May 1970

ROBOT PROBLEM SOLVING* WITHOUT STATE VARIABLES

by

Bertram Raphael

Artificial Intelligence Group

Technical Note 30

SRI Project 8259

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

*"Problem solving" in this Note refers only to the planning phase of robot activity.

I INTRODUCTION

In the original robot problem-solving system based upon an idea presented by Green,[1*] situations were described by axioms of first-order logic that explicitly included terms representing states. Theorem proving by resolution was then used to prove, constructively, the existence of a state having the properties of a desired goal. The major disadvantages of that approach were

(1) All unchanged and irrelevant properties of the world must be re-established after every state transistion;

(2) The efficiency of the system is very sensitive to the choice of domains and predicates used in the logical encoding, so that sometimes the most "natural" representations do not lead to practical task descriptions.

Waldinger[2] has proposed a scheme for getting around these problems by extending the logic to handle sets and tuples (still keeping an explicit state variable around). In addition to requiring a much more complicated unification procedure, this approach restricts severely the theorem prover's ability to generate important intermediate subgoals, and requires bookkeeping procedures whose details have not yet been worked out. Further study is necessary to determine the feasibility (and desirability) of this approach.

For several months Fikes, Munson, Nilsson, and Raphael have been exploring the possibility of a problem-solving system based upon first-order logic but without state variables. Some of these deliberations have

---

*References appear at the end of this Note.

appeared in informal memos that are partially included in Reference 3. The basic idea is to describe a state by a set of axioms, and a state transition (an action) by an operator that makes specific modifications (deletions and additions) to the set of axioms. This Note is a detailed proposal for one version of such a system.

II TERMINOLOGY

A state corresponds to our intuitive notion of a complete physical situation. The domain of our logical formalism will include physical measurements such as object positions, descriptions, etc. Therefore, every consistent sentence of first-order logic is either true or false for every state. We think of each such sentence as a predicate that defines a set of states, namely those for which it is true. We call such a set of possible states the context defined by the predicate.

We shall find it convenient to allow certain distinguished variables, called parameters, to occur in predicates. Since each such predicate with ground terms substituted for parameters defines a context, a predicate containing parameters may be thought of as defining a family of possible contexts--and each partial instantiation of parameters in the predicate defines a subfamily of contexts (or, if no parameters remain, a specific context).

For example, the predicate At(Box1,L) defines a context (the set of all states) in which object Box1 is at position L. If x and y are parameters, At(x,y) defines the family of contexts in which some object is located any place. At(Box1,y) is a subfamily of this family in which the object Box1 must be located (at some as yet unspecified place).

A problem to be solved is specified by a particular predicate called the goal predicate. The problem, implicitly, is to achieve a goal state, i.e., produce any member of the context defined by the goal predicate.

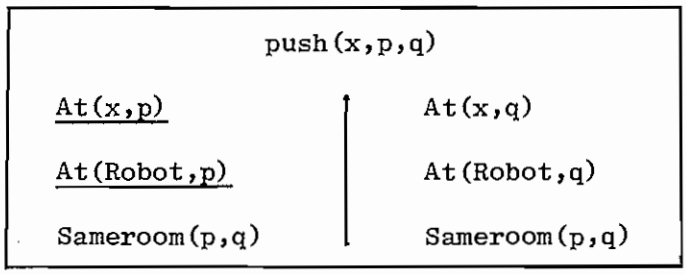
If predicate P1 defines a context C1, then the negated predicate ~P1 defines the complimentary context $\overline{C1}$, i.e., $C1 \cap \overline{C1} = \varphi$ and $C1 \cup \overline{C1}$ = the set of all possible states. If P1 and ~P2 define two contexts C1 and $\overline{C2}$ respectively, then the predicate $P1 \wedge \sim P2$ defines their intersection $C1 \cap \overline{C2}$. If $P1 \wedge \sim P2$ is contradictory, then $P1 \Rightarrow P2$; this is equivalent to saying that if $C1 \cap \overline{C2} = \varphi$, then $C1 \subset C2$, so any state in C1 is also in C2.

In the following we shall assume that each predicate is placed in the form of a set of clauses so that resolution may be used to derive consequences and deduce contradictions.

