USING GOAL STRUCTURE TO DIRECT SEARCH IN A PROBLEM SOLVER

BRIAN AUSTIN TATE

Ph.D
University of Edinburgh
1975

ABSTRACT

This thesis describes a class of problems in which interactions occur when plans to achieve members of a set of simultaneous goals are concatenated in the hope of achieving the whole goal. They will be termed "interaction problems". Several well known problems fall into this class. Swapping the values of two computer registers is a typical example.

A very simple 3 block problem is used to illustrate the interaction difficulty. It is used to describe how a simple method can be employed to derive enough information from an interaction which has occurred to allow problem solving to proceed effectively.

The method used to detect interactions and derive information from them, allowing problem solving to be re-directed, relies on an analysis of the goal and subgoal structure being considered by the problem solver. This goal structure will be called the "approach" taken by the system. It specifies the order in which individual goals are being attempted and any precedence relationships between them (say because one goal is a precondition of an action to achieve another). We argue that the goal structure of a problem contains information which is simpler and more meaningful than the actual plan (sequence of actions) being considered. We then show how an analysis of the goal structure of a problem, and the correction of such a structure in the light of any interaction, can direct the search towards a successful solution.

Interaction problems pose particular difficulties for most current problem solvers because they achieve each part of a composite goal independently and assume that the resulting plans can be concatenated to achieve the overall goal. This assumption is beneficial in that it can drastically reduce the search necessary in many problems. However, it does restrict the range of problems which can be tackled. The problem solver, INTERPLAN, to be described as a result of this investigation, also assumes that subgoals can be solved independently, but when an interaction is detected it performs an analysis of the goal structure of the problem to re-direct the search. INTERPLAN is an efficient system which allows the class of interaction problems to be coped with.

INTERPLAN uses a data structure called a "ticklist" as the basis of its mechanism for keeping track of the search it performs. The ticklist allows a very simple method to be employed for detecting and correcting for interactions by providing a summary of the goal structure of the problem being tried.

TABLE OF CONTENTS

1. INTRODUCTION

For a robot device to be self-controlling, it will certainly require a problem solving (planning) capability. Existing systems, such as STRIPS for the SHAKEY robot at Stanford Research Institute (Fikes and Nilsson, 1971), are severely restricted in that they take a long time to produce even short and straightforward plans and operate only in quite simple domains.

Michie (1974) describes a problem, the Keys and Boxes problem, whose solution poses several difficulties for current problem solving techniques and is beyond their capabilities. The work to be described in this thesis results from an investigation of the difficulties encountered by several existing problem solvers on the Keys and Boxes problem. In the process of overcoming them we have designed and tested a general and effective problem solving system.

1.1 Interaction problems

The Keys and Boxes problem, though it has other complications, is a member of the specific class of problems considered in this work, namely those in which interactions occur when plans to achieve separate members of a set of simultaneous goals are concatenated in the hope of achieving the whole goal. They will be termed "interaction problems". Several well known problems fall into this class. The problem of swapping the values of two computer registers is a typical example.

Given that register 1 holds a value C1 and register 2 holds a value C2, we wish register 1 to hold a value C2 and register 2 a value C1 when an assignment operator is available. Either of the separate parts of the simultaneous goal can easily be achieved using a single assignment. However, after doing one of the assignments, the other will not achieve the desired result. This is because conditions which must be true for an assignment to achieve the expected result are altered by the previous assignment. It is important to note thatthe achievement of either goal in any order independently will not lead to a solution to the problem. In this problem we must realize that an intermediate register should be used to hold one of the values needed.

Until recently, systems which could cope with such interaction problems did so in either a domain-dependent fashion (by knowing that an intermediate register should be used in register swapping) or by having a very much larger search space than would otherwise be necessary. Our aim in this work has been to develop a problem solving system which could deal with interaction problems but has neither of the above limitations.

A problem which is simpler than the Keys and Boxes, the 3 block problem, is used to illustrate more clearly the interaction difficulty. It is used to describe how a simple method can be employed to derive enough information from an interaction which has occurred to allow problem solving to proceed in an effective way.

1.2 Goal structure

It would be inefficient merely to extend the search space of the problem solver to allow different orderings of the achievement of sub-goals, and hope to be able to search through these for a solution using, for example, a backtracking algorithm to select between the alternatives. Instead, INTERPLAN can open up its search space selectively in view of information gleaned from any interactions which occur during an initial attempt to solve the problem.

The method used to detect interactions and derive information from them, allowing problem solving to be re-directed, relies on an analysis of the goal and subgoal structure being considered by the problem solver. This goal structure will be called the "approach" taken by the system. It specifies the order in which individual goals are being attempted as well as any precedence relationships which exist between them (say because one goal is a precondition of an action to achieve another). We will argue that the goal structure of a problem contains information which is simpler and more meaningful than the actual plan (sequence of actions) which is being constructed by the problem solver during an attempt to solve a problem We will then show how an analysis of the goal structure of a problem, and the correction of such a structure in the light of any interactions, can direct the search towards a successful solution.

Many current problem solvers achieve each part of a composite goal independently and assume that the resulting plans can be concatenated to achieve the overall goal. This assumption is beneficial in that it can effect a drastic reduction in the search necessary in many problems. However, it does also severely restrict the range of problems which can be solved. In particular, interaction problems cannot be coped with. We will describe a problem solver, INTERPLAN, which also assumes that subgoals can be solved independently and concatenated to achieve a composite goal. However, should this prove to be invalid, INTERPLAN can perform an analysis of the goal structure of the problem to derive a new "approach" which should be tried to avoid interactions. INTERPLAN is an efficient system which allows the class of interaction problems to be coped with.

The system makes productive use of the information available from a failure. Some earlier systems, such as HACKER (Sussman, 1973) and the LISP theorem prover of Boyer and Moore (1972), also used information from the failure of some process to alter or guide further problem solving efforts. INTERPLAN provides a particularily simple method of detecting important information from its failures.

1.3 Ticklists

During the study of existing systems such as STRIPS (Fikes and Nilsson, 1971) and HACKER (Sussman, 1973), a simple method of controlling the growth of the search tree of the problem solver using a data structure called a "ticklist" was devised. The ticklist provides a summary of the goal structure of the problem being tackled. It allows a simple scheme to be used for growing the search tree and for detecting any difficulties which occur during problem solving. Such a search tree growth scheme using "ticklists" has been used in INTERPLAN.

1.4 Other relevant work

While the present study was in progress, other workers have written problem solvers which are able to cope with interaction problems. WARPLAN (Warren, 1974) and a program-synthesis system written at SRI (Waldinger, 1975) assume, as earlier systems did, that independent plans can be found to achieve sub-goals. However, instead of assuming that these can be concatenated sequentially, they allow the actions found for each sub-goal to be inserted at any point in the existing partial plan for sub-goals already achieved. NOAH (Sacerdoti, 1975) takes a very different approach. It does not make assumptions about the ordering of the individual actions within a plan until such an ordering is constrained by the interactions which occur. Both WARPLAN and NOAH are described and compared with INTERPLAN later in this report.

2 ROBOT PROBLEM SOLVING

In order to introduce the terminology to be used throughout this report and to briefly describe several problem solvers upon which this work was based we will describe the control structures used by problem solvers to keep track of the growth of the search tree. We will argue that a "backup" type of goal control tree allows a localization of search information which is important if failures in a solution strategy are to be used to guide further problem solving efforts.

2.1 Problem paradigm

Many problems can be formulated as a SEARCH task. This can be represented as follows (e.g., as in Frnst and Newell, 1969):-

GIVEN: an initial state representation

a number of actions (operators) which transform one state to another if applicability conditions are met

a definition of a desired (goal) state

FIND: a sequence of actions (a plan) which will transform the initial state into a desired state.

This can be viewed as a graph search problem (see Nilsson, 1971, for background and terminology):

GIVEN: a node of a graph

a set of operators (represented by arcs of the graph)

a set of nodes satisfying a goal condition

FIND: a sequence of operator applications (arcs) which will generate a path leading from the initial node to a goal node.

2.2 Problem representation

For expository purposes let us assume that a problem state (or problem situation) is described by a list of assertions about the state. Operators can be described by giving the effects they have on a state when applied and by giving the applicability conditions for the operator. The effects of the operator can be specified by a list of statements ADDED (those made true) and DELETED (those no longer true) from the state. The applicability conditions can be specified by a list of statements which must be true in the state to which the operator is applied (often called the PRECONDITIONS). Goal states can then be specified by giving a list of statements which are required to be true in a state satisfying the goal.

This representation for a domain was proposed for STRIPS (Fikes and Nilsson, 1971) and greatly simplifies the checks needed for relevance and applicability of operators.

2.3 Forward search

Forward search can cope with a wide variety of problems formulated in the state-space paradigm, especially when heuristic control is used to guide the search across the graph, for example, as in the Graph Traverser (Doran and Michie, 1966 and Michie and Ross, 1969). A node of the graph (corresponding to the initial problem state) is identified and APPLICABLE operators are applied to it to produce successor nodes. Some node from the successors is chosen for expansion, typically the node heuristically estimated to be closest to a goal node. APPLICABLE operators are then used on this chosen node. This process continues until a node satisfying the goal conditions is generated.

2.4 Means-end analysis

In robot planning problems, the number of APPLICABLE operators is typically large (or even infinite). There may, for instance, be an action GOTO(x,y) which can move a robot between any two points, x and y, on a 1000 X 1000 grid. Forward search is not appropriate for such problems. It is necessary to use some method of restricting the number of APPLICABLE operators which need to be used. A technique was introduced in the General Problem Solver
(GPS - a full account is given in Ernst and Newell, 1969)
to cope with this difficulty. It is termed MEANS-END ANALYSIS since it considers only those operators which are RELEVANT to achieving some desired goal. Hayes (1973) found that he could not use forward search for a large scale journey planning system in which over 2000 different journey components could be used. He used means-end analysis to guide the search of his system. There is good evidence that means-end analysis is extensively used during human problem solving (Newell and Simon, 1972).

Means-end analysis has been employed by several robot planning systems, e.g., STRIPS (Fikes and Nilsson, 1971), LAWALY (Siklossy and Dreussi, 1973) and HACKER (Sussman, 1973). Such systems find which statements must be true in a desired situation, but which are not true initially. These statements become a "difference" and only operators "relevant" to reducing this difference (typically operators which can ADD one or more statements of the difference) can be considered. One of the operators is chosen and, if applicable, is applied to produce a new situation. The system then once again compares the desired situation with the newly produced one to see if there is any remaining difference.

However, it is possible that a chosen operator may not be applicable in the given situation. In this case the difference between the preconditions and the given situation is constructed and means-end analysis is again used to select from operators relevant to reducing this new difference. Once its preconditions are met, an operator can be applied. Such a process can recur to any depth if operators are chosen which are not applicable in the given situation. Search is certainly not ruled out in such a system, as often there will be more than one "relevant" operator and the order in which preconditions are satisfied may vary. Each choice must be capable of being explored if necessary.

Of course, just as forward search can be impractical when there are a large number of APPLICABLE operators, means-end analysis can be impractical when there are a large number of RELEVANT operators. A great deal of research in robot problem solving has involved ways of cutting down the number of RELEVANT operators, e.g., some way of considering individual statements of a difference by putting priorities on them (as in GPS and LAWALY).

For means-end analysis to be used, the problem must be described in such a way as to allow the RELEVANT operators to be identified for any goal. The representation of states as a list of assertions and operators as ADD, DELETE and PRECONDITION lists (as mentioned in section 2.2) fulfils this requirement and has been adopted by many problem solvers, e.g., STRIPS and HACKER. Problems to be tackled by forward search techniques can be described in different ways since only APPLICABILITY conditions need be checked before the operator's use.

2.5 Search trees in means-end analysis driven problem solvers

2.5.1 An example

We now describe a simple problem designed to illustrate means-end analysis. The solution is found without any incorrect decisions being taken. However, it does serve to explain the differences in the type of control structures built by different problem solvers.

There are 2 operators:

```
(PICKUP ?OB)                              ?OB is a variable with identifier OB.
   ADD        (HELD ?OB)
   DELETE     (HELD NOTHING)
   PRECONDS   (AT ?OB ?X) & (AT ROBOT ?X)

(GOTO ?X)
   ADD        (AT ROBOT ?X)
   DELETE     (AT ROBOT == )              "==" matches anything at all.
   PRECONDS   (HELD NOTHING)              It can be interpreted as
                                          a free variable.
                                               The DELETE statement says:
                                               ∀x DELETE (AT ROBOT x)
```

```
In an initial situation:   (AT BALL A)
                           (AT ROBOT B)
                           (HELD NOTHING).
```

Achieve a situation in which (HELD BALL) is true.

2.5.2 Means-end analysis on the example

A trace of a means-end analysis approach on the example will be given below. Two types of arrows will be used: a single shafted arrow indicates an operator considered relevant to achieving a required goal, a double shafted arrow indicates an operator application.

(HELD BALL)	(HELD BALL) the top level goal is not true in the present (initial) situation.
↑ (PICKUP BALL)	A (PICKUP BALL) is the only operator which can ADD (HELD BALL). It can be applied if its preconditions (AT BALL ?X)&(AT ROBOT ?X) are true.
(AT BALL ?X)&(AT ROBOT ?X)	(AT BALL ?X) is true if X=A. See NOTE below. However, all preconditions are not satisfied until (AT ROBOT A) is also true.
(GOTO A) ↑	A (GOTO A) is the only relevant operator. This can be applied if its precondition (HELD NOTHING) is true.
(HELD NOTHING)	(HELD NOTHING) is true in the initial situation and so the (GOTO A) action can be applied to produce a new situation, say S1, in which (AT BALL A), (AT ROBOT A), and (HELD NOTHING) were true.
(GOTO A) ⇓	
S1	Now, in S1, the preconditions of the (PICKUP BALL) operator hold and so this relevant action can be applied to produce a new situation, say S2, in which (AT BALL A), (AT ROBOT A) and (HELD BALL) were true. (HELD BALL) the top level goal now holds in S2 so the problem is solved with the plan (GOTO A); (PICKUP BALL).
(PICKUP BALL) ⇓	
S2	

NOTE: (AT ROBOT ?X) would be true if X=B, so the preconditions of (PICKUP BALL) could also be made true if (AT BALL B) was achieved. However, in this simple example there is no way to achieve (AT BALL ?X).

2.5.3 Goal control trees

It is useful to consider the data structure generated by means-end analysis as being composed of 2 parts.

1) There is the part of the structure which corresponds to the tree grown over the state-space problem graph by a forward search algorithm. We will term this part the STATE-SPACE TREE. The arcs of this tree are operator applications, the nodes are problem states (or situations). In the example of section 2.5.2, the STATE-SPACE TREE is as below.

```
                                       (AT BALL A)
                   Initial State       (AT ROBOT B)
                         ||            (HELD NOTHING)
      apply (GOTO A)     ||
                         \/            (AT BALL A)
                         S1            (AT ROBOT A)
                         ||            (HELD NOTHING)
  apply (PICKUP BALL)    ||
                         \/            (AT BALL A)
                         S2            (AT ROBOT A)
                                       (HELD BALL)
```

2) There is also the part of the structure which can be termed the GOAL CONTROL TREE. This is used to record the goals being considered at each point. The nodes of this tree represent the goals which are required to be true in a particular situation. Such nodes are represented below as a pair [situation, goal]. The arcs of the tree are of two types:

a) they can be RELEVANT operators which if applied would help to achieve a goal. A successor node below such an arc generally has a different goal to be solved (the applicability conditions of the relevant operator), but the situation the goal is to be considered in remains unchanged.

b) Another type of arc is the APPLICATION of an operator. This causes the situation the goals are being considered in to alter and causes a resetting of the goal being considered to some earlier goal.

In the example in section 2.5.2 the GOAL CONTROL TREE is as follows:

```
                                                              Is goal solved
                                                            in given situation
                                                           |
--[Initial Sitn, (HELD BALL)]                              |      NO
|                                                          |
|              (PICKUP BALL)                               |
|                                                          |
|   --[Initial Sitn, (AT BALL ?X)&(AT ROBOT ?X)]           |      NO
|   |                                                      |     X=A
|   |                 (GOTO A)                             |
|   |                                                      |
|   |      [Initial Sitn, (HELD NOTHING)]                  |      YES
|   |           ||                                         |
|   |           || apply (GOTO A)                          |
|   |           \/                                         |
|   --[Initial Sitn;(GOTO A),  (AT BALL ?X)&(AT ROBOT ?X)] |      YES
|               ||                                         |
|               || apply (PICKUP BALL)                     |
|               \/                                         |
--[Initial Sitn;(GOTO A);(PICKUP BALL),  (HELD BALL)]      |      YES
```

Note: The question answering within one particular situation is separated from the search across a space of situations (by the search for appropriate action sequences). Different mechanisms are used for these widely differing tasks.

In the above diagram dotted lines link nodes which have the same GOALS. Some means must be incorporated of knowing which goal is to be considered at each stage. In the next section two possible ways to do this will be described.

2.5.4 Push-down goal lists vs. Backup

As indicated in the diagram in section 2.5.3 the GOAL CONTROL TREE generated by means-end analysis has nodes in which we ask a question: is a certain goal true in a given situation? If the answer is YES, typically some operation is performed to generate a new situation. If the answer is NO, relevant operators are found to try to achieve the goal. In the latter case, the goal becomes the achievement of the applicability conditions of a chosen operator.

Push-down goal lists - as used in STRIPS

STRIPS has a method of keeping track of the questions to be asked in turn to solve some problem which involves the use of a push-down list of the goals to be solved. Only the top element of the push-down list is considered at any time. If the goal is solved in the given situation, the top element of the push-down list is removed. If this was the only entry the top-level goal is solved. If it is not the only entry, the goal removed was the applicability conditions of some operator which was considered relevant to achieving some earlier goal. This relevant and applicable operator is then applied to produce a new situation. The process is then repeated by asking if the top element of the push-down goal list is true in the new given situation. If the goal is not true some relevant operator is chosen and its applicability conditions are pushed onto the goal list. The process is once again repeated.

For the GOAL CONTROL TREE shown in section 2.5.3, a STRIPS-like version of this would be as follows.

Note: Push-down goal list has the top element to the left.

	Is top goal in the push-down goal list solved in given sitn
[Initial Sitn, ((HELD BALL))]	NO
↑ (PICKUP BALL) relevant	
[Initial Sitn, ((AT BALL ?X)&(AT ROBOT ?X),(HELD BALL))]	NO X=A
↑ (GOTO A)	
[Initial Sitn, ((HELD NOTHING),(AT BALL ?X)&(AT ROBOT ?X), (HELD BALL))]	YES
⇓ apply (GOTO A)	
[Initial Sitn;(GOTO A), ((AT BALL ?X)&(AT ROBOT ?X),(HELD BALL))]	YES
⇓ apply (PICKUP BALL)	
[Initial Sitn;(GOTO A);(PICKUP BALL), ((HELD BALL))]	YES

Top element of push-down goal list removed, so goal solved.

Considering goals at the top level of the push-down goal list only, means that once an operator has been chosen as relevant, the algorithm becomes single-minded in its attempts to achieve that goal. Earlier goals which were originally achieved and then made false by the efforts to solve a later goal are not noticed.

Backup

A different approach to the recording of goals and the situations they are being considered in is suggested by the links in the goal control tree diagram in section 2.5.3 between nodes which have the same goal. There is always a symmetry between two nodes which have the same goal. Note that the operator relevant to achieving the goal has been applied when the goal is reconsidered. It is therefore possible to substitute a backward arrow up the goal control tree for APPLICATIONS of relevant operators which in the push-down goal list tree caused entries lower in the tree. For the goal control tree in section 2.5.3 the backup version would be:

```
                                                              Is goal solved in
                                                              the given situation
                                                          |
[Initial Sitn,                          (HELD BALL)]      |       NO
[Initial Sitn;(GOTO A);(PICKUP BALL),   (HELD BALL)]      |       YES
                                 ↑  ⇑                     |
   (PICKUP BALL) relevant        |  ‖  apply (PICKUP BALL) |
                                 |  ‖                     |
[Initial Sitn,             (AT BALL ?X)&(AT ROBOT ?X)]    |     NO, X=A
[Initial Sitn;(GOTO A),    (AT BALL ?X)&(AT ROBOT ?X)]    |       YES
                                 ↑  ⇑                     |
       (GOTO A) relevant         |  ‖  apply (GOTO A)     |
                                 |  ‖                     |
[Initial Sitn,             (HELD NOTHING)]                |       YES
```

A NO answer to a question results in further subgoaling downwards, a YES answer causes backup and the application of the operator. Such a backup goal control tree allows goals which become false as a result of later steps in a plan to be easily detected. This localization of information about the search has been found very useful and is the basis of an idea to be described later (TICKLISTS) which can provide a simple method of checking that the search is being performed in the intended manner. Ticklists are used as a simple method of implementing a backup goal control tree in INTERPLAN.

2.6 HACKER and goal protection

HACKER (Sussman, 1973) is a system which can write programs (make plans) for the operation of a robot hand in the blocks world. It operates by suggesting a simple program (plan) which may have the intended effect on some problem, monitoring a simulation of the running of this program and then making corrections for any "bugs" which occur.

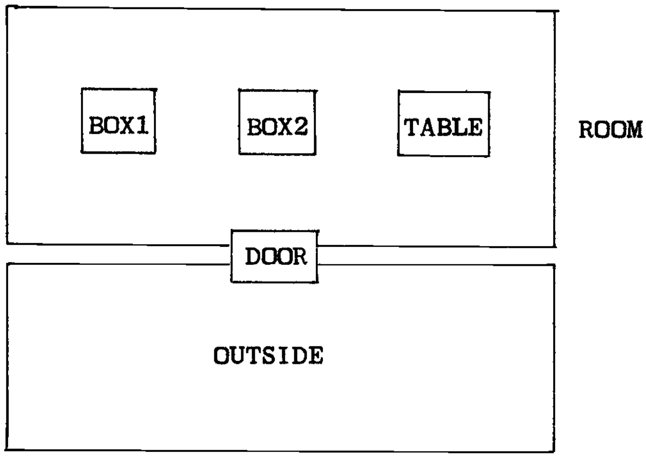
The problem solving process used in HACKER is means-end analysis with an important addition. Each goal that is achieved is noted as being PROTECTED up until the time it need no longer be kept true. If it is a top level goal, once achieved it must remain true until the whole conjunct of goals is solved. If it is a precondition it must remain true until the action it is a precondition of is applied. Any violations of this protection (i.e., an action deletes some protected goal whilst achieving some other goal) is reported to HACKER. HACKER then examines a trace of the simulation of the program and compares this trace with types of traces it knows can cause similar violations. If the trace is of known type, an appropriate change in the program is made and the program simulated again.

HACKER has many more features than the simple problem solving part outlined above. It can remember traces which caused difficulties but which were not of known type so that these can be avoided in future problem solving. It also has the ability to generalize and remember successful programs to be used as building blocks in future problem solving.

It should be noted that protection schemes are straightforward to implement using a backup goal control tree and such a scheme has been incorporated in the TICKLISTS used in INTERPLAN. The goal control tree of HACKER is of the backup type.

3 THE KEYS AND BOXES PROBLEM

The Keys and Boxes problem was devised by Michie (1974) as a benchmark test for robot problem solvers. A robot, without any capability of gathering further information than it is given at the start of problem solving, must operate in the world shown below.



The problem is defined informally below: words in capitals are special to this problem in the sense that the problem statement is meant to define them. This problem formulation differs from that given by Michie. In particular, sets of objects are used to describe the problem. The changes were made in the light of several people's attempts to solve the problem themselves (4 protocols of this sort were used to gain some insight into the methods humans may use on the problem).

3.1 Statement of the Keys and Boxes problem

The world consists of: the PLACEs named BOX1, BOX2, DOOR, TABLE and OUTSIDE; the OBJECTs, examples of which are named A, B and C; and an agent named ROBOT. OBJECTs may have properties named RED and KEY. PLACEs may have the property named INROOM. There are relations named AT, HELD and ROBOTAT. There is a (possibly empty) set of OBJECTs AT any PLACE. A set of OBJECTs (possibly empty) is HELD. NOTHING is equivalent to the empty set of OBJECTs. If a set of OBJECTs has some property, then any individual or non-empty subset of the OBJECTs has the property. The property of OBJECTs being RED or KEYs cannot be changed. The property of PLACEs being INROOM cannot be changed. The ROBOT can cause some changes by executing actions named LETGO, PICKUP and GOTO.

The LETGO action causes the parameter of HELD to be changed to NOTHING. There are no other effects of a LETGO action.

If there is a non-empty set of OBJECTs AT some PLACE and the ROBOT(is)AT the PLACE, then the PICKUP action causes the set of OBJECTs HELD to be changed to a non-empty subset of the set of OBJECTs AT the PLACE. There are no other effects of a PICKUP action.

The GOTO action takes a parameter which is a PLACE. The GOTO action primarily causes the PLACE the ROBOT(is)AT to be changed to the PLACE which is the parameter of the GOTO action. If the set of OBJECTs HELD is not empty, then the GOTO action also causes the PLACE the set of HELD OBJECTs is AT to be changed to the PLACE which is the parameter of the GOTO action. If the parameter of the GOTO action is OUTSIDE, then the GOTO action can only be applied if there is an OBJECT (and possibly others) AT the DOOR which has the property of being a KEY. Otherwise the parameter of the GOTO action should have the property of being INROOM. There are no other effects of a GOTO action.

In the initial situation there is A and possibly other OBJECTs AT BOX1.

In the initial situation there is B and possibly other OBJECTs AT BOX2.

In the initial situation there is C and possibly other OBJECTs AT the DOOR.

In the initial situation there is NOTHING AT the TABLE.

In the initial situation the PLACE the ROBOT(is)AT is unknown.

In the initial situation, either all OBJECTs AT BOX1 have the property of being KEYs or all OBJECTs AT BOX2 have the property of being KEYs.

In the initial situation all OBJECTs AT the DOOR have the property of being RED.

The PLACEs BOX1, BOX2, DOOR and TABLE all have the property of being INROOM.

The goal of the problem is to produce an action sequence (plan) which will convert the initial situation into a situation in which a subset of the OBJECTs AT the OUTSIDE have the property of being RED.

Thus an action sequence such as:-

LETGO, GOTO(DOOR), PICKUP, GOTO(TABLE),
LETGO, GOTO(BOX1), PICKUP, GOTO(DOOR),
LETGO, GOTO(BOX2), PICKUP, GOTO(DOOR),
LETGO, GOTO(TABLE), PICKUP, GOTO(OUTSIDE) will achieve the goal.

3.2 What are the difficulties?

3.2.1 There are actions with imprecisely defined outcomes.

The PICKUP action causes a SUBSET of the objects at the place the robot is at to be held. Therefore, unless we are sure there is only one object at any place, we cannot pick up particular objects. This indicates, what seems to me to be, the principal difficulty of the Keys and Boxes problem: that placing objects at any place may ruin our ability to later PICKUP objects with known properties. Thus, although we know in the initial situation that all the objects at the door are red, and therefore a PICKUP at the door will result in only red things being held, we cannot guarantee this in a situation resulting from putting keys at the door. The uncertainty of the PICKUP action gives rise to a particular case of a more general problem which I will term the INTERACTION PROBLEM. The robot is living in a "coupled world" where there can be complex interactions between the effects of some actions and the subsequent applicability of others. I will be mainly concerned with such interaction problems throughout this report (they are described in a more general way in section 4).

3.2.2 We do not know precisely which object is a key

A request to find a key will only produce the answer that either any subset of the objects at box1 or any subset of the objects at box2 has the property of being keys.

3.2.3 Keeping track of the objects at each place

The Keys and Boxes problem requires information to be stored about what objects are at certain places. We need to remember whether no objects, some particular objects, a selection of some particular objects, or an indefinite number of objects are at a place. The formulation of the problem (in section 3.1) in terms of sets of objects is intended to clarify what is required. Simple data base methods of storing a fact such as "objects OB1, OB2 and possibly others are at place BOX1" as (AT OB1 BOX1) & (AT OB2 BOX1) cannot reflect what is required if an unknown selection of these is removed (by a PICKUP).

In the next section the interaction problem mentioned above will be studied more generally. We will return to the Keys and Boxes problem in section 11 after describing INTERPLAN, a system which we have designed to deal with interaction problems.

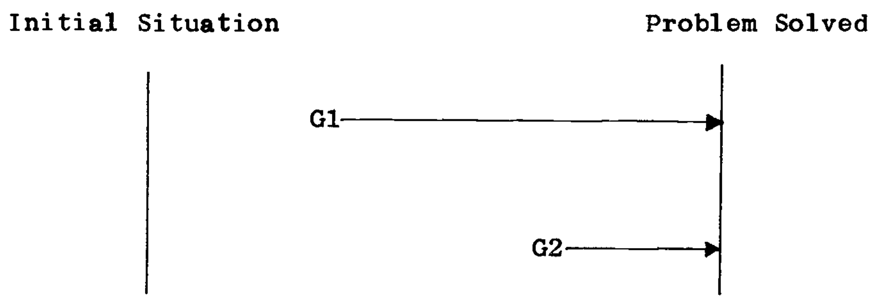
4 INTERACTING GOALS AND THEIR USE

4.1 Interacting goals

A problem is given to a means-end analysis based problem solver, such as STRIPS (Fikes and Nilsson, 1971) and the planning part of the HACKER system (Sussman, 1973), as a conjunction of goals, e.g.,

(G1 & G2)

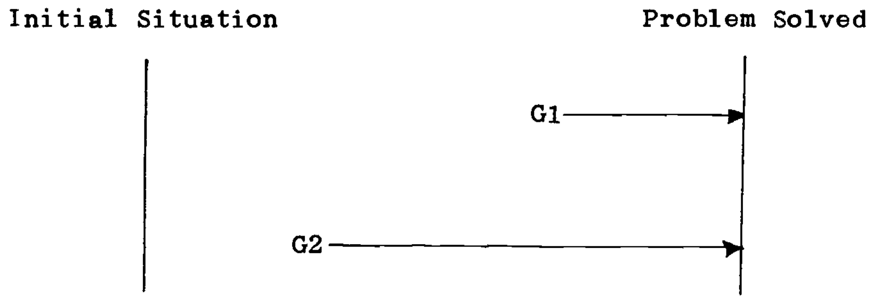
which must be true for the problem to be solved. Since the individual goals are solved sequentially, they must, once achieved, hold together for a period of time. The time for which an achieved goal must remain true will be called the goal's "holding period". I will illustrate this as follows.



Approach: G1; G2

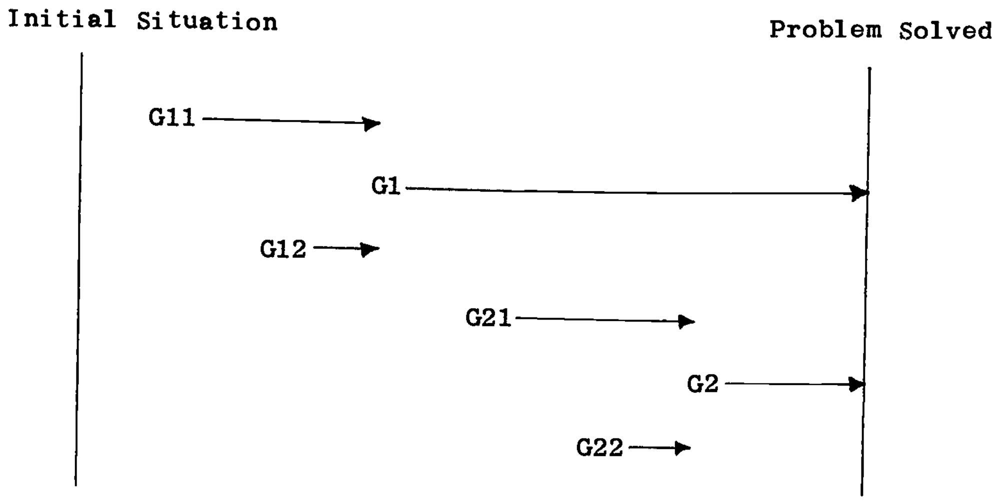
The horizontal dimension of this "holding period" diagram represents time during which actions will be applied in a final plan to achieve the given goals. APPROACH should be interpreted as: if G1 not true achieve it using some operator sequence, then do likewise for G2.

STRIPS assumes, in the absence of other information, that it can achieve the individual goals by relevant plan sequences, say, in the order in which the goals are given (Sussman calls this a linear assumption). Thus, as shown in the previous diagram, STRIPS would assume that G1 can be solved by some relevant plan sequence and then that G2 can be solved by a plan sequence following on from the first. If STRIPS can find no way to achieve the goals in the order given, it is capable of reversing the order it has attempted to achieve goals, which were initially not true, at the failure level (e.g., at the top level G1 and G2 could be reversed to give an expected holding period diagram as shown below).



Approach: G2; G1

STRIPS further assumes that for the goals not already true at the time required, the preconditions, which are required to be true for some operator to be applied to achieve the goal, can all be made true immediately before the time the goal is required to be true. Again, reversals amongst these preconditions can be made on failure backup. Thus, if the preconditions for some operator to achieve Gi are Gi1 and Gi2, then STRIPS initially assumes an approach as in the diagram below can be taken.



