MACHINE INTELLIGENCE RESEARCH UNIT
UNIVERSITY OF EDINBURGH

Subject: INTERPLAN A plan generation system which can deal with interactions between goals.

Author: Austin Tate

Memorandum: MIP-R-109
Date: December 1974

ABSTRACT

Although it is often natural to specify a task to a problem solver as a conjunction of goals to be attained, the sequential attainment of these goals is often complicated by interactions between them. The interaction problem is discussed and a process is described which allows the use of simple problem-solving techniques which assume interactions do not occur. Such techniques have received a great deal of attention, for example, in the STRIPS system (SRI). We show that when interactions do occur, information which enables problem solving to continue can easily be extracted from the existing data structures. A problem solver, INTERPLAN, has been designed and programmed which incorporates the process.

INTERPLAN tries to find a sequence of individual goals, in the order they will be considered, which will solve a task without interaction. It does this by "debugging" a given initial sequence (usually the given order of the goals in a conjunct). This process is similar to that used in the HACKER system (MIT) for a more procedural representation of a problem.

TABLE OF CONTENTS

1. INTRODUCTION

This report describes progress that has been made on a postgraduate research project at the Department of Machine Intelligence, University of Edinburgh. Michie (1974) describes a problem, the Keys and Boxes, whose solution poses difficulties for many current problem-solving techniques (the difficulties will be discussed in section 3). The aim of the project is the investigation of problem-solving techniques by considering their ability, or otherwise, to cope with the Keys and Boxes problem and the class of problems this belongs to.

The work to be described here has been under study for the period September 1973 - June 1974 and has focussed upon one particular difficulty posed by the Keys and Boxes problem, that of interactions between methods of achieving subgoals (this will be fully described later). A planning problem solver has been designed and programmed to cope, seemingly efficiently, with the difficulty.

In this report I will first describe the Keys and Boxes problem and explain why it does pose difficulties. It is not necessary for the reader to attempt to gain a full understanding of the Keys and Boxes problem in order to follow the paper. A simpler problem (the 3-block problem) which highlights the interaction problem will then be described. The problem solver, INTERPLAN, which can cope with goal interactions will then be fully specified. This part of the report is also intended to serve as documentation to the program. Further problems, currently under investigation using INTERPLAN as the research tool, will finally be mentioned.

2. THE KEYS AND BOXES PROBLEM

The problem was devised by Michie (1974) as a bench mark 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 in figure 2.1

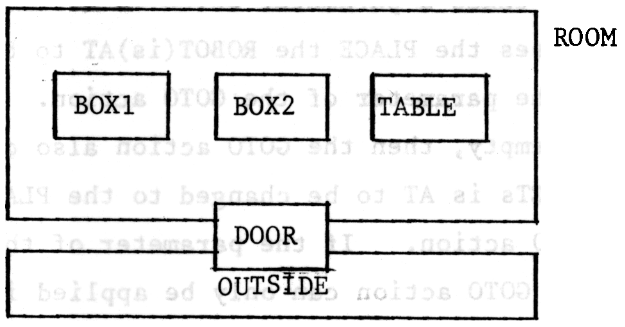


FIGURE 2.1

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 use on the problem).

English Statement of the Keys & 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. Any OBJECT is AT only one PLACE. ROBOT(is)AT only one PLACE. A set of OBJECTs is HELD. NOTHING is equivalent to the empty set of OBJECTs. If a set of items all have some property, then any individual or non-empty subset of the items 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 changes by executing the actions named LETGO, PICKUP and GOTO.

The/

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 then the PICKUP action causes the set of OBJECTs HELD to be changed to a non-empty sub-set 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. WHAT ARE THE DIFFICULTIES

3.1 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 difficulties 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 paper (they are more fully described in section 4).

3.2 We don't 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.

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 2) 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 at place BOX1" as (AT OB1 BOX1) & (AT OB2 BOX2) cannot reflect what is required if an unknown selection of these is removed (by a PICKUP).

4. INTERACTING GOALS - THE LIMITATIONS OF SOME EXISTING PROBLEM SOLVERS

Means-End Analysis

I will assume familiarity with search techniques as used in Artificial Intelligence (see Nilsson, 1971). Problems can be formulated as a search task:

GIVEN:	an initial state representation
	a number of actions (operators) which transform one state into another if applicability conditions are met
	a definition of a desired (goal) state
FIND:	a sequence of actions which will transform the initial state into a desired state.

When there are a significant number of possible actions (or operators), forward search from the initial state (i.e., trying to apply applicable operators to generate a goal state), relies heavily on an evaluation function containing problem-dependent information to guide the search along fruitful paths. A more uniform search method, called means-end analysis, was used by GPS (Ernst and Newell, 1969). There is good evidence that means-end analysis is extensively used during human problem solving (Newell & Simon, 1972).

Means-end analysis has been used in several robot planning systems, e.g., STRIPS (Fikes and Nilsson, 1971), LAWALY, (Siklóssy and Dreussi, 1973) and HACKER (Sussman, 1973). Such systems find what 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 it is applied to produce a new situation. The system then once again compares the desired situation with the newly produced one to find any remaining difference. However, a chosen operator may not be applicable in the given situation. In this case the difference between the applicability conditions (often called preconditions) and the given situation is taken and the means-end analysis driven system again selects from operators relevant to reducing this difference. Once its preconditions have been met, an operator can be applied. Such a process can recurse to any depth if operators/

operators are chosen which are not applicable in the given state. Search is not ruled out by any means 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.

4.2 Interacting Goals

A problem is given to a means-end analysis based problem solver, such as STRIPS (Fikes & Nilsson, 1971) and the planning part of the HACKER (Sussman, 1973) system, 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 in figure 4.1

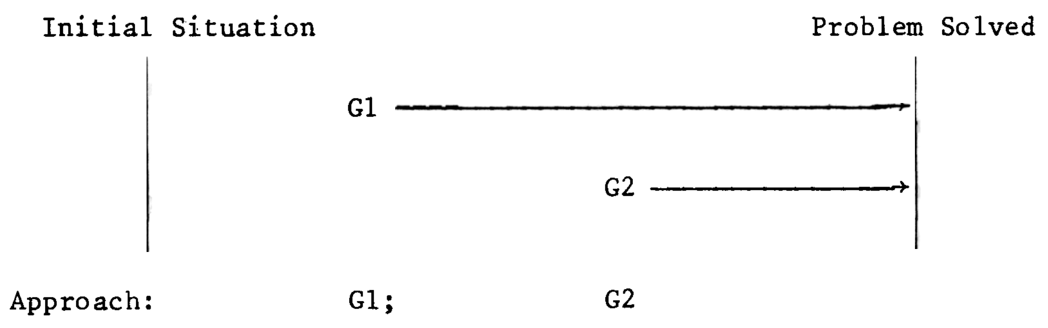


FIGURE 4.1

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 figure 4.1, it assumes G1 can be solved first by some relevant plan sequence and then that G2 can be solved/

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 in figure 4.2).

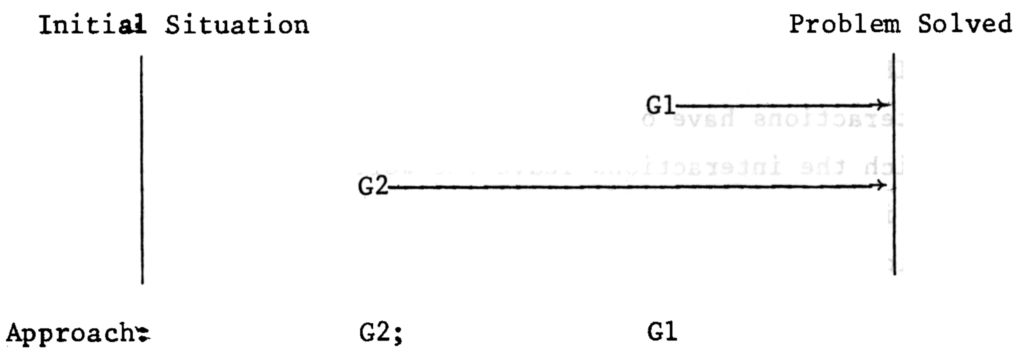
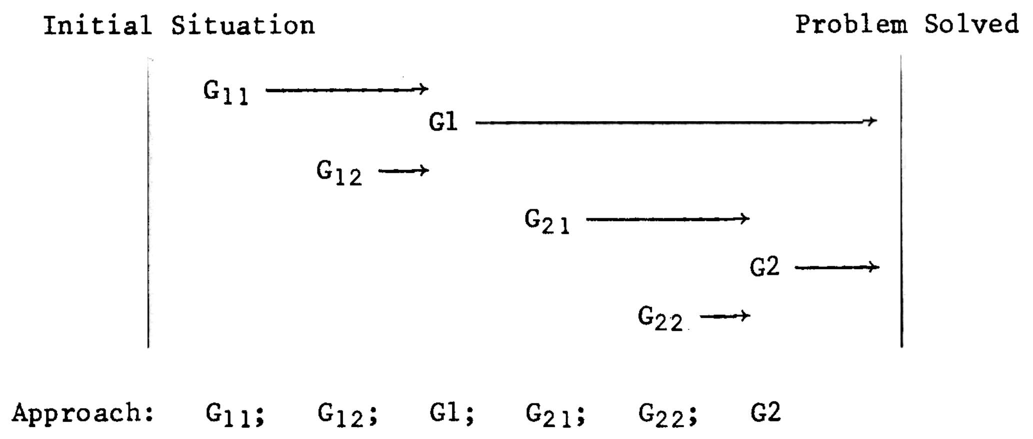