An operator consists of an operator name, a parameter list, and two predicates--the preconditions and the results. In addition, any units in the clause form of the preconditions may be designated as transient preconditions. An example of a simple robot operator is

Operator name:	push
Parameter list:	(x,p,q)
Preconditions:	{At(x,p), At(Robot,p), Sameroom(p,q)}
Transients:	At(x,p), At(Robot,p)
Results:	{At(x,q), At(Robot,q), Sameroom(p,q)} .

We shall summarize an operator description by a chart as follows (transients are underlined):



We shall generally use the symbol K_{op} to denote the preconditions of operator op, and R_{op} to denote its results.

III MAIN IDEAS

The conjunction of predicates (and tuples) in the robot's model of the world is an initial predicate I, defining as an initial context the set of all states that have, in common, all the known properties of the robot's current world. The goal context, defined by a given goal predicate, is the set of satisfactory target states. When an operator is applied in a context, it changes the defining predicate (in a certain way to be discussed below), thereby changing the context. The problem-solving task is to construct a sequence of operators that will transform the initial context into a subset of the goal context.

Any context that can be reached from the initial context by a finite sequence of operators is called an achievable context. Any context from which a subset of the goal context can be reached by a finite sequence of operators is called a sufficient context. The main task may be restated, then, as finding an operator sequence to show that the goal is achievable, or that the initial context is sufficient, or, more generally, that some achievable context is a subset of some sufficient context (and therefore is itself sufficient).

The problem-solving strategy will frequently use QA3 (and its strategies). In addition, a new executive is needed to select the calls to QA3. This executive can be extremely simple at first; eventually, it may be given sophisticated heuristics of its own for guiding its selection of trial operators.

The main loop of the problem solver consists of two steps:

(1) Test whether any known achievable context is a subset of any known sufficient context. If so, we are done.

(2) Either generate a new achievable context by applying some operator in a known achievable context ("working forwards"), or generate, as a new sufficient context, one which would become a known sufficient context by the application of some operator ("working backwards"). Then return to step 1 to test the newly generated context.

For the remainder of this Note, let P_A be a predicate defining an achievable context, and P_S define some sufficient context. Step 1 of the main loop process can be performed by a resolution theorem prover (say, QA3) simply by trying to prove $P_A \Rightarrow P_S$, i.e., deduce a contradiction from $P_A \wedge \sim P_S$. For step 2, we need to consider how to use operators.

In order to apply an operator in a context, the preconditions K of the operator must be satisfied in the context. This applicability can be assured (by QA3) by proving that the predicate of the context implies the existence of parameter values that make the preconditions true. For example, the push operator described above is applicable to an achievable context A provided

$$P_A \Rightarrow (\exists x,p,q)[At(x,p) \wedge At(Robot,p) \wedge Sameroom(p,q)]$$

is a theorem, or equivalently if

$$P_A \vee At(x,p) \vee \sim At(Robot,p) \vee \sim Sameroom(p,q)$$

is contradictory. If this proof is completed, we may generate the predicate $P_{A'}$ for the new achievable context as follows:

(1) Instantiate the operator's parameters, preconditions, and results with the parameter values determined (say, using an Ans literal) from the above applicability proof. (Parameters that were not instantiated in the proof remain as free parameters.)

(2) Delete from P_A every clause subsumed by any instantiated transient precondition (because such clauses need only be true prior to applying the operator).

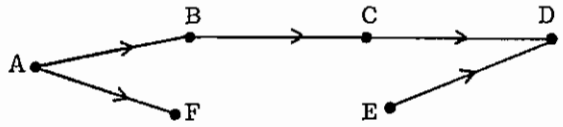
(3) Conjoin to the resulting predicate the instantiated operator's results (because such clauses must be true after applying the operator).

The resulting predicate is $P_{A'}$.

In working backwards, we wish to find an operator (with parameters p) whose results R logically imply the predicate P_S of a sufficient context, i.e., such that $(\exists p)[R \Rightarrow P_S]$ is true, or equivalently $(\forall p)[R \wedge \sim P_S]$ is contradictory. (The proof of this, of course, is a job for QA3.) If such an operator is found, then the (appropriately instantiated) preconditions of the operator constitutes the predicate $P_{S'}$ that defines the new sufficient context.

