. Thus, the sequence of several component actions needed to close the window can be readily obtained as a single macro action, and the planning time required to solve the new problem thereby reduced.

We shall begin with a brief review of the problem-solving program STRIPS. Then we shall review a novel format for storing plans that conveniently

[1] The research reported herein was supported at SRI by the Advance Research Projects Agency of the Department of Defense, monitored by the U.S. Army Research Office-Durham under Contract DAHC04 72 C 0008.

[2] The scenario is imaginary; our robot cannot actually turn off light switches or close windows.

allows most of the legitimate $2^n - 1$ subsequences of an n-step plan to be extracted as a unit in a subsequent planning activity. We then describe a process by which constants appearing in the plan can be converted to parameters so that each plan can handle a family of different tasks. Thus generalized, the plan can be stored (i.e., learned) for future use. Next, we review the operation of PLANEX and discuss how generalized plans are used during execution to increase the system's capabilities for responding to unplanned-for situations. Finally, we discuss how STRIPS uses stored plans to compose more complex ones and describe some experiments with a sequence of learning tasks.

2. Summary of Strips

2.1. Description

Because STRIPS is basic to our discussion, let us briefly outline its operation. (For a complete discussion and additional examples, see [1].) The primitive actions available to the robot vehicle are precoded in a set of action routines. For example, execution of the routine GOTHRU(D1,R1,R2) causes the robot vehicle actually to go through the doorway D1 from room R1 to room R2. The robot system keeps track of where the robot vehicle is and stores its other knowledge of the world in a model [3] composed of well-formed formulas (wffs) in the predicate calculus. Thus, the system knows that there is a doorway D1 between rooms R1 and R2 by the presence of the wff CONNECTS-ROOMS(D1,R1,R2) in the model.

Tasks are given to the system in the form of predicate calculus wffs. To direct the robot to go to room R2, we pose for it the goal wff INROOM(ROBOT,R2). The planning system, STRIPS, then attempts to find a sequence of primitive actions that would change the world in such a way that the goal wff is true in the correspondingly changed model. In order to generate a plan of actions, STRIPS needs to know about the effects of these actions; that is, STRIPS must have a model of each action. The model actions are called *operators* and, just as the actions change the world, the operators transform one model into another. By applying a sequence of operators to the initial world model, STRIPS can produce a sequence of models (representing hypothetical worlds) ultimately ending in a model in which the goal wff is true. Presumably then, execution of the sequence of actions corresponding to these operators would change the world to accomplish the task.

Each STRIPS operator must be described in some convenient way. We characterize each operator in the repertoire by three entities: an *add list*, a *delete list*, and a *precondition wff*. The meanings of these entities are straightforward. An operator is applicable to a given model only if its precondition wff is satisfied in that model. The effect of applying an (assumed applicable) operator to a given model is to delete from the model all those clauses specified by the delete list and to add to the model all those clauses specified by the add list. Hence, the add and delete lists prescribe how an operator transforms one state into another.

Within this basic framework STRIPS operates in a GPS-like manner [6]. First, it tries to establish that a goal wff is satisfied by a model. (STRIPS uses the QA3 resolution-based theorem prover [3] in its attempts to prove goal wffs.) If the goal wff cannot be proved, STRIPS selects a "relevant" operator that is likely to produce a model in which the goal wff is "more nearly" satisfied. In order to apply a selected operator the precondition wff of that operator must of course be satisfied; this precondition becomes a new subgoal and the process is repeated. At some point we expect to find that the precondition of a relevant operator is already satisfied in the current model. When this happens the operator is *applied*; the initial model is transformed on the basis of the add and delete lists of the operator, and the model thus created is treated in effect as a new initial model of the world.

To complete our review of STRIPS we must indicate how relevant operators are selected. An operator is needed only if a subgoal cannot be proved from the wffs defining a model. In this case the operators are scanned to find one whose effects would allow the proof attempt to continue. Specifically, STRIPS searches for an operator whose add list specifies clauses that would allow the proof to be successfully continued (if not completed). When an add list is found whose clauses do in fact permit an adequate continuation of the proof, then the associated operator is declared relevant; moreover, the substitutions used in the proof continuation serve to instantiate at least partially the arguments of the operator. Typically, more than one relevant operator instance will be found. Thus, the entire STRIPS planning process takes the form of a tree search so that the consequences of considering different relevant operators can be explored. In summary, then, the "inner loop" of STRIPS works as follows:

(1) Select a subgoal and try to establish that it is true in the appropriate model. If it is, go to Step 4. Otherwise:

(2) Choose as a relevant operator one whose add list specifies clauses that allow the incomplete proof of Step 1 to be continued.

(3) The appropriately instantiated precondition wff of the selected operator constitutes a new subgoal. Go to Step 1.

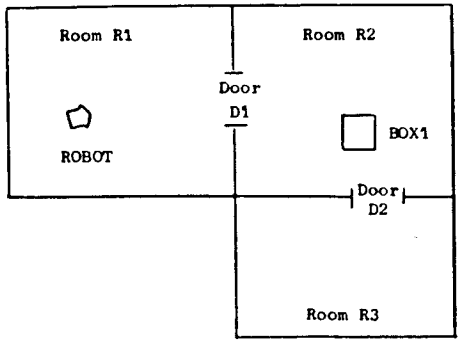
(4) If the subgoal is the main goal, terminate. Otherwise, create a new model by applying the operator whose precondition is the subgoal just established. Go to Step 1.

The final output of STRIPS, then, is a list of instantiated operators whose corresponding actions will achieve the goal.

[3] Our use of the word "model" is consistent with customary terminology in Artificial Intelligence. We hope there will be no confusion between our use of the word and its technical definition in logic, namely an interpretation for a set of formulas.

2.2. An Example

An understanding of STRIPS is greatly aided by an elementary example. The following example considers the simple task of fetching a box from an adjacent room. Let us suppose that the initial state of the world is as shown below:



Initial Model

Mo: INROOM(ROBOT,R1)
CONNECTS(D1,R1,R2)
CONNECTS(D2,R2,R3)
BOX(BOX1)
INROOM(BOX1,R2)

.
.
.

$(\forall x\ \forall y\ \forall z)$[CONNECTS$(x,y,z)\Rightarrow$CONNECTS$(x,z,y)$]

Goal wff

Go: $(\exists x)$[BOX$(x)\ \wedge$ INROOM$(x$,R1)]

We assume for this example that models can be transformed by two operators GOTHRU and PUSHTHRU, having the descriptions given below. Each description specifies an *operator schema* indexed by schema variables. We will call schema variables *parameters*, and denote them by strings beginning with lower-case letters. A particular member of an operator schema is obtained by instantiating all the parameters in its description to constants. It is a straightforward matter to modify a resolution theorem prover to handle wffs containing parameters [1], but for present purposes we need only know that the modification ensures that each parameter can be bound only to one constant; hence, the operator arguments (which may be parameters) can assume unique values. (In all of the following we denote constants by strings beginning with capital letters and quantified variables by x, y or z):

GOTHRU(d,r1,r2)

(Robot goes through Door d from Room r1 into Room r2.)

Precondition wff

INROOM(ROBOT,r1) $\wedge$ CONNECTS(d,r1,r2)

Delete List

INROOM(ROBOT,$)

(Our convention here is to delete any clause containing a predicate of the form INROOM(ROBOT,$) for any value of $.)

Add List

INROOM(ROBOT,r2)

PUSHTHRU(b,d,r1,r2)

(Robot pushes Object *b* through Door d from Room r1 into Room r2.)

Precondition wff

INROOM(b,r1) $\wedge$ INROOM(ROBOT,r1) $\wedge$ CONNECTS(d,r1,r2)

Delete List

INROOM(ROBOT,$)
INROOM(b,$)

Add List

INROOM(ROBOT,r2)
INROOM(b,r2).

When STRIPS is given the problem it first attempts to prove the goal G_0 from the initial model M_0. This proof cannot be completed; however, were the model to contain other clauses, such as INROOM(BOX1,R1), the proof attempt could continue. STRIPS determines that the operator PUSHTHRU can provide the desired clause; in particular, the partial instance PUSHTHRU (BOX1,d,r1,R1) provides the wff INROOM(BOX1,R1).

The precondition G_1 for this instance of PUSHTHRU is

G_1: INROOM(BOX1,r1)
$\wedge$ INROOM(ROBOT,r1)
$\wedge$ CONNECTS(d,r1,R1).

This precondition is set up as a subgoal and STRIPS tries to prove it from M_0.

Although no proof for G_1 can be found, STRIPS determines that if r1 = R2 and d = D1, then the proof of G_1 could continue were the model to contain INROOM(ROBOT,R2). Again STRIPS checks operators for one whose effects could continue the proof and settles on the instance GOTHRU(d,r1,R2). Its precondition is the next subgoal, namely:

G_2: INROOM(ROBOT,r1)
$\wedge$ CONNECTS(d,r1,R2).

STRIPS is able to prove G_2 from M_0, using the substitutions r1 = R1 and d = D1. It therefore applies GOTHRU(D1,R1,R2) to M_0 to yield:

M_1: INROOM(ROBOT,R2)
CONNECTS(D1,R1,R2)
CONNECTS(D2,R2,R3)
BOX(BOX1)
INROOM(BOX1,R2)
.
.
.
$(\forall x\ \forall y\ \forall z)$[CONNECTS$(x,y,z)\Rightarrow$CONNECTS$(x,z,y)$].

Now STRIPS attempts to prove the subgoal G_1 from the new model M_1. The proof is successful with the instantiations r1 = R2, d = D1. These substitutions yield the operator instance PUSHTHRU(BOX1,D1,R2,R1), which applied to M_1 yields

M_2: INROOM(ROBOT,R1)
CONNECTS(D1,R1,R2)
CONNECTS(D1,R2,R3)
BOX(BOX1)
INROOM(BOX1,R1)
.
.
.
$(\forall x\ \forall y\ \forall z)$[CONNECTS$(x,y,z)\Rightarrow$CONNECTS$(x,z,y)$].

Next, STRIPS attempts to prove the original goal, G_0, from M_2. This attempt is successful and the final operator sequence is

GOTHRU(D1,R1,R2)
PUSHTHRU(BOX1,D1,R2,R1).