FIGURE 4.2

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 used 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 a goal G_i are G_{i1} and G_{i2}, then STRIPS initially assumes an approach as in figure 4.3 can be taken. Note that the holding period diagram represents the goals to be worked upon for <u>some</u> chosen operator sequence. There is really a 3rd dimension to the diagram representing different choices

FIGURE 4.3



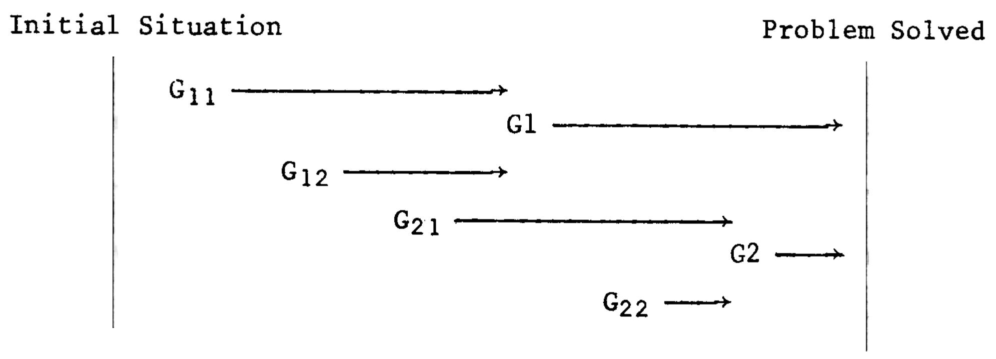
of operators.

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, interactions 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 the subgoals needed to achieve those goals. Thus, a holding period diagram and approach as shown in figure 4.4 is necessary to resolve some types of interaction.

FIGURE 4.4



Approach: G_{11}; G_{12}; G_{21}; G1; G_{22}; G2

5. 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 singles out 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.
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) and repeat if $\exists$v.ON (v,u) etc. (This operator therefore clears all blocks from the top of block x). It can always be applied.

Given an initial situation ON(C,A) & CL(C) & CL(B) as shown in figure 5.1(a) a goal of ON(A,B) & ON(B,C) is given as shown in figure 5.1(b)

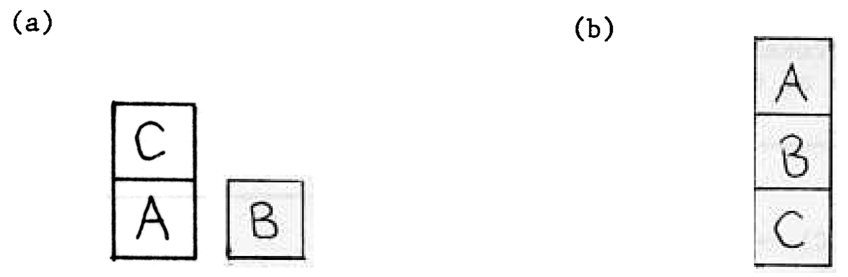


FIGURE 5.1

STRIPS can tackle (ON(A,B) & ON(B,C)) both of which are not true initially. The goals may, at first, be attempted as shown in the holding period diagram of figure 5.2

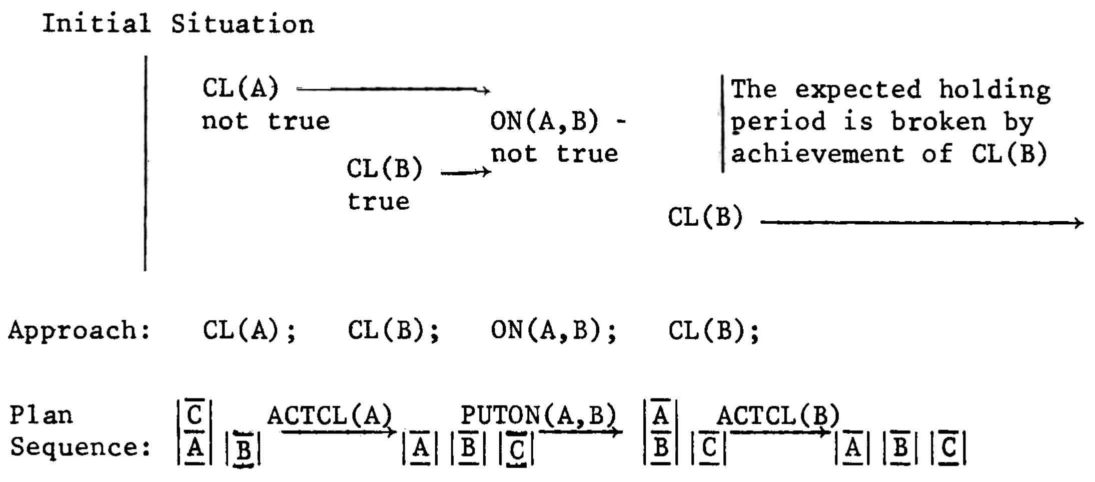


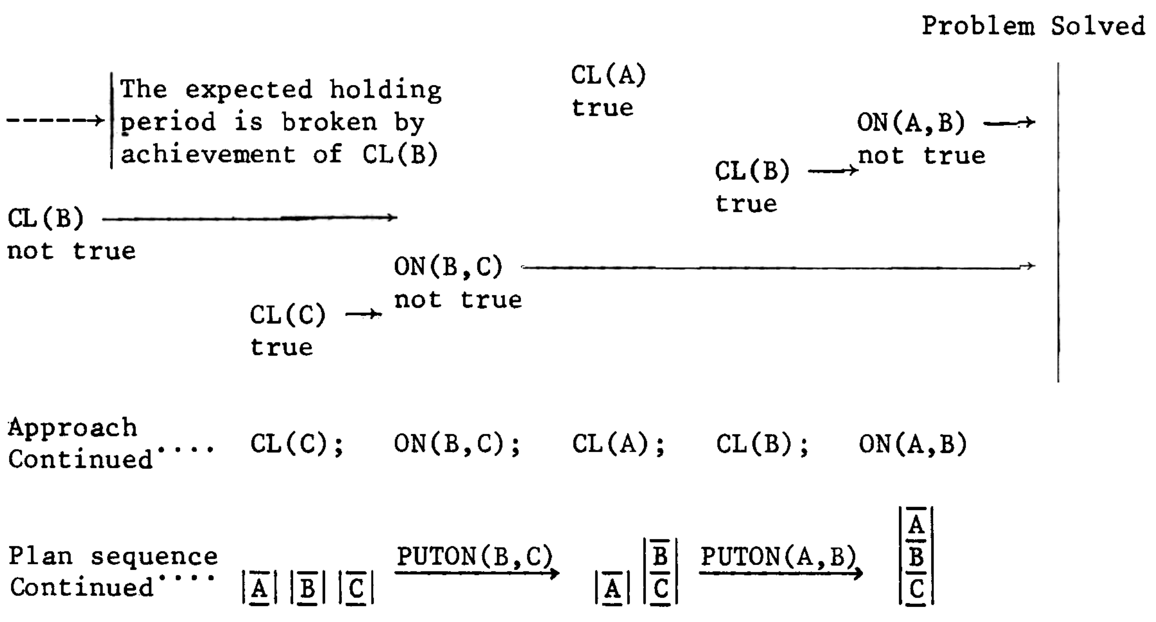
FIGURE 5.2

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 in figure 5.3. 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 longer still, though if the interactions in the first ordering produced a state of the world in which the goals could subsequently not be achieved, this would be attempted on failure backup. STRIPS is incapable of producing a shorter plan.

FIGURE 5.3


Problem Solved
The expected holding period is broken by achievement of CL(B)
CL(A) true
ON(A,B) not true
CL(B) true
CL(B) not true
ON(B,C) not true
CL(C) true
Approach Continued.... CL(C); ON(B,C); CL(A); CL(B); ON(A,B)
Plan sequence Continued....
A B C
PUTON(B,C)
A B C
PUTON(A,B)
A B C

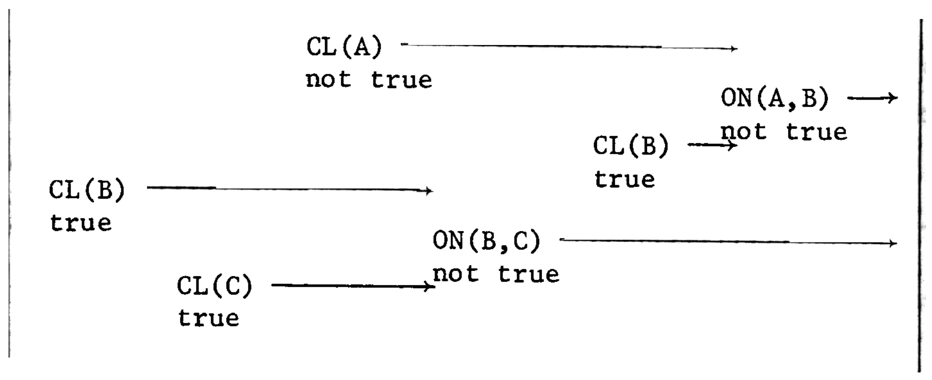
HACKER has a mechanism, called Protection, which remembers achieved goals and looks out for actions which violate them. It would notice that the previously achieved goal (ON(A,B)) ceased to hold (as a Protection Violation) and would try to reverse the order of the top level goals (to ON(B,C)&ON(A,B)) 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 STRIPS in this example.