Example: Finding a path through a directed graph provides a very simple example of this problem-solving approach. Suppose we wish to get

from A to D in the directed graph:



We shall abbreviate by $\mathcal{G}$ the predicate that gives the graph's topology:

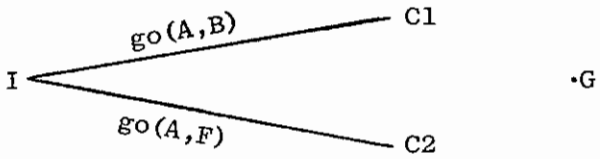
$$\mathcal{G} \equiv \text{Path}(A,B) \wedge \text{Path}(B,C) \wedge \text{Path}(C,D) \wedge \text{Path}(A,F) \wedge \text{Path}(E,D) \quad .$$

The initial predicate is $I \equiv \text{At}(A) \wedge \mathcal{G}$. The goal predicate is $G \equiv \text{At}(D)$. The operator go is defined by:

go(x,y)	
At(x)	At(y)
Path(x,y)	

.

From I and the negation of the precondition $\sim\text{At}(x) \vee \sim\text{Path}(x,y) \vee \text{Ans}(x,y)$ we can get two proofs: Ans(A,B) and Ans(A,F). Therefore, the go operator can be used two ways to generate new achievable contexts C1 and C2, with corresponding predicates $P_{C1} = \mathcal{G} \wedge \text{At}(B)$, $P_{C2} = \mathcal{G} \wedge \text{At}(F)$. To keep track of actions and instantiations, we shall draw a context graph:



Similarly, from C1 and the negation of the precondition we prove the applicability of go(B,C) which, when applied, gives C3

$$P_{C3} = \mathcal{G} \wedge \text{At}(C) \quad .$$

To illustrate working backwards, consider whether the result of a go implies G. From

$$\sim(\exists y)[\text{At}(y) \Rightarrow \text{At}(D)] \quad ,$$

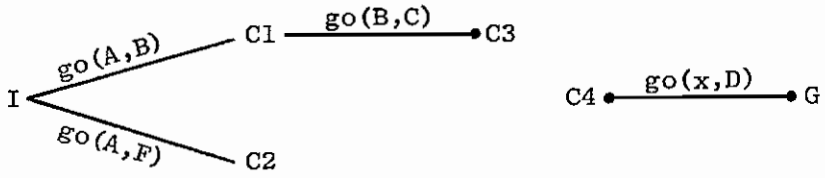
or equivalently

$$[At(y) \vee Ans(y)] \wedge \sim At(D)$$

we get a proof with Ans(D), so a new sufficient context is given by

$$P_{C4} = At(x) \wedge Path(x,D) \quad .$$

(Note that C4 is really a family of contexts, because of the parameter x. This complication will be discussed further in the next section.) The context graph is now:



Finally, we get a contradiction from $P_{C3} \wedge \sim P_{C4}$, and the proof of that contradiction determines that x = C.

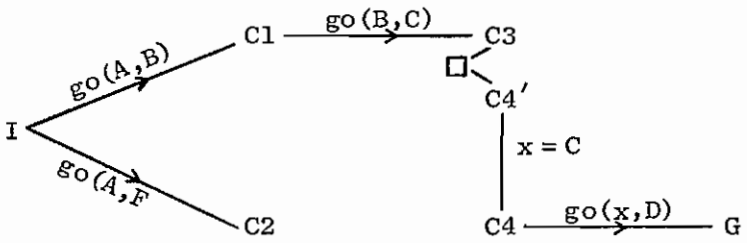
IV FURTHER DETAILS

Several difficulties arise when we try to use this approach in more complicated situations. We shall now examine the major difficulties and see how they may be overcome.

A. Parameters

Whenever a new context is generated, the defining predicate may contain uninstantiated parameters. These then should be thought of as parameters of a family of predicates defining a family of contexts. Since we have complete freedom in assigning values to these parameters, the theorem prover may treat parameters just like ordinary (universally quantified) free variables. However, the act of instantiating a parameter creates a new subfamily of contexts. Parameters do not obey the renaming

rules of variables, but rather keep their identities throughout a set of clauses. When a resolution operation causes a parameter to be instantiated, the resultant clause should be thought of as having been derived only in a context in which the parameter has been appropriately instantiated wherever it occurs. Of course, the parent context family is still available to permit other instances of the same parameter to be used separately. Therefore, the context graph in the above example should really be as follows (arrows indicate operator applications):



where C4′ is a context in the family C4.