We have just seen how STRIPS computes a specific plan to solve a particular problem. The next step is to generalize the specific plan by replacing constants by new parameters. In other words, we wish to elevate our particular plan to the status of a plan schema, or macro operator, analogous to the primitive operators we were given initially. Moreover, we would like to store a macro operator in such a way as to make any of its legitimate subsequences also available to STRIPS. In the next section we describe a storage format, called a *triangle table*, that has this property. Our procedure for plan generalization will be explained after we have discussed triangle tables and their properties.

3. Triangle Tables

Suppose STRIPS has just computed a plan consisting of the sequence of n operators $OP_1, OP_2, \ldots, OP_n$. In what form should this plan be presented to PLANEX, the system responsible for monitoring the execution of plans? In what form should it be saved? For purposes of monitoring execution, PLANEX needs at every step to be able to answer such questions as

(a) Has the portion of the plan executed so far produced the expected results?

(b) What portion of the plan needs to be executed next so that after its execution the task will be accomplished?

(c) Can this portion be executed in the current state of the world?

Also, for purposes of saving plans so that portions of them can be used in a later planning process, we need to know the preconditions and effects of any portion of the plan.

If we are to have efficient methods for answering Questions (a)–(c), we must store a plan in a way that plainly reveals its internal structure. In particular, we must be able to identify the role of each operator in the overall plan: what its important effects are (as opposed to side effects) and why these effects are needed in the plan. To accomplish this, we decided to store plans in a tabular form called a *triangle table*.[4]

A triangle table is a lower triangular array where rows and columns correspond to the operators of the plan.

An example of a triangle table is shown in Fig. 1. (The reader may temporarily ignore the heavily outlined rectangle.) The columns of the table, with the exception of Column zero, are labelled with the names of the operators of the plan, in this example $OP_1, \ldots, OP_4$. For each Column i, $i = 1, \ldots, 4$, we place in the top cell the add list A_i of operator OP_i. Going down the ith column, we place in consecutive cells the portion of A_i that survives the application of subsequent operators. Thus, $A_{1\ 2}$ denotes those clauses in A_1 not deleted by OP_2; $A_{1/2,3}$ denotes those clauses in $A_{1/2}$ not deleted by OP_3, and so forth. Thus, the ijth cell of the matrix contains those wffs added by the jth operator that are still true at the time of application of the ith operator.

We can now interpret the contents of the ith row of the table, excluding the left-most column. Since each cell in the ith row (excluding the left-most) contains statements added by one of the first $(i-1)$ operators but not deleted by any of those operators, we see that the union of the cells in the ith row (excluding the left-most) specifies the add list obtained by applying the $(i-1)$st *head* of the plain; i.e., by applying in sequence $OP_1, \ldots, OP_{i-1}$. We denote by $A_{1,\ldots,j}$ the add list achieved by the first j operators applied in sequence. The union of the cells in the bottom row of a triangle table evidently specifies the add list of the complete sequence.

The left-most column of the triangle table, which we have thus far ignored, is involved with the preconditions for the stored plan. During the formation

[4] We are indebted to John Munson who prompted us to try a tabular format.

of the plan, STRIPS produced a proof of each operator's preconditions from the model to which the operator was applied. We will define the set of clauses used to prove a formula as the *support* of that formula. We wish to ensure that the ith row of a triangle table contains all the wffs in the support of the preconditions for Operator i. In general, some clauses in the support for Operator i will have been added by the first $i-1$ operators in the plan and will therefore be included in Row i, as described in the previous paragraphs.

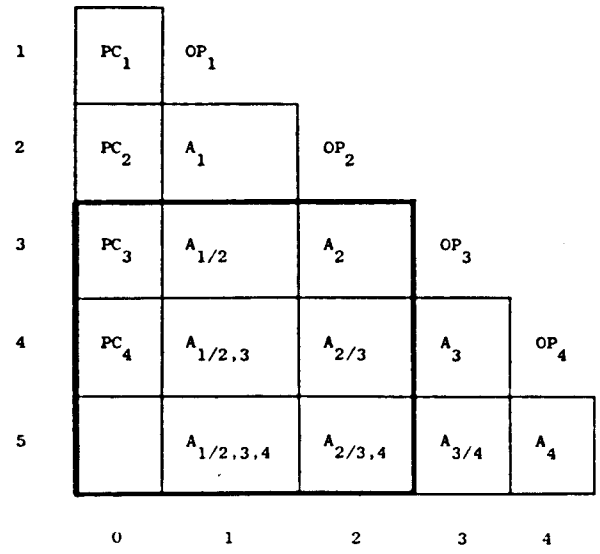


FIG. 1. A triangle table.

The remainder of the support clauses appeared in the initial model and were not deleted by any of the first $i-1$ operators. These clauses, which we denote by PC_i, are precisely the clauses that are entered into the left-most (Column 0) cell of Row i. Hence, we see that Column 0 of a triangle table contains those clauses from the initial model that were used in the precondition proofs for the plan. It is convenient to flag the clauses in each Row i that are in the support for Operator i and hereafter speak of them as *marked* clauses; by construction, all clauses in Column 0 are marked. Note that in proving the preconditions of operators, STRIPS must save the support clauses so that the triangle table can be constructed.

As an example, we show in Fig. 2 the triangle table for the plan discussed in the previous section. The clauses that are marked by an asterisk "*" were all used in the proofs of preconditions.

We have seen how the marked clauses on Row i constitute the support of the preconditions for the ith operator. Let us now investigate the preconditions for the ith *tail* of the plan—that is, the preconditions for applying the operator sequence $OP_i, OP_{i+1}, \ldots, OP_n$. The key observation here is that the ith tail is applicable to a model if the model already contains that portion of the support of each operator in the tail that is not supplied within the tail itself. This observation may be formulated more precisely by introducing the notion of a *kernel* of a triangle table. We define the ith kernel of a table to be the unique rectangular subarray containing the lower left-most cell and Row i. We assert now that the ith tail of a plan ' applicable to a model if all the marked clauses in the ith kernel are true in that model. Let us see by example why this is so.

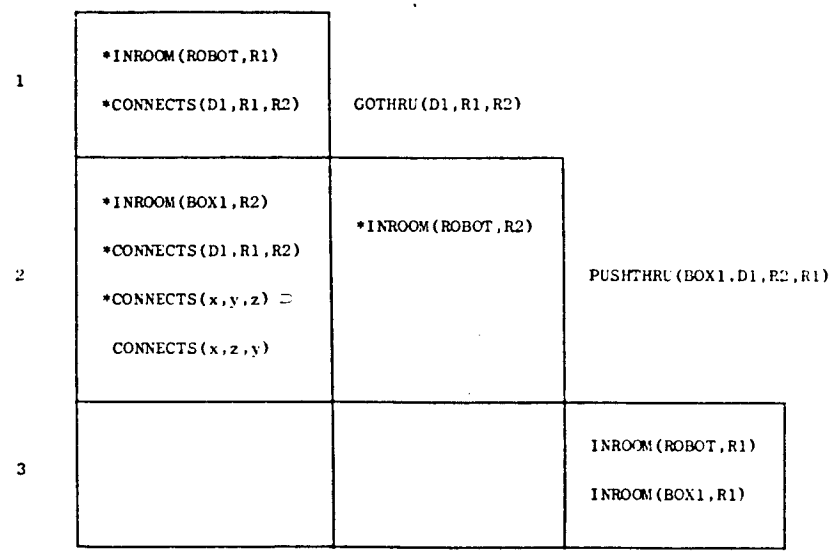


FIG. 2. Triangle table for example plan.
(A "*" preceding a clause indicates a "marked" clause.)

Consider again Fig. 1, in which we have heavily outlined Kernel 3. Let us assume that all marked clauses in this kernel are true in the current model. (When all the marked clauses in a kernel are true, we shall say that the kernel is true.) Certainly, OP_3 is applicable; the marked clauses in Row 3 are true, and these marked clauses support the proof of the preconditions of OP_3. Suppose now that OP_3 is applied to the current model to produce a new model in which A_3, the set of clauses added by OP_3, is true. Evidently, OP_4 is now applicable, since all the marked clauses in Row 4 are true; those clauses within the outlined kernel were true before applying OP_3 (and by construction of the triangle table are still true), and those outside the kernel (that is, A_3)

are true because they were added by OP_3. Thus, the truth of the marked clauses in Kernel 3 is a sufficient condition for the applicability of the tail of the plan beginning with OP_3.

We have some additional observations to make about triangle tables before moving on to the matter of plan generalization. First, notice that Kernel 1—that is, the left-most column of a triangle table—constitutes a set of sufficient conditions for the applicability of the entire plan. Thus, we can take the conjunction of the clauses in Column 0 to be a precondition formula for the whole plan.

A second observation may help the reader gain a little more insight into the structure of triangle tables. Consider again the table of Fig. 1, and let us suppose this time that Kernel 2 is true. Since Kernel 2 is true, the sequence OP_2, OP_3, OP_4 is applicable. Upon applying OP_2, which is immediately applicable because the marked clauses in Row 2 are true, we effectively add Column 2 to the table. Moreover, we lose interest in Row 2 because OP_2 has already been applied. Thus the application of OP_2 transforms a true Kernel 2 into a true Kernel 3, and the application of the operators in the tail of the plan can continue.

4. Generalizing Plans

4.1. Motivation

The need for plan generalization in a learning system is readily apparent. Consider the specific plan produced in the example of Section 2:

GOTHRU(D1,R1,R2)
PUSHTHRU(BOX1,D1,R2,R1).

While this sequence solves the original task, it probably doesn't warrant being saved for the future unless, of course, we expect that the robot would often need to go from Room R1 through Door D1 to Room R2 to push back the specific box, BOX1, through Door D1 into Room R1. We would like to generalize the plan so that it could be free from the specific constants, D1, R1, R2, and BOX1 and could be used in situations involving arbitrary doors, rooms, and boxes.

In considering possible procedures for generalizing plans we must first reject the naive suggestion of merely replacing each constant in the plan by a parameter. Some of the constants may really need to have specific values in order for the plan to work at all. For example, consider a modification of our box-fetching plan in which the second step of the plan is an operator that *only* pushes objects from room R2 into room R1. The specific plan might then be

GOTHRU(D1,R1,R2)
SPECIALPUSH(BOX1).

