STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

January 1975

THE NONLINEAR NATURE OF PLANS

by
Earl D. Sacerdoti

Artificial Intelligence Group
Technical Note 101

SRI Project 3805-2

The research reported in this paper was sponsored by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC04-72-C-008 with the U.S. Army Research Office.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **JAN 1975**	2. REPORT TYPE	3. DATES COVERED **00-01-1975 to 00-01-1975**
4. TITLE AND SUBTITLE **The Nonlinear Nature of Plans**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **SRI International,333 Ravenswood Avenue,Menlo Park,CA,94025**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**35**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

ABSTRACT

We usually think of plans as linear sequences of actions. This is because plans are usually executed one step at a time. But plans themselves are not constrained by physical limitations of linearity. This paper describes a new information structure, called the procedural net, that represents a plan as a partial ordering of actions with respect to time. By avoiding premature commitments to a particular order for achieving subgoals, a problem-solving system using this representation can deal easily and directly with problems that are otherwise very difficult to solve.

I INTRODUCTION

When we think of plans in our everyday lives, or conceive of plans for a computer to carry out, we usually think of them as linear sequences of actions. The sequence may include conditional tests or loops, but the basic idea is still to do one step after another.

This conception of linearity is misplaced, however. When we say plans are linear we mean that their execution is linear. Basically, a person or a sequential computer's processor can only carry out a single action at a time. But a plan of action is not constrained by physical limitations of linearity. Planning and execution are distinct operations. A plan to be executed need not have the same structure as the trace of its execution.

It is not necessary to represent a plan as a strict ordering of actions with respect to time. Rather, a plan may be thought of as a partial ordering of actions with respect to time. By not making arbitrary commitments to a particular ordering of actions during planning, a problem solver can avoid an entire class of failure situations.

This paper will show how, for certain classes of problems, the representation of plans as nonlinear sequences of actions enables a problem solving system to deal easily and directly with problems that are otherwise very difficult to solve.

II AN EXAMPLE

To better motivate the use of a nonlinear representation, let us develop an elementary example. We will examine a very simple problem in an environment that consists of three blocks and a table. In the initial state, Block C is on Block A, and Block B is by itself. The goal is to achieve a new configuration of blocks, as shown in Figure 1. It is expressed as a conjunction: Block A is on Block B, and Block B is on Block C.

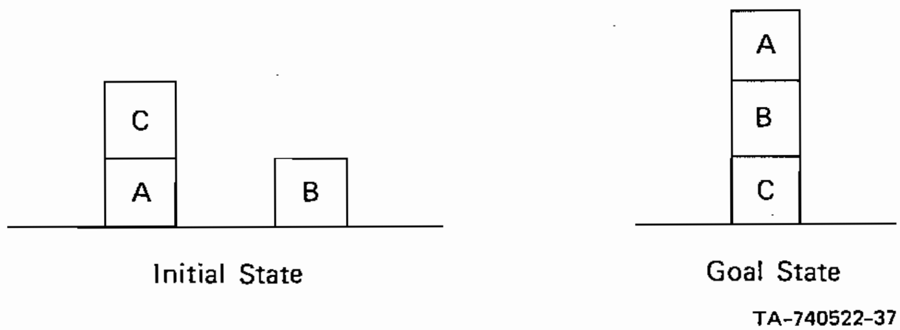


FIGURE 1 EXAMPLE PROBLEM

There is only one action that can be applied to the blocks. PUTON (X,Y) will put Block X on Y. PUTON (X,Y) is not applicable unless X has a clear top, and unless Y is the table or it has a clear top. The problem is to develop a sequence of actions that will achieve the goal state.

This example is presented by Sussman [1] as an "anomalous situation" for which his HACKER program could not produce an optimal solution. Other planning programs using means-ends analysis, for example STRIPS [2] and ABSTRIPS [3], also produce non-optimal solutions. Optimal solutions to the problem are produced by programs of Tate [4] and Warren [5], whose approaches will be discussed in Section VII below.

Let us see what a planning system using means-ends analysis would do under the assumption that plans must be linear. It will try to achieve in turn each of the conjuncts describing the goal state. Suppose it tried to put A on B first. After clearing A by doing PUTON (C,TABLE), the first subgoal can be achieved by doing PUTON (A,B). But now, in order to put B on C, B will have to be re-cleared, thus undoing the subgoal it achieved first.

On the other hand, the system might decide to put B on C first. This can be done immediately in the initial state. But now when the system tries to put A on B, it finds it is even further from its goal than it was in the initial state.

So the planner is in trouble. It must perform a more sophisticated analysis to put the subgoals in the proper order.

But the problem is easy to solve if plans are represented as partial orderings. A planner can begin with an oversimplifed plan that considers the subgoals of putting A on B and putting B on C as parallel, independent operations. When it looks at the subplans in more detail, a simple analysis will determine the interactions between them. Potential conflicts can be resolved by imposing linear constraints on some of the detailed actions.

In subsequent sections we will show how a planner that is freed from the assumption of linearity is able to solve problems of this type directly, constructively, and without backtracking.

III NOAH

NOAH (Nets of Action Hierarchies) is a problem-solving and execution monitoring system that uses a nonlinear representation of plans. The system is being used for SRI's computer based consultant project [6], and has many aspects that are not directly relevant to the point of this paper. These will be suppressed here. We will present a simplifed explanation of the procedural net, NOAH's representation for actions and plans, of SOUP, the language for giving the system task-specific knowledge, and of the planning algorithm. A complete discussion of the system will appear elsewhere [7].

NOAH is implemented in QLISP [8], and runs as compiled code on a PDP-10 computer under the TENEX time-sharing system.

A. The Procedural Net

The system's plans are built up in a data structure called the procedural net, which has characteristics of both procedural and declarative representations.

Basically, the procedural net is a semantic network of nodes, each of which contains procedural information, declarative information, and pointers to other nodes. Each node represents a particular action at some level of detail. The nodes are linked to form hierarchical descriptions of operations, and to form plans of action.