Approach: G11; G12; G1; G21; G22; G2

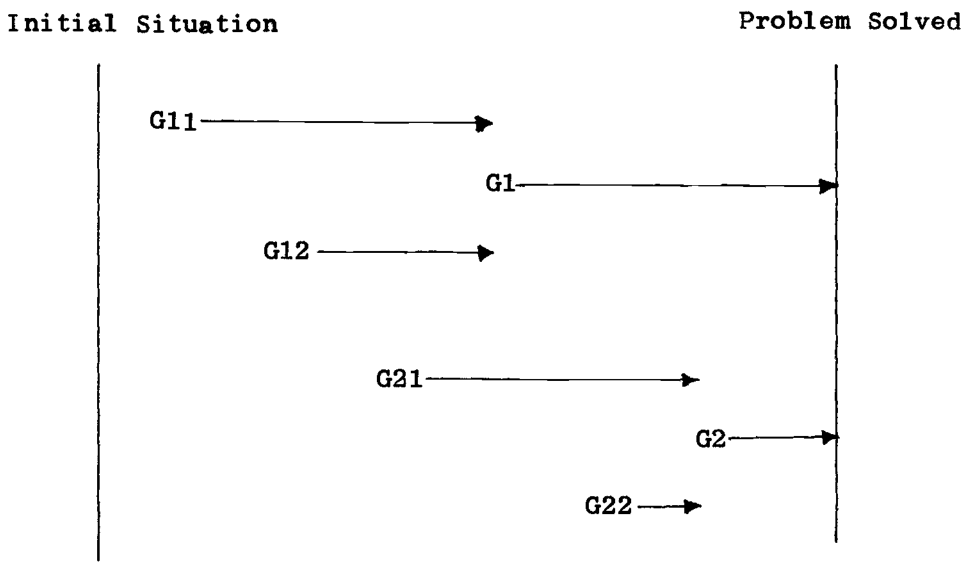
Note that the holding period diagram represents the goals to be worked upon for SOME chosen operator sequence. There is really a third dimension to the diagram representing different operator choices.

Reversals allow certain other orderings of these goals to be attempted. However, limiting reversals to goals at a particular level of the search tree hierarchy means that STRIPS (these arguments also apply to HACKER) can only tackle certain problems. Specifically, those in which interactions between top level goals can be avoided by suitable ordering of the goals and the choice of suitable operator sequences.

Since STRIPS and HACKER also allow attempts to achieve goals to be repeated if interactions have occurred, they can also handle those problems in which the interactions leave the world in some situation from which the interacted goals can be re-achieved. STRIPS will often produce longer than necessary solutions if it repeats attempts to achieve goals.

Even for very simple worlds, such as the blocks world used by

Sussman, interaction can occur. To be able to deal with all types of interaction between a set of goals, we could consider the search space as containing approaches with every interleaving of the goals and subgoals needed to achieve those goals. Thus, a holding period diagram and approach as shown below is necessary to resolve some types of interaction.



Approach: G11; G12; G21; G1; G22; G2

4.2 The 3 block problem

The 3 block problem is an example used by Sussman (1973) in his description of HACKER. It is regarded by HACKER as an ANOMALOUS SITUATION. The problem is useful as it highlights the interaction difficulty in a simple task.

A world is described by two predicates ON(x,y) and CL(x).

ON(x,y) asserts block x is on top of the (same size) block y.
Note that ON is NOT transitive, and only one block can be ON another

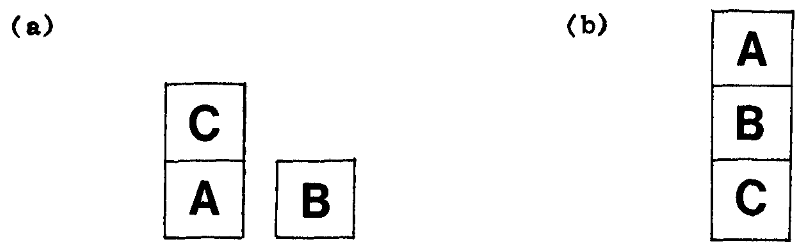
CL(x) asserts block x has a clear top.

There are two operators:-

PUTON(x,y) asserts ON(x,y) and deletes CL(y).
If $\exists$u . ON(x,u) before the application of the operator then assert CL(u) and delete ON(x,u).
It can be applied if CL(x) and CL(y) are true.

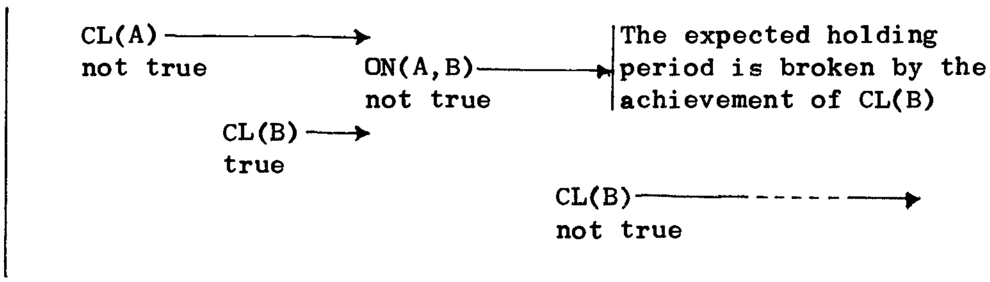
ACTCL(x) asserts CL(x).
If $\exists$u . ON(u,x) before the application of the operator then assert CL(u) and delete ON(u,x)
REPEAT if $\exists$v . ON(v,u) etc. (This operator therefore clears all blocks from the top of block x and puts them somewhere in free space). It can always be applied.

Given an initial situation ON(C,A) & CL(C) & CL(B) as shown in (a) below a goal of ON(A,B) & ON(B,C) is given as shown in (b) below.



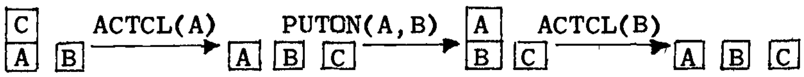
STRIPS can tackle (ON(A,B)&ON(B,C)) both parts of which are not true initially. The goals may, at first, be attempted as shown in the following holding period diagram.

Initial Situation

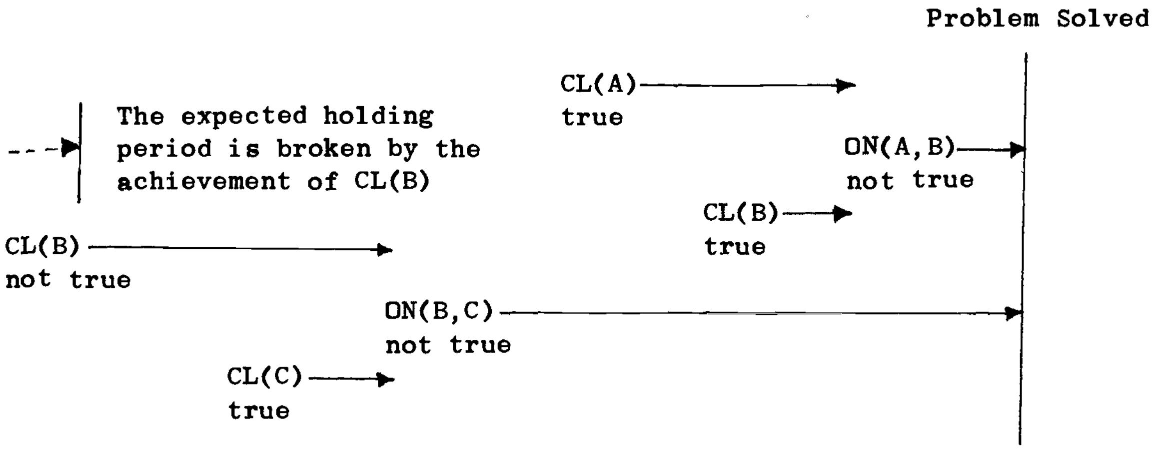


Approach: CL(A); CL(B); ON(A,B); CL(B);

Plan Sequence:

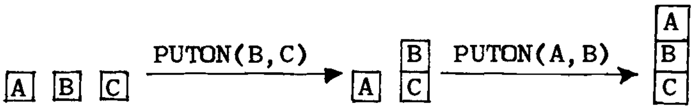


The earlier achieved goal (ON(A,B)) does not now hold (its expected holding period is broken), but this is not noticed by STRIPS, and problem solving proceeds as shown below.



Approach
Continued... CL(C); ON(B,C); CL(A); CL(B); ON(A,B)

Plan sequence
Continued...



So, STRIPS produces the longer than necessary solution:-

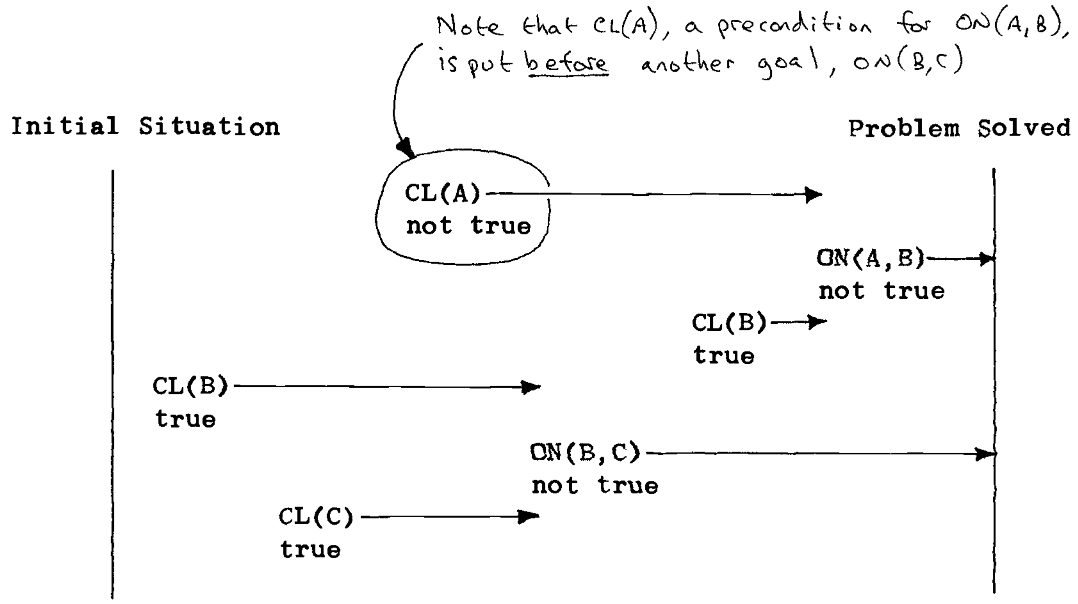
ACTCL(A), PUTON(A,B), ACTCL(B), PUTON(B,C), PUTON(A,B).

Attempting the initial goals in the opposite order would make the final solution found longer still, though if the interactions in the first ordering produced a world situation in which the interacted goals could subsequently not be achieved, this would be attempted on failure backup. STRIPS is incapable of producing a shorter plan for this problem.

HACKER has a mechanism, called protection, which remembers achieved goals and looks out for actions which violate them It would notice that the previously achieved goal (ON(A,B)) ceased to hold (as a protection violation) and would try to reverse the order of the top level goals (to ON(B,C)&ON(A,B)) at that time. However, another protection violation with the reversed approach will direct the HACKER planner to allow the protection to be violated, and the result will be the same as for STRIPS in this example.

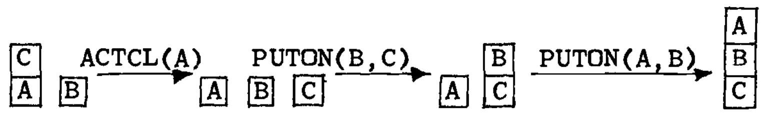
The search space should have included an approach as shown below. This approach is an ordering not allowed by reversals only within the hierarchic levels of the search tree. It would have led to a solution plan:-

ACTCL(A), PUTON(B,C), PUTON(A,B).



Approach: CL(B); CL(C); CL(A); ON(B,C); CL(B); ON(A,B)

Plan Sequence:



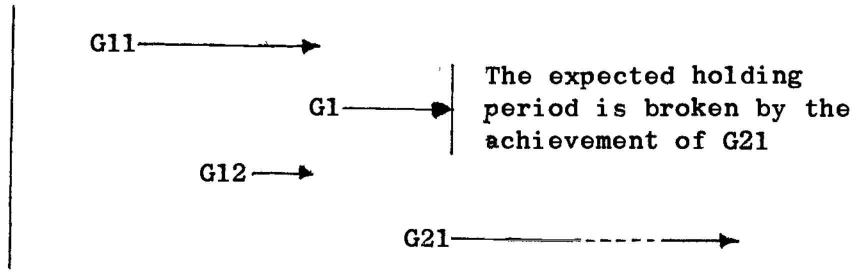
STRIPS, by re-achieving the ON(A,B) goal, can solve this problem with a longer than necessary plan because the world situation produced after interaction is such that the goals can still be achieved. The Keys and Boxes problem has interactions which would preclude a STRIPS-like problem solver from finding any solution.

4.3 Using goal interactions to suggest new approaches to a problem

Current means-end analysis problem solvers are not capable of solving problems which have certain kinds of goal interaction. Also, with the exception of some systems at MIT (e.g., HACKER), they do not use interactions amongst goals to guide the search for a solution. I mentioned earlier that all interleavings of goals should have the potential of being considered. Generally, only very few of the possible interleavings need be considered. An assumption, such as is made by many existing problem solvers, that goals can be achieved in the order given without interaction (linearily) is a very powerful heuristic. My own work in problem solving is based upon the powerful heuristics used in STRIPS and other problem solvers, but I am anxious not to let these assumptions rule the types of problems which can be dealt with. Proven contradictions of these assumptions during problem solving can direct the search to consider appropriate interleavings of plan parts to remove interactions.

The information gained from the discovery of an interaction can be used to suggest appropriate continuations. As an example, the interactions during attempts to solve the goals G1 & G2 linearily can lead us to the point in the diagram below, where the expected holding period for G1 is broken by the achievement of a subgoal G21 required for an action to achieve G2.

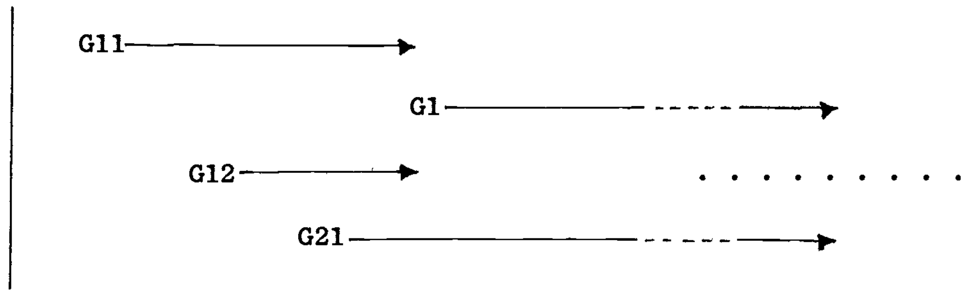
Initial Situation



Approach: G11; G12; G1; G21;

We have tried and found that G1 and G21 cannot both hold together when they have been achieved by some operator sequences in the order (G1, G21). We can either try an approach in which the goals at the higher (here the top) level are reversed to stop the conflicting goals holding periods overlapping altogether (by reversing G1 and G2) or try to achieve the conflicting goals in the opposite order. It is sufficient to try to achieve the conflicting goals in the other order only once. This can be done whilst still preserving linearity as far as possible by moving the precondition (G21) whose achievement made a previously achieved goal (G1) not hold, immediately in front of the goal as shown in the following diagram. We shall say that we PROMOTE the precondition.

Initial Situation



Approach: G11; G12; G21; G1; . . . .

Moving it further back through the goals to be worked on would, of course, still enable the conflicting goals to be achieved in the reverse order but would, however, risk the possibility that other intermediate goals would conflict with the precondition being promoted. Following Sussman (1973) we will sometimes refer to the promoted goal as a "setup" goal. Note that the promoted precondition (G21) may interact with earlier goals and may need to be shifted again due to different interactions. Subgoals intermediate between G2 and G21 if they exist may need to be promoted also.

The details of the way in which information from such a goal interaction is extracted and used to suggest new approaches to a problem will be discussed in the next section, as will other goal interactions from which information can be extracted to guide the search for a solution.

5 INTERPLAN: THE PLAN GENERATOR

In this section we will describe the problem solver, INTERPLAN.

5.1 Aims and assumptions

The plan generator is basically a STRIPS-like means-end analysis driven (or subgoaling) problem solver with the additional capability of dealing with interactions between goals. Problems are given to it by specifying an initial world situation, a goal situation, and a set of operators (or actions) which can be used to transform situations. INTERPLAN is required to find a linear, fully ordered sequence of operator applications which will transform the initial situation into a goal situation. It has been designed to produce a single solution to the problem (if one exists). It takes a suggested "approach" (usually the given order of a conjunct of individual goals) and tries to produce an operator sequence which is a concatenation of the operator sequences to solve the individual goals in the order specified in the approach. Checks are incorporated to ensure that each operator sequence does not delete the goal achieved by some earlier part. If a difficulty is encountered while pursuing the given approach, alternative approaches based upon information gathered from the nature of the difficulty itself, are suggested by INTERPLAN. INTERPLAN tries to solve the problem by showing that one such approach is valid. If the initial approach is valid, INTERPLAN will merely try to find and check appropriate operator sequences which will satisfy the individual goals, no new approaches being suggested.

During problem solving INTERPLAN makes the following assumptions:

(a) a conjunction of individual goals can be solved by tackling the goals in some order individually.

(b) a goal once solved must remain true until the other goals in the conjunct are solved.

(c) in the absence of other ordering information, the given order of goals is a reasonable first order to try. INTERPLAN is, however, capable of trying other orderings in those cases where it is proven to be of possible use to do so (e.g., on Protection Violation discoveries).

(d) to achieve a given goal, only those operators which ADD the goal directly are relevant. That is, only those operators in which the goal appears on the operator's ADD list.

(e) A goal containing variables is considered solved if it has any true instance in the required situation. No attempt is made to achieve other non-true instances in this case. This is an important restriction on the search space. However, section 5.7.3 mentions how this assumption may be relaxed if needed.

(f) Normally, the preconditions for some operator which will achieve a goal can be made true immediately before the goal they are for is to be made true. INTERPLAN is, however, capable of relaxing this assumption in those cases where it is proved to be of possible use to do so (e.g., on Protection Violation discoveries). Then, "setup" goals can be inserted into the approach.

(g) changes to the world only occur through applications of the operators given to the system.

The system separates the search across the space of world situations (regarded as a graph whose nodes are situations and whose arcs are operator applications) from the question answering about a particular situation. INTERPLAN is an operational program written in POP-2 (Burstall, Collins and Popplestone, 1971). The HBASE (Barrow, 1975) data base system is used to store situations (as CONTEXTS) and the facts known about each particular situation (as assertions). There are special INTERPLAN data structures and processes (to be described later in this chapter) which control the search across the space of world situations.

Program identifiers and syntax will be introduced and used along with the description below since this chapter is also intended to serve as documentation of the INTERPLAN program.

5.2 Specification of a problem

The plan generation system is given a task by specifying:

(a) An initial situation specified by a set of assertions.

E.g., for the 3 block problem initial situation

```
ASSERT <<ON C A>>
       <<CL C>>
       <<CL B>> ;
```

The brackets << ... >> indicate an HBASE pattern (stored as a POP-2 strip). Patterns may be nested. ASSERT takes a list of patterns and indicates that they are true in the current HBASE context (CUCTXT) which is taken to be the initial situation by INTERPLAN.

(b) Descriptions of the actions which can transform situations. These are basically specified similarily to STRIPS operator schemas (whose instances are operators) with a list of facts to be DELETED from a situation and a list of facts to be ADDED to a situation to alter it. Also specified (as PRECONDITIONS) are those facts which must hold in a situation for the operator to be applicable.

The ADD list of an operator schema is used to determine whether it is relevant to achieving some goal (i.e., whether it ADDS a statement required by the goal). However, an operator schema may make changes to a situation other than those specified in the ADD/DELETE lists since the system allows any function (the OPSCHFN) to be applied when an operator is used to transform a situation (this can be thought of as providing CONNIVER-like IFADD and IFREM method facilities - McDermott and Sussman, 1972). So, effects difficult to express assertionally or requiring testing of the situation itself can be made. However, these effects cannot be used to determine whether the operator schema is relevant.

An operator is applied to a situation by

i) notionally making a copy of all facts true in the HBASE context representing the old situation,

ii) deleting all patterns from this which match each DELETE list entries,

iii) adding all ADD list entries, and then

iv) running the operator's OPSCHFN.

An operator schema has further components mainly used by the system itself, but some allow heuristic knowledge of a particular domain to be incorporated. These will be mentioned in appropriate places throughout the text, and are given in full in appendix I.1.

A macro, OPSCHEMA, is available to construct simple operator schemas. Assignments can then be made to the empty components if more complex operator schemas are required, that is, with functions which cause side-effects, or with heuristic knowledge.

Thus for block stacking:-

```
OPSCHEMA <<ACTCL *$*X>>   *$*X is a variable local to this OPSCHEMA
   ADD    <<CL *$*X>>
   DELETE                       no deletions
   PRECONDS                     no preconditions
   VARS  X                      all local variables must be named
ENDSCHEMA -> S1;                save OPSCHEMA in POP-2 variable S1.

OPSCHEMA      <<PUTON *$*X *$*Y>>
   ADD        <<ON *$*X *$*Y>>
   DELETE     <<CL *$*Y>>
   PRECONDS   <<CL *$*X>>  <<CL *$*Y>>
   VARS       X  Y
ENDSCHEMA -> S2;
```

There are further effects of these operator schemas as specified in section 4.2. These effects are difficult to express merely in ADD and DELETE lists (see Fikes, Hart and Nilsson, 1972a). They can be written as functions in POP-2 which use HBASE primitives to search, add to and delete from the current context (CUCTXT). See section 6 for a listing of these functions.

Calling the functions CLFN and ONFN then

```
CLFN -> OPSCHFN(S1);
ONFN -> OPSCHFN(S2);
```

(c) The present system also requires the user to state which operators can be used to achieve patterns. This information is kept as an association list of patterns and a list of relevant operator schemas in a global program identifier, ACHIEVES.

For example, in block stacking:

```
[%  <<CL == >> ,      [% S1 %] ,
       <<ON == == >> ,  [% S2 %] %] -> ACHIEVES;
```

That is, the user should take each item in the ADD list of each operator schema, replace all variables by == (a pattern which matches "anything" in HBASE), and group the corresponding schema with any others which can ADD the same pattern. This list could be generated automatically.

All ADD list entries for all operator schemas need not be put on the ACHIEVES list. The "primary additions" of STRIPS can then be modelled (see Fikes, Hart and Nilsson, 1972b). For instance, a <<PUSHBOX BOX PLACE>> operator may add two facts <<AT BOX PLACE>> and <<AT ROBOT PLACE>>. We may only want to consider using PUSHBOX to achieve <<AT BOX PLACE>> goals and never merely to move the ROBOT. We could then omit the PUSHBOX operator from the ACHIEVES List associated with <<AT ROBOT == >> facts.

(d) A specification of a goal situation by giving the statements which are all required to be true in a goal situation.

For example for the 3 block problem:

GOAL <<ON A B>> <<ON B C>>;

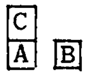
Variables are allowed in goal specifications.

5.3 Ticklists

The basic data structure used by the system is a TICKLIST. See appendix I.2 for its components. It forms the nodes of the goal control tree which INTERPLAN constructs. Basically, a ticklist is a 2-dimensional array which has a column for each of a set of goals which are all required to be true together in some situation. The root node of the goal control tree for the goal of the 3 block problem would consist of a ticklist with two columns headed <<ON A B>> and <<ON B C>>. I will refer to the set of goals represented by the columns of a ticklist as the TICKLIST HEADING. Rows of the array represent situations in which it is hoped that all the goals will be true. We thus start problem solving with a ticklist whose heading consists of the individual statements specifying the goal situation and whose single row represents the initial situation. This is shown below for the 3 block problem.

	<<ON A B>>	<<ON B C>>
Initial Situation $\boxed{C}$ / $\boxed{A}$ $\boxed{B}$		

To fill in a ticklist, we scan the last row (in the example above there is only one row initially) from left to right and for each column ask if the goal heading is true in the situation of the last row. We put a tick (√) if it is, or a cross (X) if it isn't, stopping whenever a cross is entered. If the whole conjunct of goals is true in the situation we get a complete row of ticks and have thus found a goal situation. However, if a column has a cross then this goal has to be achieved in some situation. This occurs initially in the 3 block problem where it is found that the first column has a cross entry (see diagram below).

	<<ON A B>>	<<ON B C>>
Initial Situation 	X	

5.4 INTERPLAN's search space

The space which can be potentially searched by INTERPLAN consists of all those approaches which can be obtained by using means-end analysis on all given goals and the preconditions of actions to achieve those goals (and so on for actions to achieve those preconditions, etc.) in any order, so long as the preconditions for an action are achieved before its application. For example, given two goals G1 and G2, there is an action A1 relevant to achieving G1 and an action A2 relevant to achieving G2. A1 has precondition G11 and A2 precondition G21. Both preconditions can be achieved by actions which have no preconditions. The potential search space contains the approaches obtained by trying to achieve the goals in any of the following orders.

G11	G1	G21	G2
G21	G2	G11	G1
G11	G21	G1	G2
G11	G21	G2	G1
G21	G11	G1	G2
G21	G11	G2	G1

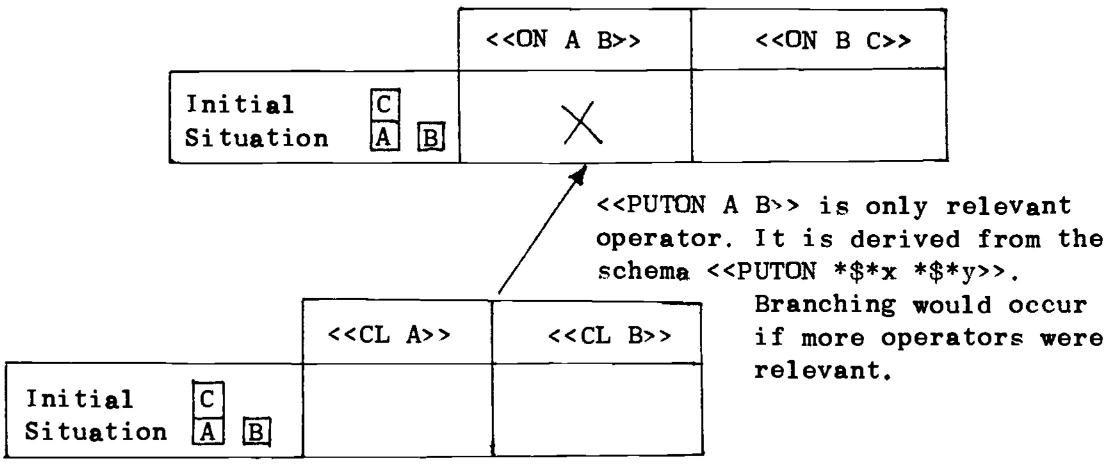
A problem solver which makes and adheres to the linear assumption would only have to consider the first two of the above six approaches (with a corresponding decrease in the range of problems which could be tackled). Simple schemes for considering alternative approaches when a failure occurs, such as backtracking, can thus be used with such systems. However, it would be very inefficient to represent the extended search space to some problem solver and expect the system to select a valid approach from this space using a simple backtrack algorithm if failures occurred.

Since there may be no way to achieve some goals and because the achievement of some goals may not in any way effect the achievement of others (no interactions), several of the above approaches could be equivalent. An initial approach is suggested to INTERPLAN by giving an ordering on the top level goals, say G1 and then G2. Since the preconditions are considered in the order in which they are found in the PRECONDS list of each relevant OPSCHEMA, the ordering on top level goals will specify a reduced number of the possible approaches. The actual reduction will depend on whether there is one or more relevant operators for each top level goal. Often, many of the approaches in the potential search space are initially locked away from consideration by INTERPLAN.

If this initial approach is successful, no further approaches are made available to INTERPLAN. However if some interaction in the initial approach occurs, this may indicate other orderings of the goals (other approaches) which may remove the interaction. Such specific approaches are then indicated as open for consideration (it depends upon the particular OR-CHOICE mechanism being used when, and if, they are actually considered). The information gleaned from an interaction thus provides "keys" to unlock specific branches along the potential search space. Tightly restricting the possible approaches in this way, and only allowing other approaches to be tried if they are indicated as being probably useful in the light of the interactions discovered, can significantly reduce the part of the potential search space actually considered in many problems.

5.5 Ticklist levels - the goal control tree

When a goal has to be achieved, for each relevant operator (i.e., instance of an operator schema) a subgoal is set up of trying to find a situation in which all the preconditions for the operator hold. A goal control tree of the BACKUP type (described in section 2.5.4) is grown by making new ticklists on a LEVEL lower to that containing the goal to be achieved. These have as column headings the preconditions of each operator, and thus represent subproblems of the higher LEVEL. They are connected to the upper level ticklist by arcs representing the particular instantiation of each relevant operator schema. For example, to continue the block stacking example:-



All ticklists at the tips of the goal control tree being constructed are suitable for further filling in, etc. Therefore, they are held in a list of choices which can be heuristically ordered. See appendix III for details of the scheme used to deal with choice points in the current implementation of INTERPLAN. The choice list is a

list of pairs, each of which consists of a heuristic value and a pointer to the ticklist on the tip of
the goal control tree (though 2 special entries are allowed on the choices lists - see sections 5.7.1 and 5.7.3). The choice list is ordered so that pairs with a lower heuristic value are nearer the head of the list and are considered "better" choices.

ADDCHOICE ε <heuristic value>, <pointer to ticklist> => ( );
splices a pair into the appropriate place in the list of choices.

MAKECHOICE removes the first (lowest value) pair from the choice list and makes the ticklist from the pair, the one for consideration next by INTERPLAN (by assigning the ticklist to GLOBTICK). It deals with the special forms allowed in the choice lists.

5.6 Protection

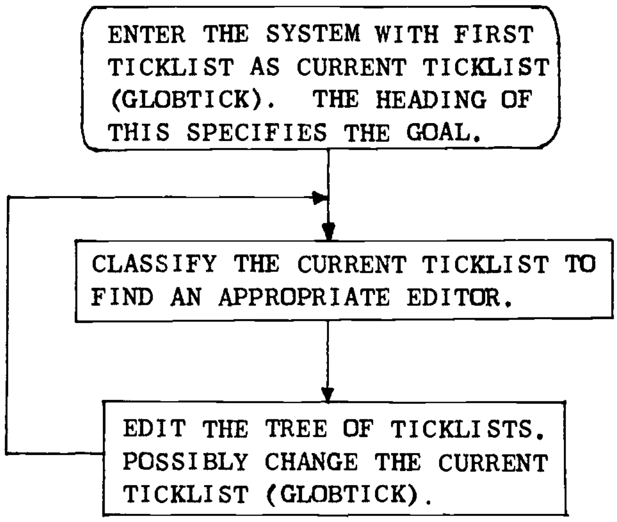
When a goal has to be achieved after other goals have already been achieved, there is a mechanism for ensuring that the previously achieved goals are not deleted. We PROTECT the previously achieved goals by adding them to the ticklist heading of all LEVELS of the goal control tree which are grown below the LEVEL where the goals were achieved. This is represented diagrammatically below. Global goals (whose truth value is not changeable - see appendix I) are not protected in this way.

	G1	G2
C1	√	X

	G1	G21	G22
C1	√		

In some situation, the protected goals must be true simultaneously with all the other goals in the ticklist heading (preconditions for some operator) for that situation to be one in which the operator is applicable (in the context of the previously achieved goals). It should become clear later how information in the protected columns of a ticklist is used by the system. For the moment, however, it will be useful to know that a system using the protection facility will look for any VIOLATION of the protection on a fact (PROTECTION VIOLATION). This is an implementation of a feature in the HACKER planning system (Sussman, 1973).

5.7 Classifiers and Editors



The basic loop of the planning system is shown above. Many different problem solvers could be written within this framework. A system is specified as pairs of classifiers for a ticklist and an editor for the tree of ticklists. See appendix I for information available within a ticklist and the tree of ticklists for use by the classifiers and editors. The following sections describe the classifiers and editors used to specify INTERPLAN.

As will be seen later, the classifiers are defined to look at the patterns of ticks and crosses in a ticklist. These patterns provide a simple language in which difficulties during problem solving can be quickly identified (cf. the analysis of the teleological trace of the problem solver's actions necessary to find bug types in HACKER - Sussman, 1973).

5.7.1

Classifier: No entries have been made in the last row of a ticklist or a tick appears in the last column of the last row of the ticklist and some other column on the row has no entry. (I.e, there remains a goal for which we have not queried the data base to see if it is true).

Editor: (FILLIN)

Scan from left to right along the last row and for any position not filled in, ask the question answerer whether the pattern heading the position is true in the situation of the last row. See appendix II for details of the Question answerer (QA). A call to QA may instantiate some variables local to the ticklist. If QA finds that a pattern has more than one true instance in the given situation the system asks the user if he would like to pre-order the instances (given in a list POSSLIST). It then hands back the first choice to FILLIN (which is thus used to set variables), but adds a special node to the choices list to be used to initialize the other choices. This special node is a STRIP of three items - see appendix II.

Filling in continues either until all the row is filled in in which case we can SUCCBACKUP, or until a cross entry is is made, in which case we must ACHIEVELAST the appropriate goal (unless it is a global goal - see appendix I.1).

5.7.2

Classifier: (ALLTICKS)

A complete row of ticks exists in some row (or more generally, the ticklist heading is satisfied by some row representing a situation).
(I e, all goals for this ticklist are solved).

Editor: (SUCCBACKUP)

Backup successfully to next higher node (ticklist) in the goal control tree, applying the operator represented by the arc of the tree which is now applicable in the situation found. The new situation produced becomes a new row in the higher ticklist and in this row a tick is entered in the column of the goal the operator achieved. The operator used to produce the new situation is remembered by assigning its name to the VALUE of the item "SITN" in the new situation (see HBASE - Barrow, 1975). An example of the use of this editor is shown below.

	P1	P2	P3
C1	×		
C2	✓	×	

after editing → gives

	P1	P2	P3
C1	×		
C2	✓	×	
C4		✓	

↑ OPx

	P1	Px1	Px2
C2	✓	×	
C3	✓	✓	✓

OPx applied to C3 gives situation C4.

5.7.3

Classifier: A cross appears for some column in the last row of a ticklist (but excluding those cases in which there are ticks further right in the row too – see section 5.7 4 for this case). (I e, a goal remains to be achieved).