The search space should have included an approach as shown in figure 5.4. 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 path:-

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

Initial Situation

Problem Solved



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

Plan Sequence:

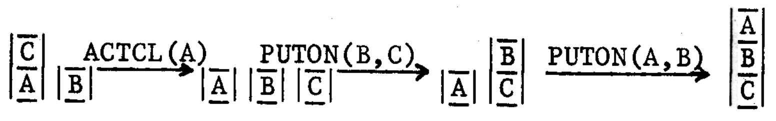


FIGURE 5.4

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

6. USING GOAL INTERACTIONS TO SUGGEST NEW APPROACHES TO TACKLING A PROBLEM

Current means-end analysis problem solvers are not capable of solving problems which have certain kinds of goal interaction, and (with the exception of some systems at MIT e.g. HACKER) do not use interactions amongst goals to guide the search for a solution. I mentioned earlier that all interleavings of goals, and the subgoals needed to achieve those goals should have the potential of being considered. Generally, only very few of the possible interleavings need be considered. An assumption that goals can be achieved in the order given without interaction (linearily) is, however, 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 type of problems I can deal with. Proven contradictions of these assumptions during problem solving can direct the search to consider appropriate interleavings of plan parts to remove interactions.

As an example, the interactions discovered during attempts to solve the goals G1 & G2 linearily lead us to the situation, in figure 6.1, where the expected holding period for G1 is broken by the

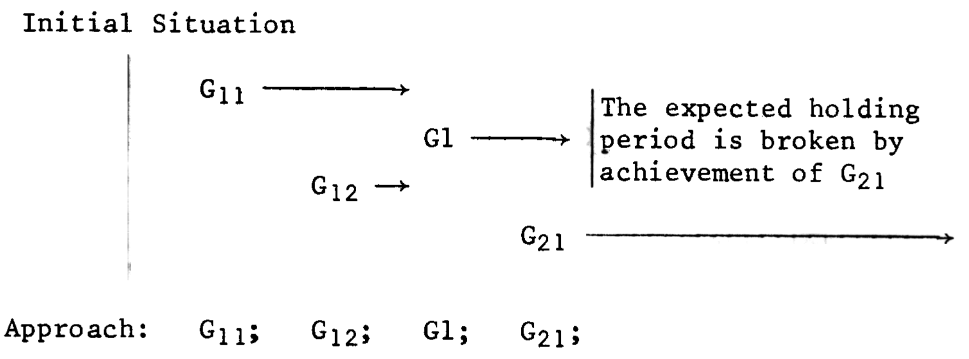

Initial Situation
G_{11}
G_{12}
G1
The expected holding period is broken by achievement of G_{21}
G_{21}
Approach: G_{11}; G_{12}; G1; G_{21};

FIGURE 6.1

achievement of a subgoal G_{21} required for an action to achieve G2. We have tried and found that G1 and G_{21} cannot both hold together when they have been achieved by some operator sequences in the order G1 and then G_{21}. We can either try to reverse goals at a higher level to stop the conflicting goal's holding periods overlapping altogether (by reversing G1 and G2) or try to achieve them in the opposite order.

It is sufficient to try to achieve the conflicting goals in the other order only once, and this can be done whilst still preserving linearity as far as possible by moving the precondition (G_{21}) whose achievement made a previously achieved goal (Gl) not hold, immediately in front of the goal as shown in figure 6.2 (we shall say that we PROMOTE the precondition). Moving it further back through the goals to be worked

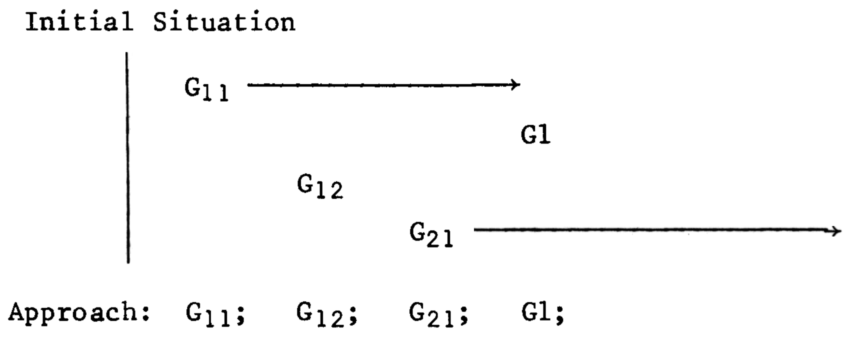


FIGURE 6.2

upon would still try to achieve the conflicting goals in the opposite order but would risk further possibilities for other intermediate goals to interact with the precondition being brought forward. Note that the precondition brought forward (G_{21}) may interact with earlier goals and may need to be shifted again due to different interactions. Subgoals intermediate between G2 and G_{21} if they exist may need to be promoted also.

If in both orders the same goals achieved by suitable operator sequences still interact and cannot hold together, the problem cannot be solved by this approach.

The details of the way in which information from such a goal interaction is extracted and used to suggest possible 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.

7. INTERPLAN: THE PLAN GENERATOR

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 and a goal situation, and giving the 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 a problem. It takes a suggested "approach" to tackling the problem (usually the given order of a conjunct of individual goals) and tries to produce an operator sequence to satisfy the goals which exactly follows the approach. By this, I mean that the goals are achieved in the order specified in the approach and can be kept true while other individual goals are achieved. If, in pursuing the given approach, a difficulty is discovered, alternative approaches are suggested by INTERPLAN which are based upon information gathered from the nature of the difficulty itself. INTERPLAN tries to solve a problem by showing one such approach is valid. If the initial approach is valid, INTERPLAN will merely try to find and check appropriate operator sequences to 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 end-on-end.

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

in the absence of other ordering information, the given order of goals is a reasonable first ordering 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).

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

operators need not be considered to achieve instances of a goal with/

with variables which already has some true instance (see section 10.3 for cases in which this assumption may need to be relaxed).

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 proven to be of possible use to do so (e.g. on Protection Violation discoveries), then, "setup' goals can be inserted into the Approach.

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 (imagined as a graph whose nodes are situations and whose arcs are operator applications) from question answering about a particular situation. INTERPLAN is an operational POP-2 (Burstall, Collins & Popplestone, 1971) program. The HBASE (Barrow, 1974) data base system is used to store different 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 section) 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 section is also intended to serve as documentation of the INTERPLAN program.

2 Specification of a Problem

The plan generation system is given a task by specifying:

An initial situation (state) 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 context, CUCTXT, which will be taken as the initial situation by INTERPLAN.

Descriptions of the actions which can transform situations. Basically these are specified similarly to the STRIPS operator schemas (whose/

(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. They also specify (as Preconditions) 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 as a 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 & Sussman, 1972). So, effects difficult to express assertionally or requiring testing of the situation itself can be caused, but 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 DELETE list entries,

iii) adding all ADD list entries,

iv) running the operators OPSCHFN.