Nodes at each level of the hierarchy are linked in a partially ordered time sequence by predecessor and successor links. Each such sequence represents a plan at a particular level of detail.

The nodes discussed in this paper are of four types: GOAL nodes represent a goal to be achieved; PHANTOM nodes represent goals that should already be true at the time they are encountered; SPLIT nodes have a single predecessor and multiple successors, and represent a forking of the partial ordering; JOIN nodes have multiple predecessors and a single successor, and represent a rejoining of subplans within the partial ordering.

Each node points to a body of code. The action that the node represents can be simulated by evaluating the body. The evaluation will cause new nodes, representing more detailed actions, to be added to the net. It will also update a hypothesized world model to reflect the effects of the more detailed actions.

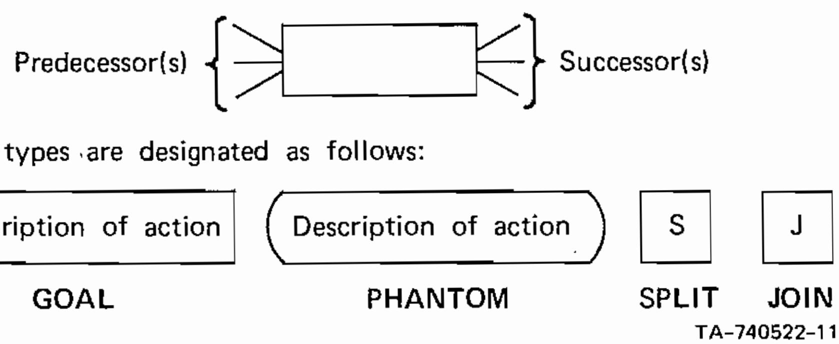


FIGURE 2 GRAPHIC REPRESENTATION OF A NODE

Associated with each node is an add list and a delete list. These lists are computed when the node is created. They contain symbolic expressions representing the changes to the world model caused by the action that the node represents.

Figure 2 shows the graphic notation used in this paper to display a node of a procedural net.

As an example, let us examine a procedural net representing a hierarchy of plans to paint a ceiling and paint a stepladder. The plan can be represented, in an abstract way, as a single node as shown in Figure 3a. In more detail, the plan is a conjunction, and might be represented as in Figure 3b. The more detailed subplans to achieve these two goals might be "Get paint, get ladder, then apply paint to ceiling," and "Get paint, then apply paint to ladder," as depicted in Figure 3c.

The pictorial representation used here suppresses much of the information associated with each node. The add and delete lists, for instance, are not indicated in the diagrams. They are not hard to infer, however. For example, "Get ladder" will cause "Has ladder" to be added to the world model, and "Apply paint to ceiling" might delete "Has paint."

Precondition-subgoal relationships are inferred by the system from pointers that indicate which nodes represent expansions in greater detail of other nodes. These pointers are also omitted in the pictorial representation. The system assumes that every action but the last in such an expansion is a precondition for the last action.

B. Task-specific Knowledge

Knowledge about the task domain is given to the system in procedural form, written in the SOUP (Semantics of User's Problem) language. SOUP is