Editor: (ACHIEVELAST)

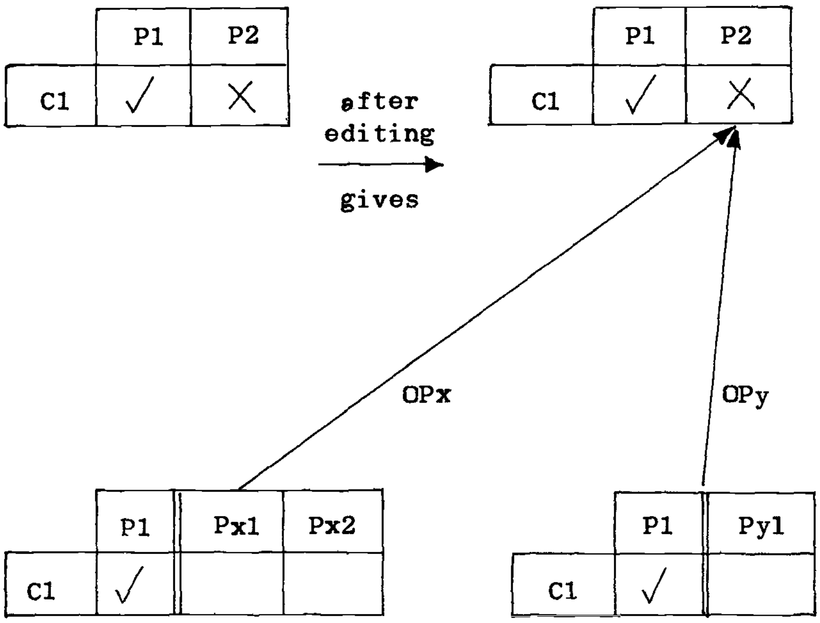
Operators which could add the pattern represented by the column with a cross to the world model in some situation are sought for. This is the recursive use of the means-end analysis technique. Before operators are found, a check is made to see if the achieve request would cause a LOOP. This is done by checking whether the achieve request already exists on the CURRACHIEVES list (see appendix I.2) and if so, whether the situation the present request is for is the same as the one for the previous request. If so, a LOOP is reported and the LOOP editor called (see section 5.7.7).

The editor finds all RELEVANT operators (i.e., those which can ADD the sought-for pattern). A function

OPSCHMODIFY ε <opschema>, <search pattern> => <opschema>,

is applied for each relevant operator when found. This normally returns the <opschema> unchanged, but can be used to change the order of preconditions etc.

The editor adds new choice points to the goal control tree corresponding to new successor nodes to the original ticklist for each relevant operator. The successor nodes are initialized when chosen from the choices list, where they are kept in a compact form, but notionally they exist after this editor has been applied. See section 5.6 on Protection for explanation of the symbols used in the example of the operation of this editor below (especially why the P1 protected goal is brought down through levels of the goal control tree).



If OPx and OPy are the only relevant operators.

Achieving goals which already have true instances

Normally, if INTERPLAN discovers some goal which is needed, already is true at the time required, it makes no attempt to APPLY operators to ACHIEVE the goal. If the goal is fully instantiated (e.g., CL(B)) this is alright as it can only have one possible instance and this is known to be true. If the goal was CL(x) and CL(B) was true, the goal would hold if the variable x was set to "B". However, another instance (e.g., CL(C)) may be required to reach a solution.

A switch (turned on by assigning "true" to the variable COMPLETE) has been provided in INTERPLAN so that goals which are not fully instantiated and which in some instances are true can be recognized and special extra choice points added to allow the non-true instances to be ACHIEVED if the already true instances prove not to be of use.

5.7.4

Classifier: A cross on some row (NOT a protected entry) is followed by a tick in a later column. That is, the achievement of a goal has made false a goal which was true previously.

Editor: (ALTERLASTORDER)

An attempt is made to shuffle the pattern of the column which was ticked, before the pattern of the column with the cross. Checks are first made to ensure that the columns to be swopped have not been swopped previously or are now not allowed to be swopped (looking at the TREVS of the ticklist for the reference numbers of the patterns - see appendix I.2), or to see if no more reversals are allowed for this ticklist (TREVS is "NOREVERSE"). An example of the use of this editor is given below when the swap is allowed. The order of goals already achieved by some operator sequence is preserved by a shuffle, as this takes into account any interactions which occurred between these earlier goals.

	P1	P2	P3
C1	✗		
C2	✓	✓	✗
C3	✗		✓

after editing → gives

	P3	P1	P2
C1			

5.7.5

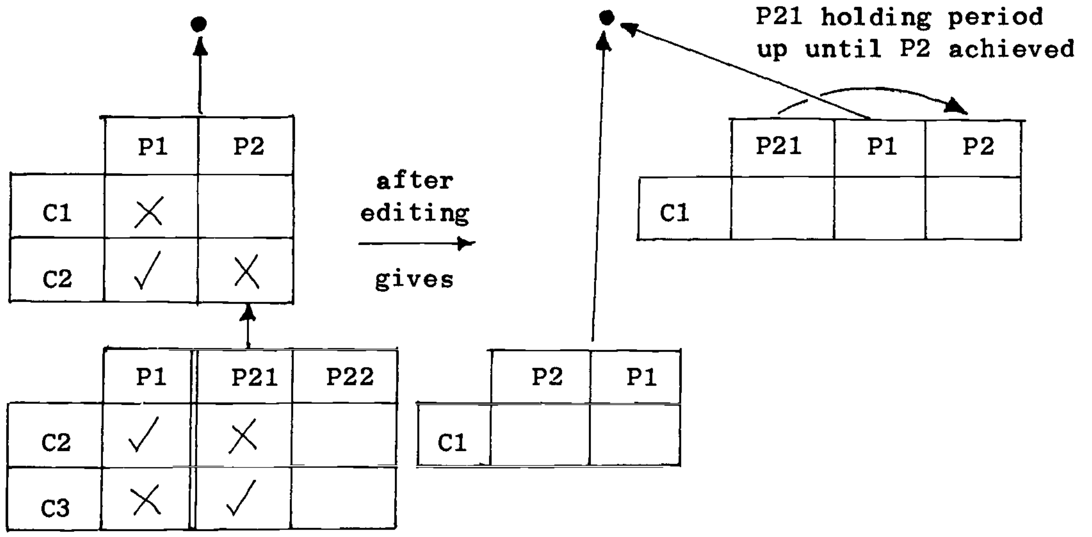
Classifier: A cross in a PROTECTED column of some row is followed by a tick in a later column. That is, a protection violation has occurred.

Editor: (PROTECTVIOLATION)

This is the editor which suggests an approach with reversed top level goals (at the level protection was placed upon the pattern which is now crossed - this is found by looking at the reference for the protected entry) or suggests an approach in which we promote the actual goal we were considering to the level at which protection was placed (see section 4.3). Before promoting a pattern, a check is made to see if the promotion would have altered the course of computation in the original case. That is, we see if the promoted pattern would already have been true at the point to which we wish to promote it. If it would have been, the promotion is attempted for the goal higher in the goal control tree for which the current goal was a subgoal. If the same applies to this we try higher still, unless the protection level itself is reached in which case no promotion is made.

If some promotion can be made, and goals higher in the goal control tree exist between the level we promoted from and the level at which protection was placed, we also try to suggest approaches in which these intermediate goals are promoted as above.

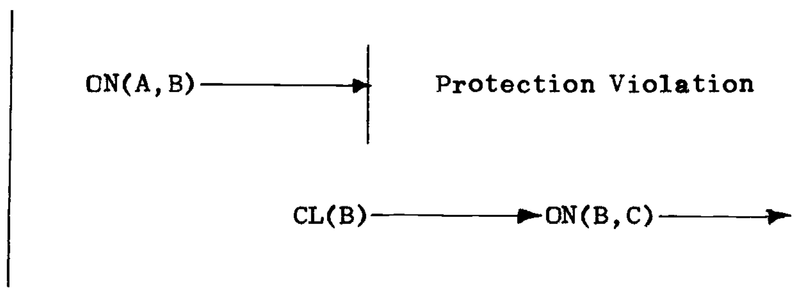
An example of the use of this editor is given below.



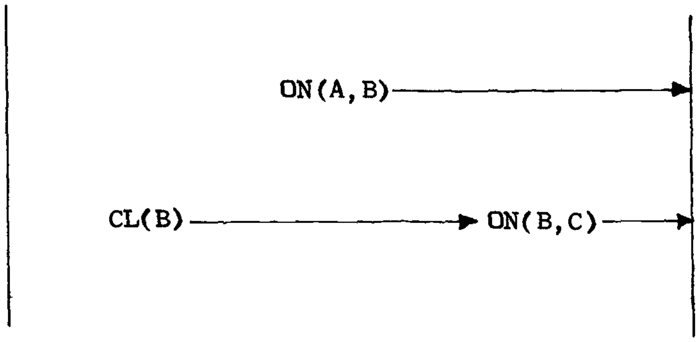
See appendix I.2 for details of how a goal with a restricted holding period is represented to INTERPLAN.

Restrictions on instances of a promoted goal

The test for rejecting promoted goals on the basis of their truth at the point required was intended to cut out those approaches which would be exactly the same as the approach before a protection violation. For example, in the 3 block problem:

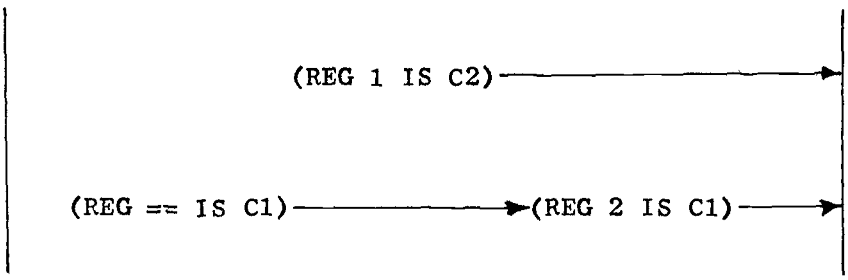


The above protection violation suggests two approaches, one of which is



However, this approach is disallowed as CL(B) is true at the point required (initial situation in the problem) and thus the approach would be exactly as in the case when the protection violation was discovered.

When the promoted goal has a variable (or variables) in it, as can often happen during promotions attempted by the LOOP editor (section 5.7.7), but is true in some particular instance, we should not reject the promoted goal outright, but should modify it to exclude the true instance (or instances). For example, in the "swap the value of two registers" example (section 8.2):



should be allowed as an approach, even though (REG 2 IS C1) is true in the initial situation. However, the promoted goal should exclude this instance to ensure that the protection violation which this approach is being suggested to avoid is not encountered again.

A scheme has been experimented with to provide variable restrictions using HBASE actors (Barrow, 1975). This scheme is outlined in appendix IV. If such actor restrictions on variables were allowed the goal to be promoted for the example above could be written:

(REG <:NON 1:> IS C1).

No promotions for an already promoted goal

All goals in a ticklist heading are given a reference number as described in Appendix I.2. When a "setup" goal is promoted it is given a reference number:

- (reference number of the goal it is a precondition of).

This simple referencing scheme disallows promotions for a goal which is itself a promoted goal. It is thus a restriction on the generality of the program.

5.7.6

Classifier: A cross appears in some column for which there is no means to achieve the relevant pattern (or no further means if some have been tried).
(I.e, no method can be found of achieving an untrue goal)

Editor: (FAILBACKUP)
Try to alter the order of the pattern which has a cross in its column with some earlier pattern in the ticklist heading (using ALTERPREV). The earlier goal's achievement may have rendered the goal on which we failed unsolvable (e.g., by wrong choice of a variable instance), in reverse order they may both be solvable. The variables of the ticklist are reset using INITVARS (see appendix I.2).

If the reversal cannot be made with any other pattern earlier in the ticklist heading (e.g., reversals already tried or this is the first pattern we are trying to achieve) then FAILBACKUP to the parent ticklist of the current one. This editor is also used when other editors have failed to do their job (e.g., cannot ALTERLASTORDER).

This backup process is mainly intended to clear the problem solvers goal control tree of useless approaches after a failure has occurred. As soon as some point is backed-up to at which there is a way to attempt to achieve the outstanding goal, backup stops and the OR-CHOICE mechanism is used to select from ANY of the outstanding choice points (which include the one just backed-up to).

5.7.7 The LOOP classifier and editor

The planning system may try to pursue an approach which causes it to loop in some way (i.e., left to itself, it may never terminate). The loop can be treated as a failure, and information extracted from the failure to suggest new problem approaches to try to avoid the loop. However, the loop must be detectable to be able to do this. At present INTERPLAN detects two types of loops.

(a) It prevents goal reversals which have already been tried from being suggested again as approaches to circumvent goal interactions (see section 5.7.4).

(b) During subgoaling, a list of all achieve requests which we are planning to satisfy (along one path through the goal control tree) are kept, together with the situation we required each one to be achieved in. This list is kept in the CURRACHIEVES of a level (see appendix I.2). If, to satisfy some lower subgoal, an achieve request is issued which is the same as some higher request and the situation both are required in is the same, a loop is reported (as mentioned in the editor in section 5.7.3).

However, for instance, the generation of similar non-linear approaches (ones with a promoted subgoal) is not detected in INTERPLAN as it is presently implemented. If a loop is not detected, as well as not providing information on which to suggest possibly useful approaches to a problem, redundancy can occur in the section of the search space looked at by the planning system (the same branch may be tried more than once). With certain OR-CHOICE mechanisms (especially those which are mainly depth-first) it would then be possible to loop without producing any solution.

The full loop editor

If a looping achieve request is detected in some situation, we have available:

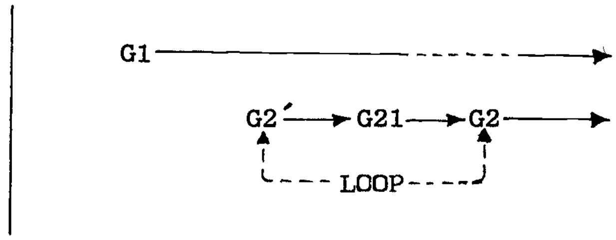
(a) the pattern causing the loop (lower occurrence)

(b) the ticklist this was required from (the lower ticklist)

(c) the pattern on CURRACHIEVES we detected loop on (the upper occurrence)

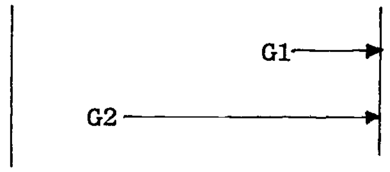
(d) the ticklist this was required from (the upper ticklist).

The editor is intended to modify the approach in the heading of the upper ticklist to try to avoid the loop. The approach being considered when a loop is detected can be typified in the holding period diagram below:

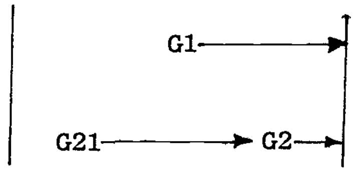


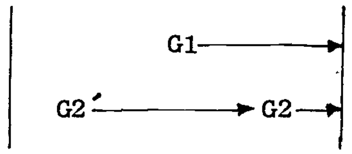
Where G2 is the looping achieve request. It should be noted that the goals may contain variables, and thus the two occurrences of the loop pattern may not be IDENTICAL, but one will be an instance of the other - hence the use of G2´ for the second (lower) occurrence.

We may be able to find a successful approach if some subgoal in the loop (above, G2, G21 and G2´) had already been true at the point required and need not have been achieved then. We have tried to achieve G2, G21 and G2´ after a goal G1 has been solved (G1 was thus protected) and found that a loop is generated with some operator sequence. As in the case of a protection violation, two courses are available to us. We could try to reorder goals at the upper ticklist containing the loop pattern. Removing the need to keep G1 true at that point may enable G2 to be solved without looping (say using facts in the initial situation altered when G1 was solved first. This occurs in the (REG 1 IS C2) & (REG 3 IS C1) example described in a note to the section on "swap the value of 2 registers" (section 8.2).



The alternative is to suggest some "setup" goal which would aid in the solution of G2. Any goal which would break the loop would be appropriate. For example,



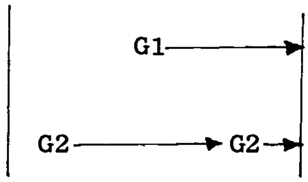


Besides the normal test of checking the promoted goal would change the actual approach being tried (by seeing if it was already true at the time required - but see NOTE), a further check must be made in those cases where the subgoal being promoted is the lower occurrence of the loop pattern (i.e., G2′). If G2′ was IDENTICAL to G2, no promotion

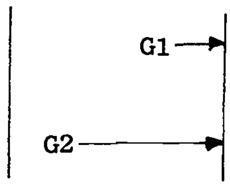
---

NOTE It can often happen that the goals to be promoted during loop correction may contain variables, and in some instances these may already be true at the point required. See note on "restrictions on the instances of a promoted goal" for how this can be handled (section 5.7.5).

should be made since



is equivalent to



The approach specifies the order in which the goals can be achieved and then kept true for the period required, the second G2 in the first holding period diagram shown is therefore superfluous.

In keeping with the above, if the lower occurrence of the loop pattern (G2′) is more general than the upper occurrence (G2 is an instance of G2′), we should disallow the promoted goal from taking an instance such that it becomes IDENTICAL to the upper occurrence (i.e., G2′ should be modified to exclude G2). If this were not done, once again an approach equivalent to G2 followed by G1 would result.

This problem occurs in the "swap the values of 2 registers" example, where the upper loop occurrence is (REG 2 IS C1) and the lower loop occurrence is (REG == IS C1). We should modify the goal to be promoted to exclude the number of the register being 2. If actor restrictions on variables were allowed (see appendix IV), this could be done by: <<REG <:NON 2:> IS C1>>.

The loop editor in the current implementation of INTERPLAN

The loop editor in the current implementation reports a loop to the user by printing on the console:

LOOP ON <lower occurrence of the loop pattern>

If a variable LOOPEDIT is set true it also prints:

WHAT SHALL I PROMOTE :

Left to itself the editor would attempt to promote subgoals being considered when the loop occurred. These would include the lower loop occurrence. If this contains variables and some instance of the pattern is true at the point to which promotion is being attempted, no promotion is made. To alleviate the defect of not having restriction facilities on variables at present, the editor can ask the user to suggest an instance of a pattern to try to promote on loop detection.

The user may go into POP-2 READY (interrupt) mode and ask such questions as what instances of the loop pattern are true at the point to which promotion will be attempted, or ask what the upper occurrence of the loop pattern is. The trace of the problem also provides information about useful instances to suggest for promotion.

The user may either type "FALSE" to indicate he does not think that correcting the loop would help, or he may suggest a goal for promotion. Normal checks for the usefulness of the suggested approaches are performed by the system.

An example of the use of this editor is given in the "swap the values of 2 registers" problem in section 8.2.

5.8 Inclusion of heuristic guidance information in INTERPLAN

The points in the current implementation of INTERPLAN at which domain-dependent knowledge can be incorporated are summarized below.

1. The ordering of preconditions in each operator schema and the ordering of the individual goals in the problem to be solved is important. This ordering is used by INTERPLAN as the approach to be considered first in each case.

2. The choice of which operators are considered "relevant" for achieving goals is important. Normally all ADD list entries of every operator should appear on the ACHIEVES list together with all those operators which can achieve them. If there is a heuristic restriction on the choice of operators for some goals this can be reflected in the ACHIEVES list. This can be used to give the same effect as the "primary additions" of STRIPS (Fikes, Hart and Nilsson, 1972b). See section 5.2(c) for more detail.

3. If there is more than one operator for any goal entry on ACHIEVES the operators can be ordered, the first being tried before others with the standard OR-CHOICE mechanism.

4. The OR-CHOICES can be made in a different order to the standard scheme by the resetting of the OR-CHOICE control parameters (see appendix III). This may be useful for example if we wish to incorporate knowledge about the probabilities of interactions in the problem domain.

5. If, for some domain, predicates can be put into hierarchies for achievement (see Siklossy and Dreussi, 1973) we can specify that reversals between members of the hierarchies should not be attempted by assigning to the SCHREVS of the operator schemas. Known hierarchies of predicates will enable us to order goals as mentioned in 1 above. Heuristic knowledge that certain orderings are equivalent may also be incorporated by assignment to SCHREVS.

```
"NOREVERSE" -  SCHREVS(<opschema>); stops any reordering attempt.
[[1.2][1.3]] -> SCHREVS(<opschema>); stops reversals between the
                   1st and 2nd or the 1st and 3rd preconditions.
```

6. A function

```
OPSCHMODIFY ε <opschema>, <achieve pattern> => <opschema>;
```

is provided. Initially this is defined to merely return the <opschema> unchanged. However, it may be redefined to allow OPSCHEMAs to be modified in the light of the environment in which they are to be used. Information can be used from the <achieve pattern> or from the ticklist this <achieve pattern> is being requested from (GLOBTICK). Schemes which reorder preconditions or set certain variables may be implemented. In particular it is possible to construct a maze-running algorithm for transfering a robot between rooms in a STRIPS-like world by assigning to certain variables in appropriate OPSCHEMAs when they are chosen (this was done for the LAWALY superworld examples run on INTERPLAN).

Operator schema withdrawal

This process allows a high degree of flexibility. For example, consider the Keys and Boxes problem where the operator GOTO(y) has different outcomes and applicability conditions depending on whether y=OUTSIDE or not, and on whether anything is HELD (see section 11.1.2). We could cause OPSCHMODIFY to select appropriate ADDs, DELETEs and PRECONDs from some data structure put in the ACHIEVES list to produce an OPSCHEMA in the light of the goal pattern required. This would alleviate the need to write out explicitly beforehand an operator schema with conditionals in its definition into the appropriate condition free OPSCHEMA structures.

7. A function

VALIDATE ε <ticklist heading> => <ticklist heading> | "INVALID";

is provided. Initially this is defined to return the <ticklist heading> unchanged. However, it may be redefined to allow domain-dependent knowledge of what conjuncts of goals are invalid to be used to check the proposed heading. It may also be written to remove repeat occurrences of goals etc. If an invalid ticklist heading is discovered "INVALID" should be returned, otherwise the valid <ticklist heading> (possibly modified) should be returned. Since the initial goal and all precondition lists of OPSCHEMAs are validated beforehand, the only way in which a heading can become invalid is if protected goals are added to a set of already valid goals or if a promoted entry is added to a set of already valid goals. If a set of protected goals are being added to a heading, the global variable NPROTECT holds the number added (they are at the front of the heading). This information can be used to cut down the amount of checking necessary to ensure validity. A useful example of how this facility may be used is decribed below.

Full expansion of search tree branches doomed to fail

INTERPLAN tries to solve a problem by TRYING OUT the problem approach it is provided with initially (the given order of goals). It solves goals in some sequence checking that previously achieved goals remain true. In many cases the system will try to achieve a goal which from the outset (if we had the information available) we could say would fail always because of the context we are trying to achieve it in. Such a problem occurs during block stacking in trying to achieve CL(B) when ON(A,B) is already true and has to be kept true. A great deal of effort may be wasted in trying different ways of achieving CL(B) when none can work if ON(A,B) must be kept true. WARPLAN (Warren, 1974) uses information about what conjunctions of facts cannot be true together to reject certain branches of its search tree. In this case an instruction such as IMPOSS(CL(y)&ON(x,y)) would be given to the planning system. A similar idea has been proposed for STRIPS (Fikes, Hart and Nilsson, 1972a, page 419). The same process could be incorporated into INTERPLAN using the VALIDATE ticklist heading facility. Whenever a new ticklist was generated, the ticklist heading would be validated using IMPOSS(...) information to reject invalid headings.

8. Any precondition of an OPSCHEMA can be preceeded by "G" to indicate that no means of achievement should be used upon it. This is intended to gain efficiency in handling global facts which are not altered by the robot's actions (e.g., <<TYPE B1 BOX>>). However, we can use the same facility to indicate preconditions which must be true but for which we do not wish actions to be applied to achieve them (even though such actions may exist in ACHIEVES).

9. Whenever the QA-system is asked a question to which it can return more than one reply (each reply causes a different choice point for planning) the system asks the user if he would like to alter the list of possibilities.

** MULTIPLE INSTANCES is printed on the console and the system goes into POP-2 READY (interrupt) mode. The instances are in the list POSSLIST which can then be examined or altered before continuing. Possibilities can be totally removed if required, or others added. This provides a useful facility to enable a user to guide problem solving.

10. When a loop is reported to the system, INTERPLAN indicates what the cause of the loop was by printing

LOOP ON <loop pattern>.

If a variable LOOPEDIT is set true, it also prints

WHAT SHALL I PROMOTE :

The user may examine the state of the search and ask questions. The user is then expected to indicate whether any attempt should be made to correct the loop. If no attempt is to be made, type FALSE (or 0), otherwise the user can indicate what goal may be worth promoting to give a new approach. The goal will usually be an instance of the loop pattern (see section on the full loop editor - section 5.7.7).

5.9 The Approach - successful Ticklist headings

The ticklist heading specifies the "approach" (the sequence chosen to attempt to achieve a set of goals) to be taken by the planning system. Any unforeseen difficulties in using this approach lead to it being discontinued, failure information being extracted as appropriate, and, possibly, new approaches being suggested. New approaches may involve reorderings of the original goals or the suggestion of certain "setup" goals in appropriate places. A successful approach fully specifies the order in which goals can be achieved and kept true without interaction. The aim of INTERPLAN is to discover such a successful approach. Successful ticklist headings contain information over which learning schemes may be devised.

Debugging the Approach

The continuous cycle of classifying the "bug" in a current ticklist and editing the tree of ticklists in the light of this can be seen as debugging the initial approach (i.e., the original goal order) to one which will in fact lead to the goals achievement. Bugs are detected by looking at the patterns of ticks and crosses in a ticklist, and alterations (edits) to the tree of ticklists (the goal control tree) are made to account for these bugs. The method used here on declarative data representations has much in common with that used in HACKER (Sussman, 1973) on more procedural representations. HACKER and INTERPLAN are examples of systems which make productive use of the information available from a failure.

6 HOW INTERPLAN SOLVES THE 3 BLOCK PROBLEM

The 3 block problem was described in section 4.2, and was used to illustrate the problem specification to INTERPLAN in section 5.2. A listing of the problem specification is given below to bring this information together. OPSCHFN functions are included. The purpose of the functions CLFN and ONFN is explained in section 5.2 (b).

```
COMMENT' BLOCK STACKING PROBLEM FOR INTERPLAN`;
VARS S1 S2;

FUNCTION CLFN; VARS B1 B2;
  INSTACT(*$*X) -> B1
  LOOPIF GETITEM(<<ON $>B2 $$B1>>,TRUE) THEN
    1 -> VALUE(<<CL $$B2>>);
    0 -> VALUE(<<ON $$B2 $$B1>>); B2 -> B1; CLOSE
END;

FUNCTION ONFN; VARS B1 B2;
  INSTACT(*$*X) -> B1; INSTACT(*$*Y) -> B2;
  IF GETITEM(<<ON $$B1 <:ET <:NON $$B2:> $>B2:> >>,TRUE)
  THEN 1 -> VALUE(<<CL $$B2>>);
       0 -> VALUE(<<ON $$B1 $$B2>>) CLOSE.
END;

OPSCHEMA <<ACTCL *$*X>>
  ADD <<CL *$*X>>
  DELETE
  PRECONDS
  VARS X
ENDSCHEMA -> S1;

OPSCHEMA <<PUTON *$*X *$*Y>>
  ADD <<ON *$*X *$*Y>>
  DELETE <<CL *$*Y>>
  PRECONDS <<CL *$*X>> <<CL *$*Y>>
  VARS X Y
ENDSCHEMA -> S2;

CLFN -> OPSCHFN(S1);
ONFN -> OPSCHFN(S2);

[% <<CL == >> , [%S1%],
   <<ON == == >> , [%S2%] %] -> ACHIEVES;

ASSERT <<ON C A>>
       <<CL C>>
       <<CL B>>;
```

No special syntax is provided for their construction in the present program. They use HBASE primitives, e.g., GETITEM, INSTACT, VALUE and the actors ET and NON (see Barrow, 1975). Many interesting problems can be specified without the need of OPSCHFNs, e.g., the STRIPS robot world and the Keys and Boxes problem. In this case the OPSCHFNs are used to allow the operator schema's effects to be dependent on some condition of the situation it is applied to. HBASE contexts have reference numbers. The current context (CUCTXT) in which the 3 facts are asserted has reference number 1. This will be taken as the initial situation by INTERPLAN. A trace of INTERPLAN on the 3 block problem is given below.

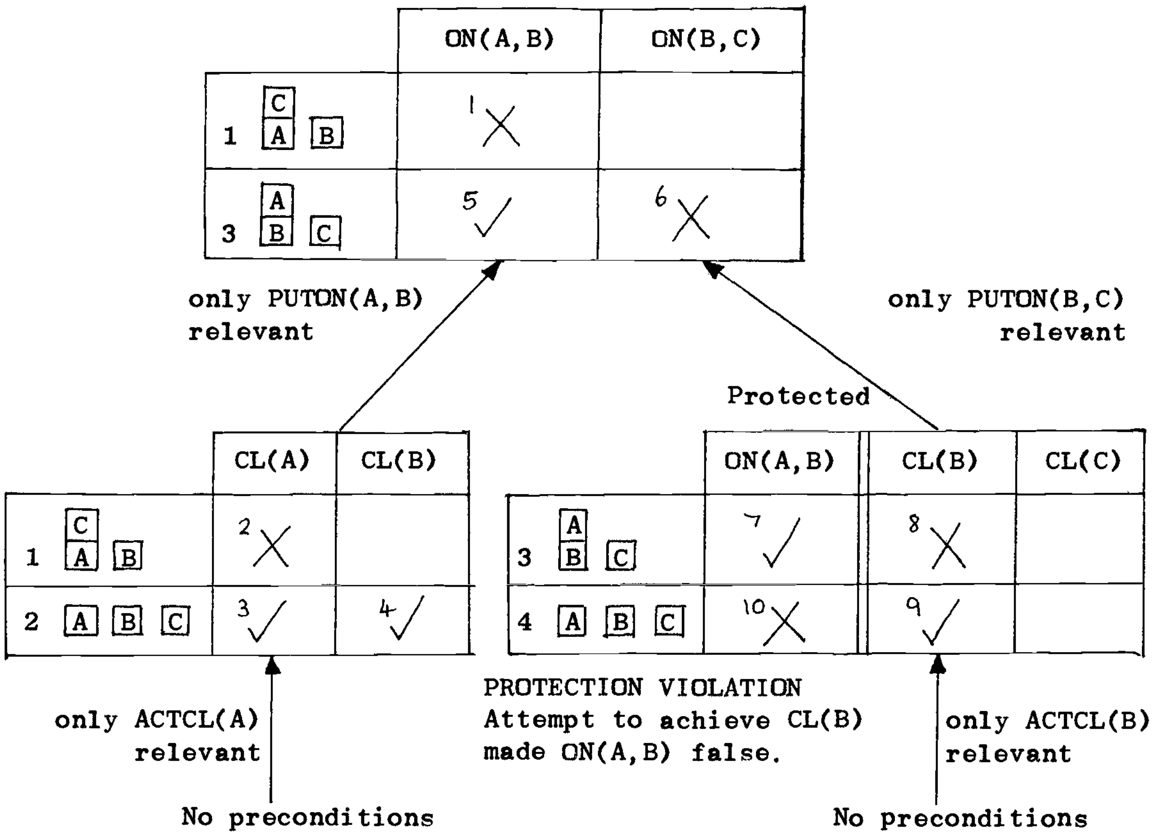
```
: GOAL <<ON A B>> <<ON B C>>;

ENTERING INTERPLAN WITH INITIAL SITUATION 1

 ** ACHIEVE << ON A B >> IN 1
 ** ACHIEVE << CL A >> IN 1
 ** APPLY << ACTCL A >> TO 1 TO GIVE 2     . . . . . . . . . . . note 1
 ** APPLY << PUTON A B >> TO 2 TO GIVE 3
 ** ACHIEVE << ON B C >> IN 3
 ** ACHIEVE << CL B >> IN 3
 ** APPLY << ACTCL B >> TO 3 TO GIVE 4
  PROTECTION VIOLATION REORDER      . . . . . . . . . . . . . . . note 2
 ** ACHIEVE << ON B C >> IN 1
 ** APPLY << PUTON B C >> TO 1 TO GIVE 5
 ** ACHIEVE << ON A B >> IN 5
 ** ACHIEVE << CL A >> IN 5
 ** APPLY << ACTCL A >> TO 5 TO GIVE 6
  PROTECTION VIOLATION PROMOTE      . . . . . . . . . . . . . . . note 3
 ** ACHIEVE << CL A >> IN 1
 ** APPLY << ACTCL A >> TO 1 TO GIVE 7
 ** ACHIEVE << ON B C >> IN 7
 ** APPLY << PUTON B C >> TO 7 TO GIVE 8
 ** ACHIEVE << ON A B >> IN 8
 ** APPLY << PUTON A B >> TO 8 TO GIVE 9

**  CPU TIME = 2.109 SECS


NOW
<< ACTCL A >>   . . . . . . . . . . . . . . . . . . . . . . . note 4
<< PUTON B C >>
<< PUTON A B >>
```

Note 1
------

2 is the reference number of the new context got by applying the operator with name <<ACTCL A>> to 1.

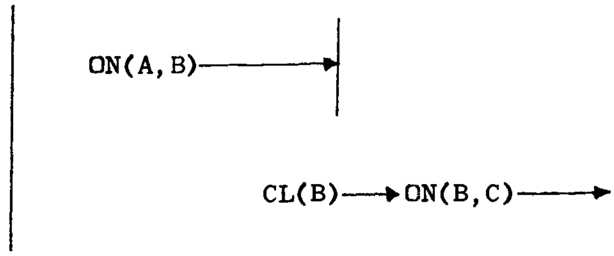
Note 2

The tree of ticklists (the goal control tree) is as below. Please note that the individual ticklists expand downwards (new rows) only as needed. The index numbers indicate the order in which the tick and cross entries were made.



The protection violation occurs when we are taking an approach as shown in the holding period diagram below.