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

A macro 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 X 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 S1

OPSCHEMA/
```

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

There are further effects of these operators as specified in section 5. 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 figure 8.1 for a listing of these functions.

Call 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.

For example, for block stacking

```
{%  <<CL  ==  >>,  {%  S1  %},
    <<ON  ==  ==  >>,  {%  S2  %}  %} → ACHIEVES;
```

That is, the user should take each item in an add list, replace all variables by == (a pattern which matches "anything" in HBASE), and group the corresponding operator with any others which can add the same pattern. This list could be generated automatically. All ADD list entries for all operators need not be put in ACHIEVES. The "primary additions" of STRIPS can then be modelled (see Fikes, Hart and Nilsson, 1972b).

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

For example, here

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

Variables in goal specifications are allowed

7.3 Ticklists

The basic data structure used by the system is a "ticklist". See APPENDIX I.II for its components. It forms the nodes of the search tree which the system explores. Basically, it 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 search tree for the goal above would consist of 2 columns headed <<ON A B>> and <<ON B C>>. I will refer to the set of goals which the ticklist columns represent as the TICKLIST HEADING. Columns of the array represent situations in which we hope to find that all the goals are true (hold). We thus start with a ticklist with the main goals as the ticklist heading and the initial situation as a row heading (as figure 7.1).

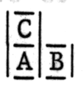
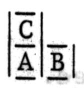
	<<ON A B>>	<<ON B C>>
Initial Situation 		

FIGURE 7.1

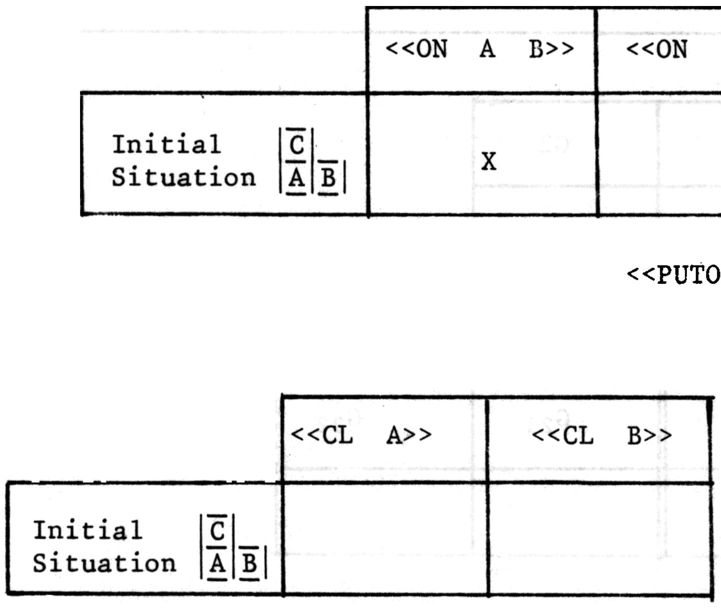
We scan the last row of the ticklist (here only one) and ask if the goal heading each column of the ticklist is true in the context of the last row. We put a tick (✓) if it is, or a cross (X) if it isn't. If the whole conjunction of goals is true in the situation we get a complete row of ticks and have thus found a goal situation. If, however, any column has a cross then this goal remains to be achieved in some situation, as initially with our problem (figure 7.2).

FIGURE 7.2

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

7.4 Ticklist Levels - The Search 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 search tree is grown by making new ticklists on a lower LEVEL to the one 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, in our case as figure 7.3.



<<PUTON A B>> is only relevant operator. It is derived from the operator schema <<PUTON $_*\$_*$X $_*\$_*$Y>>. Branching would occur if more operators were relevant.

FIGURE 7.3

All ticklists at the tips of the search tree being constructed are suitable for further filling in, etc. So 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 program. The choice list is a list of pairs of a heuristic value and a pointer to a ticklist on the tip of the search tree (though 2 special entries are allowed - see sections 7.6.2 and 7.6.3). The choice list is ordered so that pairs with a lower heuristic value are nearer the head of the list and are considered to be "better" choices.

ADDCHOICE ε <heuristic value>, <pointer to ticklist> => ( );
splices/

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 in this pair, the one for classification next. It deals with the special forms allowed in the choice list.

7.5 Protection

When a goal has to be achieved, after other goals have previously been achieved, there is a mechanism for ensuring that the previously achieved goals are kept true. We PROTECT the previously achieved goals by adding them to the ticklist heading of all levels of the search tree which are grown below the ticklist where the goals have been achieved. This is represented diagrammatically in figure 7.4.

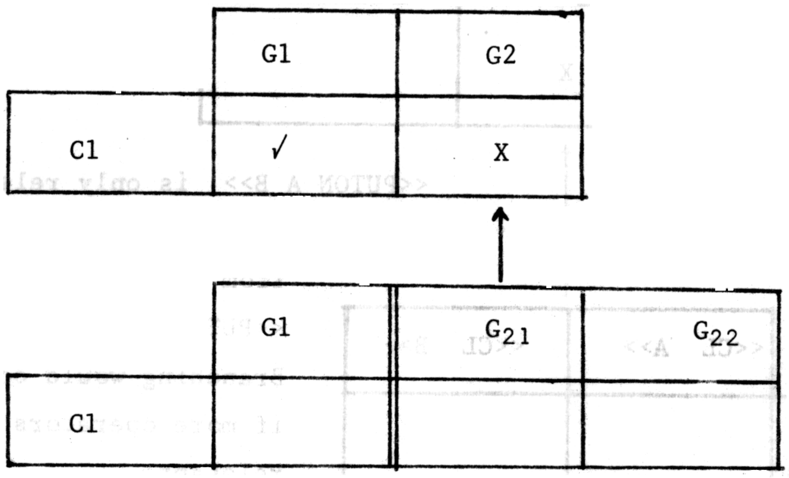


FIGURE 7.4

Then the protected goals must be true simultaneously with all the goals (preconditions for some operator) in a ticklist in some situation for that situation to be one in which the operator is then applicable (relative to 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 generally will look for any VIOLATION of the protection of a fact (PROTECTION VIOLATION). This is an implementation of a feature in the HACKER planning system (Sussman, 1973).

7.6 Classifiers and Editors

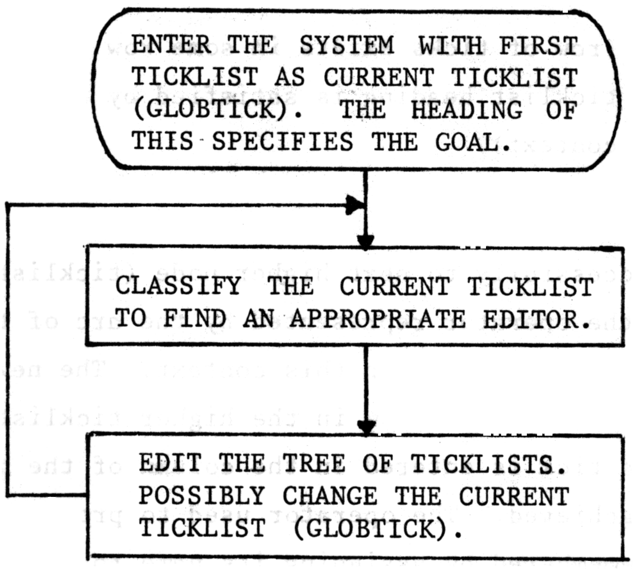

ENTER THE SYSTEM WITH FIRST TICKLIST AS CURRENT TICKLIST (GLOBTICK). THE HEADING OF THIS SPECIFIES THE GOAL.
CLASSIFY THE CURRENT TICKLIST TO FIND AN APPROPRIATE EDITOR.
EDIT THE TREE OF TICKLISTS. POSSIBLY CHANGE THE CURRENT TICKLIST (GLOBTICK).

FIGURE 7.5

The basic loop of the planning system is shown in figure 7.5. The system is thus specified as pairs of a classifier for a ticklist and an editor for the tree of ticklists. See Appendix I for information available within a ticklist and within the tree of ticklists for use by classifiers and editors.

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

7.6.1

Classifier: (ALLTICKS)

A complete row of ticks exists in some row (or more generally, the ticklist heading is satisfied by some row representing a context).

Editor: (SUCCBACKUP)

Backup successfully to next higher node (ticklist) in tree, applying the operator represented by the arc of the tree which is now applicable in this context. The new context 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 a new context is remembered by assigning its name to the VALUE (HBASE Barrow, 1974) of the item "SITN" in the new context.

For example see figure 7.6

FIGURE 7.6

	P1	P2	P3
C1	X		
C2	√	X	

$\xrightarrow[\text{gives}]{\text{after editing}}$

	P1	P2	P3
C1	X		
C2	√	X	
C4		√	

OP_x

	P1	P_{x1}	P_{x2}
C2	√	X	
G3	√	√	√

OP_x applied to C3 gives state C4.

7.6.2

Classifier: A cross appears for some column in the last row of a ticklist (and see section 7.6.4 classifier for ticks later in row too)

Editor: (ACHIEVELAST)

Operators which could add the pattern represented by the column to the world model in some context are sought for. This is the recursive use of the means-end analysis technique

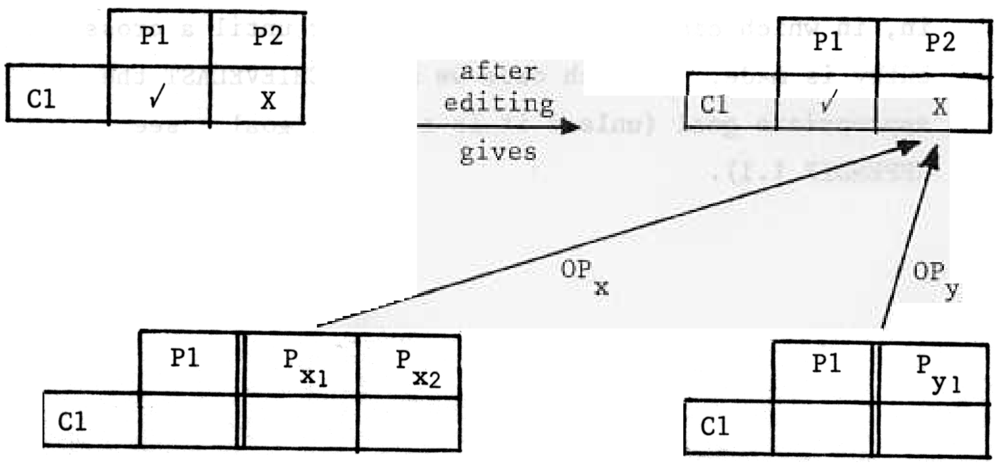
The editor finds all operators relevant (i.e., those can add the sought-for pattern to the context). A function

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

is applied for each relevant operator when found. This normally just erases the second argument to return the <opschema> unmodified but can be used to change the order of preconditions etc.

The editor adds new choicepoints to the search tree corresponding to new successor nodes for each relevant operator. The successor nodes are initialized when chosen from the CHOICEPTS list, where they are kept in compact form, but notionally they exist after this editor has been applied. See section 7.5 on Protection for explanation of symbols used in the example in figure 7.7 below (especially why the P1 protected goal is brought down through levels of the search tree).

FIGURE 7.7



if OP_x and OP_y are only relevant operators.

7.6.3

Classifier:	A tick appears in the last column of the last row of a ticklist and some other entry on the row does not have an entry.
Editor:	(FILLIN)

Scan from left to right along the last row and for any position not filled in, ask the question answerer

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

whether the pattern heading the position is true in the context of the last row. See APPENDIX II for details of QA. A call to QA may instantiate some variables local to the ticklist. If QA finds that a pattern is true in the given context with more than one instance