When we generalize this plan we cannot replace all constants by parameters, since the plan only works when the third argument of GOTHRU is R2. We would want our procedure to recognize this fact and produce the plan

GOTHRU(d1,r1,R2)
SPECIALPUSH(b1).

Another reason for rejecting the simple replacement of constants by parameters is that there is often more generality readily available in many plans than this simple procedure will extract. For example, the form of our box-pushing plan, GOTHRU followed by PUSHTHRU, does not require that the room in which the robot begins be the same room into which the box is pushed. Hence the plan could be generalized as follows:

GOTHRU(d1,r1,r2)
PUSHTHRU(b,d2,r2,r3)

and be used to go from one room to an adjacent second room and push a box to an adjacent third room.

Our plan-generalization procedure overcomes these difficulties by taking into account the internal structure of the plan and the preconditions of each operator. The remainder of this section is a description of this generalization procedure.

4.2. The Generalization Procedure

The first step in our generalization procedure is to "lift" the triangle table to its most general form as follows: We first replace every occurrence of a constant in the clauses of the left-most column by a new parameter. (Multiple occurrences of the same constant are replaced by distinct parameters.) Then the remainder of the table is filled in with appropriate add clauses assuming completely uninstantiated operators (i.e., as these add clauses appear in the operator descriptions), and assuming the same deletions as occurred in the original table. As an example, Fig. 3 shows the table from Fig. 2 in its most general form.

The lifted table thus obtained is too general; we wish to constrain it so that the marked clauses in each row support the preconditions of the operator on that row, while retaining the property that the lifted table has the original table as an instance. To determine the constraints we redo each operator's precondition proof using the support clauses in the lifted table as axioms and the precondition formulas from the operator descriptions as the theorems to be proved. Each new proof is constructed as an isomorphic image of STRIPS' original preconditions proof by performing at each step resolutions on the same clauses and unifications on the same literals as in the original proof. This proof process ensures that each original proof is an instance of the new

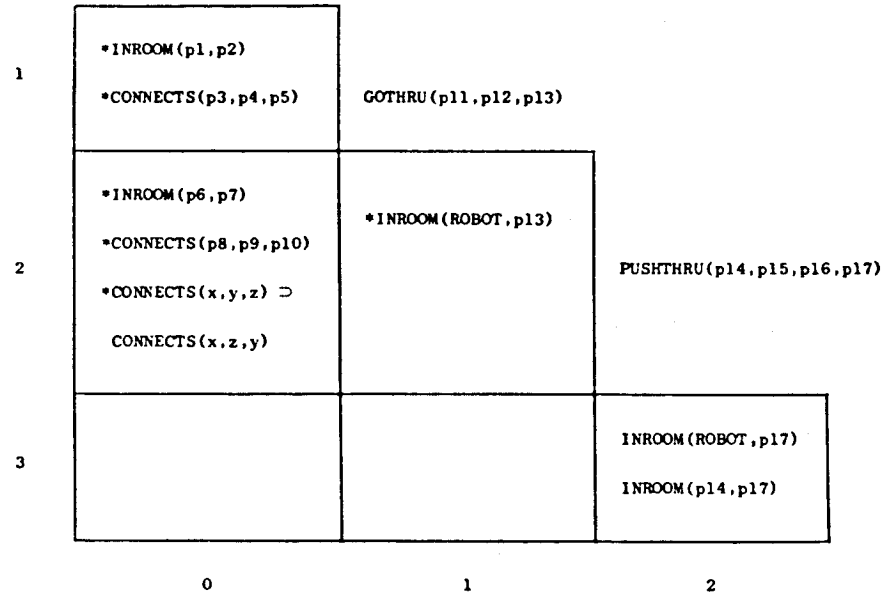
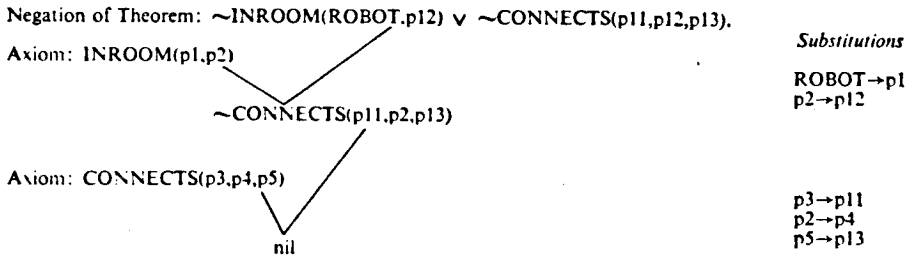


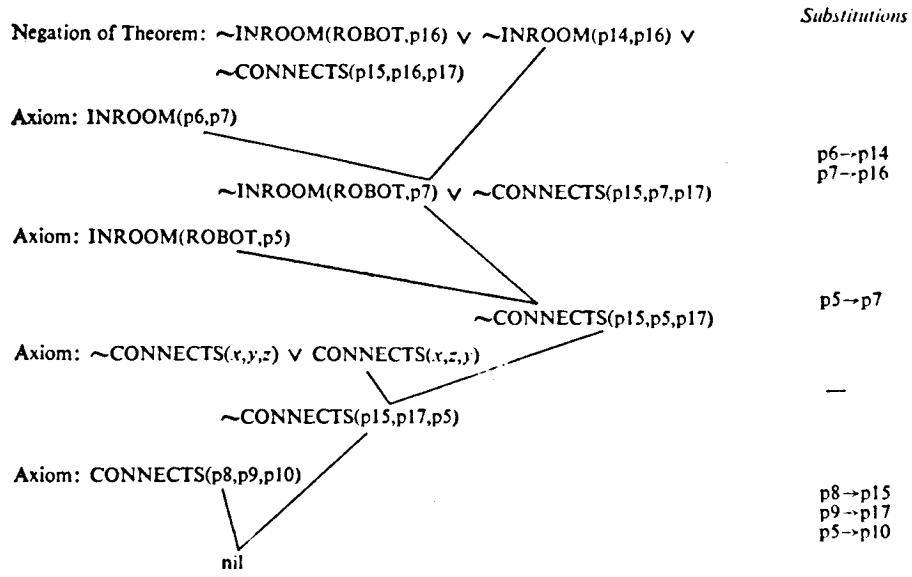
FIG. 3. Triangle table after initial lifting process.

generalized proof and therefore provides the basis for ensuring that the original table is an instance of the lifted table. Any substitutions of parameters for constants or for other parameters in the new proofs act as constraints on the generality of the plan and must be reflected in the lifted table. Hence these parameter substitutions are made throughout the lifted table and the generalized plan. The table resulting from the substitutions determined by the new proofs is constrained in the desired way.

Consider the effects of the new precondition proofs on the example table shown in Fig. 3. The precondition proof for GOTHRU(p11,p12,p13) proceeds as follows:



The substitutions from the proof are made in the table and then the following precondition proof for PUSHTHRU(p14,p15,p16,p17) is performed:



The substitutions from this proof are then used to produce the triangle table shown in Fig. 4.

The two proofs have constrained the plan so that the room into which the first operator takes the robot is the same room that contains the object to be pushed by the second operator. The robot's initial room and the target room for the push, however, remain distinct parameters constrained only by the precondition requirements that they each be adjacent to the object's initial room.

4.3. Two Refinements

Before a generalized plan is stored, two additional processing steps are carried out—one to improve efficiency and the other to remove possible inconsistencies. The first step eliminates some cases of overgeneralization produced during the lifting process and therefore makes more efficient the use of the plan by STRIPS and PLANEX. Often a clause in a plan's initial model will be in the support set of more than one operator, and therefore will appear more than once in Column 0 of the triangle table. When the table is lifted, each occurrence of the clause will generate new parameters. For example, in Fig. 3, CONNECTS(D1,R1,R2) was lifted to CONNECTS-(p3,p4,p5) and to CONNECTS(p8,p9,p10). In many cases this lifting pro-

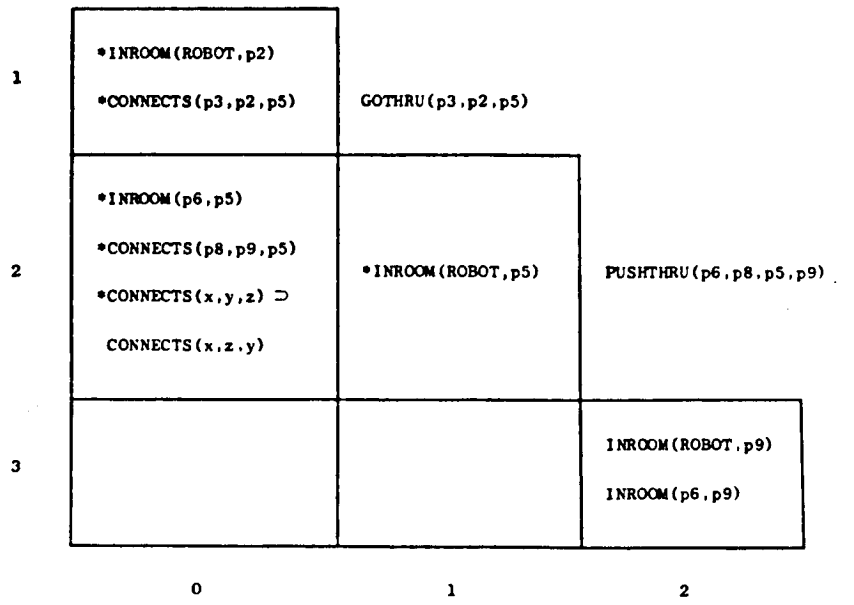


FIG. 4. Final form of triangle table for generalized plan.

cedure enhances the generality of the plan (as it did for the box-fetching plan by allowing the first and third rooms to be distinct), but it also produces cases of over-generalization that, while not incorrect, can lead to inefficiencies. For example, consider a case in which INROOM(BOX1,R1) appears twice in Column 0 of a triangle table. When the table is lifted, the occurrences of the clause in Column 0 might become INROOM(p1,p2) and INROOM(p3, p4). If the precondition proofs cause p1 to be substituted for p3, but do not constrain p2 and p4, then we have a plan whose preconditions include the clauses

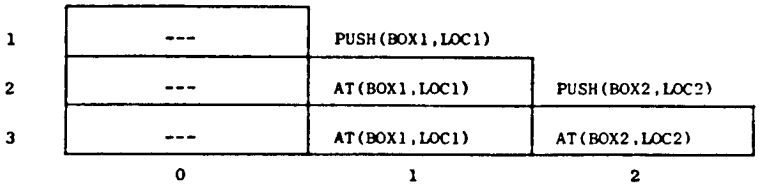
INROOM(p1,p2) and
INROOM(p1,p4).