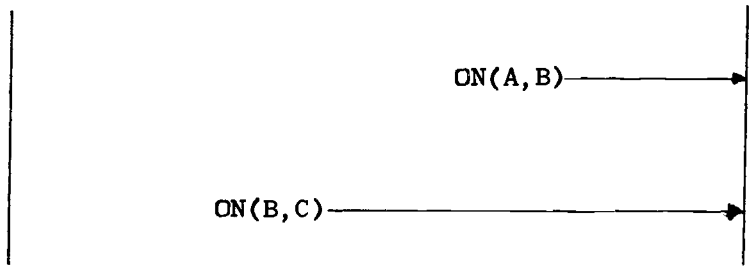
Initial Situation



Approach: ON(A,B); CL(B); ON(B,C)

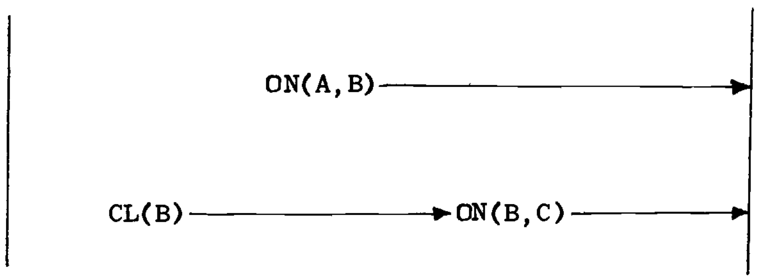
So as indicated in section 4.3, the violation may be resolved by trying one of the approaches shown below.

Initial Situation | Problem Solved



Approach: ON(B,C); ON(A,B)

Initial Situation | Problem Solved

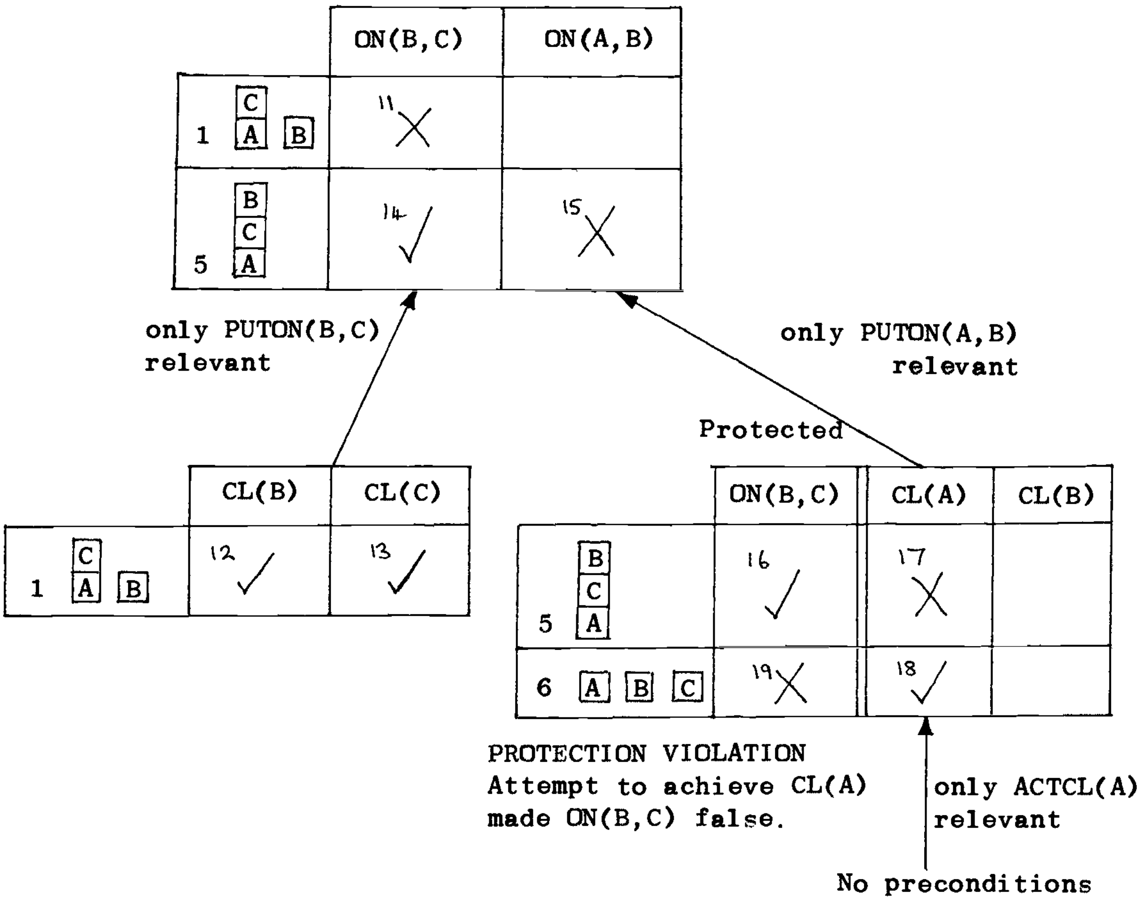


Approach: CL(B); ON(A,B); ON(B,C)

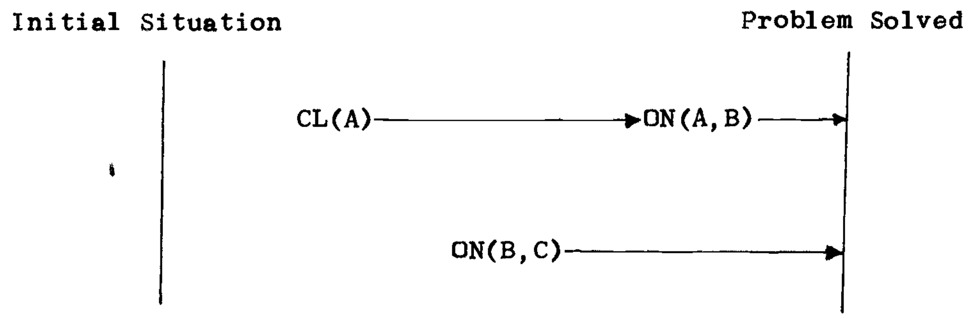
The latter cannot be used as CL(B) is already true initially and hence this approach is no different to the original which caused the violation. So, problem solving proceeds with the first (and only) suggested approach shown above. "REORDER" is printed to signify that such an approach has been suggested.

Note 3

Again a protection violation occurs while persuing this approach. The tree of ticklists then is shown below.



The approaches suggested for overcoming the violation are similar to before. However, since the top level reversal of goals has already been done, only the approach with a promoted precondition can be tried. "PROMOTE" is printed to signify this. This approach shown below is tried next as it is the only choice.



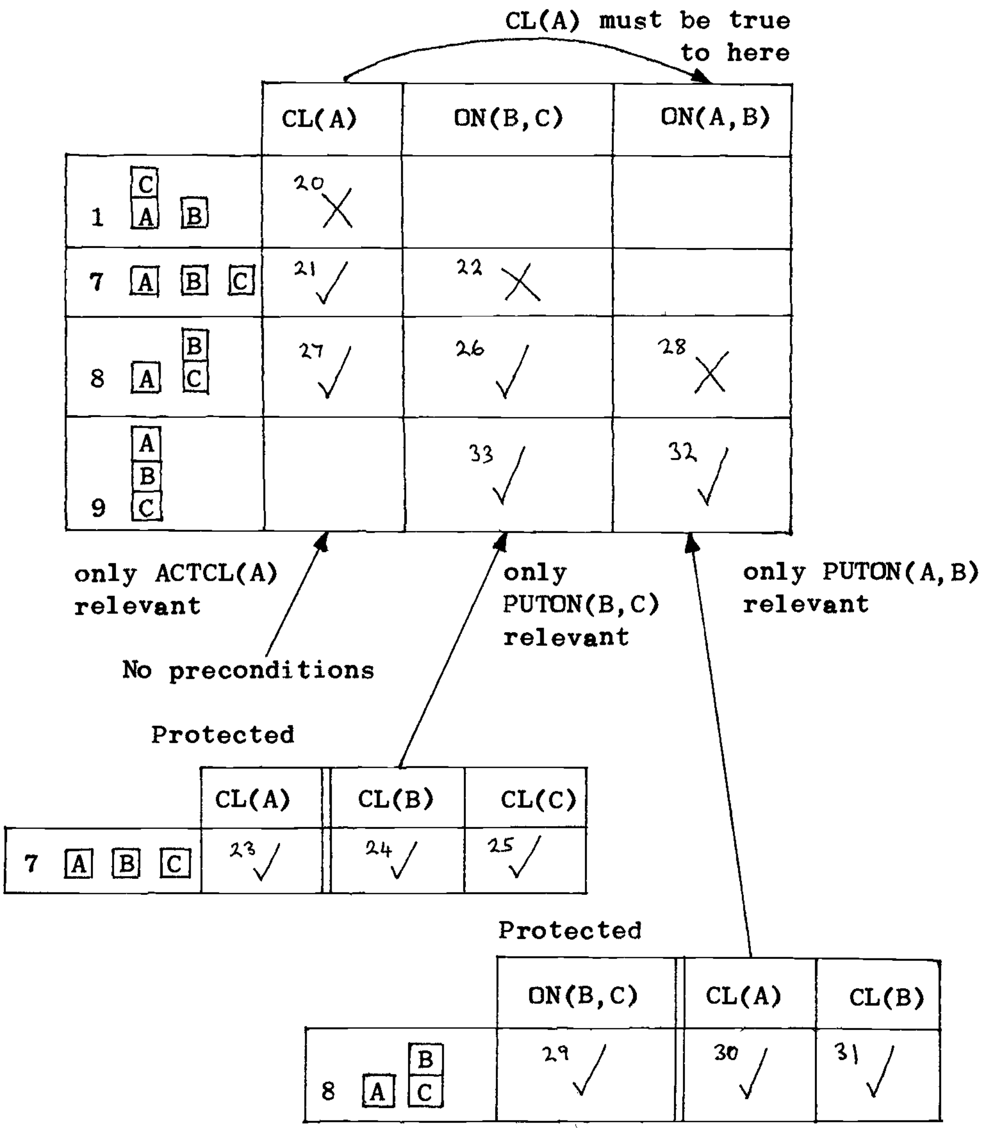
Approach: CL(A); ON(B,C); ON(A,B)

Note 4

The approach shown above is successful and produces the optimal plan

<<ACTCL A>>; <<PUTON B C>>; <<PUTON A B>>

The tree of ticklists after successful backup is shown below.



7 EXAMPLE PROBLEMS

INTERPLAN has been tried out on a variety of problems. Besides the 3-block problem (described in section 6) and a 5-block example used by Warren (1974, as described in section 9.4), the STRIPS robot world in particular was used to give some comparison between the performance of different problem solvers. The STRIPS-world is useful for comparison purposes since almost every problem solver written to date has been test run on these examples. STRIPS used this type of world to form plans for an actual robot (SHAKEY). However, it is a very simple world in which there are few serious interaction problems and in which the maximum length of a plan needed to solve any problem is limited (to 15 steps at maximum - Siklossy and Dreussi, 1973 p. 426). In view of these restrictions, problem solvers which have been written to cope with a wider class of problems than STRIPS have often extended the basic STRIPS-world by adding more actions or by changing the configuration of rooms the robot is to operate in, etc.

7.1 STRIPS-world problems

7.1.1 Operator representation

To give a background against which many of the example problems described throughout this report can be understood, the representation of the STRIPS-world actions (operators) to INTERPLAN is given below. See section 5.2 for details of how this representation specifies the problem - in particular the reason for having the ACHIEVES list of relevant operators.

```
VARS S1 S2 S3 S33 S4 S5 S6 S7 ;

OPSCHEMA <<GOTO1 *$*M>>
  ADD <<ATROBOT *$*M>>
  DELETE <<ATROBOT == >> <<NEXTTO ROBOT == >>
  PRECONDS G <<LOCINROOM *$*M *$*X>>
           <<INROOM ROBOT *$*X >> <<ONFLOOR>>
  VARS M X
ENDSCHEMA -> S1;

OPSCHEMA <<GOTO2 *$*M>>
  ADD <<NEXTTO ROBOT *$*M>>
  DELETE <<ATROBOT == >> <<NEXTTO ROBOT == >>
  PRECONDS <<INROOM ROBOT *$*X >> <<INROOM *$*M *$*X >> <<ONFLOOR>>
  VARS M X
ENDSCHEMA -> S2;

OPSCHEMA <<PUSHTO *$*M *$*N>>
  ADD <<NEXTTO *$*M *$*N>> <<NEXTTO *$*N *$*M>>
  DELETE <<ATROBOT == >> <<NEXTTO ROBOT <: NON *$*M :> >>
         <<NEXTTO <:NON ROBOT:> *$*M>>
         <<AT *$*M == >> <<NEXTTO *$*M == >>
  PRECONDS G <<PUSHABLE *$*M>> <<INROOM *$*M *$*X >>
           <<INROOM *$*N *$*X >> <<NEXTTO ROBOT *$*M>> <<ONFLOOR>>
  VARS M N X
ENDSCHEMA -> S3;

COPY(S3) -> S33; REV(ADDLIST(S3)) -> ADDLIST(S33);

OPSCHEMA <<TURNONLIGHT *$*M >>
  ADD <<STATUS *$*M ON>>
  DELETE <<STATUS *$*M OFF>>
  PRECONDS G  <<TYPE *$*M LIGHTSWITCH>> G <<TYPE *$*N BOX>>
           <<NEXTTO *$*N *$*M>> <<ON ROBOT *$*N>>
  VARS M N
ENDSCHEMA -> S4;

OPSCHEMA <<CLIMBONBOX *$*M >>
  ADD <<ON ROBOT *$*M >>
  DELETE <<ATROBOT == >> <<ONFLOOR>>
  PRECONDS G <<TYPE *$*M BOX>> <<NEXTTO ROBOT *$*M >> <<ONFLOOR>>
  VARS M
ENDSCHEMA -> S5;

OPSCHEMA <<CLIMBOFFBOX *$*M >>
  ADD <<ONFLOOR>>
  DELETE <<ON ROBOT *$*M >>
  PRECONDS <<ON ROBOT *$*M>>
  VARS M
ENDSCHEMA -> S6;

OPSCHEMA <<GOTHRUDOOR *$*K *$*L *$*M>>
  ADD <<INROOM ROBOT *$*M>>
  DELETE <<ATROBOT == >> <<NEXTTO ROBOT == >> <<INROOM ROBOT == >>
  PRECONDS <<INROOM ROBOT *$*L>> G <<CONNECTS *$*K *$*L *$*M>>
           <<NEXTTO ROBOT *$*K>> <<ONFLOOR>>
  VARS L M K
ENDSCHEMA -> S7;
```

```
[% <<ATROBOT == >> , [%S1%],
   <<NEXTTO ROBOT == >> , [%S2%],
   <<NEXTTO == == >> , [%S3,S33%],
   <<STATUS == ON>> , [%S4%],
   <<ON ROBOT == >> , [%S5%],
   <<ONFLOOR>> , [%S6%],
   <<INROOM ROBOT == >> , [%S7%] %] -> ACHIEVES;
```

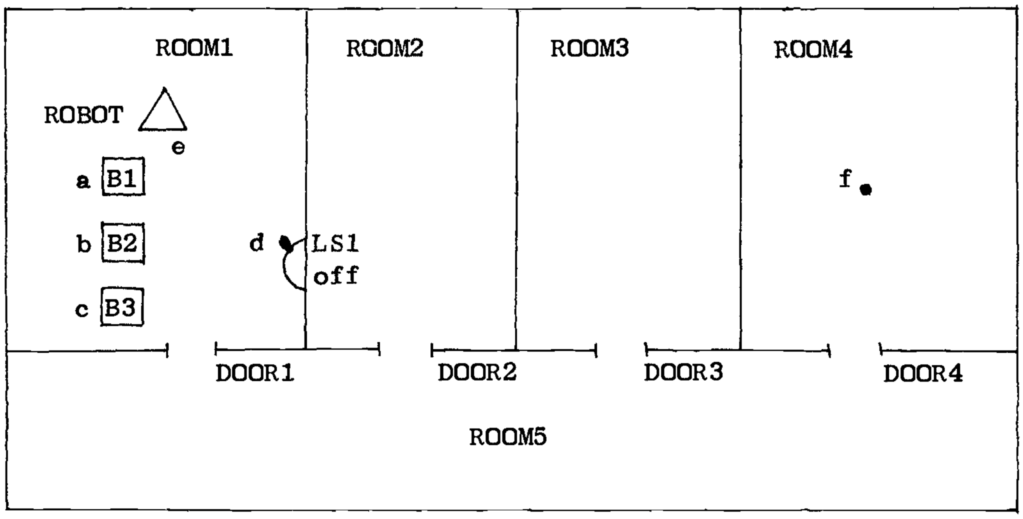
7.1.2 Implementation note

There are 7 operators, 6 of which are straightforward in that they only have one statement on their ADD list. However, operator schema S3, (PUSHTO m n), can add (NEXTTO m n) and (NEXTTO n m). So there are 2 ways to achieve e.g. (NEXTTO B1 B2), by using a (PUSHTO B1 B2) or a (PUSHTO B2 B1). In the current implementation of INTERPLAN, the variables of an OPSCHEMA are instantiated to make it relevant by matching the statement the operator is to achieve against the ADD list entries in turn from left to right until a match succeeds, the variables being set by this successful match. Normally, if it will match more than one entry in the ADD list, the 2nd and later occurrences can never be reached by the left to right matching. In the (PUSHTO m n) OPSCHEMA the achieve statement will always match the 1st entry in the ADD list (NEXTTO m n) and so to achieve, for instance, (NEXTTO B1 B2) only (PUSHTO B1 B2) would be tried whereas (PUSHTO B2 B1) is also relevant.

To overcome this implementation restriction, one must make a copy of the OPSCHEMA in which the ADD list entry which would not normally be reached in the left to right scan is put in a position in the copied ADD list such that it will be. In the STRIPS-world representation this is done by simply reversing the ADD list of OPSCHEMA S3 to give a new OPSCHEMA S33.

7.1.3 Initial situation

The initial situation used for the problems given to STRIPS is shown in the diagram below.



The following assertions represent this initial situation to INTERPLAN.

```
ASSERT
<<TYPE DOOR1 DOOR>>
<<TYPE DOOR2 DOOR>>
<<TYPE DOOR3 DOOR>>
<<TYPE DOOR4 DOOR>>
<<TYPE B1 BOX>>
<<TYPE B2 BOX>>
<<TYPE B3 BOX>>
<<TYPE LS1 LIGHTSWITCH>>
<<INROOM DOOR2 ROOM2>>
<<INROOM DOOR2 ROOM5>>
<<INROOM DOOR3 ROOM3>>
<<INROOM DOOR3 ROOM5>>
<<INROOM DOOR4 ROOM5>>
<<INROOM DOOR1 ROOM5>>
<<INROOM DOOR4 ROOM4>>
<<INROOM DOOR1 ROOM1>>
<<CONNECTS DOOR1 ROOM5 ROOM1>>
<<CONNECTS DOOR4 ROOM4 ROOM5>>
<<CONNECTS DOOR2 ROOM2 ROOM5>>
<<CONNECTS DOOR2 ROOM5 ROOM2>>
<<CONNECTS DOOR3 ROOM3 ROOM5>>
<<CONNECTS DOOR3 ROOM5 ROOM3>>
<<CONNECTS DOOR1 ROOM1 ROOM5>>
<<CONNECTS DOOR4 ROOM5 ROOM4>>
```

```
<<LOCINROOM F ROOM4>>
<<AT B1 A>>
<<AT B2 B>>
<<AT B3 C>>
<<AT LS1 D>>
<<ATROBOT E>>
<<INROOM B1 ROOM1>>
<<INROOM B2 ROOM1>>
<<INROOM B3 ROOM1>>
<<INROOM ROBOT ROOM1>>
<<INROOM LS1 ROOM1>>
<<PUSHABLE B1 >>
<<PUSHABLE B2 >>
<<PUSHABLE B3 >>
<<ONFLOOR >>
<<STATUS LS1 OFF>>
;
```

7.1.4 Different versions of the STRIPS-world problems

The time comparisons of problem solvers on STRIPS-world problems given in the literature are a little confusing since several versions of the problem domain have been used on STRIPS. The version described in sections 7.1.1 and 7.1.3 is as given in Fikes and Nilsson (1971). This version appeared in volume 2 of the journal Artificial Intelligence and will thus be refered to as version AIVol2. An earlier version of this paper was presented at the Second International Joint Conference on Artificial Intelligence and will be refered to as version IJCAI2. The main difference in this formulation is that only box B1 instead of any box may be used to stand ON to TURNON a lightswitch. Different operators, different initial situations and different problems were used in a paper by Fikes, Hart and Nilsson (1972b) to compare normal STRIPS and STRIPS with a plan saving device called MACROPS. This was published in volume 3 of the journal of Artificial Intelligence and will thus be refered to as version AIVol3.

7.2 Time comparisons - mainly on STRIPS-world problems

In the table which follows six problem solvers are compared where possible.

INTERPLAN: A program run in POP-2 (Burstall, Collins and Popplestone, 1971) and HBASE (Barrow, 1975 - a CONNIVER-like data base package written in POP-2). The times were obtained in a single session without change of any search parameters (see appendix III). The times include garbage collection and any operating system overheads when run on the Edinburgh DEC10. INTERPLAN occupies under 5K words of core on the DEC10.

STRIPS and ABSTRIPS: all forms were run in partially compiled LISP on the Stanford DEC10.
STRIPS - Fikes and Nilsson (1971).
STRIPS with MACROPS - Fikes, Hart and Nilsson (1972b).
ABSTRIPS - Sacerdoti (1974).

LAWALY: is run in interpreted LISP on a CDC-6600 and the times include garbage collection. (CDC-6600 is reputedly approx. 8 times faster than the DEC10).

WARPLAN: is interpreted in PROLOG (see Warren, 1974), which is implemented in FORTRAN and is run on the Edinburgh DEC10.

Times in seconds
To 2 significant figures

	INTERPLAN			STRIPS			MACROP STRIPS	ABSTRIP	LAWALY		WARPLAN
STRIPS ROBOT WORLD	AIvol2	IJCAI2	AIvol3	AIvol2	IJCAI2	AIvol3	AIvol3	AIvol3	AIvol2	AIvol3	IJCAI2
STATUS LS1 ON	1.6	1.5	-	113	65	-	-	-	1.6	-	8.8
ATROBOT F	2.1	2.1	-	123	125	-	-	-	4.1	-	18
NEXTTO B2 B3&NEXTTO B3 DOOR1 &STATUS LS1 ON&NEXTTO B1 B2 &INROOM ROBOT ROOM2	18	12	-	-	-	-	-	-	-	-	84
STATUS LS1 ON&NEXTTO B2 DOOR1 &NEXTTO B1 B2&NEXTTO B3 LS1 &ATROBOT F	10	11	-	-	-	-	-	-	10	-	-
STATUS LS1 ON&NEXTTO B1 B2 &NEXTTO B2 B3&ATROBOT F	13	12	-	-	-	-	-	-	-	-	73
NEXTTO B1 B2&NEXTTO B2 B3	4.4	4.4	3.3	66	122	587	180	150	4.1	14	18
NEXTTO B1 B2&INROOM ROBOT ROOM1	-	-	2.1	-	-	100	100	114	-	7.4	-
NEXTTO B2 B3&INROOM ROBOT ROOM3	-	-	2.4	-	-	344	126	175	-	11	-
INROOM ROBOT ROOM3	-	-	2.6	-	-	274	318	144	-	7.7	-
NEXTTO B1 B2&NEXTTO B3 B4	-	-	8.5	-	-	>1200*	349	401	-	19	-
BLOCK STACKING											
3-Block Problem		2.1			-		-	-		-	8.0
5-Block Problem		7.0			-		-	-		-	40

* STRIPS did not solve this problem when given 20 minutes of CPU time.

7.3 Variants of the STRIPS-world run on INTERPLAN

7.3.1 Variants with interactions

Two variants of the STRIPS-world which are similar to one another were made to introduce interaction problems. These are the 2-room problem from Siklossy and Dreussi (1973) described in section 8.1, and the SHUNT problem from Warren (1974) described in section 9.5. Both problems were used to point out shortcomings of the problem solvers described in the respective references. The action of INTERPLAN on these problems is described in the sections indicated.

7.3.2 Variants with long solution paths

Another variant of the STRIPS-world was introduced to test the effect of LAWALY (Siklossy and Dreussi, 1973) on problems requiring long sequences of individual operators to achieve some goals. A "superworld", as they termed it, was invented with 7 rooms in which a robot janitor was asked to sweep rooms, empty rubbish bins, water plants, etc. The domain has 26 operator schemas and an initial situation described by 120 assertions.

However, in this domain for any given goal, only one operator schema is relevant so eliminating branching in the search tree for operator choices. There are no serious interaction problems in the domain, and there are no interactions at all when priorities are given for the order of achievement of the individual goals and preconditions (as is done in LAWALY). Problems in this domain, though requiring long operator sequences, need only minimal problem solving capabilities in that there is only one operator relevant to each goal and the preconditions of such operators can always be satisfied. Backtracking is thus not needed for the solution of the problems in this domain. This fact is used by LAWALY so that in between partial searches to solve each component of a conjunct of goals, any choices generated are cleared leaving only the successful partial plan for earlier components of the conjunct.

Perhaps the only complexity of the LAWALY "superworld" for means-end analysis driven problem solvers is the lack of guidance available when a choice of intermediate rooms must be made to go from one room to another when these are not directly connected. LAWALY uses a maze-running algorithm to cope with this problem The maze-running algorithm computes an optimal path between any two rooms in the domain.

A listing of the "superworld" input to LAWALY was obtained and run on INTERPLAN in a similar form. The original axiomatization contained several errors which would not enable certain problems to be solved. Therefore, the version run on INTERPLAN was only changed as necessary to enable some search timings to be found. A maze-running capability was given to INTERPLAN using the OPSCHMODIFY facility (see section 5.8(6)). LAWALY solved some very long problems in this domain. A 198 step plan being found in 348 seconds and a 275 step plan being found in 433 seconds. Giving an average time per step of the final plan of 1.65 seconds. A problem in this domain was given to INTERPLAN. It was to water plants in all 7 rooms of the world. This required a 151 step plan which was found by INTERPLAN in 306 seconds, an average of just over 2 seconds per step of the final plan.

7.4 Comments on the time comparisons

7.4.1 Purpose of the time comparisons

The time comparisons of INTERPLAN on a variety of problems against other problem solvers are intended to show that it has been possible to incorporate the mechanism of protecting achieved goals and monitoring any interactions which occur to allow corrections to be made without ruining the performance of a problem solver. The range of problems which can be solved by INTERPLAN is greater than the range which can be dealt with by all the variants of STRIPS and LAWALY, yet INTERPLAN performs favourably in relation to them. The test of INTERPLAN on a single problem requiring a long plan in the LAWALY "superworld" was made for a similar reason.

Time comparisons of different systems on different computers are always difficult to make since the problem solvers are intended to cope with different aspects of planning and may have additional facilities to those being compared. Such comparisons can only be used to get a rough estimate of relative performance.

7.4.2 Comparison with STRIPS

The significant improvement of search times of INTERPLAN over STRIPS must be explained since INTERPLAN is based on many of the ideas in STRIPS but has extra abilities and mechanisms.

(a) A major factor is the use in INTERPLAN of a very simple language for performing the storage and retrieval of facts about situations in the world (the Question-answering system). INTERPLAN uses HBASE (Barrow, 1975) primitives to perform this task whereas STRIPS uses a modification of the QA3 theorem prover (Green, 1969). QA3 provides a richer language in which a situation of the world can be described (allowing implications to be used), but this power is not required for the simple problems tackled by STRIPS and the QA3 system is therefore cumbersome in this use.

(b) INTERPLAN also has a particularily straightforward method of building up its search tree using a simple iterative process of classifying and editing the structure being constructed. Ticklists provide a very simple method of allowing the appropriate edit to be chosen.

7.5 Problems run on INTERPLAN

This section lists the different problem domains given to INTERPLAN at present. Where problems in these domains are described in this report, section references are given.

Block stacking problems: especially 3 block problem (section 6) and 5 block problem (section 9.4).

STRIPS-world problems: see earlier in this chapter.

STRIPS-world variants: 2 Room problem (from Siklossy and Dreussi, 1973) see section 8.1.
SHUNT problem (from Warren, 1974) see section 9.5.
LAWALY superworld (from Siklossy and Dreussi, 1973) see section 7.3.2

A simple machine code programming task (from Warren, 1974) including the swap the values of 2 registers problem (see section 8.2).

A model car assembly task.

A simplified version of the Keys and Boxes problem (from Warren, 1974).

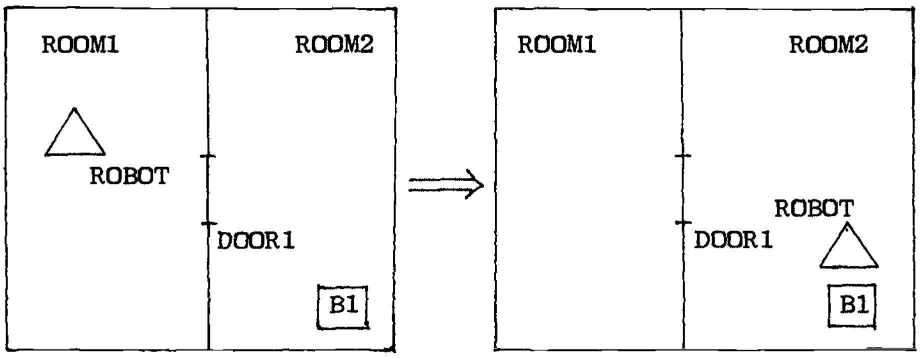
A train movement task using a common section of line.

8 OTHER PROBLEMS IN WHICH INTERACTIONS OCCUR

Interactions occur in many problems. Several of these have been mentioned previously in the literature on problem solving and have usually been dealt with in a domain specific fashion. Two of these problems will be outlined here and an interaction discovery and correction approach given for them. Such an approach does not rely upon certain domain specific facts being known before problem solving commences. Both examples have been chosen because they have influenced the design of INTERPLAN, showing the different conditions under which interactions occur.

8.1 2 Room problem

Initial Situation   Goal Situation



<<STATUS DOOR1 CLOSED>>&<<NEXTTO ROBOT B1>>

This problem is based upon the operators available in the STRIPS-AIVol3 world (see section 7.1.4). The world consists of 2 rooms connected by DOOR1 which is initially closed. The robot is in one room and a box in the other. The goal is to get the robot NEXTTO the box at the same time as the door being closed.

The problem was described by Siklossy and Dreussi (1973, sec.8) as an example of a failure of LAWALY. Though I understand that J. Roach at the University of Texas at Austin proposed the problem. It is a typical interaction problem. Concentrating on each of the component goals in either order will not achieve the goal. A similar problem, the SHUNT problem, is described in section 9.5).

An annotated trace of INTERPLAN on the problem is given below.

```
: GOAL <<STATUS DOOR1 CLOSED>> <<NEXTTO ROBOT B1>>;

ENTERING INTERPLAN WITH INITIAL SITUATION 1

 ** ACHIEVE << NEXTTO ROBOT B1 >> IN 1    .................... approach 1
 ** ACHIEVE << INROOM ROBOT  ROOM2 >> IN 1
 ** ACHIEVE << STATUS  DOOR1 OPEN >> IN 1
 ** ACHIEVE << NEXTTO ROBOT  DOOR1 >> IN 1
 ** ACHIEVE << TYPE  DOOR1 OBJECT >> IN 1
 ** APPLY << GOTOD  DOOR1 >> TO 1 TO GIVE 2
 ** APPLY << OPEN  DOOR1 >> TO 2 TO GIVE 3
  PROTECTION VIOLATION PROMOTE PROMOTE REORDER
 ** ACHIEVE << TYPE B1 DOOR >> IN 1    ....................... approach 2
 ** ACHIEVE << NEXTTO ROBOT B1 >> IN 1
 ** ACHIEVE << INROOM ROBOT  ROOM2 >> IN 1
 ** ACHIEVE << STATUS  DOOR1 OPEN >> IN 1
 ** ACHIEVE << NEXTTO ROBOT  DOOR1 >> IN 1
 ** ACHIEVE << TYPE  DOOR1 OBJECT >> IN 1
 ** APPLY << GOTOD  DOOR1 >> TO 1 TO GIVE 4
 ** APPLY << OPEN  DOOR1 >> TO 4 TO GIVE 5
 ** APPLY << GOTHRUDR  DOOR1  ROOM2 >> TO 5 TO GIVE 6
 ** APPLY << GOTOB B1 >> TO 6 TO GIVE 7
 ** ACHIEVE << STATUS DOOR1 CLOSED >> IN 7
 ** ACHIEVE << NEXTTO ROBOT DOOR1 >> IN 7
 ** ACHIEVE << TYPE DOOR1 OBJECT >> IN 7
 ** APPLY << GOTOD DOOR1 >> TO 7 TO GIVE 8
  PROTECTION VIOLATION PROMOTE
 ** ACHIEVE << STATUS  DOOR1 OPEN >> IN 1    ................. approach 3
 ** ACHIEVE << NEXTTO ROBOT  DOOR1 >> IN 1
 ** ACHIEVE << TYPE  DOOR1 OBJECT >> IN 1
 ** APPLY << GOTOD  DOOR1 >> TO 1 TO GIVE 9
```

```
** APPLY << OPEN  DOOR1 >> TO 9 TO GIVE 10
** ACHIEVE << STATUS DOOR1 CLOSED >> IN 10
** APPLY << CLOSE DOOR1 >> TO 10 TO GIVE 11
 SETUP REVERSE STOPPED
** ACHIEVE << INROOM ROBOT  ROOM2 >> IN 1     ................. approach 4
** ACHIEVE << STATUS  DOOR1 OPEN >> IN 1
** ACHIEVE << NEXTTO ROBOT  DOOR1 >> IN 1
** ACHIEVE << TYPE  DOOR1 OBJECT >> IN 1
** APPLY << GOTOD  DOOR1 >> TO 1 TO GIVE 12
** APPLY << OPEN  DOOR1 >> TO 12 TO GIVE 13
** APPLY << GOTHRUDR  DOOR1  ROOM2 >> TO 13 TO GIVE 14
** ACHIEVE << STATUS DOOR1 CLOSED >> IN 14
** ACHIEVE << NEXTTO ROBOT DOOR1 >> IN 14
** ACHIEVE << TYPE DOOR1 OBJECT >> IN 14
** APPLY << GOTOD DOOR1 >> TO 14 TO GIVE 15
** APPLY << CLOSE DOOR1 >> TO 15 TO GIVE 16
** ACHIEVE << NEXTTO ROBOT B1 >> IN 16
** APPLY << GOTOB B1 >> TO 16 TO GIVE 17

**  CPU TIME = 6.102 SECS


 NOW
 << GOTOD  DOOR1 >>
 << OPEN  DOOR1 >>
 << GOTHRUDR  DOOR1  ROOM2 >>
 << GOTOD DOOR1 >>
 << CLOSE DOOR1 >>
 << GOTOB B1 >>


: APPROACH

-1002 << INROOM ROBOT  ROOM2 >>       -1002 indicates that the goal is
 1 << STATUS DOOR1 CLOSED >>          a precondition for a goal ref. 2.
 2 << NEXTTO ROBOT B1 >>
```

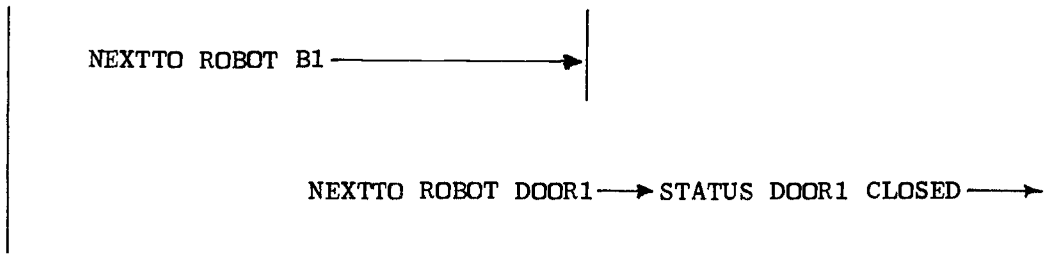
Remember that preconditions of an action to achieve a goal are written PRECOND ——▶GOAL in the diagram below. Look back at the trace to find the preconditions used.