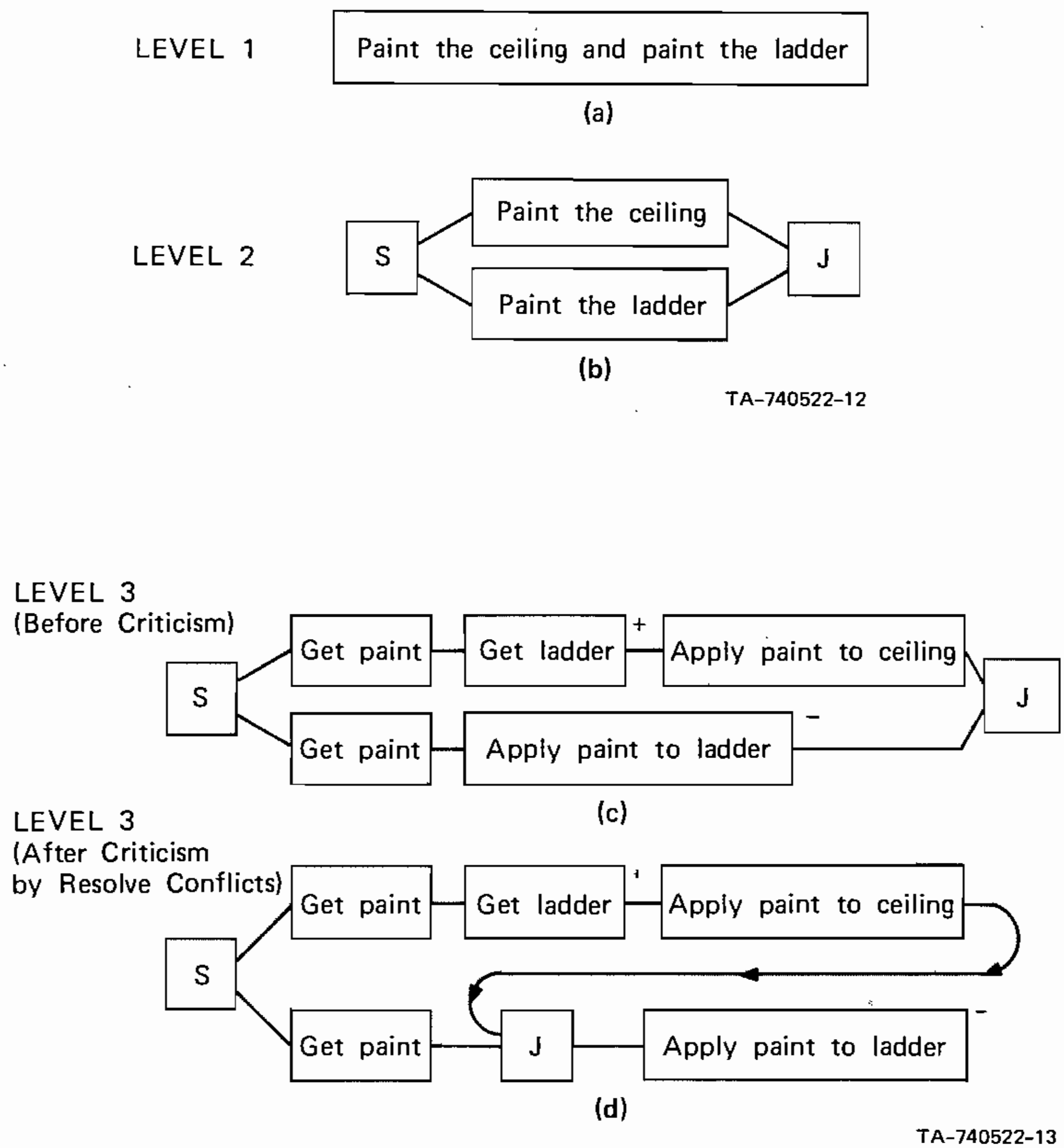


FIGURE 3 PROCEDURAL NET FOR PAINTING

an extension of QLISP [8] that is interpreted in an unusual fashion. The process of planning transforms this procedural knowledge into the hybrid procedural net form, which contains both procedural and declarative information, and which represents a hierarchy of solutions to the particular problem at hand.

As an example, let us examine the SOUP code for blocks problems such as that presented in Section II above. The complete semantics of the actions of this domain are expressed by two functions, which are shown in Figure 4. The code for the function CLEAR says, "If the variable X is TABLE, then it is already 'clear'. Otherwise, see if some block Y is on X. If so, clear Y and then remove Y by putting it somewhere else."

The code for the function PUTON says, "To put X on Y, first clear X and Y. Then place X on Y (and thus Y is no longer clear)".

C. The Planning Algorithm

Initially, NOAH is given a goal to achieve. NOAH first builds a procedural net that consists of a single goal node to achieve the given goal. This node has a list of all relevant SOUP functions as its body. This single node represents the plan to achieve the goal at a very high level of abstraction. This one-step plan may then be expanded.

The planning algorithm of the NOAH system is simple. Its input is a procedural net. It simulates the most detailed plan in the net by simulating each node of the plan in turn. In addition to building a detailed model of the effects of each action in the plan, the simulation of each node will produce child nodes. Thus by simulating the plan, a new, more detailed plan will be created.

Figure 4

SOUP Code for the Blocks Problems

```
(CLEAR
  (QLAMBDA
    (CLEARTOP ←X)
    (OR
      (EQ $X (QUOTE TABLE))
      (QPROG
        (←Y)
        (ATTEMPT (PIS (ON ←Y $X))
            THEN (PGOAL (Clear $Y)
                        (CLEARTOP $Y)
                        APPLY
                        (CLEAR) )
                 (PDENY (ON $Y $X))
                 (PGOAL (Put $Y on top of ←Z)
                        (ON $Y ←Z)
                        APPLY NIL))
        (RETURN)))))

(PUTON
  (QLAMBDA
    (ON ←X ←Y)
    (PAND
      (PGOAL (Clear $X)
             (CLEARTOP $X)
             APPLY
             (CLEAR))
      (PGOAL (Clear $Y)
             (CLEARTOP $Y)
             APPLY
             (CLEAR)))
    (PGOAL (Put $X on top of $Y)
           (ON $X $Y)
           APPLY NIL)
    (PDENY (CLEARTOP $Y))))
```

The individual subplan for each node will be correct, but there is as yet no guarantee that the new plan, taken as a whole, will be correct. There may be interactions between the new, detailed steps that render the overall plan invalid. For example, the individual expansions involved in generating the plan in Figure 3c from that in Figure 3b are correct, yet the overall plan is invalid, since it allows for painting the ladder before painting the ceiling.

Before the new detailed plan is presumed to work, the planning system must take an overall look at it to ensure that the local expansions make global sense together. This global examination is provided by a set of critics. The critics serve a purpose somewhat similar to that of the critics of Sussman's HACKER [1], except that for NOAH they are constructive critics, designed to add constraints to as yet unconstrained plans, whereas for HACKER they were destructive critics whose purpose was to reject incorrect assumptions reflected in the plans.

The algorithm for the planning process, then, is as follows:

(1) Simulate the most detailed plan in the procedural net. This will have the effect of producing a new, more detailed plan.

(2) Criticize the new plan, performing any necessary reordering or elimination of redundant operations.

(3) Go to Step 1.

Clearly, this algorithm is an oversimplification, but for the purposes of this paper we may imagine that the planning process continues until no new details are uncovered. (In fact, for the complete problem-solving and execution monitoring system, a local decision must be made at every node about whether it should be expanded.)

IV CRITICS

We are almost ready to look at some examples. But first let us examine the set of critics. The ones described here are general-purpose critics, appropriate to any problem-solving task. In addition to these, other task-specific critics may be specified for any particular domain.

A. The "Resolve Conflicts" Critic

The Resolve Conflicts critic examines those portions of a plan that represent conjuncts to be achieved in parallel. In particular, it looks at the add and delete lists of each node in each conjunctive subplan. If an action in one conjunct deletes an expression that is a precondition for a subgoal in another conjunct, then a conflict has occurred. The subgoal is endangered because, during execution, its precondition might be negated by the action in the parallel branch of the plan. (An implicit assumption being made here is that all of a subgoal's preconditions must remain true until the subgoal is executed.) The conflict may be resolved by requiring the endangered subgoal to be achieved before the action that would delete the precondition.

For example, the painting plan depicted in Figure 3c contains a conflict. "Apply paint to ladder" will effectively delete "Has ladder," which is on the add list of "Get ladder." In such a situation, a conflict would occur, since "Has ladder" is a precondition of "Apply paint to ceiling." The conflict is denoted in the pictorial representation by a plus sign (+) over the precondition and a minus sign (-) over the step that violated it. The conflict can be resolved by requiring that the endangered subgoal ("Apply paint to ceiling") be done before the violating step ("Apply paint to ladder").