Therefore we have a plan whose preconditions allow Object p1 to be in two distinct rooms at the same time, even though we know that in any semantically correct model Object p1 will be in only one room.

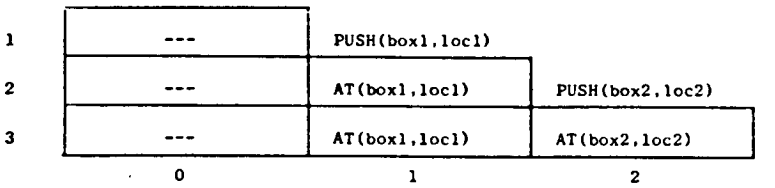
We eliminate most cases of this overgeneralization by recognizing those cases where two parameters are produced from a single occurrence of a constant in a single clause; if both such parameters do not appear as arguments of operators in the plan, then they can be bound together and one substituted for the other throughout the table without effectively inhibiting the generality of the plan. This procedure would substitute p2 for p4 in the INROOM example above, thereby making the two occurrences of the clause identical, but would not generate any constraining substitutions for the CONNECTS clause in the box-fetching example.

The second processing step that is performed before the plan is stored is needed to avoid inconsistencies that can occur in the lifted tables. The difficulty can be illustrated with the following example.

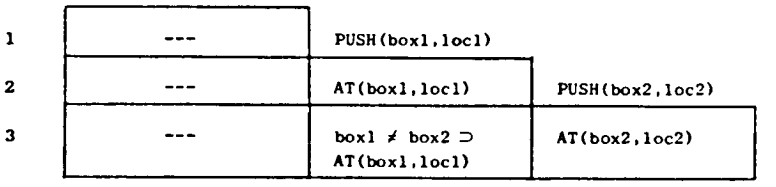
Consider a simple plan, PUSH(BOX1,LOC1), PUSH(BOX2,LOC2), for pushing two boxes to two locations. The unlifted triangle table for this plan might be as shown in Fig. 5a, where for simplicity we have not shown all clauses. When this table is lifted and the precondition proofs redone, no constraints are placed on the lifted table and it has the form shown in Fig. 5b. Suppose now that STRIPS were to use this plan with box1 and box2 instantiated to the same object and loc1 and loc2 instantiated to distinct locations. In that case STRIPS would evidently have a plan for achieving a state in which the same object is simultaneously at two different places!



(a) Unlifted Table



(b) Inconsistent Lifted Table



(c) Correct Lifted Table

FIG. 5. Triangle table for box-pushing plan

The source of this embarrassment lies in the assumption made above that the deletions in the lifted table can be the same as in the unlifted table. In our example, the clause AT(box1,loc1) should be deleted by the PUSH(box2, loc2) operator in the case where box1 and box2 are bound to the same object, but not deleted otherwise. Using the deletion algorithm described below, we represent this situation in the lifted table by replacing the clause AT(box1, loc1) in Row 3 by the clause form of

box1 ≠ box2 ⊃ AT(box1,loc1)

as shown in Fig. 5(c). This implication serves us well since the theorem prover can deduce AT(box1,loc1) as being part of the plan's additions list for exactly those cases in which box1 and box2 are distinct.

We now consider in general how deletions are correctly accounted for in the lifted triangle tables. After all the precondition proofs are redone for the lifted table, the delete list of each operator is considered beginning with the first operator and continuing in sequence through the plan. The delete list of the *i*th operator is applied to the clauses in Row *i* of the table to determine which clauses should appear in Row $i + 1$ of the table.[5] Recall that an operator's delete list is specified to STRIPS as a list of literals, and any clause that unifies with one of these literals is deleted. Application of the delete list will cause the lifted table to be modified only when a unification with a delete literal requires that a parameter p1 be replaced by another parameter p2 or by a constant C1. In that case the clause will unify with the delete literal only when p1 and p2 are instantiated to the same constant or when p1 is instantiated to C1. Hence the clause is replaced in the next row of the table by an implication as follows:

p1 ≠ p2 ⊃ clause or
p1 ≠ C1 ⊃ clause.

This implication allows the theorem prover to deduce the clause in only those cases where the operator's delete list would not have deleted it from the model.

If the clause that is replaced by the implication in a conditional deletion is part of the support of an operator in the plan (i.e., the clause is marked), then the implication must be accompanied by another addition to the table. In particular, if a clause CL1 is part of the support for the *j*th operator of the plan and CL1 is replaced in Row *j* of the table by the implication p1 ≠ p2 ⊃ CL1, then p1 ≠ p2 must be added as a marked clause to Cell $(j, 0)$ of the table. This addition to the table ensures that the *j*th operator's preconditions can be proved from the marked clauses in Row *j* of the table. The preconditions proof previously obtained will remain valid with the addition of a preliminary proof step in which clause CL1 is derived from p1 ≠ p2 and p1 ≠ p2 ⊃ CL1.

After these two processing steps are completed, the generalized plan is ready to be stored away as a macro operator, or *MACROP*, for later use by STRIPS and PLANEX.

[5] This characterization of the deletion applications requires that we include in Cell (1, 0) of the table all the clauses that appear anywhere in Column 0. The resulting redundant occurrences of Column 0 clauses can be edited out before the table is stored.

5. Execution Strategies

5.1. Requirements for the Plan Executor

In this section we shall describe how a program called PLANEX uses triangle tables to monitor the execution of plans. An early version of PLANEX was described by Fikes [2]. It is now being used in conjunction with STRIPS and the MACROP generation procedures to control the SRI robot [4].

One of the novel elements introduced into artificial intelligence research by work on robots is the study of execution strategies and how they interact with planning activities. Since robot plans must ultimately be executed in the real world by a mechanical device, as opposed to being carried out in a mathematical space or by a simulator, consideration must be given by the executor to the possibility that operations in the plan may not accomplish what they were intended to, that data obtained from sensory devices may be inaccurate, and that mechanical tolerances may introduce errors as the plan is executed.

Many of these problems of plan execution would disappear if our system generated a whole new plan after each execution step. Obviously, such a strategy would be too costly, so we instead seek a plan execution scheme with the following properties:

(1) When new information obtained during plan execution implies that some remaining portion of the plan need not be executed, the executor should recognize such information and omit the unneeded plan steps.

(2) When execution of some portion of the plan fails to achieve the intended results, the executor should recognize the failure and either direct reexecution of some portion of the plan or, as a default, call for a replanning activity.

5.2. Preparation of the MACROP for Execution

Rather than working with the specific version of the plan originally produced by STRIPS, PLANEX uses the generalized MACROP to guide execution. The generalized plan allows a modest amount of replanning by the executor should parts of the plan fail in certain ways.

Before a MACROP can be used by PLANEX, its parameters must be partially instantiated using the specific constants of the goal wff. This specializes the MACROP to the specific task at hand while it leaves as general as possible the conditions under which it can be executed. This partial instantiation process is quite simple: We put in the lower left-most cell of the triangle

table those clauses from the original model that were used by STRIPS in proving the goal wff. Then we use all of the clauses in the entire last row of the MACROP to prove the goal wff. Those substitutions made during this proof are then made on the entire MACROP. In addition we mark those clauses in the last row of the MACROP that were used to support the goal wff proof. This version of the MACROP is the one used to control execution.[6]

Let us illustrate what we have said about preparing a MACROP for execution by considering our example of fetching a box. In Fig. 4, we have the MACROP for this task. In Section 2, the goal wff for this task was given as

$$(\exists x)[\mathrm{BOX}(x) \wedge \mathrm{INROOM}(x,\mathrm{R1})].$$

In the proof of this goal wff we used the clause BOX(BOX1) from the original model, M_0. Therefore, we insert this clause in Cell (3,0) of the triangle table. We now use the clauses in Row 3 of the MACROP in Fig. 4 (together with BOX(BOX1), just inserted) to prove the goal wff. That is, we use BOX(BOX1), INROOM(ROBOT,p9) and INROOM(p6,p9) to prove $(\exists x)[\mathrm{BOX}(x) \wedge \mathrm{INROOM}(x,\mathrm{R1})]$. The substitutions made in obtaining the proof are BOX1 for p6 and R1 for p9. When these substitutions are applied to the MACROP of Fig. 4 and the support clauses for the new proof are marked, we obtain the execution MACROP shown in Fig. 6.

1	*INROOM(ROBOT,p2) *CONNECTS(p3,p2,p10)	GOTHRU(p3,p2,p10)	
2	*INROOM(BOX1,p10) *CONNECTS(p8,R1,p10) *CONNECTS(x,y,z) ⊃ CONNECTS(x,z,y)	*INROOM(ROBOT,p10)	PUSHTHROUGH(BOX1,p8,p10,R1)
3	*BOX(BOX1)		INROOM(ROBOT,R1) *INROOM(BOX1,R1)

FIG. 6. Execution MACROP for the fetch a box task.

[6] Some increase in generality can be obtained by putting in the lower leftmost cell of the triangle table *generalized* versions of the original model clauses. Some of the parameters in these generalized clauses might remain unbound in the proof of the goal wff, thereby making the table more general. In our implementation we shunned this additional complication.

5.3. The PLANEX Execution Strategy

Our strategy for monitoring the execution of plans makes use of the kernels of the execution MACROP. Recall that the ith kernel of a triangle table for an n-step plan is the unique rectangular subarray containing Row i and Cell $(n+1, 0)$. The importance of the ith kernel stems from the fact that it contains (as marked clauses) the support of the preconditions for the ith tail of the plan—that is, for the operator sequence $\{OP_i, \ldots, OP_n\}$. Thus if at some stage of plan execution the marked clauses in the ith kernel are provable, then we know that the ith tail is an appropriate operator sequence for achieving the goal. At each state of execution we must have at least one true kernel if we are to continue execution of the plan.