Approach 1:

```
|
|  STATUS DOOR1 CLOSED ———▶|  holding period is broken
|                          |  by the achievement of
|                          |  STATUS DOOR1 OPEN
|
|        STATUS DOOR1 OPEN ▶ INROOM ROBOT ROOM2 ▶ NEXTTO ROBOT B1 ▶
|
```

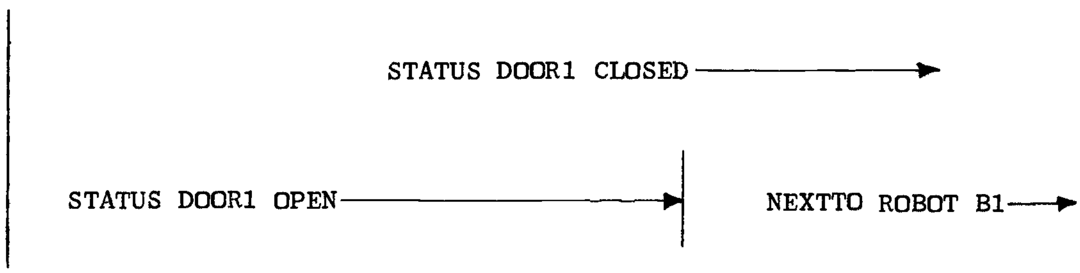
Note that the contradictory nature of the 2 goals <<STATUS DOOR1 OPEN>> and <<STATUS DOOR1 CLOSED>> is not detected as no information is known about this (IMPOSS(...) assertions could be used to save on search effort here - see section 5.8(7)). All that is known when the interaction occurs is that the achievement of the second goal deletes the first. However, INTERPLAN can still cope. The interaction suggests a REORDERING to approach 2 and 2 PROMOTIONS to approaches 3 and 4. 2 promotions are suggested as there are 2 subgoals being considered (<<STATUS DOOR1 OPEN>> and <<INROOM ROBOT ROOM2>>) when the interaction occurs, and both goals are not already true at the point at which they are being promoted to.

Approach 2:



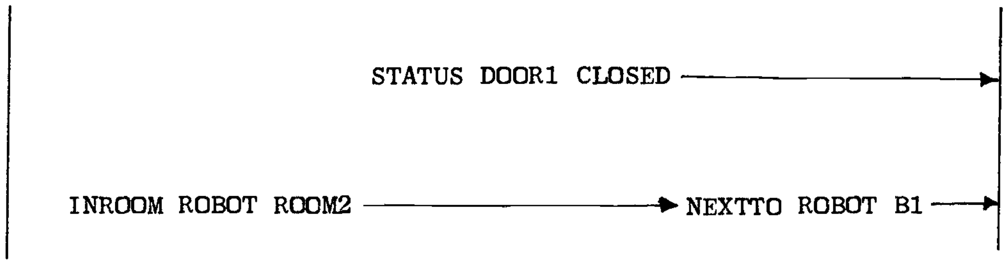
No REORDERING can be tried to correct for this interaction as it has been performed once already in response to the first interaction. However, a PROMOTION of <<NEXTTO ROBOT DOOR1>> can be made. This latter approach does not figure in the solution of the problem.

Approach 3:



Reversal of the "setup" goal (<<STATUS DOOR1 OPEN>>) is not allowed since this would place it in a position from which it had been promoted by some earlier interaction. "SETUP REVERSE STOPPED" is printed to signify this. Again note that use of IMPOSS (...) assertions could have declared the above approach INVALID.

Approach 4:



This approach is successful. Siklossy and Dreussi (1973) suggest that the problem should have been specified more exactly to a problem solver by including <<INROOM ROBOT ROOM2>> in the goal, or that this could have been done by some "transitivity of location" program. However, INTERPLAN can deal with this problem in a straightforward way using general techniques and does not rely upon domain specific knowledge which for other similar problems might not be available. It also realizes why the <<INROOM ROBOT ROOM2>> goal is needed - as a "setup" goal for <<NEXTTO ROBOT B1>> (in the context of another goal <<STATUS DOOR1 CLOSED>>). This is in contrast to its treatment as a separate top level goal in the suggestion of Siklossy and Dreussi.

8.2 Swap the values of 2 registers

A common problem in computer programming is: given 2 registers with certain values, swap their values.

Initial Situation		Goal Situation
REG 1 IS C1 REG 2 IS C2	$\Longrightarrow$	REG 1 IS C2 REG 2 IS C1

The solution involves saving one of the values in some other register before altering the two registers. This can be dealt with in a domain specific fashion by ensuring a value in one of the registers to be swapped is always saved. However below I will indicate how a general interaction detection and correction approach may be used to solve this problem.

The actions possible in this simple programming world (note) are <<STORE x / val>> which puts the value in an accumulator into REG x. <<LOAD x / val>> which loads the value in REG x into the accumulator. The entry after the "/" gives the value of the register refered to after being accessed or updated. It can be considered as a comment.

This problem requires the facilities of the full LOOP editor (see section 5.7.7). This is not available in the current implementation of INTERPLAN. However, a trace is given of the operation of INTERPLAN on this problem using the present LOOP editor which asks the user for an instance of a goal to be PROMOTED on a LOOP detection. The approaches used are described in terms of the FULL LOOP editor.

(note) This formulation of the problem was suggested by an application given to WARPLAN (see Warren, 1974) and also run on INTERPLAN in which ADD and SUBTRACT actions were also permitted. The "/" comment is needed by WARPLAN to correctly associate the ADD, DELETE and PRECOND entries for each action - these being kept in 3 separate lists (see section 9.1). It is not required by INTERPLAN.

```
: GOAL <<REG 1 IS C2>> <<REG 2 IS C1>>;


ENTERING INTERPLAN WITH INITIAL SITUATION 1

 ** ACHIEVE << REG  1 IS C2 >> IN 1        ........................ approach 1
 ** ACHIEVE << ACC IS C2 >> IN 1
 ** APPLY << LOAD  2 / C2 >> TO 1 TO GIVE 2
 ** APPLY << STORE  1 / C2 >> TO 2 TO GIVE 3
 ** ACHIEVE << REG  2 IS C1 >> IN 3
 ** ACHIEVE << ACC IS C1 >> IN 3
 ** ACHIEVE << REG == IS C1 >> IN 3
  LOOP ON << REG == IS C1 >>
  WHAT SHALL I PROMOTE: <<REG 3 IS C1>>
 PROMOTE PROMOTE REORDER
 ** ACHIEVE << REG  2 IS C1 >> IN 1        ........................ approach 2
 ** ACHIEVE << ACC IS C1 >> IN 1
 ** APPLY << LOAD  1 / C1 >> TO 1 TO GIVE 4
 ** APPLY << STORE  2 / C1 >> TO 4 TO GIVE 5
 ** ACHIEVE << REG  1 IS C2 >> IN 5
 ** ACHIEVE << ACC IS C2 >> IN 5
 ** ACHIEVE << REG == IS C2 >> IN 5
  LOOP ON << REG == IS C2 >>
  WHAT SHALL I PROMOTE: <<REG 3 IS C2>>
 PROMOTE PROMOTE
 ** ACHIEVE << REG  3 IS C1 >> IN 1        ........................ approach 3
 ** ACHIEVE << ACC IS C1 >> IN 1
 ** APPLY << LOAD  1 / C1 >> TO 1 TO GIVE 6
 ** APPLY << STORE  3 / C1 >> TO 6 TO GIVE 7
 ** ACHIEVE << REG  1 IS C2 >> IN 7
 ** ACHIEVE << ACC IS C2 >> IN 7
 ** APPLY << LOAD  2 / C2 >> TO 7 TO GIVE 8
 ** APPLY << STORE  1 / C2 >> TO 8 TO GIVE 9
 ** ACHIEVE << REG  2 IS C1 >> IN 9
 ** ACHIEVE << ACC IS C1 >> IN 9
 ** APPLY << LOAD  3 / C1 >> TO 9 TO GIVE 10
 ** APPLY << STORE  2 / C1 >> TO 10 TO GIVE 11

**  CPU TIME = 2.312 SECS


 NOW
 << LOAD  1 / C1 >>
 << STORE  3 / C1 >>
 << LOAD  2 / C2 >>
 << STORE  1 / C2 >>
 << LOAD  3 / C1 >>
 << STORE  2 / C1 >>
```

The approaches suggested here are not used in the search for a solution.

A user could have asked what instances of loop pattern were currently true and what the upper loop occurrence was to decide what to promote.

```
: APPROACH

-1002 << REG  3 IS C1 >>
 1 << REG  1 IS C2 >>
 2 << REG  2 IS C1 >>
```

Remember that preconditions of an action to achieve a goal are written
PRECOND ——→ GOAL in the diagrams below. Look back at the trace to find the preconditions used.

Approach 1:

```
REG 1 IS C2 ——————————- - - - - - - —————————→

            REG x IS C1 ——→ ACC IS C1 ——→ REG 2 IS C1 ——→
                ↑                              ↑
                └ — — — — - LOOP - — — — — — — ┘
```

A LOOP is detected on <<REG x IS C1>> as a higher level goal at that time is <<REG 2 IS C1>>. As indicated in the description of the full LOOP editor (see section 5.7.7), we may try to reorder the concurrent goals at the upper loop level (<<REG 1 IS C2>> and <<REG 2 IS C1>>). This would give approach 2 (note). Alternative approaches of suggesting a PROMOTION which would aid the solution of the upper loop occurrence of the pattern (<<REG 2 IS C1>>) while avoiding the loop are tried. PROMOTION of <<ACC IS C1>> for this purpose is straightforward, but the promotion is not used in the search for a solution. Promotion of <<REG x IS C1>> gives approach 3.

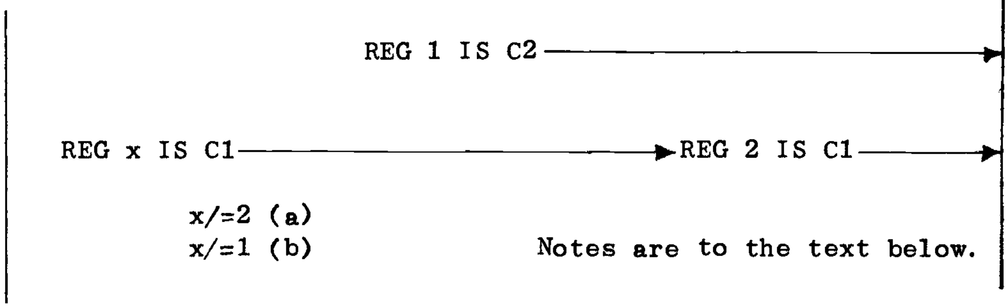
Approach 2:

```
                ┌ — — — — LOOP — — — — — ┐
                ↓                         ↓
            REG x IS C2 ——→ ACC IS C2 ——→ REG 1 IS C2 ——→


REG 2 IS C1 ——————————- - - - - - - —————————→
```

Note: If a goal of, for example, <<REG 1 IS C2>> & <<REG 3 IS C1>> is given in the same initial situation as the present problem, a straightforward reversal of the goals at the upper loop level would enable the problem to be solved.

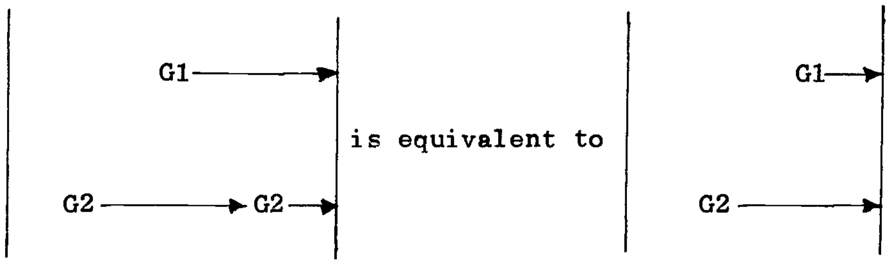
Again a LOOP is detected. A similar process to the above is performed, but the approaches which are suggested are not used in the search for a solution.

Approach 3:



The promoted goal in approach 3 can only be promoted after a LOOP has occured if

(a) the promoted goal is not IDENTICAL to the upper loop occurence of the pattern. As explained in the description of the full loop editor, this is because the approach



Thus x must not be 2.

(b) The promoted goal is not already true at the point to which it is being promoted. Since <<REG 1 IS C1>> is true initially, the goal must be restricted to exclude this instance (as explained in "restrictions on the instances of a promoted goal" - section 5.7.5). Thus x must not be 1.

A method of placing restrictions on variables has been experimented with and is outlined in Appendix IV. However, as can be seen in the trace of INTERPLAN on the swap the values of 2 registers example, the user is given the responsibility for choosing an appropriate instance of a goal to be promoted in the current implementation of the LOOP editor.

Similarity to the Keys and Boxes problem

It is interesting to note the close similarity between the approaches needed to solve the "swap the value of 2 registers" problem and those needed to solve the Keys and Boxes problem (see section 11.3).

9 WARPLAN - A COMPARISON WITH INTERPLAN

WARPLAN (Warren, 1974) is a means-end analysis driven problem solver which has been designed to solve problems described in terms similar to those used in STRIPS (initial world situation, operator schemas and the goal specification). It is intended as a method of relaxing the "linear" assumption made by earlier systems, such as STRIPS and HACKER, in which they hope that operator sequences for each individual goal can be combined end-on-end given some suitable ordering of the individuals, and that the combination of sub-plans will achieve the whole conjunct. WARPLAN, therefore, can cope with problems in which this assumption is not valid, such as the 3-block problem. Since its aims are similar to those of INTERPLAN (it being motivated to some extent by the same problem - the Keys and Boxes) it may be instructive to compare the two systems.

Before considering the detail of the method used in WARPLAN, a little background information may be useful. WARPLAN is written as 46 predicate calculus clauses which are interpreted by the PROLOG system (see appendix III of Warren, 1974). Though the program is very concise, it can cope with a wide variety of problems.

9.1 Problem specification

Operator schemas are described using 3 predicates which state which facts can be added by some operator (ADD(x,op)), what facts are deleted by some operator (DELETE(x,op)) and the preconditions required of a situation for the operator to be applicable (CAN(op,x)). Since the specification of the operator schema is in 3 different clauses, the name of the schema must contain all the variables used in its specification.

An initial situation is described using a predicate GIVEN(sitn,x) which states that the fact x is true in the situation. Facts true in all situations (global facts) can be given using a predicate ALWAYS(x). An additional predicate, IMPOSS(x), is used to state that a conjunction of facts in unattainable in any situation. This is provided for efficiency to stop fruitless goals being investigated.

9.2 Method used

The goals in a conjunct are tackled from left to right. For each goal in turn:

(a) if the goal is solved in the current situation (the initial situation for the first goal), no action is taken and we proceed to the next goal. A choice is actually being made here, it is equivalent to choosing a "do-nothing" operator at stage (b).

(b) If the goal is not solved, we seek operators which will achieve it (by looking at what operators ADD the fact).

(c) For one of the relevant operators (the others are set up by PROLOG processes as backtracking choice points in case of failure) we check if the application of the operator will delete any earlier achieved goal.

(d) If the operator is inconsistent with earlier goals, we trace back through the plan part already produced trying to find a suitable point to insert the operator. Care is taken that, at any point considered, the goal this operator is to achieve will not be deleted by actions later in the plan.

(e) Once a point of insertion for the operator is found (either after the last step of the existing plan part or some intermediate point as found in (d)), we check that the preconditions of the operator hold in the situation in which the operator will be applied.

(f) If the preconditions do not hold, a subgoal is set up of attempting to find a situation in which the operator can be applied.

NOTE: Recent work on coping with interacting goals in program synthesis is reported in Waldinger (1975). The method employed is essentially similar to that used in WARPLAN, though the two systems are not based upon one another. The discussion of WARPLAN here also applies in most part to Waldinger's system.

9.3 An Example (the 3 block problem)

Additional to the operator schemas and initial situation which are similar to those used on INTERPLAN, a fact IMPOSS(ON(x,y)&CL(y)) is given. The plan parts inserted by each step of the trace below are put in capitals in the Plan Generated column.

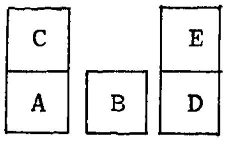
Goals Considered	Plan Generated	Comments
none	now	
ON(A,B)	now;ACTCL(A); PUTON(A,B)	Actcl(a) inserted to achieve a pre-condition for Puton(a,b) which achieves the given goal.
ON(A,B)&ON(B,C)	now;actcl(a); PUTON(B,C); puton(a,b)	Puton(b,c) to achieve ON(B,C) cannot be put on the end of the sequence since a precondition, CL(B) is inconsistent with an earlier achieved goal, ON(A,B), using IMPOSS(ON(x,y)& CL(y)). A suitable point of insertion is found just before Puton(a,b).

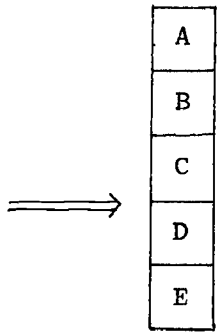
The partial plan generated holds enough information to enable the system to compute from the ADD and DELETE entries what facts hold in the situations produced by application of each operator along the plan sequence.

9.4 A problem with interleaving given operator sequences

Consider an example problem run on WARPLAN and based upon the 3 block problem. It is a 5 block problem For a detailed description of the method WARPLAN uses on this see Warren (1974). A trace of the important steps is given here. The problem is

Initial Situation





Goal Situation

The trace is for the first solution generated to this problem when using a depth-first search strategy. Other choice points could be used by backtracking.

Goals Considered	Plan Generated	Comments

ON(A,B)&ON(B,C)	now;actcl(a);
	puton(b,c);
	puton(a,b)
		found as explained previously.

ON(A,B)&ON(B,C)
&ON(C,D)
	now;actcl(a);
	ACTCL(D);
	PUTON(C,D);
	puton(b,c);
	puton(a,b)
		Puton(c,d) requires CL(C) which cannot be true if ON(B,C) is, using IMPOSS(ON(x,y)&CL(y)) once again. Therefore the operator must be put before Puton(b,c). In this position a precondition, CL(D) does not hold. It can be achieved by an Actcl(d).

ON(A,B)&ON(B,C)
&ON(C,D)&ON(D,E)
	now;actcl(a);
	actcl(d);
	PUTON(D,E);
	puton(c,d);
	puton(b,c);
	puton(a,b)
		Final goal achieved by insertion of Puton(d,e) operator.

Note in the above that the constraint to use the already existing plan sequence in the solution to subsequent goals results in a redundant step, ACTCL(A), being left in the final plan. This is due to the fact that an operator is chosen with regard to the facts which must be made to hold in a particular situation. If the operator is later shifted to a different position so that it is applied in a different situation, it may become redundant.

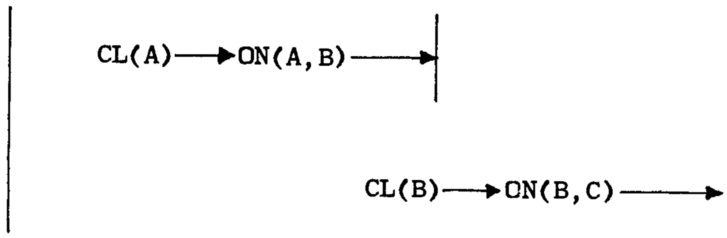
INTERPLAN modifies the order of goals it is to consider when interactions are discovered. The sequence of approaches suggested as each interaction is discovered follows similar lines to the sequence of partial plans generated by WARPLAN (as in the block stacking domain there is only one operator to achieve each goal). However, since at any point at which a goal is already true when it is tackled, no operators are applied, no redundant steps are inserted. See the trace below which shows INTERPLAN working on the 5 block problem annotated with the approaches being considered at each phase.

```
: GOAL <<ON A B>> <<ON B C>> <<ON C D>> <<ON D E>>;


ENTERING INTERPLAN WITH INITIAL SITUATION 1

 ** ACHIEVE << ON A B >> IN 1     ......................... approach 1
 ** ACHIEVE << CL A >> IN 1
 ** APPLY << ACTCL A >> TO 1 TO GIVE 2
 ** APPLY << PUTON A B >> TO 2 TO GIVE 3
 ** ACHIEVE << ON B C >> IN 3
 ** ACHIEVE << CL B >> IN 3
 ** APPLY << ACTCL B >> TO 3 TO GIVE 4
  PROTECTION VIOLATION REORDER
 ** ACHIEVE << ON B C >> IN 1     ......................... approach 2
 ** APPLY << PUTON B C >> TO 1 TO GIVE 5
 ** ACHIEVE << ON A B >> IN 5
 ** ACHIEVE << CL A >> IN 5
 ** APPLY << ACTCL A >> TO 5 TO GIVE 6
  PROTECTION VIOLATION PROMOTE
 ** ACHIEVE << CL A >> IN 1       ......................... approach 3
 ** APPLY << ACTCL A >> TO 1 TO GIVE 7
 ** ACHIEVE << ON B C >> IN 7
 ** APPLY << PUTON B C >> TO 7 TO GIVE 8
 ** ACHIEVE << ON A B >> IN 8
 ** APPLY << PUTON A B >> TO 8 TO GIVE 9
 ** ACHIEVE << ON C D >> IN 9
 ** ACHIEVE << CL C >> IN 9
 ** APPLY << ACTCL C >> TO 9 TO GIVE 10
  PROTECTION VIOLATION REORDER
 ** ACHIEVE << ON C D >> IN 1     ......................... approach 4
 ** ACHIEVE << CL D >> IN 1
 ** APPLY << ACTCL D >> TO 1 TO GIVE 11
 ** APPLY << PUTON C D >> TO 11 TO GIVE 12
 ** ACHIEVE << ON B C >> IN 12
 ** APPLY << PUTON B C >> TO 12 TO GIVE 13
 ** ACHIEVE << ON A B >> IN 13
 ** APPLY << PUTON A B >> TO 13 TO GIVE 14
 ** ACHIEVE << ON D E >> IN 14
 ** ACHIEVE << CL D >> IN 14
 ** APPLY << ACTCL D >> TO 14 TO GIVE 15
  PROTECTION VIOLATION PROMOTE REORDER
 ** ACHIEVE << ON D E >> IN 1     ......................... approach 5
 ** ACHIEVE << CL D >> IN 1                           2 choices - Reorder
 ** APPLY << ACTCL D >> TO 1 TO GIVE 16               is prefered.
 ** APPLY << PUTON D E >> TO 16 TO GIVE 17
 ** ACHIEVE << ON C D >> IN 17
 ** APPLY << PUTON C D >> TO 17 TO GIVE 18
 ** ACHIEVE << ON B C >> IN 18
 ** APPLY << PUTON B C >> TO 18 TO GIVE 19
 ** ACHIEVE << ON A B >> IN 19
 ** APPLY << PUTON A B >> TO 19 TO GIVE 20

**   CPU TIME = 7.712 SECS
```

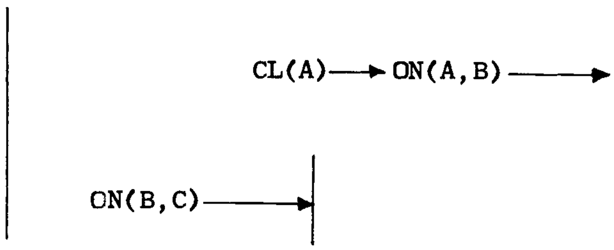
```
NOW
<< ACTCL D >>
<< PUTON D E >>
<< PUTON C D >>
<< PUTON B C >>
<< PUTON A B >>

: APPROACH

 4 << ON D E >>
 3 << ON C D >>
-1001 << CL A >>
 2 << ON B C >>
 1 << ON A B >>
```

-1001 indicates that the goal is a precondition for the goal ref. 1.

Approach 1:

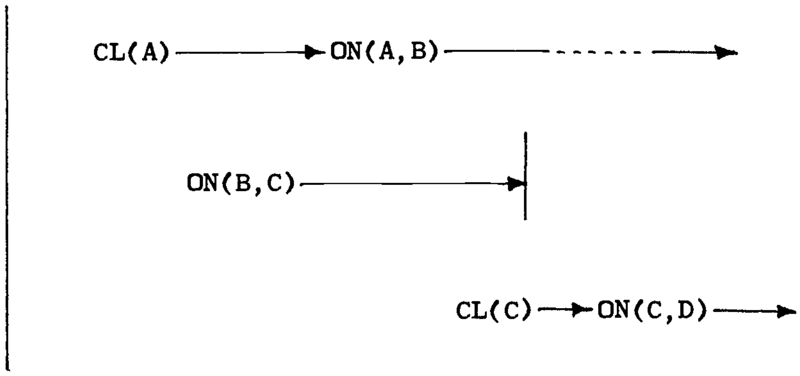


Approach 2:



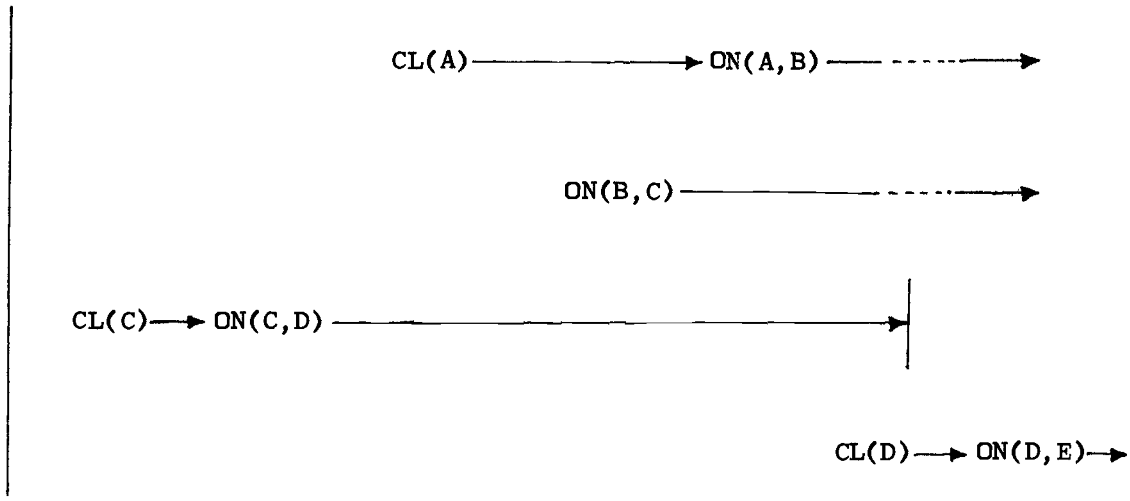
The first part of this problem proceeds exactly as for the 3-block problem (see section 6).

Approach 3:



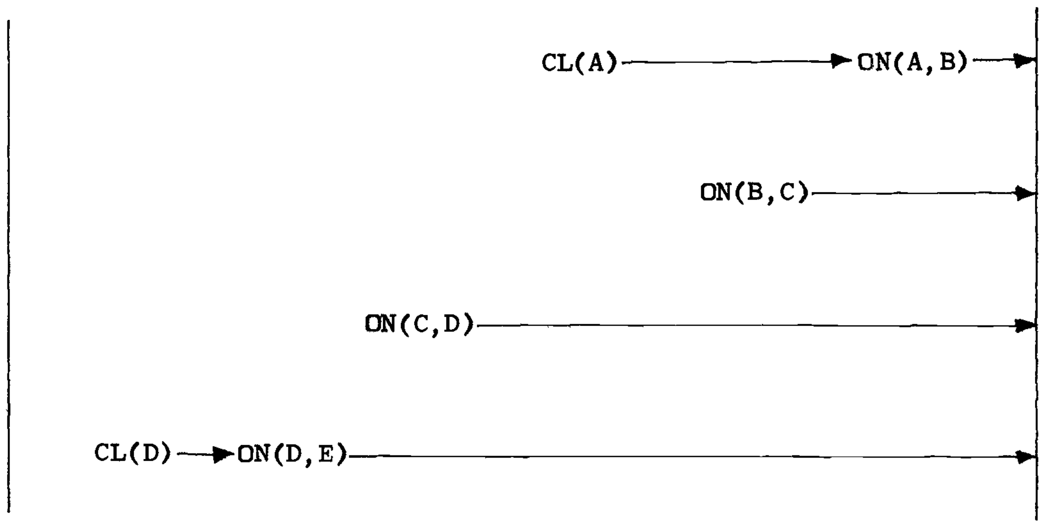
Interaction suggests a REORDERING to approach 4. PROMOTION is not allowed as CL(C), the goal to be promoted, is true before ON(B,C) (the point to which promotion is attempted).

Approach 4:



The interaction suggests a REORDERING to approach 5 and a PROMOTION of CL(D) to before ON(C,D). This latter approach is not used in the search for a solution.

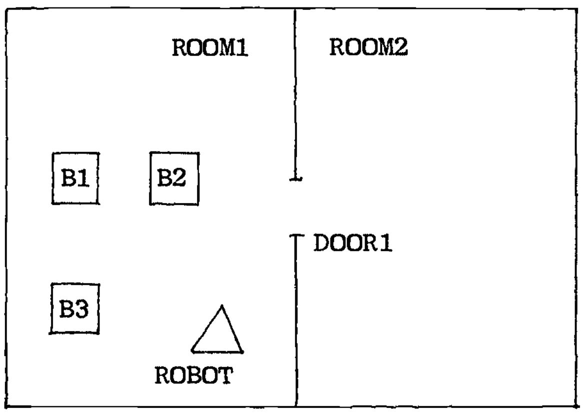
Approach 5:



9.5 The SHUNT problem

The SHUNT problem is an extension to the STRIPS-world (see section 7.1) proposed by Warren (1974) to illustrate the difficulty, outlined above of having to use a previously discovered subplan for earlier goals in the solution of further goals in a conjunct. It is similar to the 2 ROOM problem of Siklossy and Dreussi (1973).

There is one additional operator to those given in the STRIPS-world. It is <<SHUNTTHRU bx dxy rx ry>> which shunts the robot into box bx in room rx and both box and robot go through door dxy into room ry. However, the robot is not left NEXTTO the box bx. Therefore there are two ways to achieve <<INROOM ROBOT == >> using the normal GOTHRUDR or using a SHUNTTHRU. Also, additionally to the STRIPS world there is a way that a box may change the room it is in, using SHUNTTHRU. A goal of <<INROOM ROBOT ROOM2>> & <<NEXTTO ROBOT B1>> is given in the following world situation:



Warren noted that the most obvious way to achieve <<INROOM ROBOT ROOM2>> using a GOTHRUDR would not contribute to the solution of the whole goal. Since WARPLAN relies on straightforward backtracking to select continuation points after a failure, WARPLAN may have to search through many possibilities before the correct SHUNT on B1 was chosen and the correct box "accidently" shunted into ROOM2 in an attempt just to move the robot. Then this partial plan could be used to go on to achieve both goals by executing a <<GOTO2 B1>>.

Systems, such as WARPLAN, which reorder the chosen operators in the light of interactions are really most suited to tasks in which there is only one or few ways in which a goal can be achieved. If the choice of operator was inappropriate for some goal, or becomes inappropriate because of a change of position of the operator in a plan, no information is available from the resulting failure to guide the choice of another operator. This argument also applies to Sacerdoti's NOAH system (see section 10).