If the conflict were resolved in this manner, the resulting plan would appear as in Figure 3d.

A similar conflict occurs if an action in one conjunct deletes an expression that is a precondition for a following subgoal. In this case, the precondition must be re-achieved after the deleting action.

Conflicts of this type are very easy to spot. The critic simply builds a table of multiple effects. This table contains an entry for each expression that was asserted or denied by more than one node in the current plan. A conflict is recognized when an expression that is asserted at some node is denied at a node that is not the asserting node's subgoal.

Note that a precondition may legally be denied by its own subgoal. For example, to put Block A on Block B, B must have a clear top. This precondition will be denied by the action of putting A on B.

B. The "Use Existing Objects" Critic

In addition to specifying the right actions in the right order, a complete plan must specify the objects that the actions are to manipulate. For NOAH, this specification is accomplished by binding the unbound variables (those prefixed by a left arrow) in the PGOAL statements of the SOUP code.

During the course of planning, NOAH will avoid binding a variable to a specific object unless a clear best choice for the binding is available. When no specific object is clearly best, the planner will generate a formal object to bind to the variable. The formal object is essentially a place holder for an entity that is as yet unspecified. The formal objects described here are similar in spirit to those used by Sussman in his HACKER program [1], and to the uninstantiated parameters in relevant operators as used by ABSTRIPS [3].

The strategy of allowing actions with unbound arguments to be inserted into a plan has several advantages. First, it enables the system to avoid making arbitrary, and therefore possibly wrong, choices on the basis of insufficient information. Furthermore, it allows the system to deal with world models that are only partially specified by producing plans that are only partially specified.

However, after a plan has been completed at some level of detail, it may be clear that a formal object can be replaced by some object that was mentioned elsewhere in the plan. The Use Existing Objects critic will replace formal objects by real ones whenever possible. This may involve merging nodes from different portions of the plan, resulting in reordering or partial linearization.

For example, a more detailed expansion of the painting plan might specify putting the ladder at Place001 to paint it, and at Under-Ceiling for painting the ceiling. The Use Existing Objects critic would optimize the plan by replacing Place001 with Under-Ceiling.

C. The "Eliminate Redundant Preconditions" Critic

During the simulation phase of the planning process, every precondition that is encountered is explicitly stored in the procedural net. This is so that the critics will be able to analyze the complete precondition-subgoal structure of each new subplan. But after the other critics have done their work, and the plan has been altered to reflect the interactions of all the steps, the altered plan may well specify redundant preconditions.

For instance, in our painting example, "Get paint" appears twice in the plan. This critic recognizes the redundancy be examining the same table of multiple effects that was used by Resolve Conflicts. The extra preconditions are eliminated to conserve storage and avoid redundant planning at more detailed levels.

V THE EXAMPLE, AGAIN

We are now ready to see how NOAH solves the problem posed in Section II. The initial state is expressed to the system as a set of QLISP assertions:

(ON C A)

(CLEARTOP B)

(CLEARTOP C).

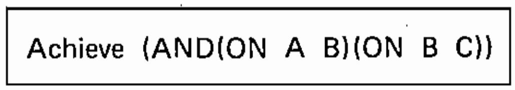
NOAH is invoked with the goal: (AND (ON A B) (ON B C)).

The system builds an initial procedural net that consists of a single GOAL node. The node is to achieve the given goal; its body is a list of the task-specific SOUP functions, in this case CLEAR and PUTON. It then applies the planning algorithm to this one-step plan, which is depicted in Figure 5a.

The conjunction is split up, so that each of its conjuncts is achieved independently. PUTON is the relevant function for achieving both conjuncts, but the system does not immediately invoke PUTON. Rather, the system builds a new GOAL node in the procedural net to represent each invocation. The nodes are to achieve (ON A B) or (ON B C), and have PUTON as their body. The original plan has now been completely simulated to a greater level of detail, and so the critics are applied. At this level, they find no problems with the plan that was generated. The new plan is shown in Figure 5b.

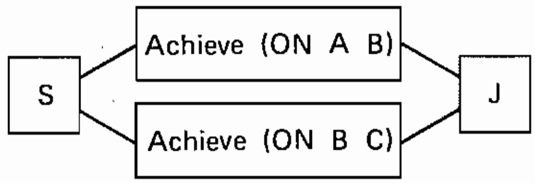
The new plan is now expanded. When the GOAL nodes for achieving (ON A B) and (ON B C) are simulated, PUTON is applied to each goal

LEVEL 1



(a)

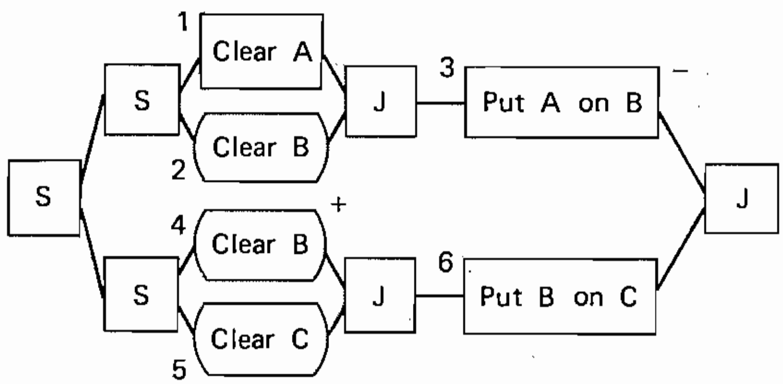
LEVEL 2



(b)