At the beginning of execution we know that the first kernel is true, since the initial model was used by STRIPS when the plan was created. But at later stages, unplanned outcomes might place us either unexpectedly close to the goal or throw us off the track completely. Our present implementation adopts a rather optimistic bias. We check each kernel in turn starting with the highest numbered one (which is the last row of the MACROP) and work backwards from the goal until we find a kernel that is true. If the goal kernel (the last row) is true, execution halts; otherwise we determine if the next-to-last kernel is true, and so on, until we find a true kernel k_i and a corresponding tail of the plan $\{OP_i, \ldots, OP_n\}$. The execution strategy then executes the action corresponding to OP_i and checks the outcome, as before, by searching for the highest-numbered true kernel. In an "ideal" world this procedure merely executes in order each operator in the plan. On the other hand, the procedure has the freedom to omit execution of unnecessary operators and to overcome failures by repeating the execution of operators. Replanning by STRIPS is initiated when no kernels are true.[7]

When checking to see if a kernel is true, we check to see if some instance of the conjunction of marked clauses in the kernel can be proved from the present model. Once such an instance is found, we determine the corresponding instance of the first operator in the tail of the plan and execute the action corresponding to that instance. Thus the generality of representation of the execution MACROP allows a great deal of flexibility in plan execution. For example, consider a case where PLANEX is executing a plan that takes the robot from one room through a second room into a third room. If, when the robot attempts to go through the door connecting the second and third rooms, the door is found to be locked, then PLANEX may be able to

[7] Typically, when replanning is necessary it is sufficient to produce a short sequence of operators to "get back onto the track" of the original plan. Since STRIPS has the MACROP for the original plan in its repertoire of operators, the new plan can often be formed by composing a sequence of operators and appending it to an appropriate tail of the MACROP.

reinstantiate parameters so that the first part of the plan can be reexecuted to take the robot from the second room through some new fourth room and then into the target third room.

An interesting by-product of our optimistic strategy of examining kernels in backwards order is that PLANEX sometimes remedies certain blunders made by STRIPS. Occasionally, STRIPS produces a plan containing an entirely superfluous subsequence—for example, a subsequence of the form OP, OP^{-1}, where OP^{-1} precisely negates the effects of OP. (Such a "detour" in a plan would reflect inadequacies in the search heuristics used by STRIPS.) During plan execution, however, PLANEX would effectively recognize that the state following OP^{-1} is the same as the state preceding OP, and would therefore not execute the superfluous subsequence.

5.4. The PLANEX Scanning Algorithm

The triangle table is a compact way of representing the kernels of a MACROP; most cells of the table occur in more than one kernel. We have exploited this economy of representation by designing an efficient algorithm for finding the highest-numbered true kernel. This algorithm, called the *PLANEX scan*, involves a cell-by-cell scan of the triangle table. We give a brief description of it here and refer the reader to Fikes [2] for more details. Each cell examined is evaluated as either *True* (i.e., all the marked clauses are provable from the current model) or *False*. The interest of the algorithm stems from the order in which cells are examined. Let us call a kernel "potentially true" at some stage in the scan if all evaluated cells of the kernel are true. The scan algorithm can then be succinctly stated as: *Among all unevaluated cells in the highest-indexed potentially true kernel, evaluate the left-most. Break "left-most ties" arbitrarily.* The reader can verify that, roughly speaking, this table-scanning rule results in a left-to-right, bottom-to-top scan of the table. However, the table is never scanned to the right of any cell already evaluated as false. An equivalent statement of the algorithm is "Among all unevaluated cells, evaluate the cell common to the largest number of potentially true kernels. Break ties arbitrarily." We conjecture that this scanning algorithm is optimal in the sense that it evaluates, on the average, fewer cells than any other scan guaranteed always to find the highest true kernel. A proof of this conjecture has not been found.

As the cells in the table are scanned we will be making substitutions for the MACROP parameters as dictated by the proofs of the cells' clauses. It is important to note that a substitution made to establish the truth of clauses in a particular cell must be applied to the entire table. When there are alternative choices about which substitutions to make, we keep a tree of possibilities so that backtracking can occur if needed.

6. Planning with MACROPS

In the preceding sections, we described the construction of MACROPS and how they are used to control execution. Now let us consider how a MACROP can be used by STRIPS during a subsequent planning process.

6.1. Extracting a Relevant Operator Sequence from a MACROP

Recall that the $(i + 1)$st row of a triangle table (excluding the first cell) represents the add list, $A_{1,\ldots,i}$, of the ith head of the plan, i.e. of the sequence $OP_1, \ldots, OP_i$. An n-step plan presents STRIPS with n alternative add lists, any one of which can be used to reduce a difference encountered during the normal planning process. STRIPS tests the relevance of each of a MACROP's add lists in the usual fashion, and the add lists that provide the greatest reduction in the difference are selected. Often a given set of relevant clauses will appear in more than one row of the table. In that case only the lowest-numbered row is selected, since this choice results in the shortest operator sequence capable of producing the desired clauses.

Suppose that STRIPS selects the ith add list $A_{1,\ldots,i}$, $i < n$. Since this add list is achieved by applying in sequence $OP_1, \ldots, OP_i$, we will obviously not be interested in the application of $OP_{i+1}, \ldots, OP_n$, and will therefore not be interested in establishing any of the preconditions for these operators. Now in general, some steps of a plan are needed only to establish preconditions for subsequent steps. If we lose interest in a tail of a plan, then the relevant instance of the MACROP need not contain those operators whose sole purpose is to establish preconditions for the tail. Also, STRIPS will, in general, have used only some subset of $A_{1,\ldots,i}$ in establishing the relevance of the ith head of the plan. Any of the first i operators that does not add some clause in this subset or help establish the preconditions for some operator that adds a clause in the subset is not needed in the relevant instance of the MACROP.

Conceptually, then, we can think of a single triangle table as representing a family of generalized operators. Upon the selection by STRIPS of a relevant add list, we must extract from this family an economical parameterized operator achieving the add list. In the following paragraphs, we will explain by means of an example an editing algorithm for accomplishing this task of operator extraction.

6.2. The Editing Algorithm

Consider the illustrative triangle table shown in Fig. 7. Each of the numbers within cells represents a single clause. The circled clauses are "marked" in the sense described earlier; that is, they are used to prove the precondition of the operator whose name appears on the same row. A summary of the structure

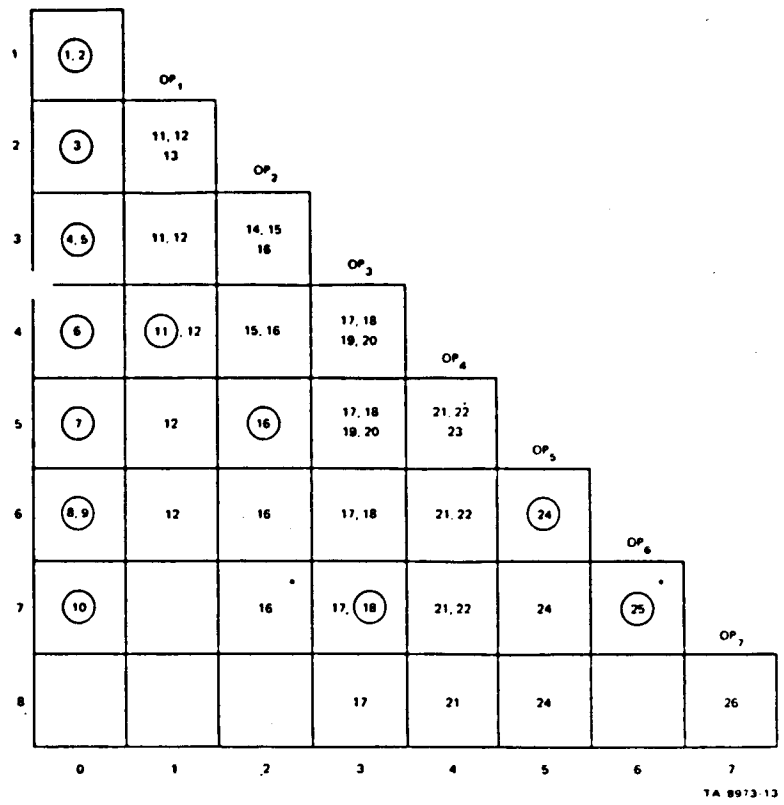


FIG. 7. MACROP with marked clauses.

of this plan is shown below, where "I" refers to the initial state and "F" to the final state:

OPERATOR	PRECONDITION SUPPORT SUPPLIED BY	PRECONDITION SUPPORT SUPPLIED TO
OP_1	I	OP_4
OP_2	I	OP_5
OP_3	I	OP_7,F
OP_4	I,OP_1	F
OP_5	I,OP_2	OP_6,F
OP_6	I,OP_5	OP_7
OP_7	I,OP_3,OP_6	F

Suppose now that STRIPS selects $A_{1,\ldots,6}$ as the desired add list and, in particular, selects Clause 16 and Clause 25 as the particular members of the add list that are relevant to reducing the difference of immediate interest. These clauses have been indicated on the table by an asterisk (*). The editing algorithm proceeds by examining the table to determine what effects of individual operators are not needed to produce Clauses 16 and 25. First, OP_7 is obviously not needed; we can therefore remove all circle marks from Row 7, since those marks indicate the support of the preconditions of OP_7. We now inspect the columns, beginning with Column 6 and going from right to left, to find the first column with no marks of either kind (circles or asterisks). Column 4 is the first such column. The absence of marked clauses in Column 4 means that the clauses added by OP_4 are not needed to reduce the difference and are not required to prove the pre-condition of any subsequent operator; hence OP_4 will not be in the edited operator sequence and we can unmark all clauses in Row 4. Continuing our right-to-left scan of the columns, we note that Column 3 contains no marked clauses. (Recall that we have already unmarked Clause 18.) We therefore delete OP_3 from the plan and unmark all clauses in Row 3. Continuing the scan, we note that Column 1 contains no marked entries (we have already unmarked Clause 11), and therefore we can delete OP_1 and the marked entries in Row 1.

The result of this editing process is to reduce the original seven-step plan to the compact three-step plan, $\{OP_2, OP_5, OP_6\}$, whose add list specifically includes the relevant clauses. The structure of this plan is shown below.