B. Augmented Predicates and Restricted Contexts

Suppose P_B is a resolvent of a clause CL1 ϵ P_A and a clause CL2 ϵ $\sim P_S$. If $P_B = \square$ (or, equivalently, if we can prove immediately that P_B is not satisfiable), we are through, for we have shown that $P_A \Rightarrow P_S$. Otherwise, however, what shall we do with P_B? We should not just forget it because it may be a useful intermediate result in the overall proof.

Observe that, if we augment a predicate by conjoining a new clause to it, we restrict the context defined by the predicate by selecting that subset of the context which satisfies the new clause. Any such restriction of a sufficient context is therefore itself sufficient.

In the present case, we have proven that $[CL1 \wedge \sim P_S \Rightarrow P_B]$, which is equivalent to proving $[CL1 \wedge \sim P_B \Rightarrow P_S]$. If we augment P_S by

CL1, this establishes that $\sim P_B$ defines a new sufficient context, provided we insure as a side condition that CL1 is true at each step in proving the achievability of the $\sim P_B$ context. This will not be difficult, since a clause true in any context is true in all following contexts unless explicitly deleted (by subsumption by a transient precondition of an operator). One way to keep track of augmenting literals is to include them in the new sufficient predicate, but with a special identification flag that makes them invisible to the problem solver until all other literals are gone (proven). At the final step, all augmenting literals reappear and their truth must be verified.

C. Conditional Operations

Suppose we cannot completely prove the applicability of an operator's preconditions in an achievable context (or, similarly, the adequacy of an operator's results for guaranteeing a sufficient context). The intermediate results generated during such attempts may still be useful, so we must determine their roles in the overall scheme.

Consider first the case of establishing applicability. Let A be an achievable context defined by predicate P_A. Suppose from $P_A \wedge \sim K$, QA3 cannot deduce a contradiction but instead deduces some resolvent P_D. If P_D could be immediately proven false in A, then the operator would be applicable; however, in that case $\square$ is deducible from $P_A \wedge \sim K$ and we need not be concerned with P_D. In one significant case, on the other hand, it is more convenient to delay this deduction of $\square$. This is the case in which P_D contains parameters. Although P_D may be provably false for certain parameter values, it is more flexible to leave

the choice of parameters until a later stage (in a following context). This can be done by applying the operator as before, but replacing the operator's results R by $[R \vee P_D]$ in the predicate of the new achievable context. If P_D is subsequently proven false, this tentative operator usage will have been justified. If P_D turns out to be satisfiable, then R automatically will become irrelevant for future inferences and the new context will essentially degenerate to a useless superset of its predecessor. Caution: Any eventual proof of the falsehood of P_D must be achieved with clauses that remain continuously true from the context of P_A through the context in which the proof of falsehood is achieved. It would be meaningless to prove that an operator was applicable at one time by using facts introduced by another operator at a later time. This time dependence of literals creates a significant bookkeeping problem.

An analogous problem exists in the "working backwards" case. If from $[R \wedge \sim P_S]$ we deduce P_D, then K defines a new sufficient context only if P_D is false in the context of P_S. If K' is the appropriate instance of K, then the new context is defined not by K' but by $[K' \wedge \sim P_D]$--and we must do the bookkeeping necessary to insure that the literals of $\sim P_D$ are "resolved away" only by clauses that remain true through the context defined by P_S.

V EXAMPLE

Without worrying yet about the details of bookkeeping, answer tracing, or proof strategies, we now present a solution illustrating "working forwards," "working backwards," augmenting literals, and conditionally applied operators, to "the three-box problem": Three boxes are scattered around the room, and the robot is sitting in the corner.

He must bring the three boxes together. (A context graph appears at the end of the solution.)

P_I: $At(a,\ell_a) \wedge At(b,\ell_b) \wedge At(c,\ell_c) \wedge At(rob,\ell_d)$

P_G: $(\exists x)[At(a,x) \wedge At(b,x) \wedge At(c,x)]$

Operators:

go(x,y)	
K_{go}	R_{go}
<u>At(rob,x)</u>	At(rob,y)

push(u,v,z)	
K_{push}	R_{push}
<u>At(rob,v)</u>	At(rob,z)
<u>At(u,v)</u>	At(u,z)

1. (Working forwards.) Is <u>go</u> applicable to context I? From $[P_I \wedge \sim K_{go}]$ deduce $\square$ with $x = \ell_d$. Apply $go(\ell_d,y)$ producing the class of contexts C1.