TA-740522-14

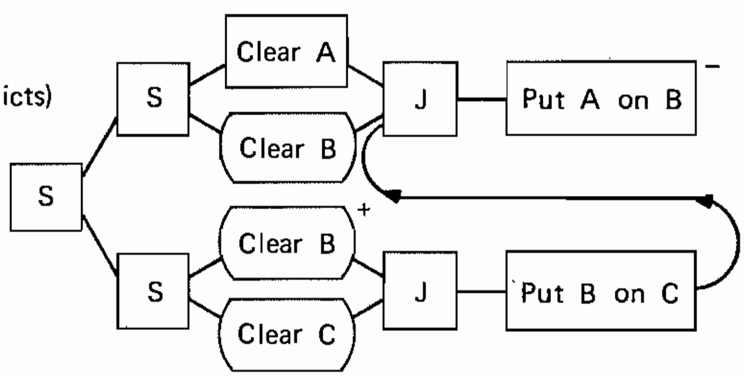
LEVEL 3
(Before Criticism)



(c)

TA-740522-15

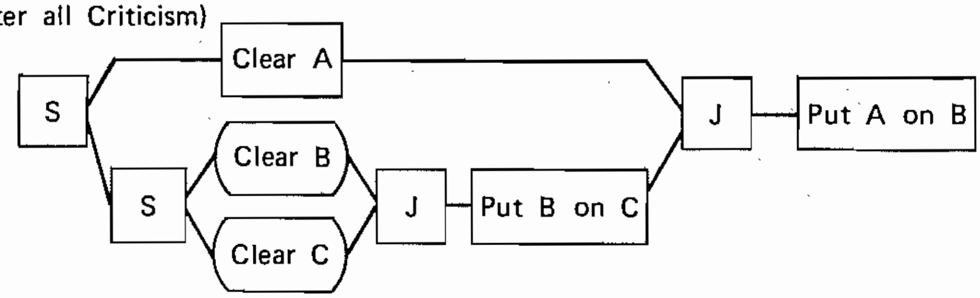
LEVEL 3
(After Criticism
by Resolve Conflicts)



(d)

TA-740522-16

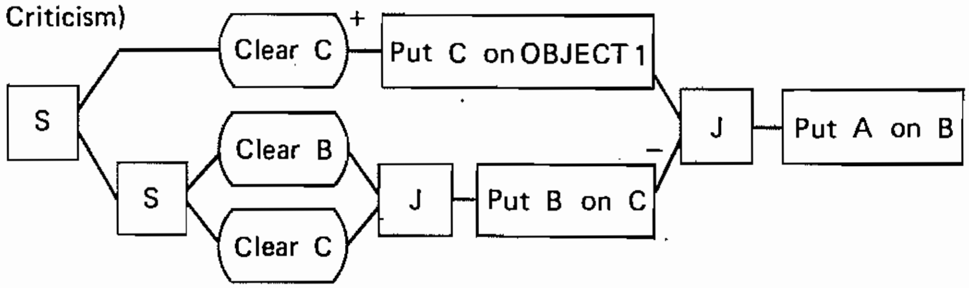
LEVEL 3
(After all Criticism)



(e)

TA-740522-17

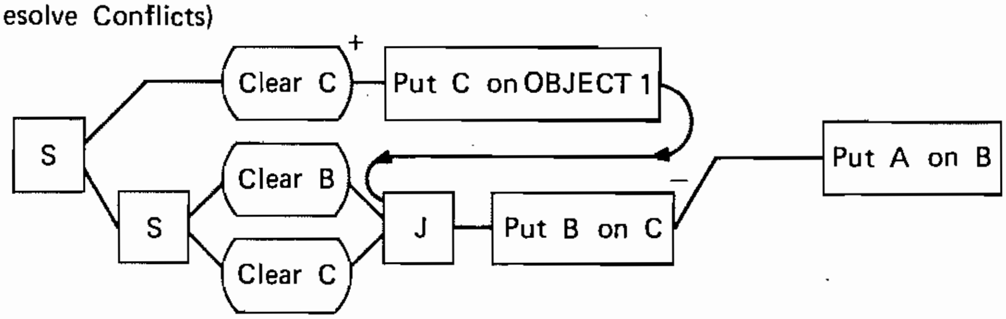
LEVEL 4
(Before Criticism)



(f)

TA-740522-18

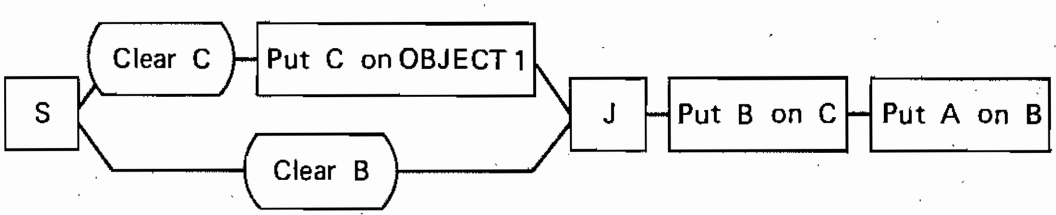
LEVEL 4
(After Criticism
by Resolve Conflicts)



(g)

TA-740522-19

LEVEL 4
(After all Criticism)



(h)

TA-740522-20

expression. PUTON causes the generation of a new level of GOAL nodes. When the entire plan has been simulated, the resulting new plan appears as in Figure 5c. The nodes of the plan are numbered to aid in explaining the actions of the critics.

The critics are now applied to the new plan. Resolve Conflicts generates a table of all the expressions that were asserted or denied more than once during the simulation. The table is shown in Figure 6a. This table is then pared down by eliminating from consideration those preconditions that are denied by their own subgoals. For example, (CLEARTOP C) is a precondition for the subgoal (ON B C), so it is not a conflict that achieving (ON B C) at Node 6 makes (CLEARTOP C) false. Now, any expression for which there is only a single remaining effect is removed from the table. The resulting table, shown in Figure 6b, displays all the conflicts created by the assumption of nonlinearity.

Resolve Conflicts now resolves the conflict by reordering the plan to place the endangered subgoal [the node achieving (ON B C)] before the violating step [the node achieving (ON A B)]. The transformed plan is shown in Figure 5d.