OPERATOR	PRECONDITION SUPPORT SUPPLIED BY	PRECONDITION SUPPORT SUPPLIED TO
OP_2	I	OP_5,F
OP_5	I,OP_2	OP_6,F
OP_6	I,OP_5	F

6.3. Use of Edited MACROPS as Relevant Operators

Once an edited MACROP has been constructed, we would like STRIPS to use it in the same manner as any other operator. We have some latitude though, in specifying the preconditions of the MACROP. An obvious choice would be to use the conjunction of the clauses in the left-most column, but there is a difficulty with this straightforward choice that can be made clear with the aid of a simple example. Suppose we are currently in a state in which the first kernel of an edited MACROP—that is, its left-most column—is false, but suppose further that, say, the third kernel is true. Since the third kernel is true, the tail of the MACROP beginning with OP_3 is immediately applicable and would produce the desired relevant additions to the model. If STRIPS were to ignore this opportunity and set up the left-most column of the MACROP as a subgoal, it would thereby take the proverbial one step backward to go two steps forward.

This example suggests that we employ a PLANEX scan on the edited table

so that all tails of the relevant MACROP will be tested for applicability. If an applicable tail is found, STRIPS applies, in sequence, each operator in this tail to produce a new planning model. Each operator application is performed in the usual manner using the add and delete lists of the individual operators. If the PLANEX scan fails to find a true kernel, then no tail is applicable and the conjunction of the marked clauses in the first kernel is set up as a subgoal to be achieved by STRIPS. Actually, any kernel would constitute a perfectly good subgoal and, in principle, the disjunction of all the kernels would be better still. Unfortunately, this disjunction places excessive demands on both the theorem prover and the STRIPS executive, so we restrict ourselves to consideration of the first kernel.

We have seen that STRIPS uses a MACROP during planning by extracting a relevant subsequence of the MACROP's operators, and then including that subsequence in the new plan being constructed. When the new plan is made into a MACROP it is often the case that it will contain add lists that are subsets of add lists in already existing tables. For example, if an entire existing MACROP is used in the construction of a new plan, and the parameter substitutions in the new MACROP correspond to those in the old MACROP, then each add list in the old MACROP will be a subset of an add list in the new MACROP. To assist STRIPS in its use of MACROPS, we have designed a procedure that will remove redundant add lists from consideration during planning, and in cases where an entire MACROP is contained within another, will delete the contained MACROP from the system.

Our procedure takes the following action: If every instance of the operator sequence that is the ith head of some MACROP is also an instance of a sequence occurring anywhere else in the same or some other MACROP, then all the add lists in that head (i.e. Rows 2 through $i + 1$) are disallowed for consideration by STRIPS.[8] For example, consider the following two generalized plans:

Plan A:OPA(p1),OPB(p1,p2),OPC(p3),OPD(p3,C1),OPA(p3),OPB(p4,p5)
Plan B: OPC(p6),OPD(p6,C1),OPA(p7),OPF(p6,p7).

Rows 2 and 3 of Plan A are disallowed for consideration as add lists since every instance of the sequence, OPA(p1),OPB(p1,p2), is also an instance of the sequence, OPA(p3),OPB(p4,p5), that occurs at the end of Plan A. Rows 2 and 3 of Plan B are disallowed because of the sequence, OPC(p3),OPD(p3,C1), that occurs in Plan A. Note that Row 4 of Plan B could not be disallowed for consideration by Plan A since there are instances of the sequence, OPC(p6),OPD(p6,C1),OPA(p7), that are not instances of OPC(p3),OPD(p3,C1),OPA(p3).

This procedure is applied whenever a new MACROP is added to the system. It has proved to be quite effective at minimizing the number of MACROP add lists that STRIPS must consider during planning. (See Section 7, for examples.) A difficulty arises in the use of this procedure when the same operator appears in two MACROPs and the support sets for the precondition proofs of that operator differ markedly in the two triangle tables. This can occur, for example, when the precondition is a disjunction of two wffs and in one case the first disjunct was proven to be true and in the other case the second disjunct was proven to be true. In those situations the two occurrences of the operator should not be considered as instances of the same operator since each occurrence effectively had different preconditions. A refinement of our procedure that would include an appropriate comparison of the support sets could be employed to overcome this difficulty.

[8] Note that the first row of a MACROP contains no add clauses.

7. Experimental Results

The mechanisms we have described for generating and using MACROPS have been implemented as additions and modifications to the existing STRIPS and PLANEX systems. In this section we will describe the results of some of the experiments we have run with the new system. Problems were posed to the system in the SRI robot's current experimental environment of seven rooms, eight doors, and several boxes about two feet high. The robot is a mobile vehicle equipped with touch sensors, a television camera, and a push bar that allows the robot to push the boxes [4]. A typical state of this experimental environment is modeled by STRIPS using about 160 axioms.

7.1. Operator Descriptions

The operator descriptions given to STRIPS for these experiments model the robot's preprogrammed action routines for moving the robot next to a door in a room, next to a box in a room, to a location in a room, or through a door. There are also operators that model action routines for pushing a box next to another box in a room, to a location in a room, or through a door. In addition, we have included operator descriptions that model fictitious action routines for opening and closing doors. These descriptions are as follows:

GOTOB(bx) *Go to object bx.*
Preconditions: TYPE(bx,OBJECT),($\exists rx$)[INROOM(bx,rx) $\wedge$ INROOM(ROBOT,rx)]
Deletions: AT(ROBOT,$1,$2),NEXTTO(ROBOT,$1)
Additions: *NEXTTO(ROBOT,bx)

GOTOD(dx) *Go to door dx.*
Preconditions: TYPE(dx,DOOR),($\exists rx$)($\exists ry$)[INROOM(ROBOT,rx) $\wedge$ CONNECTS(dx,rx,ry)]
Deletions: AT(ROBOT,$1,$2),NEXTTO(ROBOT,$1)
Additions: *NEXTTO(ROBOT,dx)

GOTOL(x,y) *Go to coordinate location (x,y).*
Preconditions: ($\exists rx$)[INROOM(ROBOT,rx) $\wedge$ LOCINROOM(x,y,rx)]
Deletions: AT(ROBOT,$1,$2),NEXTTO(ROBOT,$1)
Additions: *AT(ROBOT,x,y)

PUSHB(*bx*,*by*) *Push bx to object by.*
Preconditions: TYPE(*by*,OBJECT),PUSHABLE(*bx*),NEXTTO(ROBOT,*bx*),
(∃*rx*)[INROOM(*bx*,*rx*) ∧ INROOM(*by*,*rx*)]
Deletions: AT(ROBOT,$1,$2),NEXTTO(ROBOT,$1),AT(*bx*,$1,$2),NEXTTO(*bx*,$1),
NEXTTO($1,*bx*)
Additions: *NEXTTO(*bx*,*dx*),NEXTTO(ROBOT,*bx*)

PUSHD(*bx*,*dx*) *Push bx to door dx.*
Preconditions: PUSHABLE(*bx*),TYPE(*dx*,DOOR),NEXTTO(ROBOT,*bx*)
(∃*rx*)(∃*ry*)[INROOM(*bx*,*rx*) ∧ CONNECTS(*dx*,*rx*,*ry*)]
Deletions: AT(ROBOT,$1,$2),NEXTTO(ROBOT,$1),AT(*bx*,$1,$2),NEXTTO(*bx*,$1),
NEXTTO($1,*bx*)
Additions: *NEXTTO(*bx*,*dx*),NEXTTO(ROBOT,*bx*)

PUSHL(*bx*,*x*,*y*) *Push bx to coordinate location* (*x*,*y*).
Preconditions: PUSHABLE(*bx*),NEXTTO(ROBOT,*bx*),(∃*rx*)[INROOM(ROBOT,*rx*) ∧
LOCINROOM(*x*,*y*,*rx*)]
Deletions: AT(ROBOT,$1,$2),NEXTTO(ROBOT,$1),AT(*bx*,$1,$2),NEXTTO(*bx*,$1),
NEXTTO($1,*bx*)
Additions: *AT(*bx*,*x*,*y*),NEXTTO(ROBOT,*bx*)

GOTHRUDR(*dx*,*rx*) *Go through door dx into room rx.*
Preconditions: TYPE(*dx*,DOOR),STATUS(*dx*,OPEN),TYPE(*rx*,ROOM),
NEXTTO(ROBOT,*dx*) (∃*ry*)[INROOM(ROBOT,*ry*) ∧ CONNECTS(*dx*,*ry*,*rx*)]
Deletions: AT(ROBOT,$1,$2),NEXTTO(ROBOT,$1),INROOM(ROBOT,$1)
Additions: *INROOM(ROBOT,*rx*)

PUSHTHRUDR(*bx*,*dx*,*rx*) *Push bx through door dx into room rx.*
Preconditions: PUSHABLE(*bx*),TYPE(*dx*,DOOR),STATUS(*dx*,OPEN),TYPE(*rx*,
ROOM),NEXTTO(*bx*,*dx*),NEXTTO(ROBOT,*bx*),(∃*ry*)[INROOM(*bx*,*ry*) ∧
CONNECTS(*dx*,*ry*,*rx*)]
Deletions: AT(ROBOT,$1,$2),NEXTTO(ROBOT,$1),AT(*bx*,$1,$2),NEXTTO(*bx*,$1),
NEXTTO($1,*bx*),INROOM(ROBOT,$1),INROOM(*bx*,$1)
Additions: *INROOM(*bx*,*rx*),INROOM(ROBOT,*rx*),NEXTTO(ROBOT,*bx*)

OPEN(*dx*) *Open door dx.*
Preconditions: NEXTTO(ROBOT,*dx*),TYPE(*dx*,DOOR),STATUS(*dx*,CLOSED)
Deletions: STATUS(*dx*,CLOSED)
Additions: *STATUS(*dx*,OPEN)

CLOSE(*dx*) *Close door dx.*
Preconditions: NEXTTO(ROBOT,*dx*),TYPE(*dx*,DOOR),STATUS(*dx*,OPEN)
Deletions: STATUS(*dx*,OPEN)
Additions: *STATUS(*dx*,CLOSED)

Note: The addition clauses preceded by an asterisk are the *primary additions* of the operator. When STRIPS searches for a relevant operator is considers only these primary addition clauses.

7.2. Example Problems

7.2.1. SUMMARY. A sequence of five problems was designed to illustrate the various ways in which MACROPs are used during planning. We show in the next subsection an annotated trace of the system's behaviour for each problem in the sequence. Each trace is preceded by a diagram of the problem's initial and final states, and includes the sequence of subgoal generations and operator applications actually occurring in the STRIPS solution. STRIPS' attention was directed to the rooms shown in the diagrams by closing the doors connecting all other rooms.