P_{C1}: $At(a,\ell_a) \wedge At(b,\ell_b) \wedge At(c,\ell_c) \wedge At(rob,y)$

2. Is <u>push</u> applicable to any context of C1? From $[P_{C1} \wedge \sim K_{push}] = P_{C1} \wedge (\sim At(rob,v) \vee \sim At(u,v))$ can deduce $\square$ with, e.g., $u = a$, $y = v = \ell_a$. Apply $push(a,\ell_a,z)$ to C1′, the instance of C1 in which $y = \ell_a$. This produces a new context class C2 with parameter z.

P_{C2}: $At(b,\ell_b) \wedge At(c,\ell_c) \wedge At(rob,z) \wedge At(a,z)$

3. (Let's test for a solution.) Is (any instance of) C2 a subset of G? Construct $\sim P_G$:

$\sim P_G$: $\sim At(a,x) \vee \sim At(b,x) \vee \sim At(c,x)$.

From $P_{C2} \wedge \sim P_G$ we cannot deduce $\square$ but we <u>can</u> deduce, e.g., (by resolving on the <u>b</u> literals) $\sim At(a,\ell_b) \vee \sim At(c,\ell_b)$.

Enclosing augmenting literals by wedges, we thus get a new sufficient context G':

$$P_{G'}: \quad At(a,\ell_b) \wedge At(c,\ell_b)\langle \wedge\ At(b,\ell_b)\rangle$$

4. (Let's try working backwards.) Can push lead to G'? From $R_{push} \wedge \sim P_{G'} = At(rob,z) \wedge At(u,z) \wedge [\sim At(a,\ell_b) \vee \sim At(c,\ell_b)\langle \vee \sim At(b,\ell_b)\rangle]$ we cannot deduce $\square$ but we can resolve, say, on the c literal and get $\sim At(a,\ell_b)\langle \vee \sim At(b,\ell_b)\rangle$ with $u = c$, $z = \ell_b$. Conditionally, apply $push(c,v,\ell_b)$ backwards to G' to get context class H1 (with parameter v)

$$P_{H1}: \quad At(rob,v) \wedge At(c,v) \wedge \overset{(G')}{At(a,\ell_b)}\langle \wedge\ \overset{(G')}{At(b,\ell_b)}\rangle$$

where two literals have been flagged to be proven only from facts true in context G'. (Note that if we ever do execute the push transition from H1 to G', then the robot will be at ℓ_b in context G'--but this fact was not used and therefore is irrelevant to the present proof structure.)

5. (Another solution test.) Does P_{C2} imply P_{H1}? From

$$P_{C2} \wedge \sim P_{H1} = P_{C2} \wedge [\sim At(rob,v) \vee \sim At(c,v) \vee \sim \overset{(G')}{At(a,\ell_b)}\langle \vee \sim \overset{(G')}{At(b,\ell_b)}\rangle]$$

we can resolve the last clause on the a literal, which instantiates z in C2 producing $C2'$:

$$P_{C2'}: \quad At(b,\ell_b) \wedge At(c,\ell_c) \wedge At(rob,\ell_b) \wedge At(a,\ell_b)$$

and a new sufficient augmented context $H1'$:

$$P_{H1'}: \quad At(rob,v) \wedge At(c,v)\langle \wedge\ At(a,\ell_b) \wedge \overset{(G')}{At(b,\ell_b)}\rangle$$

(If the a literal augments H1, it will satisfy the G' constraint.)

6. (Continuing the solution test.) Does $P_{C2'}$ imply $P_{H1'}$? With $v = \ell_c$ we get, from $P_{C2'} \wedge \sim P_{H1'}$,

$$P_{H1''}: \quad At(rob,\ell_c)\langle \wedge\ At(c,\ell_c) \wedge At(a,\ell_b) \wedge \overset{(G')}{At(b,\ell_b)}\rangle \quad .$$

7. (Working backwards again.) Can the go operator guarantee $H1''$? From $R_{go} \wedge \sim P_{H1''}$, with $y = \ell_c$, we can deduce only the augmenting literals. Apply $go(x,\ell_c)$ conditionally backwards to $H1''$.

$$P_{H2}: \quad At(rob,x)\langle \wedge\ \overset{(H1'')}{At(c,\ell_c)} \wedge \overset{(H1'')}{At(a,\ell_b)} \wedge \overset{(G,H1'')}{At(b,\ell_b)}\rangle$$

where new flags have been added to constrain the proof of the carried-over literals.