Since no formal objects were generated at this level of detail, Use Existing Objects does not transform the plan further. Eliminate Redudant Preconditions is now applied, and the resulting plan is shown in Figure 5e. Note that the major restriction in the solution to the problem, that B must be placed on C before A is placed on B, has been incorporated into the plan. This has been accomplished directly, constructively, and without backtracking.

Figure 6

Table of Multiple Effects for Example Problem

(Node numbers refer to Figure 5c.)

6a - Original Table

CLEARTOP B:	Asserted - Node 2 ("Clear B")
	Denied - Node 3 ("Put A on B")
	Asserted - Node 4 ("Clear B")
CLEARTOP C:	Asserted - Node 5 ("Clear C")
	Denied - Node 6 ("Put B on C")

6b - Refined Table

CLEARTOP B:	Denied - Node 3 ("Put A on B")
	Asserted - Node 4 ("Clear B")

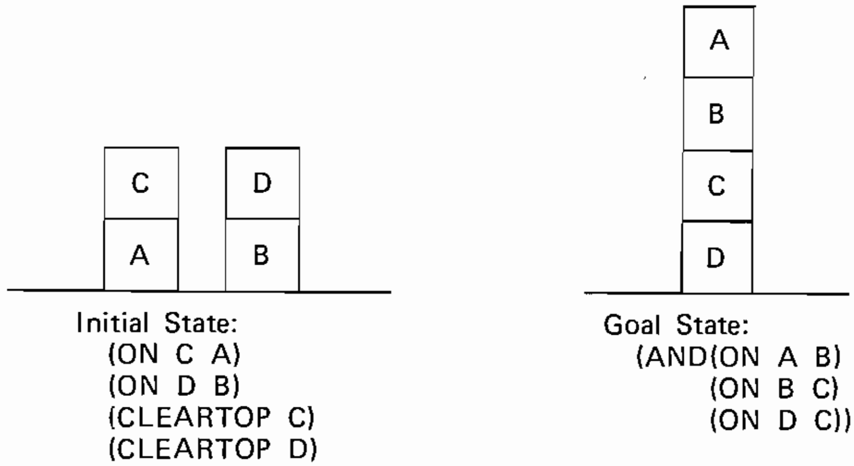
The critics having been applied, the system simulates the new plan. This results in the generation of a new, yet more detailed plan, shown in Figure 5f. The critics are then applied. An analysis similar to that described above enables Resolve Conflicts to discover that (CLEARTOP C) might be violated when achieving (ON B C). Thus, the plan is rearranged, as shown in Figure 5g, so that (ON C Object1), the endangered subgoal, is achieved before (ON B C).

Use Existing Objects again finds no formal objects that can be unified with existing ones. After Eliminate Redundant Preconditions cleans up the plan, it appears as in Figure 5h. The final plan is: Put C on Object1; Put B on C; Put A on B. The total time required to produce the plan was 26 seconds. Essentially, the plan is now completely linearized. The planning system has chosen the correct ordering for the subgoals, without backtracking or wasted computation. By avoiding a premature commitment to a linear plan, the system never had to undo a random choice made on the basis of insufficient information.

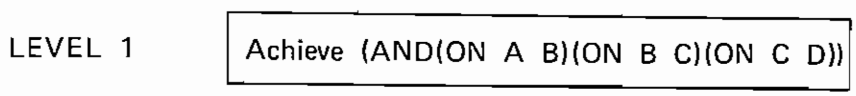
VI OTHER EXAMPLES

In this section a number of other blocks world examples will be presented. The problems and their solutions will be displayed graphically, and only points of special interest will be discussed in the text.

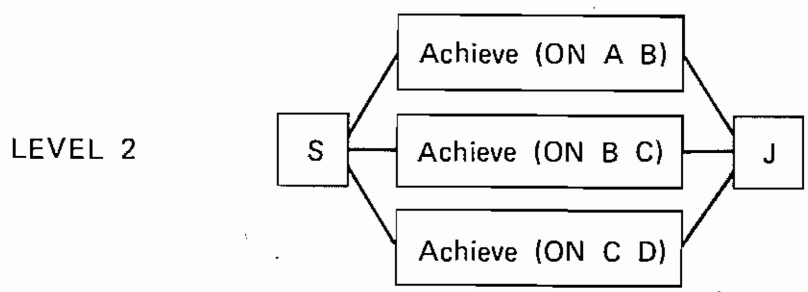
A. Four Blocks



TA-740522-28

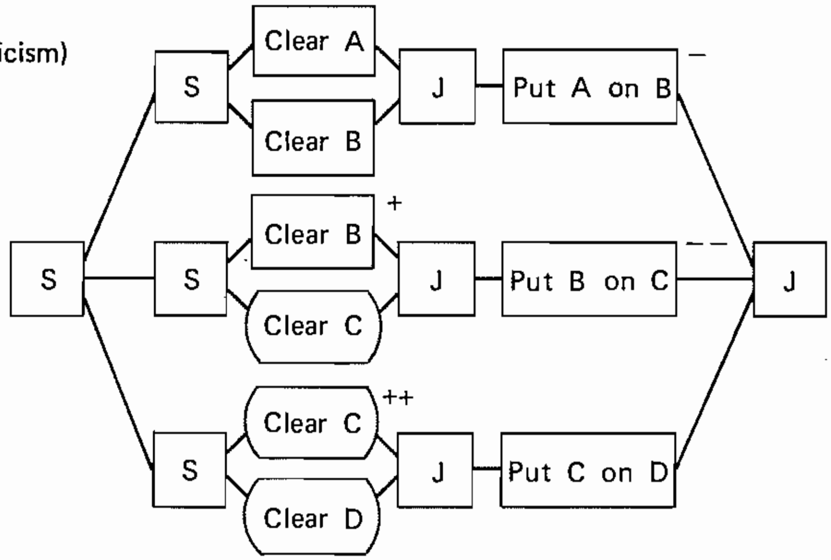


The conjunctive goal is split into parallel goals.