```
e.g. <<TYPE *$*X BOX>> matches <<TYPE B1 BOX>>
                               <<TYPE B2 BOX>>
                               <<TYPE B3 BOX>>
```

the system asks the user if he would like to pre-order the choices (POSSLIST), then proceeds as above with the first choice, 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). These choices other than the first are demoted on the choices list.

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

7.6.4

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 in the ticklist heading. Checks are first made to see if the columns have been swopped about previously or are now not allowed to be swopped (looking at TREVS of the ticklist for the reference numbers of the patterns see APPENDIX I.II) or to see if no more reversals are allowed for this ticklist (TREVS is "NOREVERSE"). For example see figure 7.8.

FIGURE 7.8

	P1	P2	P3
C1	X		
C2	✓	✓	X
C3	X		✓

after editing $\longrightarrow$ gives

	P3	P1	P2
C1			

7.6.5

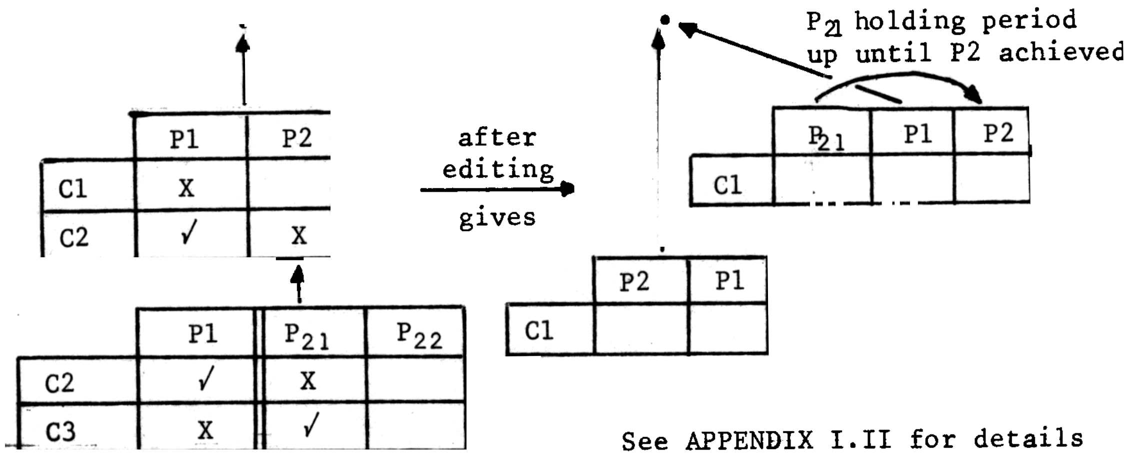
Classifier: A cross in a protected column of some row is followed by a tick in a later column.

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 6). Before promoting an entry 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 entry 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 tree which this subgoal was for. If the same applies to this we try higher still, unless the protection level itself is reached, in which case no promotion is made. For example see figure 7.9.

If some promotion can be made, and goals higher in the search 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.

FIGURE 7.9



See APPENDIX I.II for details of how a goal with a restrictec holding period is represented to INTERPLAN.

7.6.6

Classifier: A cross appears in some column for which there are no means to achieve the relevant pattern (or no further means if some have been tried).

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 goals achievement may have rendered the goal on which we failed insoluble (e.g., by wrong choice of a variable instantiation), in reverse order they may both be solvable. The variables of the ticklist are reset using INITVARS (see APPENDIX I.II).

If the reversal cannot be made with any other pattern earlier in the ticklist heading (e.g., reversal already done in opposite order or this is 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).

7.7 The Approach-Successful Ticklist Headings

The ticklist heading specifies the "Approach" (the sequence chosen to attempt to achieve 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 re-orderings of the original goals or the suggestion of certain "set-up" 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 the INTERPLAN system can be interpreted as discovering 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 suggested to achieve a goal (i.e., the goal ordering itself) to one which will in fact achieve the goal. Bugs are detected by looking at the patterns of ticks and crosses in a ticklist and alterations (edits) to the search tree of ticklists 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.

8. HOW INTERPLAN SOLVES THE 3-BLOCK PROBLEM

The 3-block problem was described in section 5, and was used to illustrate the problem specification to INTERPLAN in section 7.2. A listing of the problem specification is given in figure 8.1 to bring this information together. OPSCHFN functions are included. The purpose of the functions CLFN and ONFN is specified in section 7.2(b). No special snytax is provided for their construction in the present program. They use HBASE primitives, e.g. GETITEM, INSTACT, VALUE and

FIGURE 8.1

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

FUNCTION CLFN; VARS B1 B2;
  INSTACT(*$*X) -> B1;
  COMMENT 'GET INSTANCE OF X LOCAL TO OPSCHEMA S1
  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>> ;
```

the actors ET and NON (see Barrow, 1974). 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 shown in figure 8.2.