A trace of INTERPLAN on the SHUNT problem is given below with an annotation of the approaches being considered at each point.

```
: GOAL <<INROOM ROBOT ROOM2>> <<NEXTTO ROBOT B1>>;

ENTERING INTERPLAN WITH INITIAL SITUATION 1

 ** ACHIEVE << INROOM ROBOT ROOM2 >> IN 1    ............... approach 1
 ** ACHIEVE << NEXTTO ROBOT  DOOR1 >> IN 1
 ** APPLY << GOTO2  DOOR1 >> TO 1 TO GIVE 2
 ** APPLY << GOTHRUDO  DOOR1  ROOM1 ROOM2 >> TO 2 TO GIVE 3
 ** ACHIEVE << NEXTTO ROBOT B1 >> IN 3
 ** ACHIEVE << INROOM B1  ROOM2 >> IN 3
 ** ACHIEVE << INROOM ROBOT  ROOM1 >> IN 3
 ** ACHIEVE << NEXTTO ROBOT  DOOR1 >> IN 3
 ** APPLY << GOTO2  DOOR1 >> TO 3 TO GIVE 4
 ** APPLY << GOTHRUDO  DOOR1  ROOM2  ROOM1 >> TO 4 TO GIVE 5
  PROTECTION VIOLATION PROMOTE REORDER
```

```
 ** ACHIEVE << NEXTTO ROBOT B1 >> IN 1      .................. approach 2
 ** APPLY << GOTO2 B1 >> TO 1 TO GIVE 6
 ** ACHIEVE << INROOM ROBOT ROOM2 >> IN 6
 ** ACHIEVE << NEXTTO ROBOT  DOOR1 >> IN 6
 ** APPLY << GOTO2  DOOR1 >> TO 6 TO GIVE 7
  PROTECTION VIOLATION PROMOTE
  MULTIPLE INSTANCES            Trying a different way to achieve
READY                           <<INROOM ROBOT ROOM2>> IN 6 using a
:: GOON                         SHUNTTHRU.  This allows a choice of box.
                                B3 happens to be chosen first.

 ** ACHIEVE << NEXTTO ROBOT  B3 >> IN 6
 ** APPLY << GOTO2  B3 >> TO 6 TO GIVE 8
  PROTECTION VIOLATION PROMOTE
 ** ACHIEVE << INROOM B1  ROOM2 >> IN 1     ................. approach 3
 ** ACHIEVE << NEXTTO ROBOT B1 >> IN 1
 ** APPLY << GOTO2 B1 >> TO 1 TO GIVE 9
 ** APPLY << SHUNTTHR B1  DOOR1  ROOM1  ROOM2 >> TO 9 TO GIVE 10
 ** ACHIEVE << NEXTTO ROBOT B1 >> IN 10
 ** APPLY << GOTO2 B1 >> TO 10 TO GIVE 11

**  CPU TIME = 6.164 SECS


 NOW
 << GOTO2 B1 >>
 << SHUNTTHR B1  DOOR1  ROOM1  ROOM2 >>
 << GOTO2 B1 >>

: APPROACH

-1002 << INROOM B1  ROOM2 >>
 1 << INROOM ROBOT ROOM2 >>
 2 << NEXTTO ROBOT B1 >>
```

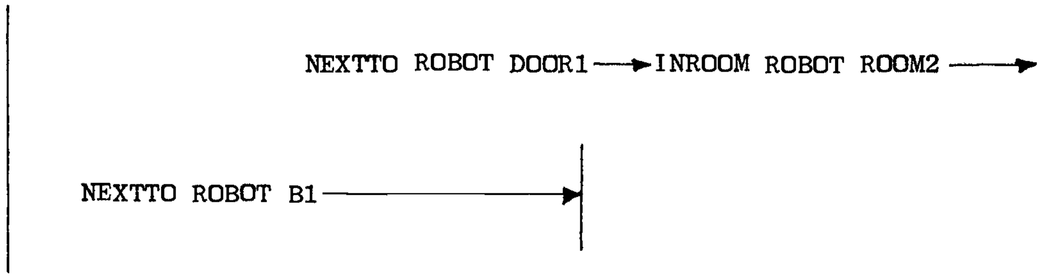
Remember that preconditions for an action to achieve a goal are written PRECOND ——▶ GOAL in the diagrams below. Look back at the trace to find the preconditions used.

Approach 1:

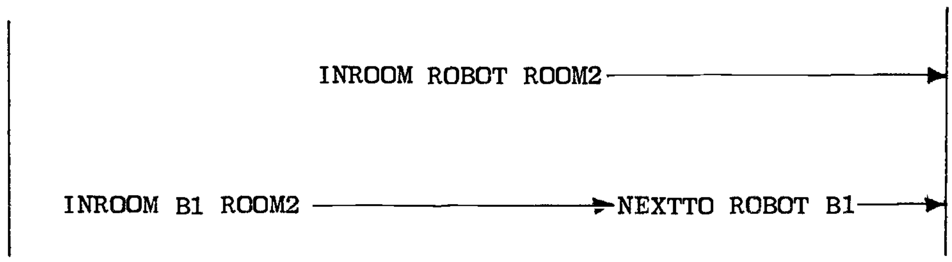
```
    |
    |      INROOM ROBOT ROOM2 ————▶| Holding period of this goal is
    |                              | broken by the achievement of
    |                              | INROOM ROBOT ROOM1
    |
    |          INROOM ROBOT ROOM1 ——▶ INROOM B1 ROOM2 ——▶ NEXTTO ROBOT B1 ——▶
    |
```

The approaches suggested to remove the interaction are a REORDERING to approach 2 and a PROMOTION to approach 3. The latter approach proves successful, the choice of the SHUNTTHRU on B1 then being constrained. It is chosen to achieve INROOM B1 ROOM2 on purpose and not as a fortunate accident.

Approach 2:



Approach 3:



Using "primary additions" only

To make this point clear, if we disallowed SHUNTTHRU as an operator relevant to achieving <<INROOM ROBOT == >>, using SHUNTTHRU only to achieve <<INROOM box == >> and GOTHRUDR to achieve <<INROOM ROBOT == >> (i.e., primary additions only are on the ACHIEVES list given to INTERPLAN - see section 5.8(2)), the problem would still be solved by INTERPLAN.

9.6 Goal Ordering vs. operator reordering

WARPLAN has taken the extreme of considering the goals in a fixed order and re-arranging the operators of suggested partial plans for each goal to form the plan for the conjunct of goals. This has led to the difficulty discussed above. However, INTERPLAN takes another extreme position. It considers some ordering of the goals in the conjunct and tries to form operator sequences to solve the individual goals and combine these in THE ORDER GIVEN. Any interactions are corrected for by discontinuing the former approach and suggesting a reordering of the goals or some promotion of a subgoal to try to remove the cause of interaction. INTERPLAN then tries to find operator sequences for the individual goals to be combined in the new order. Interactions may be localised and so not require a restart on the top level goals. Since some of the operator sequences may be virtually the same regardless of the position in the plan, this can lead to a serious duplication of effort. For example, a long operator sequence is needed to ensure a key is taken to the door in the Keys and Boxes problem (see section 11.2.2). After the discovery of this sequence an interaction occurs and planning with a different goal ordering requires virtually the same long operator sequence to be found.

Operator recommendations

Early designs for INTERPLAN considered the notion of keeping an association list of all relevant operators for each goal in an operator's precondition with the operator data structure (see appendix I.2) which is kept at the appropriate Levels of the goal control tree. When a goal was first to be attempted, the relevant operators would be found by the normal process by looking for all operators which could ADD the goal. The association list entry for relevant operators for a goal would have 3 components:

(a) previously successful operators

(b) untried operators

(c) previously failed operators.

On initialization only component b would have any entries. If the goal was G1 and 2 operators were relevant, after initialization the association pair would be:

(G1 , < nil , [% op1 , op2 %] , nil >).

Whenever a choice of an operator is to be made for G1, it is then done in the following way:

i) get relevant operator's association value for the goal.

ii) if not yet initialized, do so as above.

iii) Set up choice points for the alternative operators available, heuristically ordered so that relevant operators from the previously successful list are chosen first, untried operators next and previously failed operators last.

INTERPLAN normally performs step iii) by finding the operators which can ADD the given goal, and it orders them according to the order they are put in by the user.

Whenever backup occurs to a ticklist (whose heading represents the precondition of some operator) then:

ON SUCCESS: if the successful operator is not on the previously successful operators list of the operator whose precondition is represented by the successful ticklist, remove it from its present list and add it to the previously successful operators list.

ON FAILURE: do likewise for the failed operator to the previously failed operators list.

Now, whenever a re-arrangement of goals is made on an interaction, the relevant operator's recommendations can be passed from the failed approach to the new one.

This scheme only accounts for the outcome of the last use of an operator. Instead a count of the number of successful and failed uses could be used to order operators within one list (+1 for success, -1 for failure). A disadvantage of the operator recommendation notion would be that much of the data structures generated during problem solving would be retained after use while the recommendations were kept.

10 NOAH - A COMPARISON WITH INTERPLAN

It is obvious that many interaction problems arise because the goals are tackled in a linear way. Given a suitable ordering of the components of a conjunct of goals many interactions can be avoided. The techniques of this report allow interactions to be found, and corrected for, under the assumption that we wish to tackle the goals linearily. This is because efficient problem solvers can be written which tackle goals linearily.

Sacerdoti (1975) has described a non-linear approach to problem solving embedded in NOAH (Nets of Action Hierarchies), a program written in QLISP (Bobrow and Raphael, 1974). The system is intended only to make assumptions about the ordering of individual actions when this is necessary to the solution of the problem at hand.

Problem actions are described to the system as QLISP functions which embed the ADD, DELETE and PRECOND entries of an OPSCHEMA. When some goal is given, the system works by progressively refining a "procedural net" for the problem. Refinement occurs by finding actions to achieve the goals, then running the QLISP code for the chosen action which in turn asks for the achievement of the action's preconditions, and when this is done updates the world model to reflect the effects of the action.

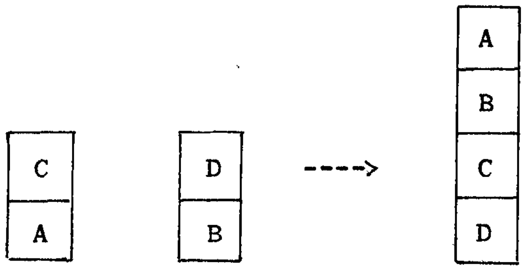
Generally there are two steps which are performed in turn until the net is fully refined (the problem solved).

(a) Choice of an action to achieve an unsolved goal. This choice may in turn introduce new precondition goals.

(b) "Criticism" of the structure of the net to look for interactions between suggested actions etc.

Sacerdoti (1975) shows how the procedural net is used within a particular problem solver (NOAH) to handle block stacking problems. An example will be used to show the operation of the system. It is a 4 block problem, the 3 block problem described in section 4.2 is included in this. The problem is chosen as it shows more features of the system than the 3 block problem would.

10.1 NOAH on the 4 block problem

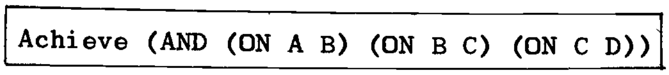


The following notation is used in the diagrams below:

Achieve	A goal which is not satisfied in the situation it is required in.
	A goal which is satisfied in the situation it is required in.
	An action to achieve a goal.
S	A special "split" node for parallel branches.
J	A special "join" node for parallel branches.
+x, -x	An action labelled -x deletes some precondition x (labelled +x) for a parallel action.
⟶	An arc specifying an ordering constraint between a pair of nodes.

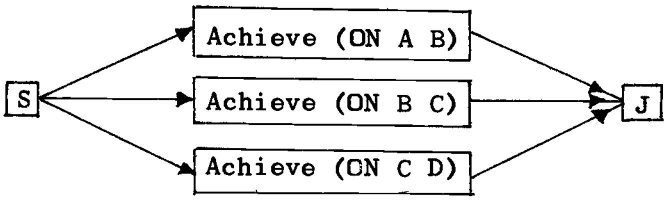
Levels in the
Procedural Net
(Refinements)

1.



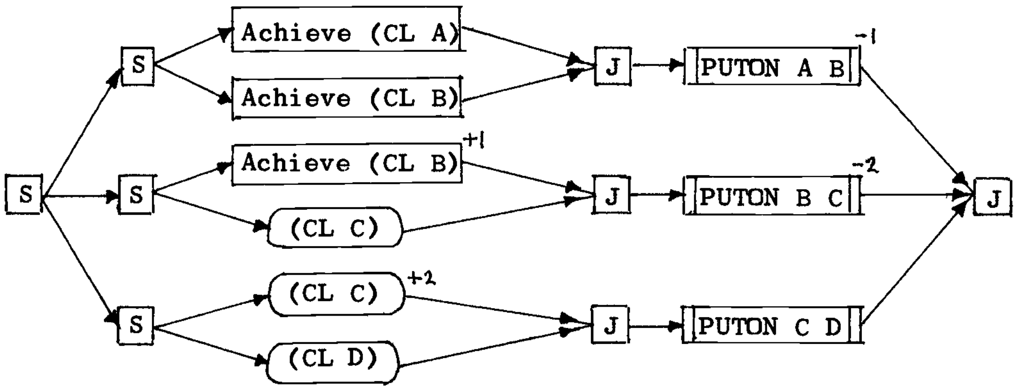
The function for Achieve (AND ...) suggests parallel branches for the components.

2.



The function for Achieve (ON x y) suggests a PUTON x y with preconditions (CL x) and (CL y). N.B. If there was more than one relevant operator, different procedural nets would have to be made available for consideration at this point.

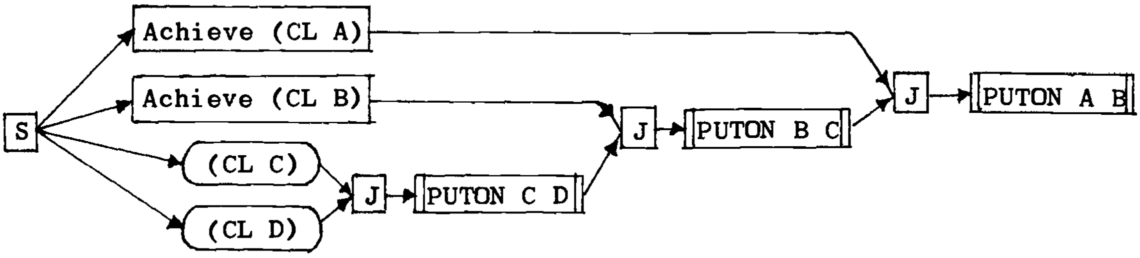
3.



The System notices that in 2 cases a precondition (+x) is deleted by a parallel operation (-x). The recognition is done by building a structure called the "table of multiple effects" and allowing several critics to look for interactions indicated in the table (see later). These critics suggest appropriate linearizations when interactions are found. The plan is thus partially linearized to put the goal which has a deleted precondition before the negating action.

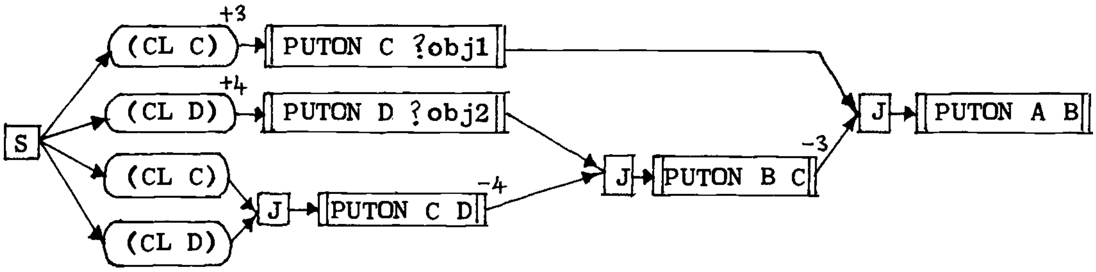
Redundant preconditions in parallel branches are eliminated.

3′ (after criticism)



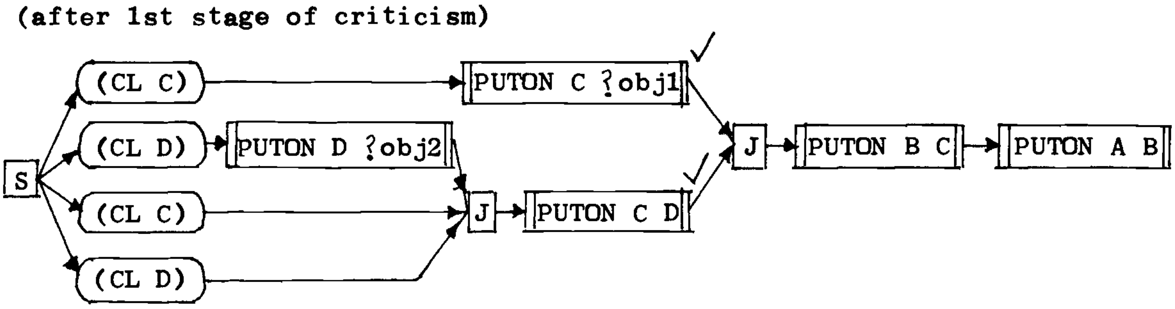
The function for Achieve (CL x) suggests moving a block y which is (ON y x). A PUTON y z for some z is used. This is different from the block stacking problems run on INTERPLAN but the same interactions occur.

4.



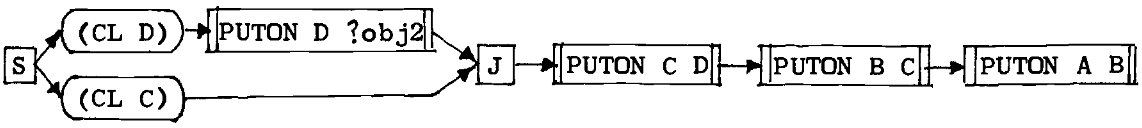
2 preconditions are again deleted by parallel operations. Further linearization takes place as a result of criticism.

4′ (after 1st stage of criticism)



The system tries to make actions with unspecified arguments redundant by trying to unify them with a parallel action using a suitable choice of variable specification (i.e. here ?obj1="D"). The 2 merged operations are ticked (√) in the diagram
Final criticism removes redundant preconditions in parallel branches.

4′′ (after criticism)



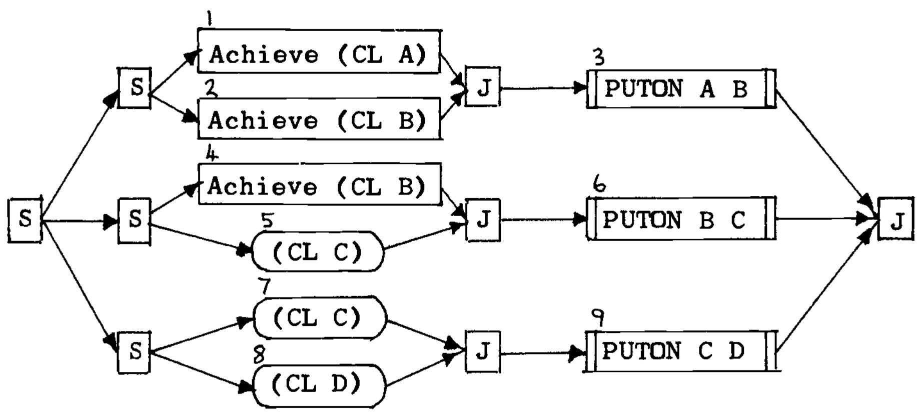
10.2 The multiple effects table

At each stage of plan expansion after new nodes have been added to the procedural net, various "critics" are allowed to look at the net and make appropriate changes if they see fit. One of these critics (called Resolve Conflicts) looks for interactions between parallel branches.

It behaves thus:

1. A table of multiple effects is built by making an entry for each expression (goal) that is asserted or denied by more than one node in the current net.

   E.g. At level 3 of the example block stacking problem given in the previous section we had the following situation (nodes are numbered for use in the explanation to follow).



The table of multiple effects would initially be:

CL B:	asserted at node 2 deleted at node 3 asserted at node 4
CL C:	asserted at node 5 deleted at node 6 asserted at node 7
CL D:	asserted at node 8 deleted at node 9

2. Eliminate from the table those expressions which are deleted at the node they are a precondition for.

   E.g. CL B at node 2 is a precondition for the action PUTON A B at node 3 and is deleted at node 3. No interaction is involved in such a deletion. Drop any expression from the table which only has one entry left after this elimination of preconditions. The table above then becomes:

   CL B: deleted at node 3
   asserted at node 4

   CL C: deleted at node 6
   asserted at node 7

3. The Resolve Conflicts critic then uses the interaction information in the table to partially linearize the procedural net being considered as shown in the previous section.

Ticklists and the table of multiple effects

The table of multiple effects performs a similar function to the ticklists of INTERPLAN (and indeed were based upon ticklists and the notion of looking for interactions by the simple examination of a table of the effects of different actions upon the goals required - Sacerdoti, 1975 p.29).

Such tabular formats provide a simple means of detecting interactions between subgoals and allows the locality of the interaction to be identified. The discovery of an interaction can thus be a constructive thing in that suitable corrections can easily be made when definite information as to what goals are interacting and how they interact is available. This is quite different from the procedure in many existing problem solvers which would simply backtrack to other choice points on discovering an interaction, or worse still, fail to detect the interaction at all.

10.3 Some limitations of the current version of NOAH

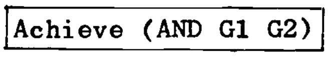
10.3.1 Choice of an operator if several are relevant to one goal

Actions are only put into the procedural net of a problem if they are relevant to the achievement of a goal being considered. If there is more than one relevant operator, some single action must be chosen from those available. The other choices giving rise to different nets which are kept as backup possibilities. If it turns out that the choice was incompatible with the other parallel goals being considered, the net currently being worked upon cannot lead to a solution and a failure is reported to the problem solver. However, as in WARPLAN (see section 9), no information as to the cause of the failure is given and blind backtracking is used to select a new alternative net from the backup possibilities available. Thus in those problems where the choice of the obvious relevant operator will not lead to a solution, the procedural net will not perform well. An example in which this would arise is the SHUNT problem detailed earlier in the comparison with WARPLAN (see section 9.5). The difficulty is explained there and the action of INTERPLAN on such problems described.

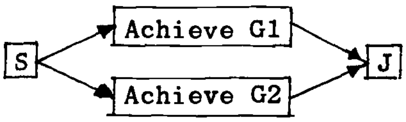
10.3.2 Restrictions on the legal linearizations to correct for an interaction

If 2 goals are given, G1 and G2, and there are relevant actions A1 and A2 with preconditions G11 and G22 respectively, the net may be refined thus:

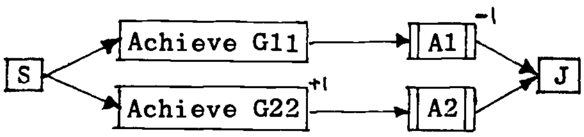
level 1:



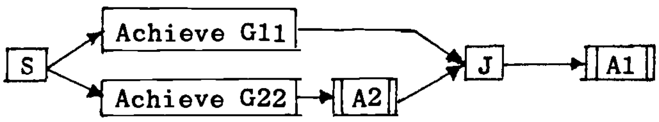
level 2:



level 3:

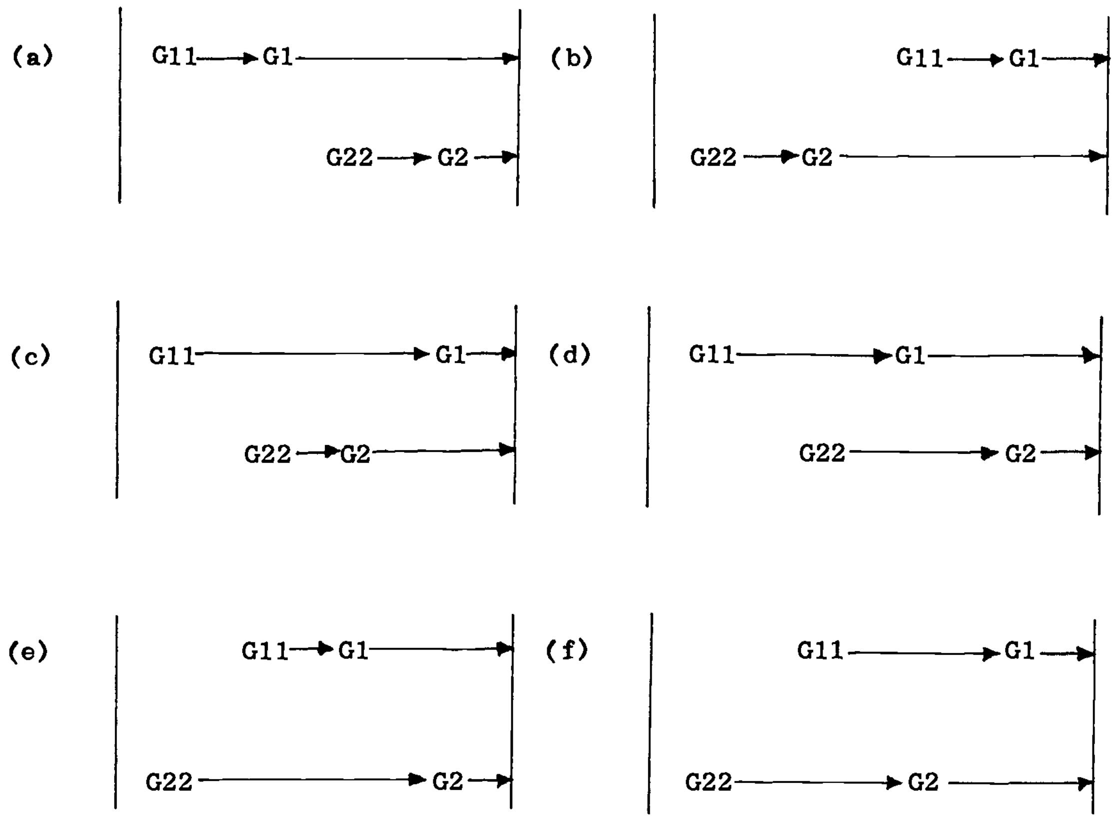


Say there are interactions as indicated in the final diagram where A1 deletes G22, a precondition for a parallel action A2. Then the current version of NOAH has the critic described in section 10.2 to resolve the indicated conflict and it suggests the following linearization.



Let us consider the holding period diagrams of the approach which lead to interaction and the suggested linearization. Remember that in a holding period diagram the time at which a goal is achieved is indicated from left to right (see section 4.1).

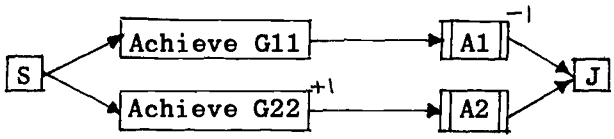
The approach before linearization specifies any of the following linear approaches (where A1 achieves G1 and A2 achieves G2):



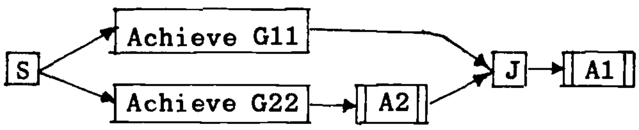
The indicated interaction says that: if G22 is true and A1 (to achieve G1) is applied, G22 will be made false. So any approach which requires that G22 be true while A1 is applied (G1 achieved) is illegal. This should reject approaches (d) and (e) only (i.e. those cases where A1 intersects the holding period of G22).

However, the linearization suggested by the resolve conflicts critic in NOAH specifies the linear approaches (b), (c) and (f). However, as indicated, approach (a) should also be allowed for consideration but is excluded by the linearization suggested. Now, since approach (a) is the simple linear sequence of trying to achieve G1 first and G2 second (the sort of approach attempted first by most problem solvers), we must be wary of excluding the possible use of this approach.

Accepting Sacerdoti's thesis that decisions about ordering choices should be made only as is necessary to remove interactions, the information available in the interaction

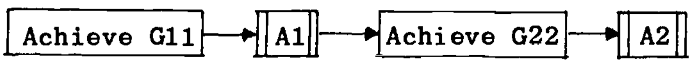


only specifies the ordering constraint that A1 should not be applied after G22 has been achieved and before A2 is applied (G22 is a precondition of A2). This constraint cannot be expressed within a single procedural net diagram by incorporating ordering lines between goals and actions. Thus either a new type of ordering constraint which excluded actions from appearing between some pair of nodes must be allowed or alternatively, 2 or more separate procedural nets should be suggested as appropriate linearizations for the interaction described. In the case above 2 separate nets would suffice, the one already suggested by the critics of NOAH



specifying approaches (b),(c),(f)

and the alternative

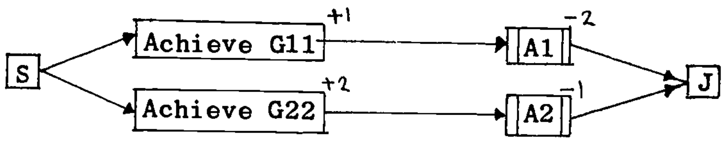


which specifies approach (a).

The addition of more backup possibilities as separate nets to be considered on failure makes the lack of guidance as to a suitable choice after failure (described in section 10.3.1) even more critical.

10.3.3 Double interactions

The problem solving routines in the version of NOAH described in Sacerdoti (1975) are not capable of dealing with problems in which there are "double interactions". The general case is given below:



A typical STRIPS-world problem which fits this case is:

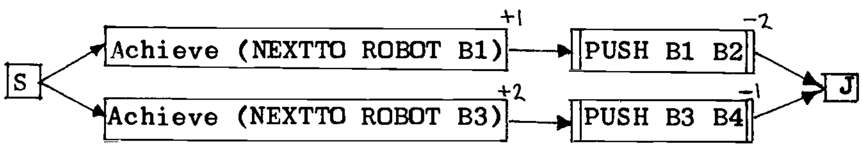
Achieve (AND (NEXTTO B1 B2) (NEXTTO B3 B4))

when the robot is initially not NEXTTO any of B1, B2, B3 or B4.

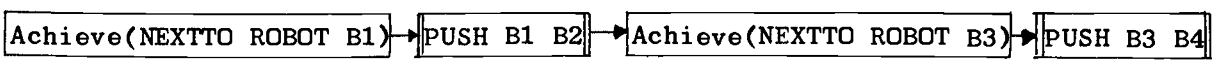
An action (PUSH bx by) exists with definition

```
     PRECONDS   (NEXTTO ROBOT bx)
     DELETE     (NEXTTO ROBOT == ) (NEXTTO bx == )
     ADD        (NEXTTO bx by) (NEXTTO ROBOT by)
```

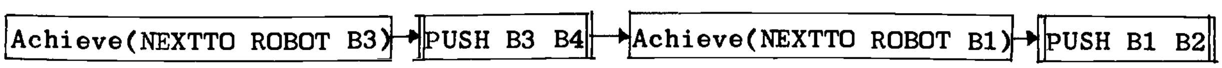
This problem generates at some stage the procedural net:



We can thus see that very straightforward problems fall into this category. It is possible to make 2 simple linearizations which may resolve the conflict.



or



Once again (as in section 10.3.2) the generation of the 2 different nets for consideration may be avoided if some "restriction" ordering was allowed in the net to disallow an action from appearing between 2 nodes. Such a method would be more in line with the procedural net philosophy of only making linearizations as necessary.

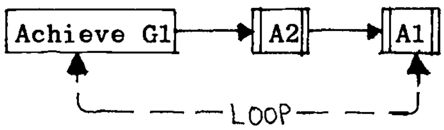
N.B.

In the criticism contained in sections 10.3.2 and 10.3.3 it can be seen that the present NOAH system is not considering those linear approaches most frequently considered first by existing problem solvers.

Sections 10.3.1,10.3.2 and 10.3.3 outline cases under which several nets may have to be generated, only one of which can be considered at any one time. Choice mechanisms between these nets would have to be considered and the use of failure information for this. Also duplication of effort on several similar nets could arise in these cases.

10.3.4 Loop detection and correction

Loops generated in the procedural net are not detected, e.g.



where A1 is inserted to achieve G1.

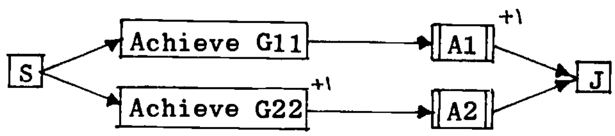
As detailed in section 5.7.7 in the description of INTERPLAN, loop detection can be as important as many forms of interaction in outlining a defect in the approach on some problems. If corrected for it may enable a solution to be found, as for example, in the "Swap the values of 2 Registers" and the "Keys and Boxes" problems. Both these problems would be coped with by other mechanisms in NOAH.

10.3.5 "Formal object" problems

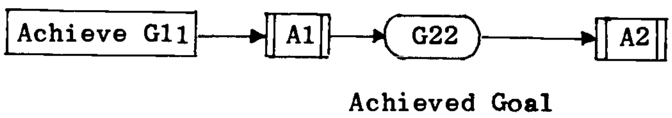
For example, in block stacking if ?OB1 or ?OB2 (see section 10.1 (4)) were set to any of the blocks for which (CL x) was later needed in a plan, problems would occur. There is really an implicit exclusion of any instance causing an interaction from the values any "formal object" may take. Some sort of variable restriction scheme (possibly as outlined in appendix IV) would be necessary to ENSURE that this was done in longer and more difficult problems.

10.4 Beneficial side effects

A precondition for some action may be achieved as a side effect of a parallel action as shown below where A1 achieves some main goal G1 but also achieves G22 (a precondition for A2) as well.



Then we could suggest a linearization to make the achievement of G22 unnecessary as follows:



which is equivalent to

Though, it should be remembered that the other linearizations are not illegal (no interaction prevents them being used) and for some problems explicit achievement of G22 may be necessary.

The table of multiple effects provides the information which would enable a critic to be written to look for beneficial side effects.

11 KEYS AND BOXES PROBLEM SIMULATION

As mentioned earlier in this report, an aim of the work was to discover the reasons why existing problem solvers could not cope with a particular problem, the Keys and Boxes. The work on interacting goals stemmed directly from this investigation. We will now return to this problem to illustrate how it could be represented to INTERPLAN and to simulate the action of the program on the problem. To actually run the problem on the current implementation of INTERPLAN would require, in particular, the matcher to be extended to cope with sets. and the full loop editor to be used (section 5.7.7). The provision of set matching would be tedious and would not aid our understanding of the processes involved. However, to make clear what would actually be required of the matcher, all set matches have been noted in the simulation and are listed in section 11.2.1.