TA-740522-29

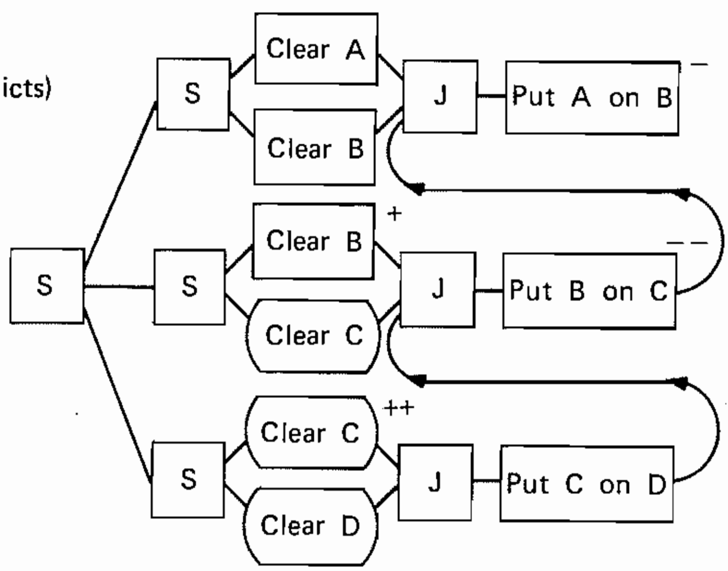
LEVEL 3
(Before Criticism)



TA-740522-30

Resolve Conflicts notices two cases of a precondition (+) negated by a parallel operation (–).

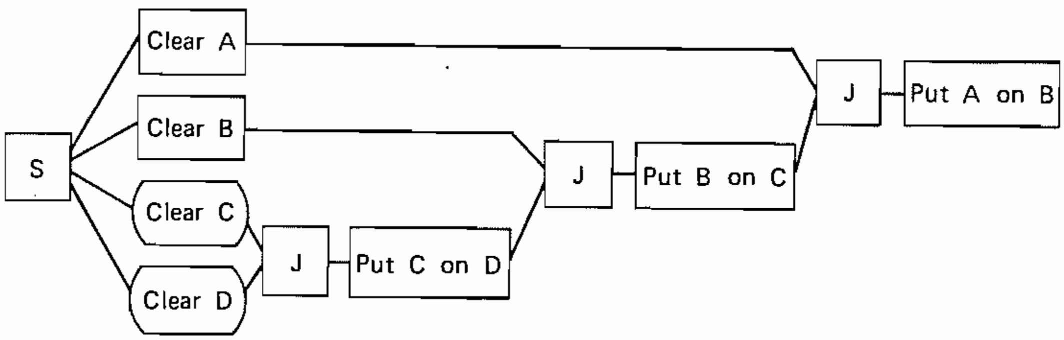
LEVEL 3
(After Criticism
by Resolve Conflicts)



TA-740522-31

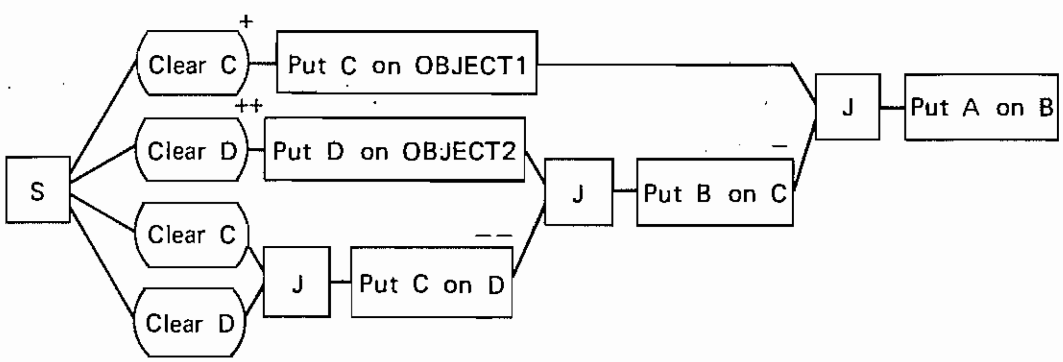
Eliminate Redundant Preconditions cleans up the plan.

LEVEL 3
(After all Criticism)



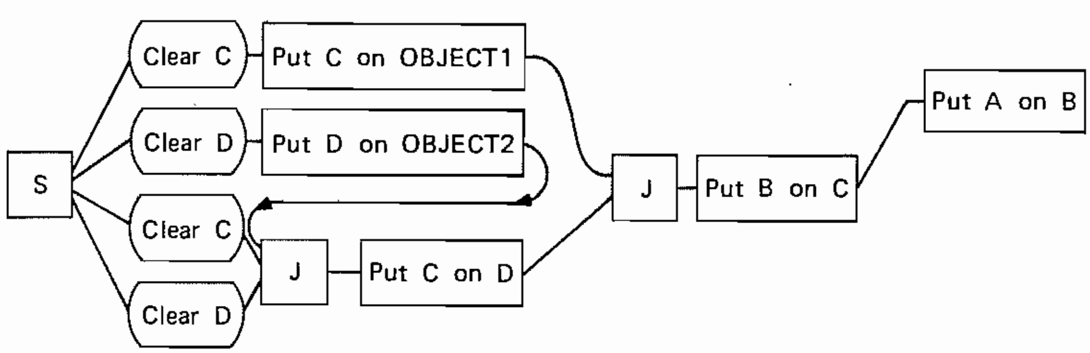
TA-740522-32

LEVEL 4
(Before Criticism)