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


ENTERING PLANNING SYSTEM WITH INITIAL STATE 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 >>
```

FIGURE 8.2

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 at this stage is as below. Please note that the individual ticklists expand downwards (rows) only as needed. The index numbers indicate the order in which tick and cross entries are made. See figure 8.3.

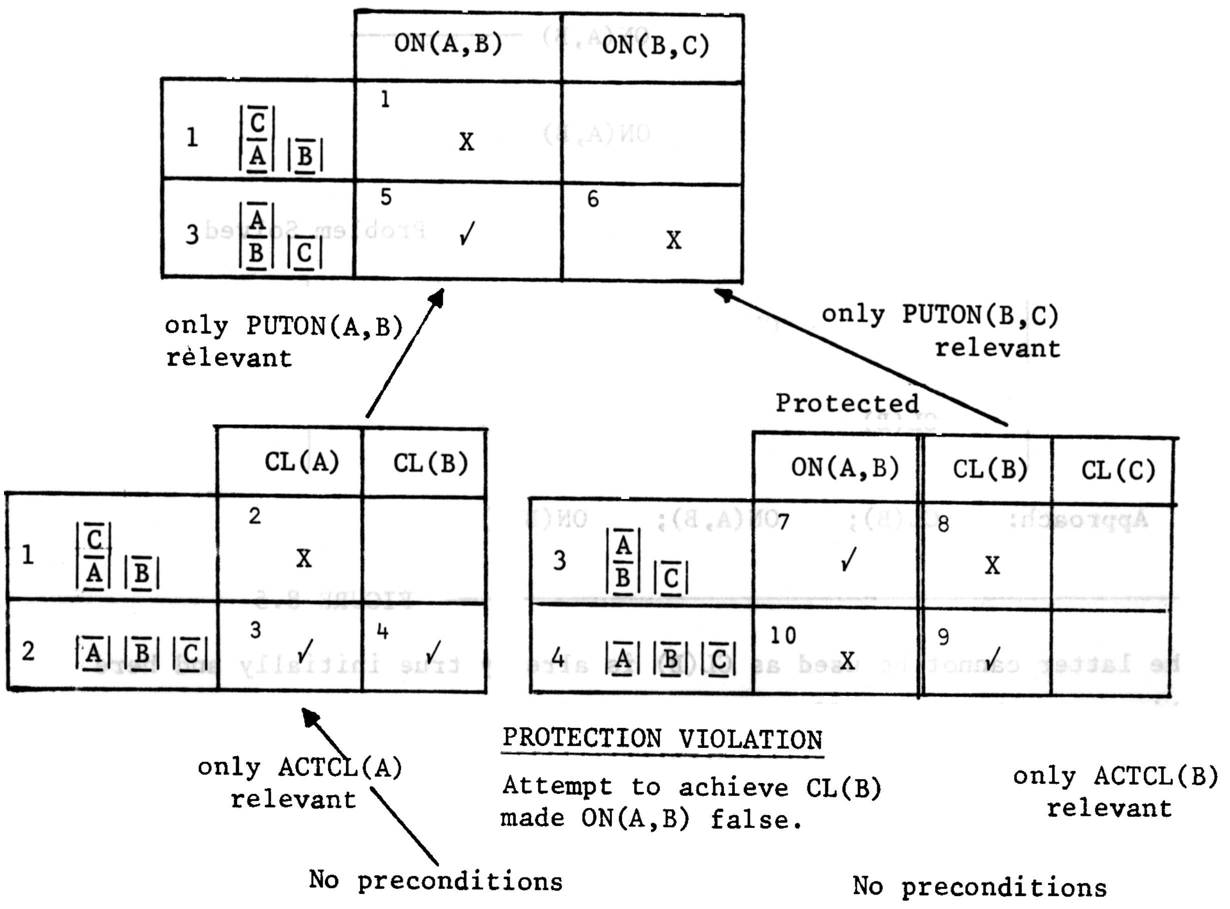
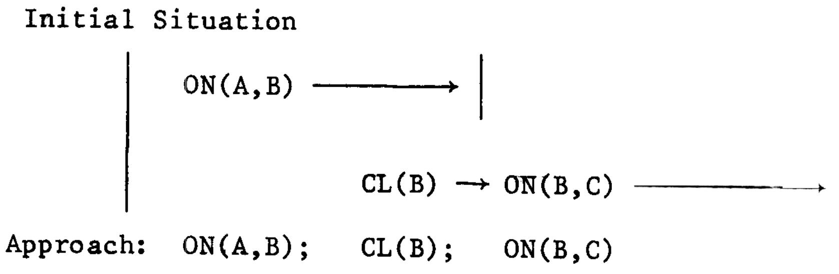


FIGURE 8.3

The protection violation occurs when taking the approach as shown in the holding period diagram of figure 8.4.

FIGURE 8.4



So, as indicated in section 6, the violation may be resolved by trying an approach as shown in figure 8.5(a) or one as shown in figure 8.5(b).

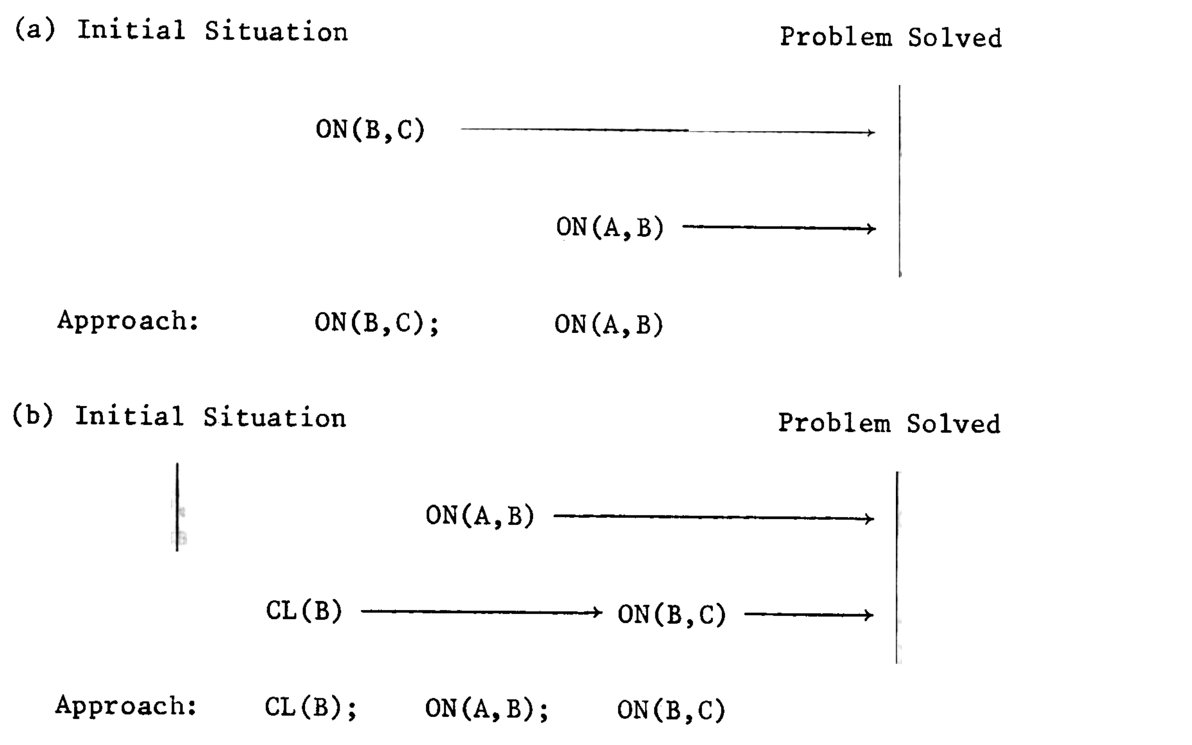


FIGURE 8.5

The latter cannot be used as CL(B) is already true initially and here this approach is no different to the original which caused the violation So, problem solving proceeds with the first (and only) suggested approach (figure 8.5(a)). "REORDER" is printed to signify this.

<u>Note 3</u>

Again a protection violation occurs while pursuing this approach. The tree of ticklists then is shown in figure 8.6. The approaches suggested for getting around the violation are similar to before. Since, however, the top level reversal of goals has already been done, only the approach with a promoted precondition can be tried (see figure 8.7). "PROMOTE" is printed to signify this. This is attempted next as again it is the only choice.

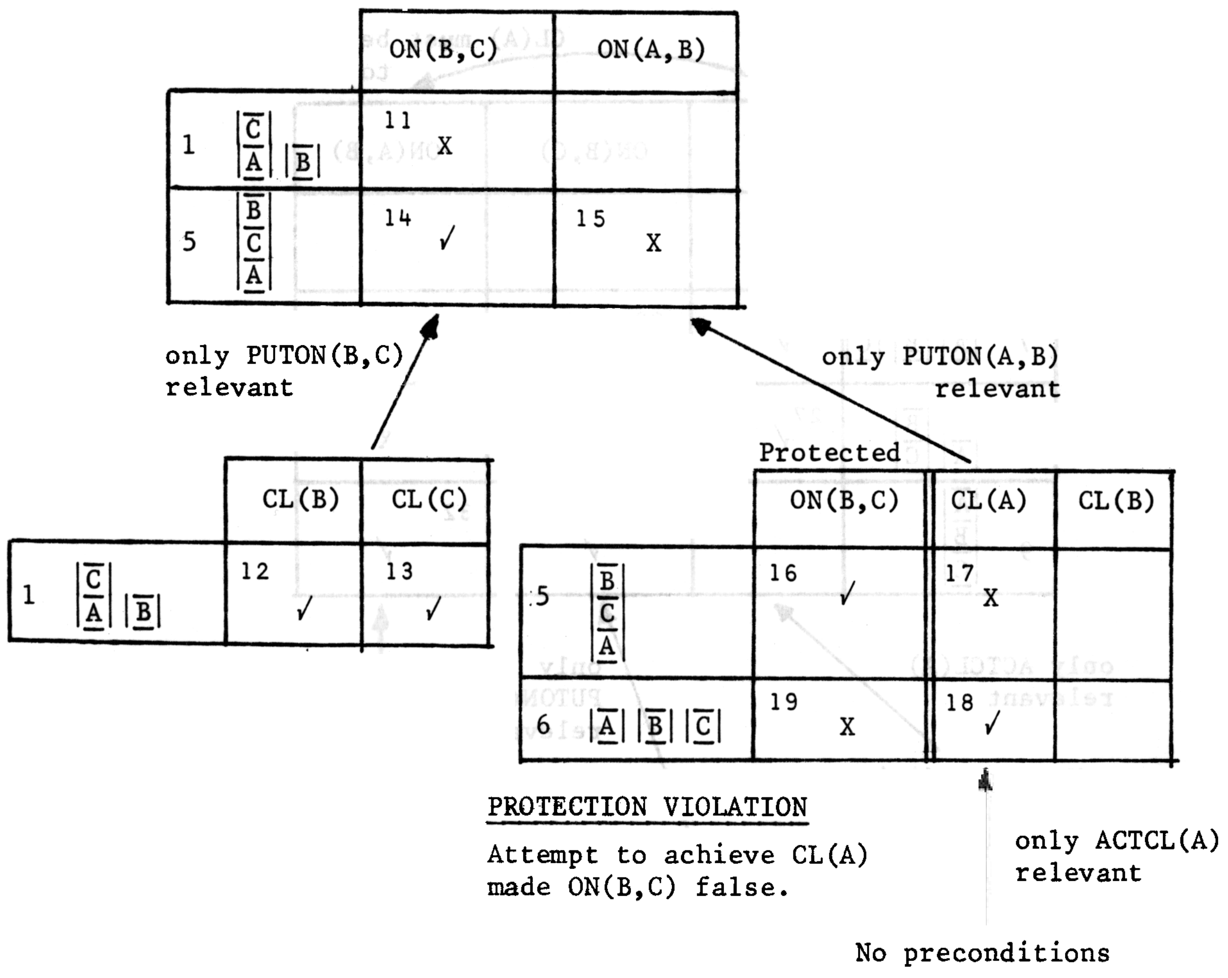


FIGURE 8.6

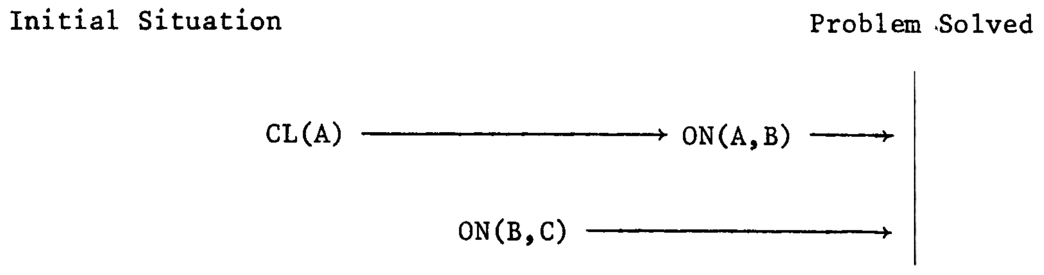
Note 4

The approach of figure 8.7 is successful and produces the optimal plan

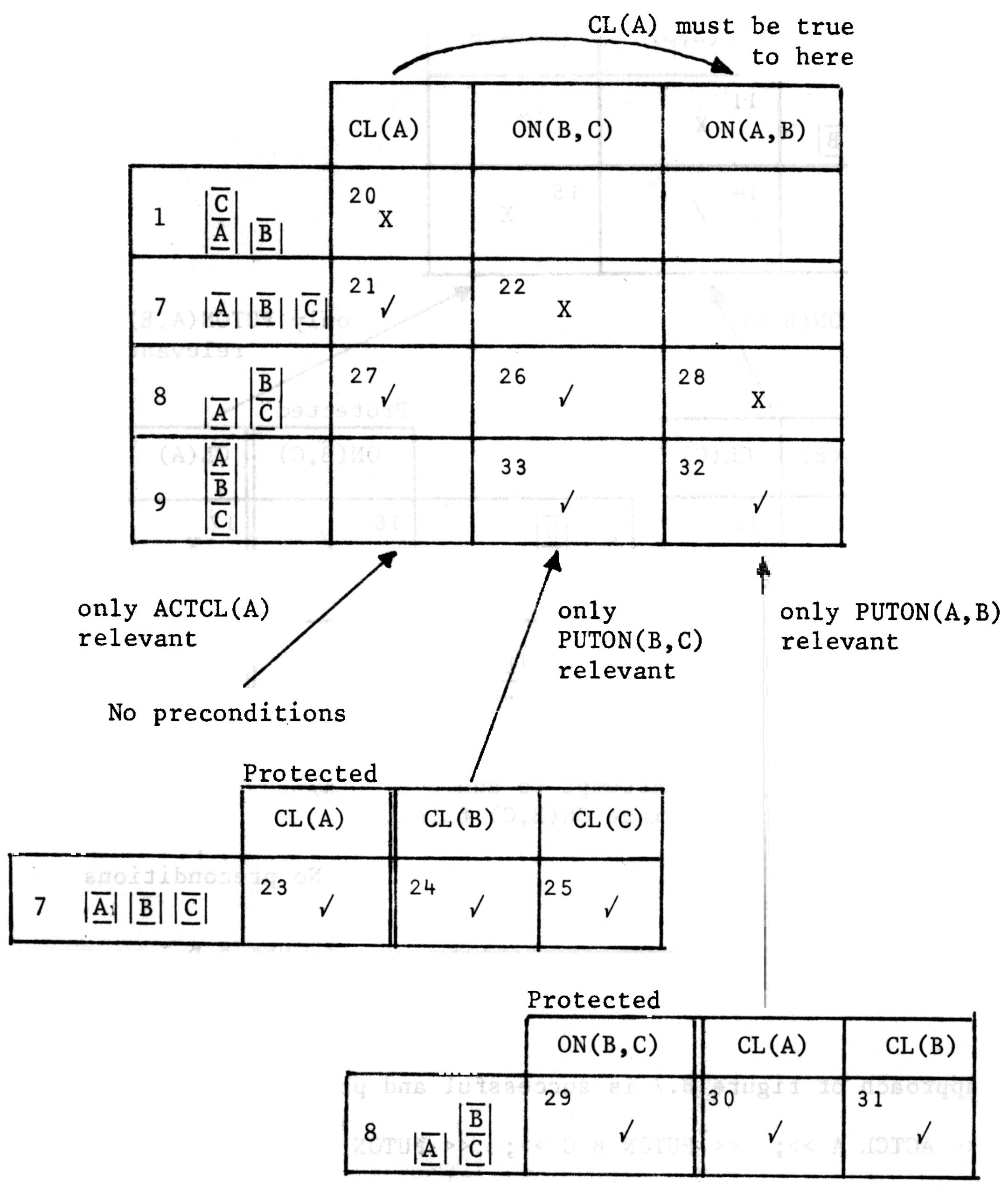
<< ACTCL A >>; << PUTON B C >>; << PUTON A B >>

The tree of ticklists after successful back-up is shown in figure 8.8.

FIGURE 8.7



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



	CL(A)	ON(B,C)	ON(A,B)
1	20 X		
7	21 ✓	22 X	
8	27 ✓	26 ✓	28 X
9		33 ✓	32 ✓

	CL(A)	CL(B)	CL(C)
7	23 ✓	24 ✓	25 ✓

	ON(B,C)	CL(A)	CL(B)
8	29 ✓	30 ✓	31 ✓

FIGURE 8.8

9. EXAMPLE PROBLEMS

INTERPLAN has been tried out on a variety of problems. Besides the block stacking problem (3 block as given and a 5 block example used by Warren, 1974), the STRIPS robot world in particular was used to give some comparison between the different problem solvers' performance. Especially to test the degradation of performance on longer problems. Some provisional times for a series of problems are given below. Comparison times are given where available for STRIPS (3 versions of the axioms were used for their problems: AI Journal Vol. 2 (Fikes and Nilsson, 1971), an earlier version of this paper as presented at IJCAI2, and AI Journal Vol. 3 (Fikes, Hart and Nilsson,1972b)), ABSTRIPS (Sacerdoti, 1974), LAWALY (Siklóssy and Dreussi, 1973) and WARPLAN (Warren, 1974). These times should not be taken too seriously as I have felt free to modify the axiomatizations to fit into the INTERPLAN framework.

Notes (these refer to the table below)

1. INTERPLAN is a program run in POP-2 and HBASE (a CONNIVER-like data base package written in POP-2) on a DEC10 computer. The times were obtained in a single session and include garbage collection and any operating system overhead. INTERPLAN occupies approx. 5K words of store.
2. STRIPS and ABSTRIPS in all forms were run in partially compiled LISP on a DEC10.
3. LAWALY is run in interpreted LISP on a CDC-6600 and the times include garbage collection. (CDC-6600 is approx. 8 times faster than the DEC10).
4. WARPLAN is run in PROLOG, run in optimized FORTRAN on a DEC10.
5. STRIPS did not solve this problem given 20 minutes of CPU time.

Times in seconds To 2 significant figures	INTERPLAN_1			STRIPS_2			MACROP STRIPS	ABSTRIP	LAWALY_3		WARPLAN_4
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	-	-	$>1200_5$	349	401	-	19	-

BLOCK STACKING											
3-Block Problem		2.1			-		-	-		-	8.0
5-Block Problem		7.0			-		-	-		-	40

10. ENHANCEMENTS TO THE METHODS USED IN INTERPLAN

The current program, as described here, has been kept simple so that the mechanisms used to cope with the interacting goal problem could be identified and described. Some of the range of problems which INTERPLAN has tackled have indicated several enhancements which could prove useful or, for some problems, necessary.

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 under any circumstances. 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 exploring ways to achieve a goal (e.g. CL (B)) when none can work because of the particular context we are working in (e.g. ON(A,B) to remain true). WARPLAN (Warren, 1974) uses information about what conjunctions of facts cannot be true together to reject certain branches of the search tree. In this case an instruction to the planning system such as

IMPOSS(CL(y) & ON (x,y)) would be given.

I propose to incorporate a similar idea into INTERPLAN. Whenever a new ticklist is generated, the ticklist heading will be validated using IMPOSS(...) information to reject invalid headings.

Pre-ordering the Approach

When a sequence of goals is given, some may already hold in the initial situation. We should prefer plans which did not destroy the goal, and then re-achieve it again later. However, the ordering as given may allow this, since INTERPLAN only preserves the truth of a goal once it has actually considered it in the approach it was given. If all the individual goals which are true in the initial situation are moved to the front of the goal list (the approach is pre-ordered in this way) INTERPLAN will prefer plans which preserve the truth of these goals.