The plan for the first problem in the sequence pushes two boxes together and then takes the robot into an adjacent room. The second problem is similar to the first except that different rooms and different boxes are involved, and the robot begins in a room adjacent to the room containing the boxes. STRIPS uses a tail of MACROP1 to get the robot into the room with the boxes and then uses the entire MACROP1 to complete the plan.

The third problem involves taking the robot from one room through a second room and into a third room, with the added complication that the door connecting the second and third rooms is closed. STRIPS first decides to use MACROP2 with the box-pushing sequence edited out and then finds that the door must be opened; to get the robot next to the closed door, a head of MACROP2 is selected with the box-pushing sequence again edited out. After formation of the plan to go to the door and open it, the PLANEX scan observes that only the final operator of the first relevant instance of MACROP2 is needed to complete the plan.

The fourth problem requires that three boxes be pushed together, with the robot beginning in a room adjacent to the room containing the boxes. A head of MACROP2 is used to get the robot into the room with the boxes and to push two of them together; the box-pushing sequence of MACROP2 is used to complete the plan, again with the assistance of the PLANEX scan.

The fifth problem requires the robot to go from one room into a second room, open a door that leads into a third room, go through the third room into a fourth room, and then push together two pairs of boxes. The plan, which is formed by combining all of MACROP4 with all of MACROP3, is well beyond the range of plans producible by STRIPS without the use of MACROPs. Note that although MACROP4 was created by lifting a plan that pushed three boxes together, it has enough generality to handle this form of a four-box problem. Note also that MACROP1, MACROP3, and MACROP4 have been recognized as redundant and deleted, so that the net result of this learning sequence is to add only MACROP2 and MACROP5 to the system.

In Table I we present a table showing the search tree sizes and running times for the five problems. The problems were run both with and without the use of MACROPs for comparison. Even when MACROPs were not being used for planning we include the MACROP production time since PLANEX needs the MACROP to monitor plan execution. Note that the times and the search tree sizes are all smaller when MACROPS are used and that the MACROPs allow longer plans to be formed without necessarily incurring an exponential increase in planning time.

TABLE I
Statistics for STRIPS behavior

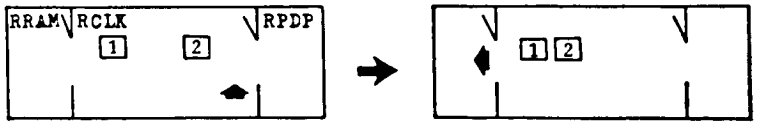
	PROBLEM 1	PROBLEM 2	PROBLEM 3	PROBLEM 4	PROBLEM 5
Without MACROPS					
Total time (minutes)	3 : 05	9 : 42	7 : 03	14 : 09	–
Time to produce MACROP	1 : 00	1 : 28	1 : 11	1 : 43	–
Time to find unlifted plan	2 : 05	8 : 14	5 : 52	12 : 26	–
Total nodes in search tree	10	33	22	51	–
Nodes on solution path	9	13	11	15	–
Operators in plan	4	6	5	7	–
With MACROPS					
Total time (minutes)	3 : 05	3 : 54	6 : 34	4 : 37	9 : 13
Time to produce MACROP	1 : 00	1 : 32	1 : 16	1 : 37	3 : 24
Time to find unlifted plan	2 : 05	2 : 22	5 : 18	3 : 00	5 : 49
Total nodes in search tree	10	9	14	9	14
Nodes on solution path	9	9	9	9	14
Operators in plan	4	6	5	6	11

STRIPS is written in BBN-LISP and runs as compiled code on a PDP-10 computer under the TENEX time-sharing system.

STRIPS could not solve Problem 5 without using MACROPs.

7.2.2. Annotated Trace of System Behavior for Each Example Problem.

Problem 1



G1: INROOM(ROBOT,RRAM) ∧ NEXTTO(BOX1,BOX2)

*****

G1 is the task statement.

*****

G2: Preconditions for PUSHB(BOX2,BOX1)

G3: Preconditions for GOTOB(BOX2)

Apply GOTOB(BOX2)

Apply PUSHB(BOX2,BOX1)

G4: Preconditions for GOTHRUDR(par18,RRAM)

G6: Preconditions for GOTOD(DRAMCLK)

*****

G5 was the precondition for an operator that did not appear in the completed plan.

*****

Apply GOTOD(DRAMCLK)

Apply GOTHRUDR(DRAMCLK,RRAM)

Solution

Form MACROP1(par29,par37,par45,par54,par33)

*****

The parameter list for a MACROP contains all the parameters that occur in the triangle table.

*****

GOTOB(par29)

PUSHB(par29,par37)

GOTOD(par45)

GOTHRUDR(par45,par54)

*****

The generalized plan pushes two boxes together and takes the robot into an adjacent room, given that the robot and the boxes are initially all in the same room.

*****

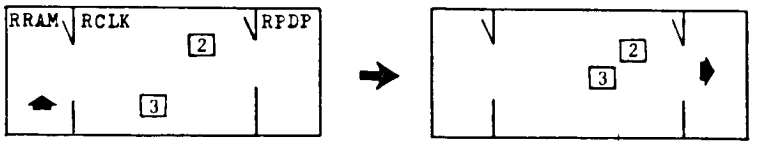
Set first additions row of MACROP1 to 3.

*****

STRIPS will consider only rows numbered 3 and higher as add lists during planning. Rows 1 and 2 of a triangle table are never considered as add lists since there are no add clauses in Row 1, and the add clauses in Row 2 are redundant with respect to the operator description of the first operator in the MACROP.

*****

Problem 2



G1: INROOM(ROBOT,RPDP) ∧ NEXTTO(BOX2,BOX3)

*****

G1 is the task statement.

*****

G2: Preconditions for MACROP1:5(BOX3,BOX2,par3,RPDP,par5)

*****

The notation MACROP1:5 means that Row 5 of MACROP1 is selected as a relevant add list. MACROP1 is instantiated so that Row 5 contains the relevant clauses INROOM(ROBOT,RPDP) added by GOTHRUDR(par3,RPDP) and NEXTTO(BOX2,BOX3) added by PUSHB(BOX3,BOX2). All four operators in MACROP1 are needed to produce these relevant clauses. No kernels in the triangle table are satisfied. A difference consisting of the single clause INROOM (ROBOT,RCLK) is extracted from the first kernel.

*****

G3: Preconditions for MACROP1:5(par17,par18,par19,RCLK,par21)

*****

Row 5 of MACROP1 is again selected as a relevant add list. MACROP1 is instantiated so that Row 5 contains the relevant clause INROOM-(ROBOT,RCLK) added by GOTHRUDR(par19,RCLK). Only the last two operators in MACROP1 are needed to produce the relevant clause.

*****

Kernel 3 satisfied

*****

Kernel 3 is the precondition for the last two operators in MACROP1.

*****

Apply GOTOD(DRAMCLK)
Apply GOTHRUDR(DRAMCLK,RCLK)
Kernel 1 satisfied
Apply GOTOB(BOX3)
Apply PUSHB(BOX3,BOX2)
Apply GOTOD(DPDPCLK)
Apply GOTHRUDR(DPDPCLK,RPDP)
Solution

Form MACROP2(par27,par52,par72,par91,par111,par38,par40)
GOTOD(par27)
GOTHRUDR(par27,par40)
GOTOB(par52)
PUSHB(par52,par72)
GOTOD(par91)
GOTHRUDR(par91,par111)

*****

The generalized plan takes the robot from one room into an adjacent room, pushes two boxes together in the second room, and then takes the robot into a third room adjacent to the second.

*****

Erase MACROP1.

*****

MACROP1 is completely contained in MACROP2.

*****

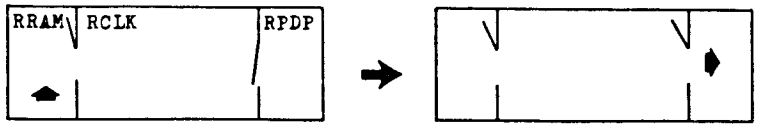
Set first additions row of MACROP2 to 4.

*****

The first two operators of MACROP2 match the last two operators of MACROP2.

*****

Problem 3



G1: INROOM(ROBOT,RPDP)

*****

G1 is the task statement.

*****

G2: Preconditions for MACROP2:7(par1,par2,par3,par4,RPDP,par6,par7)

*****

Row 7 of MACROP2 is selected as a relevant add list. MACROP2 is instantiated so that Row 7 contains the relevant clause INROOM-(ROBOT,RPDP) added by GOTHRUDR(par4,RPDP). Only the first, second, fifth, and sixth operators are needed to produce this relevant clause. No kernels in the triangle table are satisfied. A difference consisting of the single clause STATUS(DPDPCLK,OPEN) is extracted from the first kernel.

*****

G5: Preconditions for OPEN(DPDPCLK)

*****

After considering two other relevant operators for achieving G1, STRIPS returns to the solution path. OPEN(DPDPCLK) is found to be a relevant operator and a difference consisting of the single clause NEXTTO(ROBOT,DPDPCLK) is extracted from the preconditions.

*****

G9: Preconditions for MACROP2:6(par15,par16,par17,DPDPCLK,par19, par20,par21)

*****

After considering three other relevant operators for achieving G5, STRIPS selects Row 6 of MACROP2 as a relevant add list. MACROP2 is instantiated so that Row 6 contains the relevant clause NEXTTO-(ROBOT,DPDPCLK) added by GOTOD(DPDPCLK). Only the first, second, and fifth operators are needed to produce this relevant clause.

*****

Kernel 1 satisfied.
Apply GOTOD(DRAMCLK)
Apply GOTHRUDR(DRAMCLK,RCLK)
Apply GOTOD(DPDPCLK)
Apply OPEN(DPDPCLK)
Kernel 6 satisfied

*****

A PLANEX scan is used so that all kernels are checked. Kernel 6 is the precondition for the final operator in the relevant instance of MACROP2.

*****

Apply GOTHRUDR(DPDPCLK,RPDP)
Solution

Form MACROP3(par24,par59,par82,par32,par42)
GOTOD(par24)
GOTHRUDR(par24,par42)
GOTOD(par59)
OPEN(par59)
GOTHRUDR(par59,par82)

*****

The generalized plan takes the robot from one room into an adjacent room, then to a closed door in the second room, opens the closed door, and then takes the robot through the opened door into a third room.