A simplified version of the Keys and Boxes problem which does not require the use of a set matcher is described in Warren (1974). This was successfully run on INTERPLAN.

11.1 Representation of the Keys and Boxes problem to INTERPLAN

This representation closely follows the English statement of the problem given in section 3.1.

11.1.1 Predicates

There are 3 predicates which can be altered by the robot's actions. With the parameter types they take they are:

AT(<set of objects>,<place>)

Note there will be only one AT statement for each place.

ROBOTAT(<place>)

HELD(<set of objects>).

There are 3 global predicates:

RED(<set of objects>)

KEY(<set of objects>)

INROOM(<place>).

"NOTHING" is equivalent to {} the empty set of objects. {...x} represents a particular set of objects, possibly empty, whose individuals are not explicitly known. The value of x distinguishes different such sets, it may be omitted if no distinction is required.

The statement in the Keys and Boxes problem description in section 3.1 which says that in the initial situation there is A and possibly other objects at BOX1 can be represented as << AT {A, ...1} BOX1 >>. A is a particular object and ...1 represents the other elements which may be at BOX1 initially. The other elements, if they exist are treated as unique. We assume a limited set matching facility is available to the system as specified in the following sections.

<<SUBSET x>> can mean the set x itself or else represents a non-empty set of objects from x.

<<UNION x y>> means the set union of sets x and y.

<<SETMINUS x y>> means the set y with the elements of set x removed.

N.B. <<SUBSET x>> <<UNION x y>> and <<SETMINUS x y>> are patterns which are to be matched against others and do not behave as set functions.

11.1.2 Operator schemas

There are 3 actions, LETGO, PICKUP and GOTO(<place>). LETGO and PICKUP are straightforward and each convert to an operator schema as follows:

```
OPSCHEMA LETGO
  ADD <<HELD NOTHING>>
  DELETE <<HELD == >>          "==" matches any item (HBASE).
  PRECONDS
  VARS
ENDSCHEMA
```

```
OPSCHEMA PICKUP
  ADD <<HELD <<SUBSET *$*X>> >>
  DELETE <<HELD == >>
  PRECONDS <<AT <<SUBSET *$*X>> *$*Y>>  <<ROBOTAT *$*Y>>
  VARS X Y
ENDSCHEMA
```

N.B. It is only necessary to have a SUBSET of the set x at the place to be able to hold a SUBSET of x after a PICKUP. This is the weakest precondition which will specify the PICKUP effects and should be used to ensure the PICKUP is useful in as many cases as possible.

The GOTO(<place>) action is a little more involved since it has several conditions in its definition. It therefore expands out to several operator schemas (though ways of withdrawing appropriate operator schemas as needed from a single representation of GOTO(<place>) can be provided - see section 5.8(6)). Following our English statement of the problem we can write:

```
GOTO(y) is defined as follows
  IF y="OUTSIDE"
  THEN precondition is KEY(t) & AT(UNION(SUBSET(t), {...} ), DOOR)
  ELSE precondition is INROOM(y) CLOSE;
  deletes ROBOTAT(z) and adds ROBOTAT(y).
  IF HELD(x);  x/="NOTHING"
  THEN deletes AT(w,z) and deletes AT(v,y)
       adds AT(UNION(x,v),y) and adds AT(SETMINUS(x,w),z)
  CLOSE;
```

Since there are 2 conditionals in this definition, we obtain 4 different operator schemas all with the same name, GOTO(y). To aid the explanation to follow we shall, however, rename them - though this is not necessary for the operation of the program. TAKE(y) will describe the actions in which we do a GOTO(y) with something HELD.

```
OPSCHEMA <<GOTO *$*Y>>
  ADD <<ROBOTAT *$*Y>>
  DELETE <<ROBOTAT == >>
  PRECONDS G <<INROOM *$*Y>> <<HELD NOTHING>>
  VARS Y
ENDSCHEMA

OPSCHEMA <<GOTO OUTSIDE>>
  ADD <<ROBOTAT OUTSIDE>>
  DELETE <<ROBOTAT == >>
  PRECONDS G <<KEY *$*T>> <<AT <<UNION <<SUBSET *$*T>> {...} >> DOOR>>
          <<HELD NOTHING>>
  VARS T
ENDSCHEMA

OPSCHEMA <<TAKE *$*Y>>
  ADD <<ROBOTAT *$*Y>> <<AT <<UNION *$*X *$*V>> *$*Y>>
      <<AT <<SETMINUS *$*X *$*W>> *$*Z>>
  DELETE <<ROBOTAT *$*Z>> <<AT *$*W *$*Z>> <<AT *$*V *$*Y>>
  PRECONDS G <<INROOM *$*Y>> <<AT *$*V *$*Y>> <<HELD *$*X>>
  VARS V W X Y Z
ENDSCHEMA

OPSCHEMA <<TAKE OUTSIDE>>
  ADD <<ROBOTAT OUTSIDE>> <<AT <<UNION <<SUBSET *$*X>> *$*V>> OUTSIDE>>
      <<AT <<SETMINUS *$*X *$*W>> *$*Z>>
  DELETE <<ROBOTAT *$*Z>> <<AT *$*W *$*Z>> <<AT *$*V OUTSIDE>>
  PRECONDS G <<KEY *$*T>> <<AT *$*V OUTSIDE>>
          <<AT <<UNION <<SUBSET *$*T>> {...} >> DOOR>>
          <<HELD *$*X>>
  VARS T V W X Z
ENDSCHEMA
```

The ADD/DELETE lists fully specify the effects of the actions, so OPSCHFNs are not needed.

The changeable predicates (AT, ROBOTAT and HELD) may have a definite order of priority put upon them. This does not always indicate which goals are easier to solve, but gives information about the interactions possible in the domain. If the ROBOT(is)AT a place, we cannot go on to achieve an already untrue AT statement without first deleting the ROBOTAT fact. So, AT must have greater priority than ROBOTAT. Also, if we achieve some HELD goal and require it to be kept true we may not be able to solve some AT goal, but it is usually possible in the other order. Using such domain specific information we can order the predicates by priority thus:

1. AT   2. HELD   3. ROBOTAT.

The ordering can be seen in the operator schemas given earlier. It can be used to disallow reversals of goals of different priorities by setting SCHREVS of each OPSCHEMA appropriately (see appendix I.1). Theoretically, predicates of the same priority can be solved in any order. So, the system accepts whatever order it is given, but is prepared to alter this if an approach fails.

In fact the preconditions for the TAKE operator schemas are insufficient if <<AT <<SETMINUS x w>> z>> is allowed as an achieve request to them. However, the modifiation would be to add two preconditions to them (<<AT *$*W *$*Z>> and <<ROBOTAT *$*Z>>). We will ignore this request knowing that it will not arise in the Keys and Boxes problem

11.1.3 Initial situation and Rules (IFNEEDS)

We assert in the initial sitaution (CUCTXT):

<<AT {A,...1} BOX1>>
<<AT {B,...2} BOX2>>
<<AT {C,...3} DOOR>>
<<AT NOTHING TABLE>>
<<INROOM BOX1>>
<<INROOM BOX2>>
<<INROOM DOOR>>
<<INROOM TABLE>>

N.B. There are no assertions for <<AT x OUTSIDE>> or <<ROBOTAT x>>.

The following rules are available to compute true instances of an achieve request (these would be stored as IFNEEDED methods - McDermott and Sussman, 1972). IFNEEDED methods cannot be given to the data base system used in the current implementation of INTERPLAN.

true ==> <<AT {...} y>> .
i.e., there is a possibly empty set of objects at any place

(<<AT u BOX1>> & <<AT v BOX2>>) in context NOW ==> <<KEY <<EITHEROF u v>> >>.

<<AT u DOOR>> in context NOW ==> <<RED u>>.

<< p x >> ==> <<p <<SUBSET x>> >>. If a set of objects has some property p, then a subset of the set also has the property.

11.1.4 Goal

The goal, following the English statement in section 3.1, can be expressed as:

<<RED x>> & <<AT <<UNION <<SUBSET x>> {...}>> OUTSIDE>>.

11.1.5 ACHIEVES list

```
[% <<HELD NOTHING>>, [% LETGO %],
   <<HELD <<SUBSET == >> >>, [% PICKUP %],
   <<ROBOTAT <:NON OUTSIDE:> >>, [% GOTO(y) %],
   <<ROBOTAT OUTSIDE>>, [% GOTO(OUTSIDE) %],
   <<AT <<UNION == == >> <:NON OUTSIDE:> >>, [% TAKE(y) %],
   <<AT <<UNION == == >> OUTSIDE>>, [% TAKE(OUTSIDE) %] %] -> ACHIEVES;
```

N.B. (a) <:NON OUTSIDE:> is an HBASE actor which will not match OUTSIDE. <:NON OUTSIDE:> instances are put first so that attempts to achieve AT(x,y) where y="==" (i.e., put some objects anywhere) only attempt to put x AT places INROOM, not OUTSIDE.

(b) TAKE(y) and TAKE(OUTSIDE) also achieve ROBOTAT goals. Also in the ACHIEVES list above <<AT <<SETMINUS == == >> == >> achievements are ignored. So, only the important achieve requests with their primary method of achievement are on ACHIEVES (i.e., the primary additions of STRIPS - Fikes, Hart and Nilsson, 1972b).

11.2 The Simulation

11.2.1 Set matching for the Keys and Boxes

A matcher, say MATCH1, is required which behaves thus:

MATCH1 = MATCH (normal HBASE matcher) except in the case where both arguments are sets.

The set matcher must have the following minimal properties to solve the Keys and Boxes problem. Matches are one way only.

i) NOTHING or {} matches only NOTHING or {}.

ii) {...} matches any set.

iii) a set matches another if each element of the set matches each element of the other in some order.

iv) <<SUBSET x>> matches y if x matches y. I.e. <<SUBSET x>> can be equal to the set x itself.

v) ...x only matches ...x. for any number x. I.e. set remainders are considered as unique.

The letters in brackets refer to points in the figures to follow in section 11.2.2 which explain the action of INTERPLAN on the Keys and Boxes problem.

(a) MATCH1(<<UNION <<SUBSET {C,...3}>> {...} >>, {...}) => undefined.

(b) MATCH1({...} , {...}) => true.

(c) MATCH1(<<UNION <<SUBSET <<EITHEROF {A,...1} {B,...2}>> >> {...} >>,
C,...3) => undefined.
or
MATCH1(<<UNION <<SUBSET {A,...1}>>
<<UNION <<SUBSET {B,...2}>> {...} >> >>,
{C,...3}) => undefined.

(d) MATCH1(<<UNION <<SUBSET {B,...2}>> {...} >>, {C,...3}) => undefined.

(e) MATCH1({...} , {C,...3}) => true.

(f) MATCH1(<<SUBSET {B,...2}>>, {}) => undefined.

(g) MATCH1(<<SUBSET {B,...2}>>, {B,...2}) => true.

```
(g') MATCH1(<<SUBSET {A,...1}>>, {A,...1}) => true.

(g") MATCH1(<<SUBSET {C,...3}>>, {C,...3}) => true.

(h)  MATCH1(<<SUBSET {C,...3}>>,
            <<UNION <<SUBSET {A,...1}>>
                     <<UNION <<SUBSET {B,...2}>> {C,...3}>> >>) => undefined.

     MATCH1(<<SUBSET {C,...3}>>, {} ) => undefined.

     MATCH1(<<SUBSET {C,...3}>>,
            <<INTERSECT <<SUBSET {A,...1}>> {A,...1}>>) => undefined.

     MATCH1(<<SUBSET {C,...3}>>,
            <<INTERSECT <<SUBSET {B,...2}>> {B,...2}>>) => undefined.

(i)  MATCH1({}, {}) => true.
```

11.2.2 Simulation

We present the simulation of the action of INTERPLAN on the Keys and Boxes problem by giving a series of 4 "snapshots" of the state of the goal control tree of the system at interesting points on the way to a solution.

STATE 1: Search has proceeded in a straightforward way to this point. To achieve the goal, a red thing must be outside. This can be achieved using a TAKE(OUTSIDE) operator. To take anything outside, a key must be at the door. We can be sure of getting a key outside if we get a subset of the things now at Box1 to the door, and a subset of the things now at Box2 to the door. We plan to take a subset of the things at Box2 to the door first. State 1 is the stage at which operators have been chosen to get a subset of the things at Box2 to the door. Successful backup is about to take place.

STATE 2: The successful backup from state 1 is shown giving entries up to index number 35. We then have a subset of the objects from Box2 at the door. Now a subset of the objects from Box1 must be taken to the door. State 2 shows the goal control tree after this sub-plan is found and after successful backup has taken place. By the time entry 61 is made we have planned to get a key at the door.

STATE 3: Now we have a key at the door, we could achieve our top level goal goal of getting a red thing outside by holding a red thing and executing a TAKE(OUTSIDE). However, in this state we have tried to hold a red thing and have run into a LOOP. Information is available within the goal control tree upon which to suggest a new approach (see section 11.2.3).

STATE 4: The new suggested approach is tried and proves successful. The stage shown is just after planning to remove a red thing from the door to the table for "safe-keeping". When this approach has been fully expanded the optimal plan is generated:

LETGO, GOTO(DOOR), PICKUP, TAKE(TABLE),
LETGO, GOTO(BOX2), PICKUP, TAKE(DOOR),
LETGO, GOTO(BOX1), PICKUP, TAKE(DOOR),
LETGO, GOTO(TABLE), PICKUP, TAKE(OUTSIDE).

SIMULATION STAGE 1:

GOAL

FINISH

	GLOBAL RED x	AT (UNION (SUBSET x) {...}) OUTSIDE
NOW	1 ✓ x = {C, .3}	2 X (a)

TAKE (OUTSIDE) operator chosen by matching against ACHIEVES list entry (AT (UNION == ==) OUTSIDE)

	GLOBAL KEY t	AT {...} OUTSIDE	AT (UNION (SUBSET t) {...}) DOOR	HELD (SUBSET {C, ...3})
NOW	3 ✓	4 ✓ (b)	5 X (c)	

t = (EITHEROF {A, 1}{B, .2})
NOTE 1 (see section 11 2 3)

TAKE (DOOR) operator chosen by matching (AT (UNION (SUBSET {A, 1}) (UNION (SUBSET {B, ..2}) {...})) DOOR) against ACHIEVES list entry (AT (UNION == ==) <: NON OUTSIDE :>)
NOTE 2 (see section 11.2.3)

	AT {. } OUTSIDE	GLOBAL INROOM DOOR	AT (UNION (SUBSET {B, ...2}) {...}) DOOR	HELD (SUBSET {A, ...1})
NOW	6 ✓	7 ✓	8 X (d)	

TAKE (DOOR) operator chosen by matching against ACHIEVES list entry (AT (UNION == ==) <: NON OUTSIDE :>)

	AT {...} OUTSIDE	GLOBAL INROOM DOOR	AT {...} DOOR	HELD (SUBSET {B, ...2})
NOW	9 ✓	10 ✓	11 ✓ (e)	12 X (f)

PICKUP operator chosen by matching against ACHIEVES list entry (HELD (SUBSET ==))

	AT {..} OUTSIDE	AT {..} DOOR	AT (SUBSET {B, ...2}) y	ROBOTAT y
NOW	13 ✓	14 ✓	15 ✓ (g)	16 X

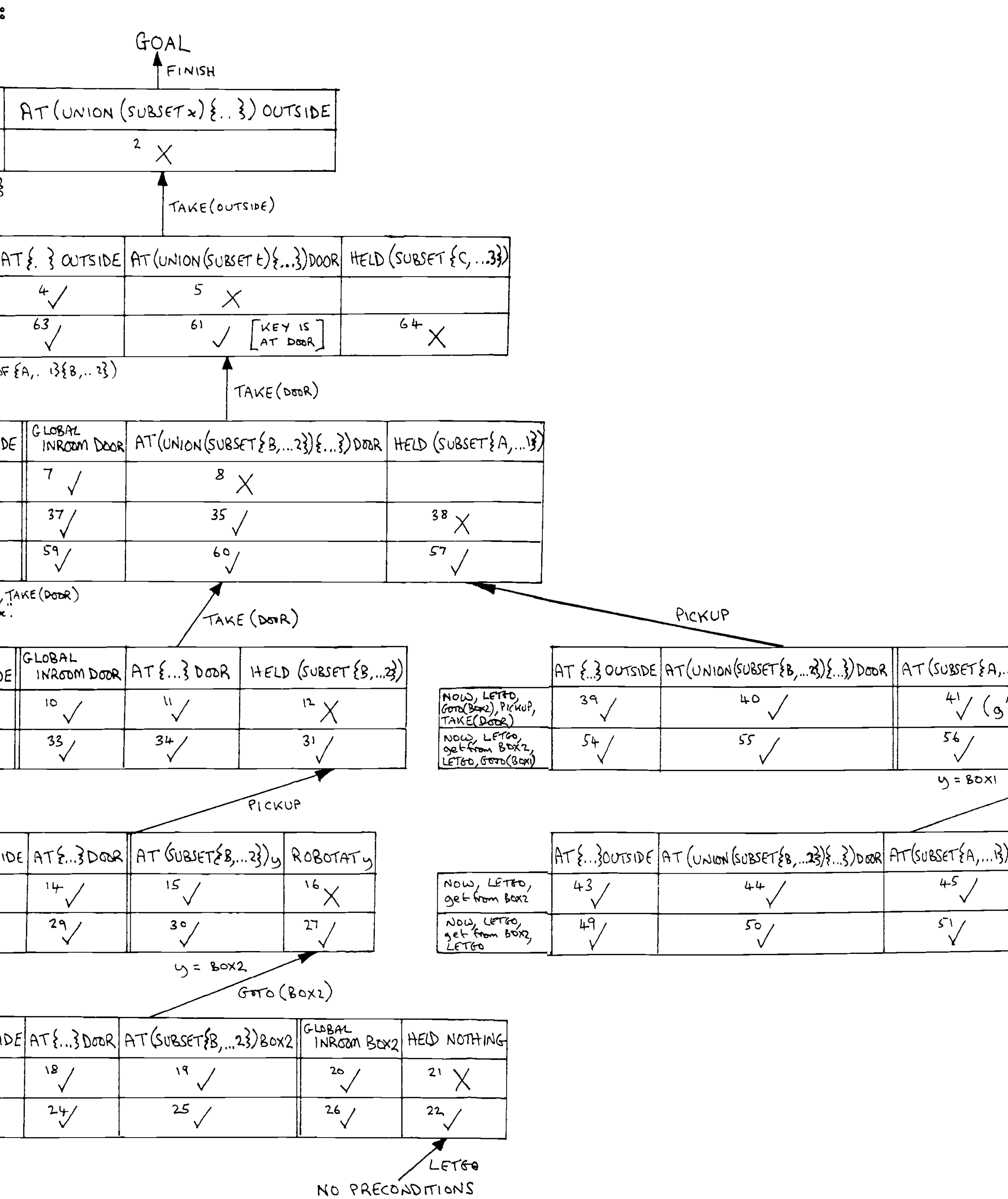
y = BOX2

GOTO (BOX2) operator chosen by matching against ACHIEVES list entry (ROBOTAT ==)

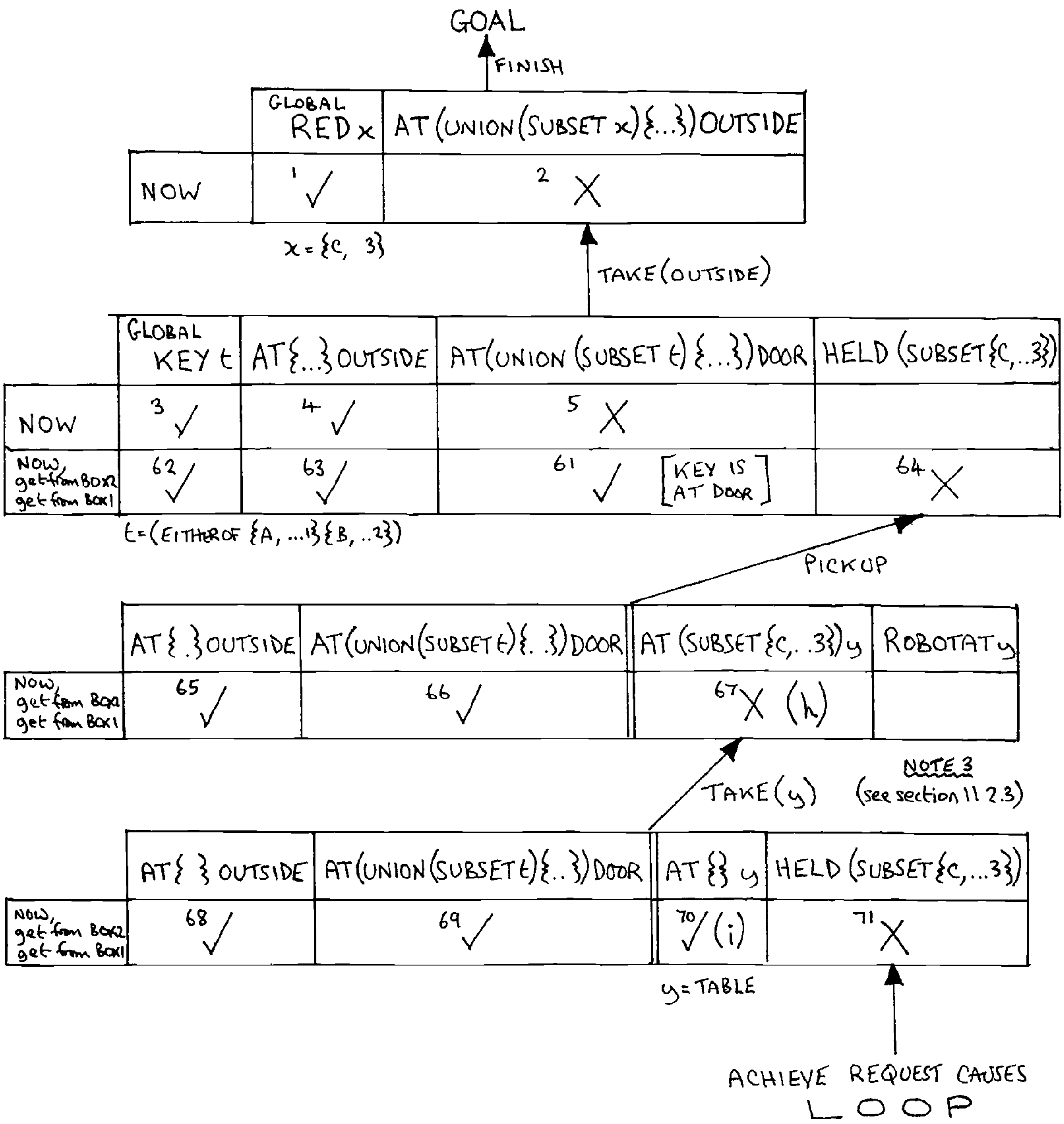
	AT {...} OUTSIDE	AT {...} DOOR	AT (SUBSET {B, ...2}) BOX2	GLOBAL INROOM BOX2	HELD NOTHING
NOW	17 ✓	18 ✓	19 ✓	20 ✓	21 X

LETGO operator chosen by matching against ACHIEVES list entry (HELD NOTHING)
NO PRECONDITIONS

NOW BACKUP OCCURS

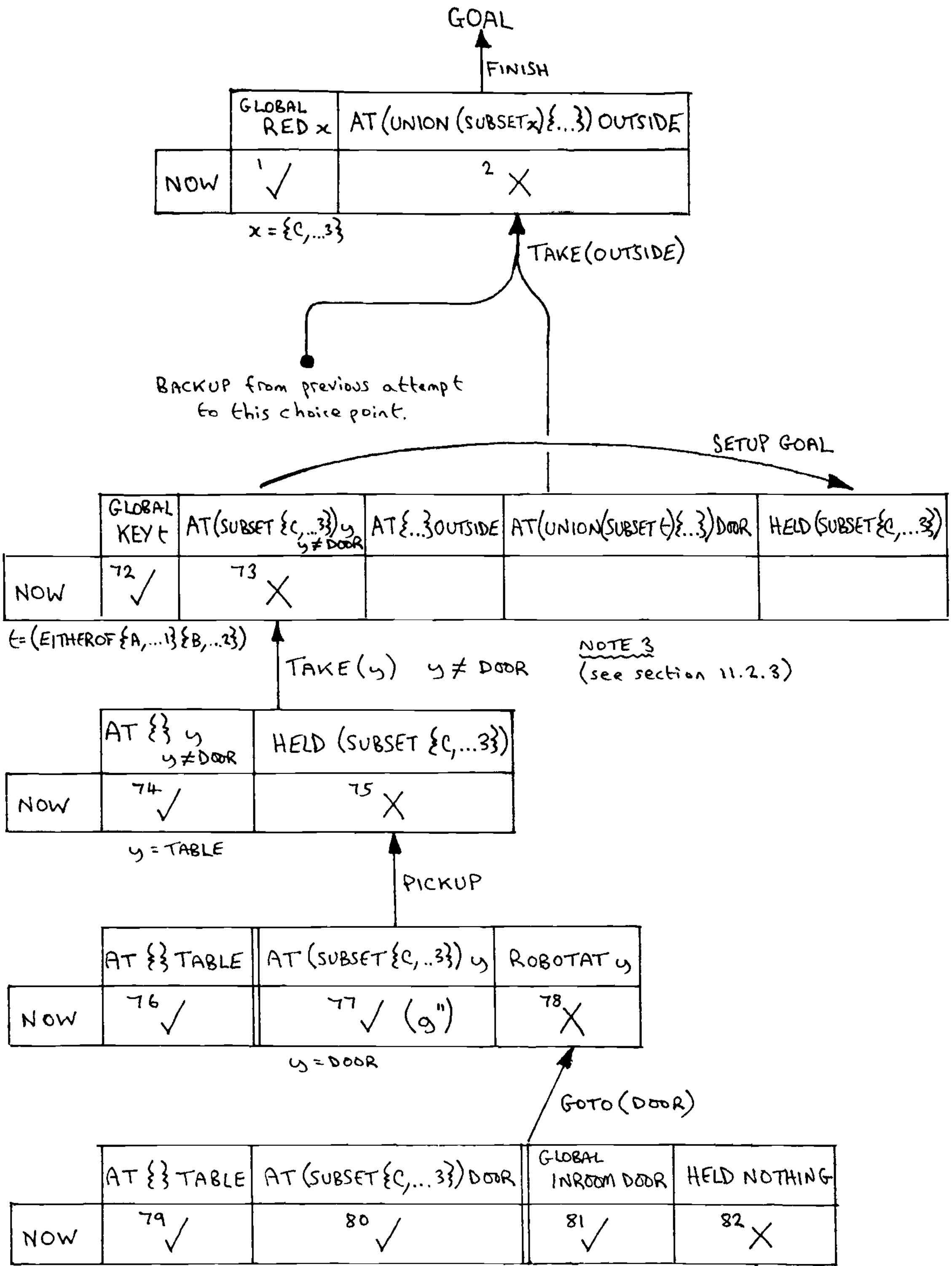

GOAL
FINISH
AT (UNION (SUBSET x) {.. }) OUTSIDE
2 X
TAKE (OUTSIDE)
AT {. } OUTSIDE
AT (UNION (SUBSET t) {...}) DOOR
HELD (SUBSET {C, ...3})
4 ✓
5 X
63 ✓
61 ✓
KEY IS AT DOOR
64 X
F {A,.. 1}{B,.. 2})
TAKE (DOOR)
GLOBAL INROOM DOOR
AT (UNION (SUBSET {B,...2}) {...}) DOOR
HELD (SUBSET {A,...1})
7 ✓
8 X
37 ✓
35 ✓
38 X
59 ✓
60 ✓
57 ✓
, TAKE (DOOR)
TAKE (DOOR)
PICKUP
GLOBAL INROOM DOOR
AT {...} DOOR
HELD (SUBSET {B,...2})
10 ✓
11 ✓
12 X
33 ✓
34 ✓
31 ✓
NOW, LETGO, GOTO(BOX2), PICKUP, TAKE(DOOR)
NOW, LETGO, get from BOX2, LETGO, GOTO(BOX1)
AT {...} OUTSIDE
AT (UNION (SUBSET {B,...2}) {...}) DOOR
AT (SUBSET {A,..
39 ✓
40 ✓
41 ✓ (g
54 ✓
55 ✓
56 ✓
y = BOX1
PICKUP
AT {...} DOOR
AT (SUBSET {B,...2}) y
ROBOTAT y
14 ✓
15 ✓
16 X
29 ✓
30 ✓
27 ✓
NOW, LETGO, get from BOX2
NOW, LETGO, get from BOX2, LETGO
AT {...} OUTSIDE
AT (UNION (SUBSET {B,...2}) {...}) DOOR
AT (SUBSET {A,...1})
43 ✓
44 ✓
45 ✓
49 ✓
50 ✓
51 ✓
y = BOX2
GOTO (BOX2)
AT {...} DOOR
AT (SUBSET {B,...2}) BOX2
GLOBAL INROOM BOX2
HELD NOTHING
18 ✓
19 ✓
20 ✓
21 X
24 ✓
25 ✓
26 ✓
22 ✓
LETGO
NO PRECONDITIONS

SIMULATION STAGE 3:



The LOOP detected editor now acts. It suggests a new approach at the level where we are trying to find a situation in which the preconditions of TAKE(OUTSIDE) are satisfied. See the notes on how the full loop editor works (section 5.7.7). Reversal of goals at the upper loop level is not possible since PRIORITY(AT) > PRIORITY(HELD).

SIMULATION STAGE 4:



BACKUP now takes place. We have succeeded in isolating some red things at the table. The process of filling in the ticklists then proceeds without further interaction to produce the successful plan.

11.2.3 Notes on the simulation

1. <<KEY t>> = <<KEY <<EITHEROF {A,...1} {B,...2}>> >>

   t is expected to be the set of all possible objects which are keys.

2. The question answerer and the operator selector perform appropriate matching and transformation of set descriptions to obtain a match. A special facility must be provided additionally to deal with EITHEROF goals or facts.

   Using: RELATION ON x & RELATION ON y ==> RELATION ON (EITHEROF x y)

   the question answerer should transform

   <<AT <<UNION <<SUBSET <<EITHEROF {A,...1} {B,...2}>> >> {...} >> DOOR>>

   to 2 questions both of which must be true:-

   <<AT <<UNION <<SUBSET {A,...1}>> {...} >> DOOR>>

   <<AT <<UNION <<SUBSET {B,...2}>> {...} >> DOOR>>.

   I.e., If both of above are true, the relation on EITHEROF is also true. But, note that the above is

   <<AT <<UNION <<SUBSET {A,...1}>>

   <<UNION <<SUBSET {B,...2}>> {...} >> >> DOOR>>

   = AT ((SUBSET {A,...1}) U (SUBSET {B,...2}) U {...}) DOOR.

   Also if a relation on an EITHEROF is required to be ACHIEVED, we can try to achieve the relation on both parts, as this is the only way of being sure that the EITHEROF is satisfied if testing of a state of the robot's world is not allowed. So, an achieve request should also be transformed as above.

3. <<AT <<SUBSET {C,...3}>> y>> can only match <<AT <<UNION == == >> == >> in the ACHIEVES list (see section 11.1.5). Since we are trying to ACHIEVE the pattern, we require: <<AT <<UNION <<SUBSET {C,...3}>> {} >> y>>. Only other instance of <<AT <<UNION <<SUBSET {C,...3}>> <<SUBSET {C,...3}>> >> y>> would merely cause a LOOP. This is a general heuristic principle which could be incorporated into the set match routines.

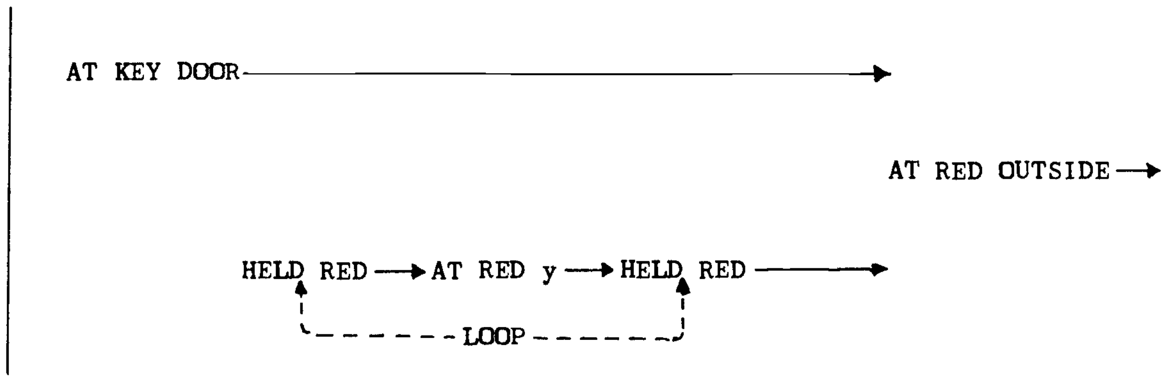
11.3 The approaches used in the Keys and Boxes problem

We can describe the approaches used and tried during the simulation of the Keys and Boxes problem using the "holding period" diagram. I will abbreviate

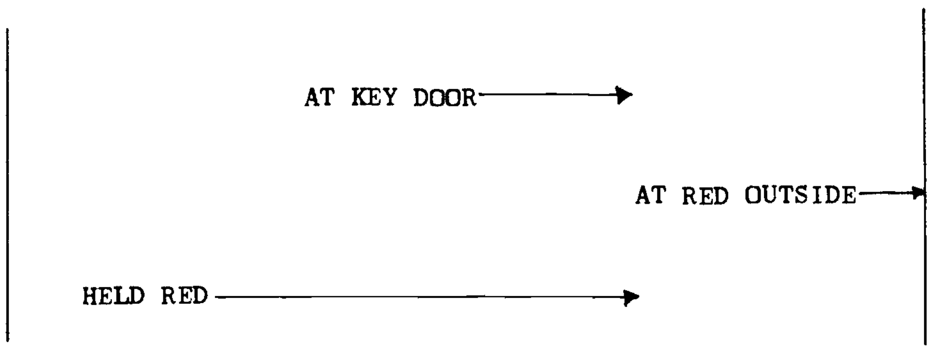
<<UNION <<SUBSET {C,...3}>> {...} >> as "RED" and

<<UNION <<SUBSET {A,...1}>> <<UNION <<SUBSET {B,...2}>> {...}>> >> as "KEY". The approach (the initial approach) being considered when the important LOOP detection occurs is shown below.
Remember that preconditions of an action to achieve a goal are written

PRECOND ⟶ GOAL in the diagrams below.