8. (Finally.) Does $C2'$ imply H2? Yes! From $P_{C2'} \wedge \sim P_{H2}$ we first deduce only the augmenting literals (by setting $x = \ell_b$), and then they become visible and can all be resolved away, completing the proof. In justifying these final resolutions with flagged literals, we must check that the operators that will get us to $H1''$ and G' will not delete the required literals from $C2'$.

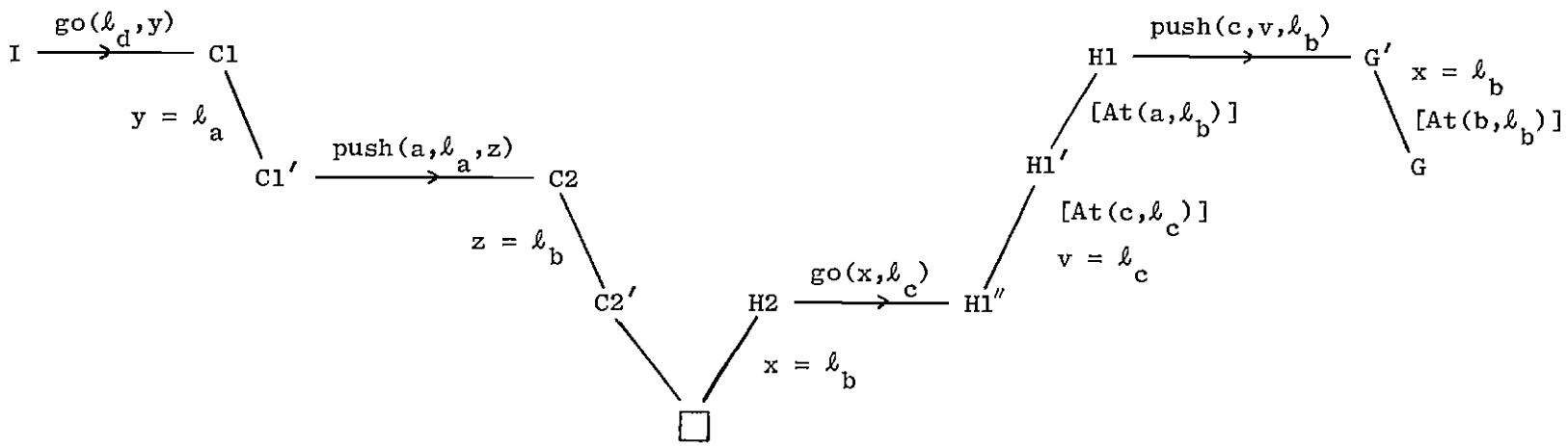
From the context graph (below) we can now read the solution: Go from ℓ_d to ℓ_a, push a from ℓ_a to ℓ_b, go from ℓ_b to ℓ_c, push c from ℓ_c to ℓ_b, and we will have achieved the goal.

This context graph contains a single path, rather than numerous multiple branches, because we judiciously chose only correct steps in the above example. In practice the graph may be expected to show many false tries. The efficiency of the entire scheme depends upon bookkeeping and strategic matters such as those discussed below.

VI IMPLEMENTATION CONSIDERATIONS

A. Bookkeeping

Several bookkeeping problems threaten the practicality of a problem-solving system such as the one discussed above. Our purpose



CONTEXT GRAPH

in this section is to identify the major difficulties and suggest how they might be handled; of course, all the details of an implementation have not yet been worked out.

1. Context Graph

The partial solutions developed during the problem-solving process are not automatically kept within the derived logical expressions, as they would be in a system using state variables. Therefore, some extra-logical structure is necessary to "remember" the actions and instantiations that lead to any particular clause. The context graph holds precisely this information. Therefore one needs to store with each clause the identification of its context, and maintain a separate list-structure representation of the context graph from which to determine the relationships between contexts and to reconstruct the problem's solution after its existence has been determined.