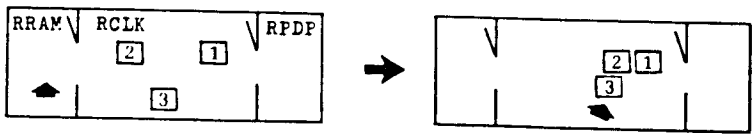
*****

Set first additions row of MACROP3 to 4.

*****

The first two operators of MACROP3 match the first two operators of MACROP2.

Problem 4



G1: NEXTTO(BOX1,BOX2) ∧ NEXTTO(BOX2,BOX3)

*****

G1 is the task statement.

*****

G2: Preconditions for MACROP2:5(par1,BOX2,BOX1,par4,par5,par6,par7)

*****

Row 5 of MACROP2 is selected as a relevant add list. MACROP2 is instantiated so that Row 5 contains the relevant clause NEXTTO-(BOX1,BOX2) added by PUSHB(BOX2,BOX1). All of the first four operators in MACROP2 are needed to produce this relevant clause.

*****

Kernel 1 satisfied

Apply GOTOD(DRAMCLK)

Apply GOTHRUDR(DRAMCLK,RCLK)

Apply GOTOB(BOX2)

Apply PUSHB(BOX2,BOX1)

G3: Preconditions for MACROP2:5(par19,BOX3,BOX2,par22,par23,par24,par25)

*****

Row 5 of MACROP2 is selected as before. The instantiation is so that Row 5 contains the relevant clause NEXTTO(BOX2,BOX3) added by PUSHB(BOX3,BOX2). Again all of the first four operators are included in the relevant instance of MACROP2.

*****

Kernel 3 satisfied

*****

A PLANEX scan is used so that all kernels are checked. Kernel 3 is the precondition for the third and fourth operators.

*****

Apply GOTOB(BOX3)

Apply PUSHB(BOX3,BOX2)

Solution

Form MACROP4(par37,par80,par102,par123,par134,par57,par59)

GOTOD(par37)

GOTHRUDR(par37,par59)

GOTOB(par80)

PUSHB(par80,par102)

GOTOB(par123)

PUSHB(par123,par134)

*****

The generalized plan takes the robot from one room into an adjacent room, pushes one box to a second box, and then pushes a third box to a fourth box.

*****

Set first additions row of MACROP2 to 6.

*****

The first 4 operators of MACROP2 match the first 4 operators of MACROP4.

*****

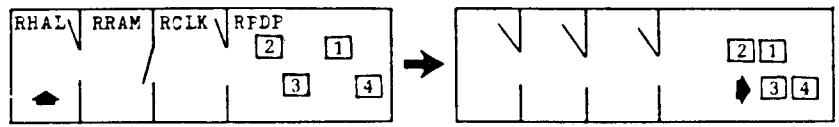
Set first additions row of MACROP4 to 4.

*****

The first 2 operators of MACROP4 match the last 2 operators of MACROP2.

*****

Problem 5



G1: NEXTTO(BOX1,BOX2) ∧ NEXTTO(BOX3,BOX4)

*****

G1 is the task statement.

*****

G2: Preconditions for MACROP4:7(par13,BOX2,BOX1,BOX3,BOX4,par18,par19)

*****

Row 7 of MACROP4 is selected as a relevant add list. MACROP4 is instantiated so that Row 7 contains the relevant clauses NEXTTO (BOX1,BOX2) added by PUSHB(BOX2,BOX1) and NEXTTO (BOX3,BOX4) added by PUSHB(BOX3,BOX4). All six operators in MACROP4 are needed to produce these relevant clauses. No kernels in the triangle table are satisfied. A difference consisting of the single clause INROOM(ROBOT,RCLK) is extracted from the first kernel.

*****

G3: Preconditions for MACROP3:6(par27,par28,RCLK,par30,par31)

*****

Row 6 of MACROP3 is selected as a relevant add list. MACROP3 is instantiated so that Row 6 contains the relevant clause INROOM (ROBOT,RCLK) added by GOTHRUDR(par28,RCLK). All five

operators in MACROP3 are needed to produce this relevant clause.
*****
Kernel 1 satisfied
Apply GOTOD(DRAMHAL)
Apply GOTHRUDR(DRAMHAL,RRAM)
Apply GOTOD(DRAMCLK)
Apply OPEN(DRAMCLK)
Apply GOTHRUDR(DRAMCLK,RCLK)
Kernel 1 satisfied
Apply GOTOD(DPDPCLK)
Apply GOTHRUDR(DPDPCLK,RPDP)
Apply GOTOB(BOX2)
Apply PUSHB(BOX2,BOX1)
Apply GOTOB(BOX3)
Apply PUSHB(BOX3,BOX4)
Solution

Form MACROP5(par44,par87,par151,par208,par237,par265,par294,par180,par130,
par64,par66)
GOTOD(par44)
GOTHRUDR(par44,par66)
GOTOD(par87)
OPEN(par87)
GOTHRUDR(par87,par130)
GOTOD(par151)
GOTHRUDR(par151,par180)
GOTOB(par208)
PUSHB(par208,par237)
GOTOB(par265)
PUSHB(par265,par294)
*****
The generalized plan takes the robot from one room into a second room, opens a door leading to a third room, takes the robot through the third room into a fourth room, and then pushes together two pairs of boxes.
*****
Erase MACROP3.
Erase MACROP4.
*****
MACROP3 and MACROP4 are completely contained in MACROP5.
*****
Set first additions row of MACROP5 to 4.
*****
The first two operators of MACROP5 match the sixth and seventh operators of MACROP5.
*****

7.3. Further Experiments

In another set of experiments that were run with the new system, the primary goal was to produce long plans. We ran a sequence of eight problems in our robot environment that culminated in the production of a 19-operator plan for fetching three boxes from three different rooms and then pushing the three boxes together. This final MACROP subsumed the seven earlier ones so that only one MACROP was retained by the system. Subsequences of the 19-step MACROP could be used to fetch boxes, push boxes together, move the robot from room to room, etc.

The experiments we have been discussing show the use of MACROPs during planning. We have also run experiments with PLANEX to illustrate the use of MACROPs during plan execution. One such experiment is documented in a report [4] and film [5] that illustrate how PLANEX monitors robot task execution in the seven-room experimental environment. One interesting sequence in this experiment involves the robot attempting to go from one room through a second room into a third room. After entering the second room, the robot discovers that a box is blocking the door that leads into the third room. Since PLANEX is working with a generalized plan, the difficulty can be overcome by finding a different instance of the plan's first kernel that is satisfied. This new instantiation of the plan's parameters causes the robot to be sent from the second room into a fourth room and then into the target third room.

8. Conclusions

We have presented in considerable detail methods by which a problem-solving program can "learn" old solutions and use them both to monitor real-world execution and to aid in the solution of new problems. We view these methods as representing only a preliminary excursion into an area that, in the long run, may hold high potential for the design of "intelligent" robots. Before such potential is realized, however, there are a number of substantial technical problems to be solved; in this final section we briefly point out a few of these.

8.1. Abstracting Preconditions

It is a commonplace observation that successful problem solvers (human or machine) must plan at a level of detail appropriate to the problem at hand. In typical problem-solving programs, the level of detail is set a priori by the experimenter when he carefully selects the representations employed. This situation changes when the problem solver can create its own MACROPS. Now we have the possibility of creating powerful macro operators whose specification is at the same level of detail as each component operator. In terms of our system, we may create a large triangle table whose preconditions (its first kernel) is the conjunction of so many literals that the theorem prover has little hope of success. What we need is a way of appropriately

abstracting the preconditions of a MACROP so that only its "main" preconditions remain. A plan would first be attempted using these abstract preconditions; if successful, a subsequent planning process would fill in the details (and perhaps suggest changes to the abstract plan) as needed. As a rough example of the sort of process we have in mind, suppose we have a MACROP that requires the robot to travel through several doors. An abstract precondition for the MACROP might not contain the requirement that the doors be open on the supposition that, should they be closed, the robot could easily open them at the appropriate time. In whatever manner such a scheme is ultimately implemented, it seems clear that a problem solver will be able to increase its power with experience only if it can use this experience at an appropriate level of abstraction.

8.2. Saving MACROPS

We discussed previously a method for discarding a MACROP when it is subsumed by another, more powerful MACROP. In general, any system that learns plans must also either incorporate a mechanism for forgetting old plans or else face the danger of being swamped by an ever-increasing repertoire of stored plans. One straightforward approach to this problem would be to keep some statistics on the frequencies with which the various MACROPS are used, and discard those that fall below some threshold. We have not, however, experimented with any such mechanisms.

8.3. Other Forms of Learning

The generalization scheme discussed in this paper is but one of many possible forms of machine learning. Another form of learning that would be interesting to investigate involves reconciling predicted and observed behavior. Suppose, by way of example, that an operator *OP* is originally thought to add Clause *C* whenever it is applied, but suppose we notice that the action corresponding to *OP* consistently fails to add *C*. We would like the system to remedy this situation by taking one of three steps: drop *C* from the add list of *OP*, restrict the preconditions of *OP* to those (if any) that guarantee that *C* is added by the action, or change the actual action routine so that it does in fact behave as originally advertised. While we offer no algorithms for accomplishing these forms of learning, it is interesting to note that the problem itself arises only when we deal with real, as opposed to simulated, robot systems. It is the occurrence of problems of this sort that persuades us of the continuing interest and importance of robot problem solving.

REFERENCES

1. Fikes, R. E. and Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* **2** (1971), 189–208.
2. Fikes, R. E. Monitored execution of robot plans produced by STRIPS. *Proc. IFIP Congress* 71, Ljubljana, Yugoslavia (August 23–28, 1971).
3. Garvey, T. D. and Kling, R. E. User's Guide to QA3.5 Question-Answering System. Technical Note 15, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (December 1969).
4. Raphael, B. et al. Research and Applications—Artificial Intelligence. Final Report, Contract NASW-2164, Stanford Research Institute, Menlo Park, California (December 1971).
5. Hart, P. E. and Nilsson, N. J. Shakey: Experiments in Robot Planning and Learning. Film produced at Stanford Research Institute, Menlo Park, California (1972).
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.