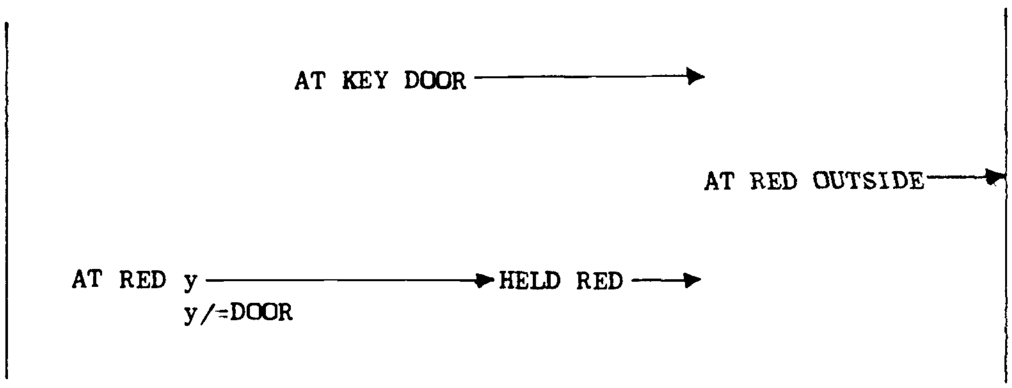


As indicated in the description of the full loop editor (section 5.7.7), we may try to reorder the concurrent goals at the upper loop level (AT KEY DOOR and HELD RED) to avoid the loop. This would give an approach as shown below.



However, if knowledge of the predicate priorities has been incorporated into the operator schemas (as mentioned in section 11.1.2), this ordering would not be allowed as priority(AT) > priority(HELD).

The alternative of suggesting some "setup" goal to aid the solution of HELD RED while avoiding the loop is also tried. The pattern we looped upon (HELD RED) contains no variables and thus, the two loop occurrences of the pattern are IDENTICAL. Using some instance of the loop pattern as a "setup" goal is thus equivalent to the approach with goals at the upper loop level reordered (as shown in the previous diagram). This is fully explained in section 5.7.7. However, the intermediate subgoal between the loop patterns in the approach (AT RED y) could be used as a setup goal if it had no true instance in the initial situation (the point to which it is to be promoted). Since (AT RED DOOR) is true initially, we must restrict y to not be the DOOR to allow its promotion. The approach suggested by this promotion is the one which allows us to go on to solve the problem. It is shown below.



Similarity to the swap the values of 2 registers problem

It is interesting to note the close similarity between the approaches needed to solve the swap the values of 2 registers problem (see section 8.2) and those needed to solve the Keys and Boxes problem.

12 CONCLUSIONS

12.1 Interaction problems

We have described a class of problems in which the solution of individual goals in sequence will not lead to a solution of a conjunct of goals. The Keys and Boxes problem falls into this class, as do other well known problems, such as swapping the values of two computer registers. Such problems have been termed interaction problems. A very simple block stacking problem was used to point out the interaction difficulties encountered by linear problem solvers and to describe our approach to overcome these difficulties.

Several problems which previously have been dealt with using special domain-dependent facts can be tackled in a natural fashion without this information if dealt with as interaction problems. We have shown our system, INTERPLAN, coping in a general way with the problem of swapping the values of two computer registers and with other problems which have been considered anomalous by other problem solvers. These have included the 2-room problem of Siklossy and Dreussi (1973), see section 8.1, and the Shunt problem of Warren (1974), see section 9.5.

12.2 Extending the scope of linear problem solvers

Linear problem solvers which assume that plans to achieve individual goals can be concatenated to solve a conjunct of goals have been studied extensively. For example, in STRIPS (Fikes and Nilsson, 1971), LAWALY (Siklossy and Dreussi, 1973) and HACKER (Sussman, 1973). Such systems often gain their efficiency by being able to restrict the operators which need be considered as relevant because goals which are true in the initial or intermediate situations can be used to restrict the instantiations of the relevant operators.

A process has been described which allows the use of linear problem solving techniques on the class of interaction problems. The process provides a monitoring system which looks out for interactions in the plan being built up in a linear fashion, and provides the ability to make simple corrections if interactions occur to allow linear problem solving to resume. A problem solver which incorporates this process, INTERPLAN, has been programmed and tested.

The provision of an ability to deal with interaction problems by a problem solver has extended the scope of linear means-end analysis driven systems to an inportant class of problems. This ability provides the mechanism which could be used to solve the Keys and Boxes problem (Michie, 1974). We have given a simulation of the action of INTERPLAN on this problem.

12.3 Use of goal structure

The monitoring system which checks for interactions does not consider the individual sequences of actions which comprise the plan, but rather considers the effects these plan sequences have on the goals being achieved.

Initially some order of the top level goals is chosen as an "approach" to the problem. If the conjunct of top level goals can be achieved by the concatenation of operator sequences for the individual goals in the order specified in the approach, the problem is solved normally by the system. However, the monitoring system keeps a check that the approach is being strictly followed. If the chosen operator sequence for some individual goal deletes some previously achieved goal a violation of the approach is reported by the monitor. Corrections are made to the approach which will probably remove the difficulty (for example, by reordering the goals in the approach or the insertion of some necessary intermediate step). An attempt is then made to solve the individual goals by plan sequences in the order specified in the new approach and to concatenate these in that order. Many other legal approaches to the problem are not tried since they are not indicated as useful.

This process can be seen as "debugging" an initial approach suggested to achieve some conjunct of goals to an approach which does in fact allow the achievement of the conjunct. The method used here on declarative data representations (operators represented basically as ADD, DELETE and PRECONDITION lists) has much in common with that used in HACKER (Sussman, 1973) on more procedural representations.

The aim of the INTERPLAN system can be interpreted as finding a successful approach which fully specifies the order in which goals can be achieved by some operator sequence and kept true (without interaction) whilst the other goals are achieved. Such a successful approach provides much information over which learning schemes can be devised.

12.4 Use of Ticklists

The goal control tree of INTERPLAN is of the "backup" type described in the introductory section on robot problem solving (see section 2.5.4). This structure allows the localization of the information about which goals are effected by the operator sequences which are used to achieve some individual goal. This localization led to the use of a straightforward tabular form for keeping track of the interactions between plan sequences to achieve individual goals. This tabular structure is called a "ticklist" since goals which are asserted by some plan sequence are ticked in the table and goals which are deleted are crossed.

It has been found possible to define a set of classifiers which look for certain patterns of ticks and crosses in the ticklist currently being considered and a set of editors each of which is paired with a classifier and which perform the appropriate actions on the tree of ticklists (which is the goal control tree of INTERPLAN). An iterative process of classifying and editing the tree of ticklists can therefore be used to solve a problem.

The tabular format of ticklists and the pattern of ticks and crosses within a ticklist provides a simple means of detecting interactions between subgoals and allows the locality of an interaction to be identified. Compare this with the analysis of the teleological trace of the problem solver's actions necessary to find the cause of an interaction in HACKER (Sussman, 1973). The discovery of an interaction can be constructive in that suitable corrections to the approach being tried by the problem solver can easily be made when definite information is available as to what goals are interacting and how that interact. This is quite different from the procedure in many existing problem solvers which would simply backtrack to other choice points on discovering an interaction, or worse still, fail to detect the interaction at all.

12.5 Comparisons with other systems

During the course of this project two other research workers designed problem solvers which are capable of dealing with interaction problems. The methods employed by these problem solvers, WARPLAN (Warren, 1974) and NOAH (Sacerdoti, 1975), have been compared with INTERPLAN. NOAH is particularily interesting since it is probably the first robot problem solver to use a non-linear approach to solving the components of a conjunct of goals. NOAH uses a table in which the effects of plan sequences on the GOALS being achieved are recorded and this table is used to decide on the action to be taken by the problem solver. This tabular form was based upon a description of ticklists given in an earlier paper (Tate, 1974).

Time comparisons of several problem solvers against INTERPLAN, particularily on problems in the STRIPS robot world and variants of this, show that INTERPLAN performs better even though it can cope with a wider class of problems than most.

INTERPLAN has been written in such a way as to be easily modifiable to allow its use in further problem solving research. In this context it has been used in a study on the usefulness of pre-processing routines on STRIPS-world problems (Davis, 1975).

12.6 Future considerations

12.6.1 A more flexible search strategy

The work presented in this report has concentrated upon the development of a problem solver which can use a means-end analysis (or problem reduction) approach to solving a problem. It was argued in section 2.4 that means-end analysis was useful, and in some problems necessary, when a large number of operators were APPLICABLE to a current situation. However, in some problems there may be a large number of RELEVANT operators, but only a few which are APPLICABLE. Normal forward search procedures would then be most useful. Such an alternative strategy is not open to INTERPLAN and other means-end analysis driven systems. What is required is a problem solver which can exploit the most restrictive kind of search technique at EACH choice point during the search for a problem solution.

Kowalski (1974) describes a means of representing a problem to a theorem prover called "connection graphs". In theory, this representation provides information upon which a decision could be based as to what is the most restrictive operation which can be performed to aid the solution of a problem at each choice point. Investigations would be needed to find techniques to enable the information contained within such a representation of the problem to be used to guide a problem solver's search without the need to fully analyse the potentially very large structure.

12.6.2 Consideration of several goals simultaneously for QA purposes

Consider a question such as (AT ?X ?Y) & (AT ROBOT ?Y). If the two parts were asked separately in the order given when the data base contained

(AT BALL A)
(AT BLOCK B)
.
.
(AT ROBOT B)

we could instantiate such that (AT ?X ?Y) matched (AT BALL A) setting Y=A. Then the question (AT ROBOT A) would be asked and would not be true. It would thus require achievement. If a different instance had been chosen for Y we could have avoided making such an achievement. We would like to obtain matching instances for the WHOLE goal first, and only as a second best, matches for part of the goal. We would need to have the other goals available when asking some question and extend the Question Answerer to take these other goals into consideration when ordering the possibility list of true instances of some individual question. A better method may be to still ask the questions singly, but allow the possibility lists of answers (e.g., above the QA system returns (AT BALL A), (AT BLOCK B), etc. in reply to the question (AT ?X ?Y)) to RESTRICT the values of the variables X and Y as appropriate. Further questions would then contain enough information to enable the QA system to order their possibility lists.

12.6.3 An improved problem solving philosophy

Many interaction problems arise because of the linear way in which most current problem solvers tackle individual goals of a conjunct of goals. The work of Sacerdoti (1975) makes the point that linearization of components of a plan should only be made when interactions actually dictate that they must be made. Sacerdoti demonstrated the usefulness of such an approach on block stacking problems. The question answering strategy outlined in section 12.6.2 is a special case of such a relaxation of the linear problem solving approach.

Linear problem solvers generate a plan which can be represented very simply. This report shows that it is also straightforward to represent the structure of the goals being considered in a linear system, such structure being important to help guide problem solving. However, except in the simplest problems, the same cannot be said of the problem solvers of the type advocated by Sacerdoti. This is because there are many instances when restrictions on legal linearizations of the non-linear plan representation must be made. This cannot be done by simple orderings of actions within the representation (e.g., see section 10.3.2).

Search problems, similar to those which occur in linear systems, arise in non-linear problem solvers because operator choices have to be made and the alternatives must be kept available as backup choices. Decisions must be made as to whether to continue working with the constraints of some particular operator choice or whether to choose another operator. The search problem is particularly acute in non-linear systems because alternative choices can be generated in more cases than for linear problem solvers (e.g., see sections 10.3.1, 10.3.2 and 10.3.3). It would be valuable now to investigate the use of goal structure to direct alternative choices in a problem solver which used a non-linear approach.

APPENDIX I PROGRAM IDENTIFIERS

I.1 The Components of an OPSCHEMA

An OPSCHEMA can be constructed using a function CONOPSCHEMA. The macro OPSCHEMA makes default settings for most components, see example later.

(a) OPSCHNAME — A pattern (possibly with variables local to the OPSCHEMA) which is used as the name of the operator for output.

(b) ADDLIST — A list of patterns (possibly with local variables) which when an operator from this OPSCHEMA can be applied in some situation, can be instantiated from the values of variables local to this OPSCHEMA and asserted (made true) in the successor situation.

(c) DELETELIST — A list of patterns as above which are no longer known to be true in the successor situation All patterns which match a DELETELIST entry are marked as having an undefined truth value.

(d) OPSCHFN — A function to be applied to the successor situation after the additions and deletions have been made. Generally, this may act like the IFADD and IFREM theorems of CONNIVER (McDermott and Sussman, 1972).

(e) PRECONDS — A list of pairs

[<REF NUMBER> . <PATTERN>]

where <REF NUMBER> will usually be a positive integer (see Appendix I.2 (b)). The PRECONDS are joined onto any PROTECTED patterns to become the ticklist heading of ticklists for operators which are instances of this OPSCHEMA. The PRECONDS specify the applicability conditions of the OPSCHEMA.

(f) SCHREVS This is a list of pairs of the reference numbers of preconditions for which reversals should never be attempted. It will generally be left null, but can be used to incorporate heuristic knowledge of a problem domain. For instance, a scheme preventing reversals between groups of goals arranged in a precedence ordering (see Siklossy and Dreussi, 1973) can be implemented using this feature. SCHREVS can be "NOREVERSE" if it is known that no reversals should be attempted.

(g) VARSLIST An association list ("ALIST") which contains all the local variables of this OPSCHEMA. Usually their values will be UNDEF initially,

e.g., [X UNDEF Y UNDEF].

This component is used to initialize the TICKVARS of each ticklist generated from this OPSCHEMA.

(h) MAXREVS Specifies the maximum number of pairwise reversals which can be made for ticklists generated from this OPSCHEMA. A function, NUMREVS(n), is provided to give this number. MAXREVS is used only for computational convenience in checking if all reversals have been tried.

The macro OPSCHEMA

When the macro OPSCHEMA is used, default settings are provided for many components, e.g.,

```
OPSCHEMA    <NAME>      ⎤  maps to  ⎧<NAME>,
  ADD       <A1> <A2>   ⎥           ⎪[% <A1> , <A2> %],
  DELETE    <D1> <D2>   ⎥           ⎪[% <D1> , <D2> %],
                        ⎥           ⎪(lambda; end),        no action OPSCHFN
  PRECONDS  <P1> <P2>   ⎬           ⎨[% [1 . <P1>] [2 . <P2>] %],
                        ⎥           ⎪[],                        null SCHREVS
  VARS X Y              ⎥           ⎪[ X UNDEF Y UNDEF],
                        ⎥           ⎪NUMREVS(2)                      MAXREVS
ENDSCHEMA               ⎦           ⎩.CONOPSCHEMA;
```

If "G" proceeds any precondition, the pattern is given a reference number 0 to indicate it is a GLOBAL precondition with no means of achievement (see Appendix I.2).

I.2 The components of TICKLIST, OP and LEVEL

The components of a TICKLIST (constructor CONSTICK) are:-

(a) TICKARR The actual 2-dimensional array represented as a STRIP of 2 bit elements (initiator INIT2, access doublet SUBSCR2). The entries are initially 0, but can also be a cross (2) or a tick (3). The strip is initially given a length appropriate to 4 rows (i.e. 4*COLMBOUND - see (i) later) but can be expanded as needed.

(b) TICKPATTS Is a list, COLMBOUND long.
Its entries are pairs [<REF NUMBER> . <PATTERN>].
It is accessed using the doublets:

PATTREF(i,ticklist) and PATT(i,ticklist).

<PATTERN> ::= goal pattern which may have variables.

<REF NUMBER> ::= INTEGER >= 1

A goal which must be true when the whole ticklist heading is satisfied.

| 0

A goal for which there are no means of achievement (a global goal). This is provided for efficiency in some problems. It can also be used to indicate that no means of achievement should be used for a goal.

| INTEGER =< -1 but >= -1000

A goal which need only be true until the goal with reference number equivalent to the absolute value of this goals reference number is satisfied. Typically these goals are ones found to be generally required to be true before another harder to achieve goal can be satisfied. These are often called SETUP goals, as they SETUP the facts in some situation to make it easier to solve a later goal.

INTEGER =< -1000

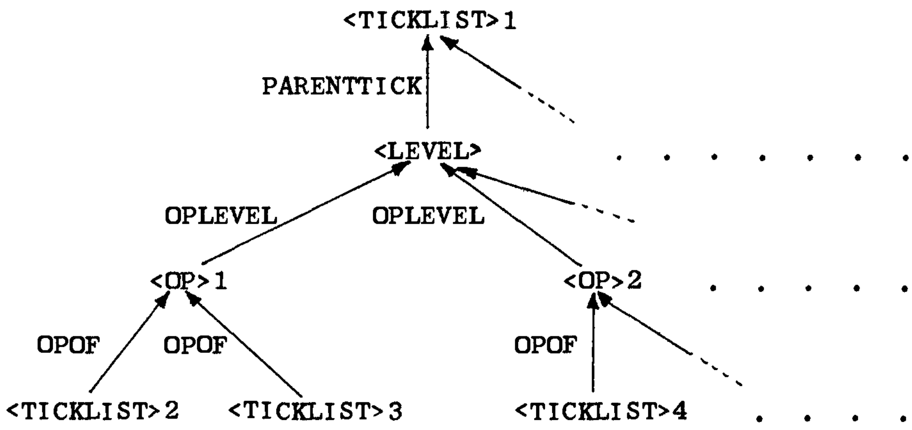
A setup goal as above whose corresponding main goal is already true. -1000 is added to such a setup reference number.

[<TICKLIST . <COLUMN NUMBER>]

A reference number which is a pair indicates that the corresponding pattern is a PROTECTED entry. In the pair, the ticklist is the one at which the PROTECTION was placed and to which any PROTECTION VIOLATIONS should be reported. The column number is the column in which the fact on which PROTECTION was placed is in the ticklist.

(c) TICKSITNS Accessed by the doublet SITN(i,ticklist). It is a list of contexts which represent the headings of each row of the ticklist.

(d) OPOF A pointer to the operator which will be applied to some situation which satisfies the heading of this ticklist. Via OPOF the system can gain access to nodes (ticklists) higher in the search tree. The intermediate data structures between a ticklist and its parent ticklist can be thought of as an arc of the goal control tree of INTERPLAN. There are two such connecting structures which are both always used to specify an arc as shown below.



See later for components of OPs other than OPLEVEL and components of LEVELs other than PARENTTICK.

(e) TICKVARS An association list ("ALIST") of variable names local to the OP being used, with their values (values are UNDEF if not set).

E.g., if X="BOX1" and Y is not set, TICKVARS is

[X BOX1 Y UNDEF].

When a ticklist is created, its TICKVARS is initialized from the VARSLIST of the OPSCHEMA.

(f) TREVS A list of pairs of reference numbers of major goals (ones which initially have reference numbers >= 1) for which column reversals at this ticklist have been attempted. E.g., if there were 3 goals initially with reference numbers 1, 2 and 3 and reversals have been tried between 1 and 2, and between 1 and 3, TREVS would be [[1 . 2] [1 . 3]]. This component is used to check that repeat reversals are not tried. TREVS can also be "NOREVERSE". The system assigns "NOREVERSE" to TREVS when all reversals have been tried. TREVS is initialized from the SCHREVS component of the OPSCHEMA of the OPOF this ticklist. Heuristic knowledge as to what reversals are not useful can be incorporated into the SCHREVS of OPSCHEMAs.

(g) LASTROW The row number corresponding to the context in which we are trying to see if the ticklist heading is satisfied.

(h) LASTCOLM The column number we last made an entry in. It will point to a column with no entry (value of entry=0) if the ticklist has no entries yet.

(i) COLMBOUND The total number of columns in the ticklist heading.

(j) NUMPROTECTEDS The number of columns of the ticklist occupied by PROTECTED entries. For convenience PROTECTED entries are always put in the first NUMPROTECTED columns of the ticklist.

The components of an OP (constructor CONOP) are :-

(a) SCHEMA A pointer to the OPSCHEMA data structure from which this OP is descended (i.e. this OP is an instance of the OPSCHEMA).

(b) OPLEVEL A pointer to the LEVEL data structure (see later) to connect with the parent ticklist as shown in the diagram above.

(c) ACHPATT The pattern (which usually refers to local variables in this OP) which will be used to match against the pattern in the parent ticklist which we are trying to achieve. This match transfers the values of variables between the two ticklists.

(d) INITVARS This is a copy of the ALIST from the appropriate OPSCHEMA after instantiation by matching the pattern we expect to be achieved against the appropriate ADDLIST entry (to set some variables). INITVARS is used to RESET the TICKVARS of ticklists in certain cases if column reversals etc. have been performed and a search for some satisfactory situation is begun again.

The components of a LEVEL (constructor CONSLEVEL) are:-

(a) PARENTTICK A pointer to a ticklist in which some goal is desired to be true (see the previous diagram).

(b) CURRACHIEVES A list used in LOOP detection which holds information on what patterns have been asked to be achieved in what contexts, the entries being notionally grouped into three components:-

1. An instance of the pattern we have asked to be achieved (any unset variables are "==" - see Barrow, 1975).
2. The context we asked for the pattern to be achieved in.
3. The ticklist in which it was found necessary to make this pattern true.

(c) CHOICES Used to hold a list of the different ways to achieve the achieve pattern of the LEVEL. See Appendix III on the Or-choice mechanism.

APPENDIX II THE QUESTION ANSWERER (QA)

The Question Answerer is used to gain access to facts about a particular situation. It is given a pattern and a context, and is expected to find all instances of the pattern which are true in the context. If there are none, it return "cross", if there is a least one it returns "tick".

QA ε <pattern> , <context> => <tick or cross>.

If there is more than one instance

** MULTIPLE INSTANCES is printed out and the system goes into POP-2 READY (interrupt) mode. The instances are in the list POSSLIST which can then be examined or altered before continuing. The first (or only) possibility is matched against the input pattern to cause instantiation of variables. Any other possibilities are kept as choice points in the goal control tree by adding a special node to the CHOICES lists, this holds:

1. the rest of the possibility list (other than the first item),
2. the ticklist the call to QA was made for, and
3. the input pattern (to be used to instantiate variables when the other possibilities are used).

The instances of a given pattern are found using a function

FETCHALL ε <pattern> => <possibility list of instances of patterns>

This is simply defined at present to find all patterns in the context

CUCTXT which have VALUE true, using APPITEMS (see HBASE - Barrow, 1975).

The deduction of facts which may be true in some context is not at present allowed in the QA module. Simple extensions have been experimented with to provide this facility by the use of a restricted type of IFNEEDED theorem as provided in CONNIVER (McDermott and Sussman, 1972). But, in the present implementation of INTERPLAN, the incorporation of rules such as

AT(x,y) & ON(z,x) ==> AT(z,y) is not possible.

APPENDIX III OR-CHOICES

The mechanisms provided within the classifier/editor framework describing INTERPLAN are intended to cope intelligently with the generation of a solution to a problem composed of a conjunct of goals. When the planner is confronted with a choice of several ways to proceed to achieve a goal pattern, it uses the information it is given (e.g., the given ordering of different operator schemas which can be used to achieve a given request) to make a reasonable first choice, then proceeds. The alternative choices (OR-CHOICES) must be stored in some way which will enable them to be chosen if the first choices are poor. The mechanism presently used in INTERPLAN will be described here.

Or-choices occur when there are several ways in which a goal pattern can be made true. These occur mainly when:

(a) there are several true instances of a goal, or,

(b) there are several different operator schemas which can be be used to achieve instances of the goal.

Other or-choices can occur if INTERPLAN, in discovering some goal interaction, has suggested alternative approaches to the main problem (the original conjunct of goals) or to subproblems of it.

The basic way in which or-choices are ordered is that when interactions occur, an alternative way to proceed is taken from the or-choice point which was most recently used. That is, we use depth-first backtracking to find an alternative way to proceed. Alternative choices are taken from the immediate vicinity of some interaction discovered in the goal control tree.

We could just use a list, like a backtrack trail, in which all choices were added to the front of the list when they were generated, and alternative choices could be made by removing the first choice in the list. However, INTERPLAN generates some choices (e.g., alternative choices to avoid a protection violation) which are alternative ways to proceed at different points in the search tree to the point at which an interaction occurred. If these were merely added onto the front of a choices list, they would be chosen at inappropriate times.

We therefore keep or-choices with the points in the goal control tree at which they are intended to be used. The "LEVEL" data structure (see Appendix I.2) provides the point to which or-choices can be anchored. When an interaction occurs, a failure causes a choice to be made from the appropriate alternatives at this LEVEL. When success reaches some choice point, the untried choices are not forgotten, but are released to a global list of untried choice points (called CHOICES(TOPLEVEL)).

Ordering schemes may be used to order choices at any choice point including the global CHOICES(TOPLEVEL) list. Each choice is inserted into the appropriate choice list by comparing a heuristic value it may have with others on the list. The lists are ordered so that lower values are considered "better" and are earlier in the lists. Choices are made from the head of the appropriate list. Whenever a choice is made from the global CHOICES(TOPLEVEL) list "GLOBAL CHOICE USED" is printed. This signifies that a choice has had to be made which may not be immediately relevant to the interactions which have just occurred - there being no choices left in this position. The ordering scheme can easily be altered by setting parameters but is arranged at present to prefer in order:

(a) alternative operators to achieve a goal,

(b) suggested re-orderings of goals (new approach),

(c) suggested promotion of a precondition (new approach), then

(d) alternative instantiation choice for a goal with variables.

If a first choice of an instance of a goal which is true in some context proves to be of no value, we have no cause to believe that merely substituting alternative instantiations will work (e.g., if it did not work with BOX1, why should it work with BOX2 - BOX99 ?). Different operators or approaches suggested in the light of interactions provide a more definite way to reconsider the problem Therefore choices of type (d) need not be chosen immediately at the point at which interactions occur. We therefore put alternative instantiation choices (type (d)) immediately on the global CHOICES(TOPLEVEL) list. Once again this scheme can easily be altered by a change of parameter.

Or-Choice Control Parameters

(a) There are parameters which give the heuristic values of different choice types. These are used for inserting the choices into the list held in the CHOICES of the appropriate level, or in the global CHOICES(TOPLEVEL) list if this is indicated.

type	(a)	OPCHOICE	default is 10
	(b)	REVCHOICE	11
	(c)	EXTCHOICE	12
	(d)	INSTCHOICE	20

A parameter CHOICELEVEL (default is 15) can be set to give the value below which choices are routed to the CHOICES list of the appropriate LEVEL, and above which are routed to the global CHOICES(TOPLEVEL) list.

(b) An additional choice point type may be generated when the switch COMPLETE is set to true. These are choices which indicate attempts to achieve instances of a goal which has some true instance in the context required. They have a parameter giving their heuristic value:

type (e) COMPCHOICE default is 50.

Thus with CHOICELEVEL as given they are routed immediately to the global CHOICES(TOPLEVEL) list.

APPENDIX IV ACTOR RESTRICTIONS ON VARIABLES

As mentioned in "restrictions on instances of a promoted goal" (section 5.7.5) and "The loop classifier and editor" (section 5.7.7) it is sometimes necessary to give a precondition or goal which, though it contains variables, has certain restrictions on the instances these variables can take. It was mentioned in the sections indicated how this could be done if actor restrictions on variables were allowed. A scheme has been tested which allows this process.

Normally, when a value is being matched against a variable used in INTERPLAN (i.e., a variable prefixed by *$*), this is done using a function

QAGIVEN(s,x) where s is the value being matched, and
x is a variable name.

(a) the value of x is found in the appropriate TICKVARS(TICKLIST).
IF the value=UNDEF THEN the variable has no value. So s can be assigned to x and the match succeeds.
ELSE we match the present value of x against s.

(b) Within the outer call of the MATCH function, any variable set (i.e., match made against some variable with value UNDEF) are remembered on a list SETVARS. If the match fails at top level, these variables are reset to their UNDEFined values.

We could modify this process to provide actor restrictions on variables thus:

(a) the value of x is found in the appropriate TICKVARS(TICKLIST).
IF the value is an actor AND the actor matches s (note)
THEN s can be assigned to the value of x and the match succeeds
ELSE proceed as before.

(b) Since some variables when they are first set may have values which were not UNDEF (i.e., ACTORS), we must save not only the variables set as before in SETVARS, but also the values they had before being set. If the outer level MATCH fails, variables are reset to their UNDEFined or actor values as appropriate.

Useful additional facility

It is useful to allow the initial value of an OPSCHEMA's VARSLIST to be set with actor values as well as UNDEF. For example, if a precondition was ON(x,y) & CL(x) where y/=FLOOR we could restrict y to not be the FLOOR in the initial VARSLIST. The macro OPSCHEMA can easily be modified to allow optional actor values to be given to variables initially.

Note: an actor is a facility provided in HBASE (Barrow, 1975) and is a function which can be run on any item to determine if it matches the item.

ACKNOWLEDGEMENTS

In the course of 3 years study many people have contributed towards the theme and content of this thesis. It would be impractical to name all those people here but I would like to offer my thanks to all who have taken an interest in the work.

The work has been financially supported by a Research Studentship from the Science Research Council of Great Britain. Professor Donald Michie supervised the work and offered encouragement when it was needed.

The work has developed through discussions with Harlyn Baker, Chris Brown, John Darlington, Alistair Duncan, Bill Scherlis, Jerry Schwarz, Rodney Topor and Dave Warren.

I would particularily like to express my thanks to Harry Barrow and Earl Sacerdoti, both at the Stanford Research Institute. Robert Ross of the Machine Intelligence Research Unit at Edinburgh read through a draft of the thesis and has managed to remove most of the expressions which betray my Yorkshire up-bringing, but a few may have managed to squeeze through his scrutiny.

REFERENCES

Bobrow, D.G., and Raphael, B. (1974) New Programming Languages for Artificial Intelligence. Computing Surveys Vol.6 No.3, Sept. 1974.

Barrow, H.G. (1975) HBASE POP-2 library documentation. Edinburgh: Dept. of Artificial Intelligence, Univ. of Edinburgh. also a LISP version is documented in: HBASE: A fast, clean and efficient database. Draft report from Stanford Research Institute AI Center.

Boyer, R.S and Moore, J.S. (1972) Proving theorems about LISP functions DCL memo no. 60. Dept of Computational Logic, Univ of Edinburgh.

Burstall, R.M., Collins, J.S. and Popplestone, R.J. (1971) Programming in POP-2. Edinburgh: Edinburgh University Press.

Davis, M. (1975) On constructing a pre-processor for STRIPS-world problem solvers. Research Memorandum MIP-R-111. Edinburgh: Machine Intelligence Research Unit, University of Edinburgh.

Doran, J. and Michie, D. (1966) Experiments with the Graph Traverser program. Proc. Roy. Soc., (A), 294, pp 235-259.

Ernst, G.W. and Newell, E. (1969) GPS: A case study in generality and problem-solving. New York: Academic Press.

Fikes, R.E., Hart, P.E. and Nilsson, N.J. (1972a) Some new directions in robot problem-solving. In Machine Intelligence 7, (eds. Meltzer, B. and Michie, D.) pp 413-438. Edinburgh: Edinburgh University Press.

Fikes, R.E., Hart, P.E. and Nilsson, N.J. (1972b) Learning and executing generalised robot plans. Artificial Intelligence, 3, pp 251-288.

Fikes, R.E. and Nilsson, N.J. (1971) STRIPS: a new approach to the application of theorem proving to problem-solving. Artificial Intelligence, 2, pp 189-208.

Green, C.C. (1969) Application of theorem proving to problem solving. Advance papers of IJCAI1, pp 219-240. Washington DC, USA.

Hayes, P.J. (1973) Structuring of robot plans by successive refinement and decision dependency. M.Phil. Thesis, Univ. of Edinburgh.

Kowalski, R. (1974) Logic for problem solving. DCL memo 75. Edinburgh: Dept. of Computational Logic, Univ. of Edinburgh.

McDermott, D.V. and Sussman, G.J. (1972) The CONNIVER reference manual. MIT AI Memo No.259.

Michie, D. (1974) On Machine Intelligence pp 149-151. Edinburgh: Edinburgh University Press.

Michie, D. and Ross, R. (1969) Experiments with the adaptive Graph Traverser. In Machine Intelligence 5, (eds. Meltzer, B. and Michie, D.), pp 301-318. Edinburgh: Edinburgh Univ. Press.

Newell, A. and Simon, H.A. (1972) Human Problem Solving pp 808. New Jersey: Prentice Hall Inc.

Nilsson, N.J. (1971) Problem solving methods in Artificial Intelligence. New York: McGraw-Hill.

Sacerdoti, E.D. (1974) Planning in a hierarchy of abstraction spaces. In Artificial Intelligence, 5, pp 115-135.

Sacerdoti, E.D. (1975) The nonlinear nature of plans. SRI AI Group Technical Note 101.

Siklossy, L. and Dreussi, J. (1973) An efficient robot planner which generates its own procedures. In Advance Papers of IJCAI3, Stanford, USA, pp 423-430.

Sussman, G.J. (1973) A computational model of skill aquisition. MIT Technical Report AI TR-297.

Tate, A. (1974) INTERPLAN: a plan generation system which can deal with interactions between goals. Research memorandum MIP-R-109. Edinburgh: Machine Intelligence Research Unit, University of Edinburgh.

Waldinger, R. (1975) Achieving several goals simultaneously. Stanford Research Institute AI Center Technical Note 107. May, 1975.

Warren, D.H.D. (1974) Warplan: a system for generating plans. DCL Memo No. 76. Edinburgh: Dept. of Computational Logic, University of Edinburgh.