Heuristic knowledge about goal orderings, or orderings learned during planning may also be used to pre-order an approach. A pre-ordering of each new ticklist heading could be done to allow the above. Repeated goals in a ticklist heading could also then be weeded out.

10.3. Achieving goals which already have true instances.

Normally, if INTERPLAN discovers some goal which is required to be true 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 okay 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 be achieved 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 further choice points constructed to allow the non-true instances to be achieved if the already true instances prove not to be of use.

10.4. Loop detection - Search space redundancies.

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 loop.

a) It prevents goal reversals which have already been tried being suggested as approaches to circumvent goal interaction problems (see section 4).

b) During subgoaling a list of all achieve requests which we are planning to satisfy (along one path through the ticklist search tree) are kept, together with the situation we required each one to be achieved in. If, to satisfy some lower subgoal, a 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 (see section

7.6.2). However, for instance, the generation of similar non-linear approaches (ones with a promoted sub-goal) is not checked.

If a loop is not detected, as well as not providing information on which to suggest possibly useful problem approaches, redundancy can occur in the search space of the planner (the same branch may be tried more than once). When depth first expension of the search tree is being used we then can loop without producing any solution.

10.5. Planning in abstraction Spaces

As mentioned in section 10.1, planning in INTERPLAN proceeds by making a plan which in all the details given (in add and delete lists of operators) works. Where there is an excessive amount of detail this can be tedious and small errors can throw the planner off track (with a depth first search expansion strategy). Sacerdoti (1974) has suggested and implemented a scheme in the ABSTRIPS problem solver which checks that suggested plans will work at certain abstract levels of detail before going on to check them on more details. His scheme is to assign each precondition of a STRIPS-like operator schema to an "abstraction space" and then to plan in higher abstraction spaces before going deeper to check details. ABSTRIPS will re-plan in a higher level abstraction space if details cannot be filled in without interaction. If this occurs, the higher level planning space is not given an indication of what caused the failure. The system relies on good plans being produced at high levels of abstraction.

I have started to consider a simple extension to INTERPLAN based on Sacerdoti's assignment of operator schema preconditions to abstraction spaces, which will allow its full power to be used to produce an "approach" which seems suitable at a high abstraction level. Then, further details will be added to this "approach" and planning in lower abstraction spaces can be done until a plan which works for the most detailed level is available. All goal interactions, whether within an abstraction level, within details, or between both, will be dealt with by the normal classifiers and editors of INTERPLAN. Failure information in this way is passed between planning efforts in different abstraction spaces.

11. CONCLUSION

This report has outlined the problems of interactions between goals which occur in "coupled worlds". A process has been described which allows the use of problem solving techniques which ignore the possibility of interactions and hence which can be simple. Such techniques have received a greal deal of attention, e.g. in STRIPS (Fikes & Nilsson, 1971). Information can easily be extracted from any interactions which do occur to allow planning to continue. A problem solver, INTERPLAN, has been designed and programmed which incorporates this process.

INTERPLAN tries to find an approach (sequence of individual goals) which will solve a task which comprises a conjunction of goals. It does this by "debugging" a given initial approach (usually the given order of a conjunct of goals). This process is similar to that used in HACKER (Sussman 1973) for a more procedural representation of the problem information.

INTERPLAN, and the classifier/editor framework it is described in, provides a tool which can be used for the further study and comparison of problem-solving techniques for creating linear plans. In particular the ticklist and its heading provide a source of information which can be used to improve the planner's information.

The Keys and Boxes problem described in section 2 can still not be tackled by the system though I still consider this to be a benchmark test. Further pattern matching facilities must be provided to deal with sets of objects (see section 3.3) before this will be possible.

APPENDIX 1 PROGRAM IDENTIFIERS

1.1 The Components of an OPSCHEMA

An OPSCHEMA can be constructed using 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 purposes.
(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 to be made false in the successor situation.
(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 & Sussman, 1972).
(e)	PRECONDS	A list of pairs {<REF NUMBER>. <PATTERN>} where <REF NUMBER> will usually be a positive integer. The PRECONDS are joined to any PROTECTEDS 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 <u>never</u> be attempted. It will generally be left null, but can be used to incorporate heuristic knowledge of a problem domain. For instance, a scheme of preventing reversals between groups of goals arranged in a precedence ordering (see Siklőssy 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 only used 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>      )              ( <NAME>,