2. Implicit Context Membership

Perhaps the conceptually simplest logical-expression memory would consist of a set of complete predicates, one for each context occurring in the context graph. However, this would entail much wasted space and time spent copying clauses from one predicate to another. In particular, the initial context may be defined by a large number--perhaps hundreds and, eventually, thousands--of clauses describing the robot's world, almost all of which are unaffected by any contemplated actions. Each operator causes the deletion and insertion of only a small number of clauses as it creates a new context. Therefore, it seems preferable to associate with each clause the identity of the context

in which it was first known or created, and those contexts at which it was deleted; its presence in any context can then be determined from the context graph. This treatment of context identity would also permit the problem solver to keep all intermediate results in a single uniform clause memory.

3. Instantiation Bookkeeping

Instantiation of variables that arise from the negation of existentially quantified variables in the goal predicate can be traced automatically, as usual, by an Ans literal; the instances selected are not important until the construction of the final plan. Parameters introduced by operators behave somewhat differently. They must be traced during the test of an operator's applicability (or sufficiency), then immediately substituted into the appropriate places upon generation of a new context. The Ans idea will probably still work, provided special rules are worked out for inserting and deleting these bookkeeping literals. (Remember that augmenting literals, and literals introduced by the conditional application of operators, also must be handled in special ways by the theorem prover.)

4. Complementary Contexts

One complication in our example solution to the three-box problem was a frequent negating of key predicates. This occurred because we wished to exhibit the predicates that defined certain contexts, but in trying to deduce a contradiction we needed the predicates of the complementary contexts. However, except for expository purposes, these negating operations do not seem necessary. We always work with

the predicates of achievable contexts (and operator results), and the predicates of the complements of sufficient contexts (and negations of operator preconditions). (These complemented sufficient contexts contain what we have previously called "shadow states.")

B. Strategies

Everything discussed thus far in this note concerns the specification and manipulation of a formal space containing problem descriptions and solutions. They key question we have ignored so far is, what strategy should be used to search for a solution in this formal space?

A quick answer is, any strategy you wish. We have defined a formalism in which states of the world, operators, and tasks may be easily and naturally stated. Once a few (hopefully straightforward) details and conventions are worked out, e.g., an internal representation for the context graph, all the mechanical operations of applying operators and testing contexts can be implemented. At that time any strategy for guiding the planning processes may be superimposed. Discovery of a "best" planning strategy is a major research task, and may be worked on during the next year in parallel with other aspects of the overall robot task, such as the execution of any plan and the use of sensory data to update the model.

Of course, several extremely simple (and inefficient) strategies are immediately apparent, and one or more of these should be implemented as soon as possible just to get things started. For example, these three:

(1) Man-machine. Every strategic decision is made at the teletype (or display). The man commands, e.g., "Apply operator go to context C2," or "Test whether context H1 is implied by C1′," and the system responds with the results and an updated context graph.

(2) Breadth-first backwards. The system builds a breadth-first tree of sufficient contexts, trying to reach the goal with all possible operators, and testing each new context against the initial context (but ignoring partial results from unsuccessful tests). This is similar to the natural strategy used by Waldinger's state-variable system,[2] and requires considerably less bookkeeping than many other strategies. Other strategies of the same type include breadth-first forwards, depth first with level bound, etc.

(3) QA3 strategy. The initial model clauses, the negation of the goal predicate, and all the operator results and negated preconditions are placed in one big collection and QA3 is turned loose to find a proof, with the negated goal predicate as set of support. Of course, extra filters are inserted to test the legitimacy of resolving clauses that may come from different contexts, and various QA3 subroutines are souped up to maintain the context graph and to handle parameters correctly. I have no idea how effective a problem solver this would

be, but suspect that it is significantly better that those proposed in (2) above, while being easier to implement than a special difference-oriented problem-solving strategy such as the one currently being developed by R. E. Fikes.

VII FUTURE PLANS

The system described above is now a prime candidate for implementation as the Planner for the first PDP-10 robot system. Comments, criticisms, detailed counter proposals, etc., should be presented within the next few weeks so that a firm decision can be made.

REFERENCES

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

2. Richard Waldinger, "Robot and State Variable," Technical Note 26, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (April 1970).

3. Richard Duda et al., "Research and Application--Artificial Intelligence," Interim Scientific Report, Contract NAS12-2221, Stanford Research Institute, Menlo Park, California (April 1970).