   ADD     <A1>  <A2>  |   maps to    | {% <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 which has no means of achievement (see Appendix I.II).

.II 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.
It's entries are pairs { <REF> . <PATTERN> }
It is accessed using the doublets:
  PATTREF(i,ticklist) and PATT(i,ticklist).
<PATTERN> ::= goal pattern which may have variables.
<REF> ::= 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 goal's 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, often called SETUP goals as they SETUP the facts in some context 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 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 the 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 search tree. There are two such connecting structures which are both always used to specify an arc as shown in figure APP.1.

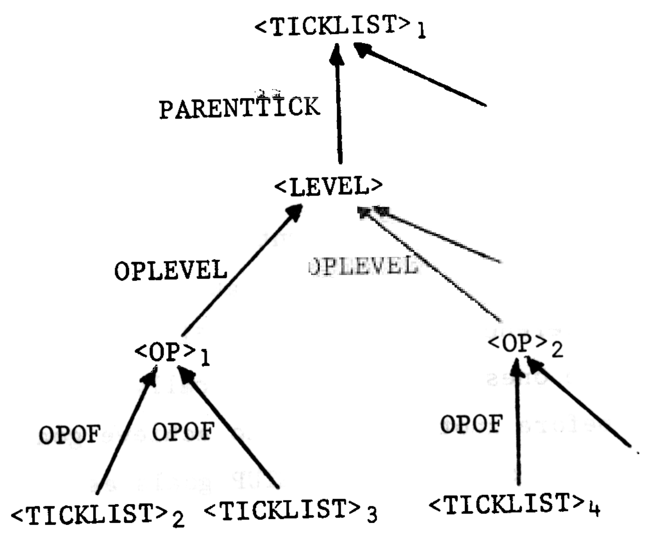


FIGURE APP.1

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, it's TICKVARS is initialized from the VARSLIST of an 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. For example, 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 done. 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 (or 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) NUMPROTECTED	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. See figure APP.1.

(c) ACHPATT The pattern (which usually will refer 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 the variables between 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 figure APP.1).

(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 3 components:-

1. An instance of the pattern we have asked to be achieved - any unset variables are "==" (see Barrow, 1974).
2. The context we asked for the pattern to be true in.
3. The ticklist in which it was found to be 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.

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 returns "cross", if there is at 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 ticklist search 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 pattern>

This is simply defined at present to find all patterns in the context CUCTXT which have VALUE true, using APPITEMS (see HBASE - Barrow, 1974). It is intended to expand FETCHALL to be able to deal with the equivalent of CONNIVER-like IFNEEDed theorems (McDermott and Sussman, 1972).

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 1st 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 used to try to achieve instances of the goal.

Other or-choices can arise 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 would like to use simple depth-first backtracking to find an alternative way to proceed. Alternative choices are taken from the immediate vicinity of some interaction discovered in the search 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 made by removing the first choice in the list. However, INTERPLAN generates some choices (e.g. alternative approaches 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, would be chosen at inappropriate times

We therefore keep or-choices with the points in the search tree which they were intended for. The "level" data structure (see Appendix I.II) 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 choicepoints (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)
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 didn't 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 re-consider 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(TOP LEVEL) list. Once again this scheme can easily be altered by a change of a 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 CHOICES of the appropriate level, or in the global CHOICES(TOPLEVEL) list.

(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 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.

ACKNOWLEDGEMENTS

This work has benefited from interactions with members of the Department of Machine Intelligence (now the Machine Intelligence Research Unit) of Edinburgh University, especially Harry Barrow and Robert Ross. Discussions with my fellow student, Dave Warren, tackling similar problems from a different viewpoint have helped clarify the ideas.

My debt to the designers of STRIPS and HACKER should be clear.

The work is being supported by an SRC research studentship and is supervised by Professor D. Michie. I thank Jean Duckman and Gloria Ketchin for their care and patience during the preparation of the typescript.

REFERENCES

Barrow, H.G. (1974) HBASE POP-2 library documentation (in preparation).

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

Ernst, G.W. and Newell, E. (1969) GPS: A Case Study in Generality and Problem-solving Academic Press.

Fikes, R.E., Hart, P.E. and Nilsson, N.J. (1972a) Some New Directions in Robot Problem-solving Machine Intelligence 7 pp.413. 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.

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.

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 McGraw-Hill.

Sacerdoti, E.D. (1974) Planning in a Hierarchy of Abstraction Spaces Artificial Intelligence, 5 pp. 115-135.

Siklössy, L. and Dreussi, J. (1973) An Efficient Robot Planner which Generates its own Procedures. Advance Papers of IJCAI3 pp. 423-430.

Sussman, G.J. (1973) A Computational Model of Skill Acquisition MIT Technical Report AI TR-297.

Warren, D.H.D. (1974) Warplan: A system for Generating Plans DCL Memo No. 76, University of Edinburgh.