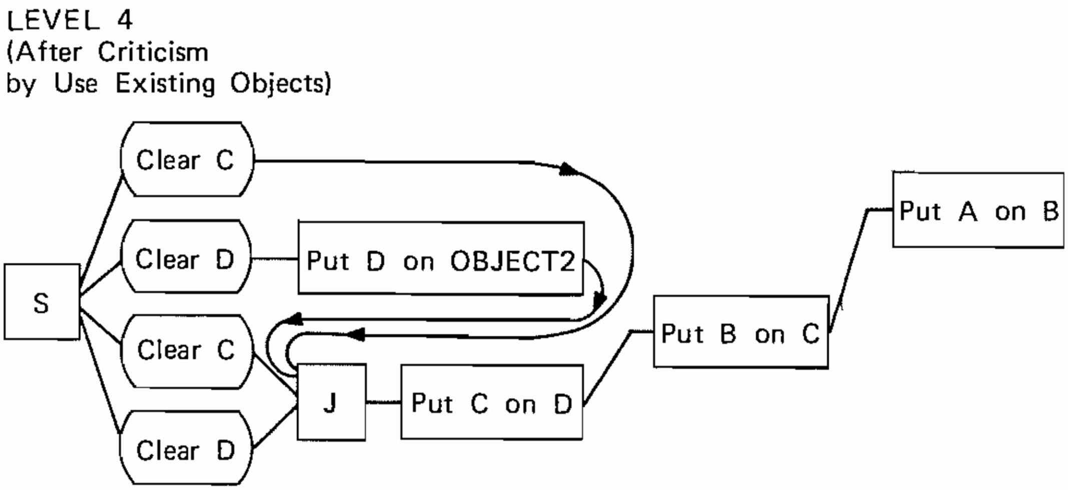
TA-740522-33

LEVEL 4
(After Criticism
by Resolve Conflicts)

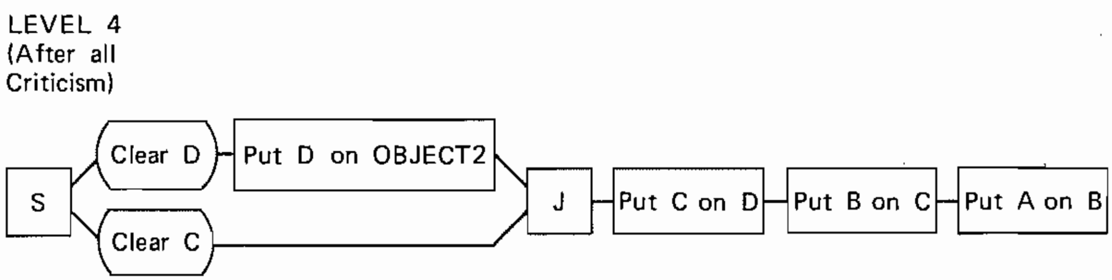


TA-740522-34

Use Existing Objects notices that the plan can be simplified by unifying the formal object, Object1, with Block D. The nodes which refer to putting C on D and Object1 are merged.



TA-740522-35

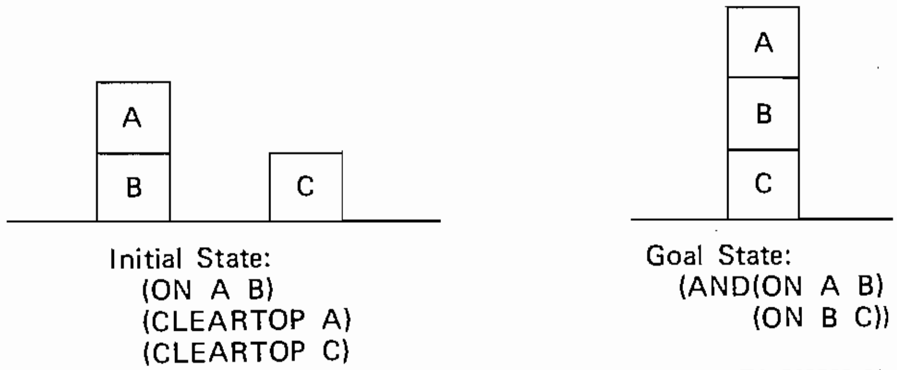


TA-740522-36

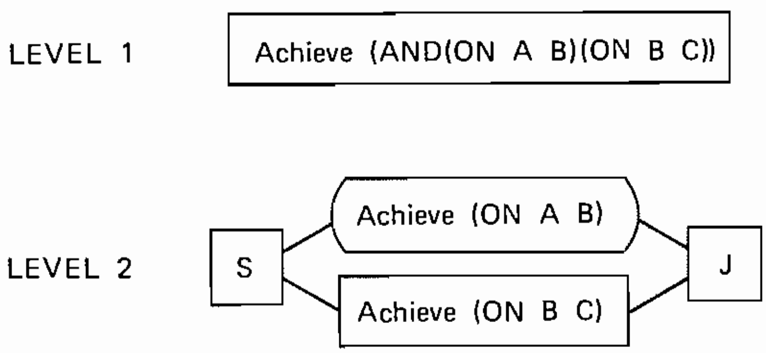
The final plan is: Put D on Object2, Put C on D, Put B on C, Put A on B. The total time to generate the plan was 41 seconds.

B. Creative Destruction

This problem can only be solved by undoing a subgoal which is already achieved.



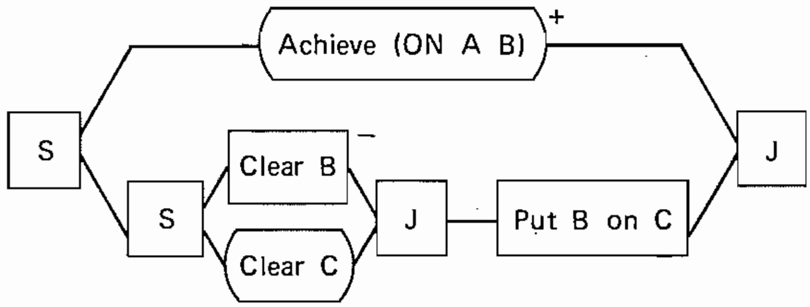
TA-740522-21



TA-740522-22

The first conjunct is a PHANTOM goal, since it is already true in the initial world model.

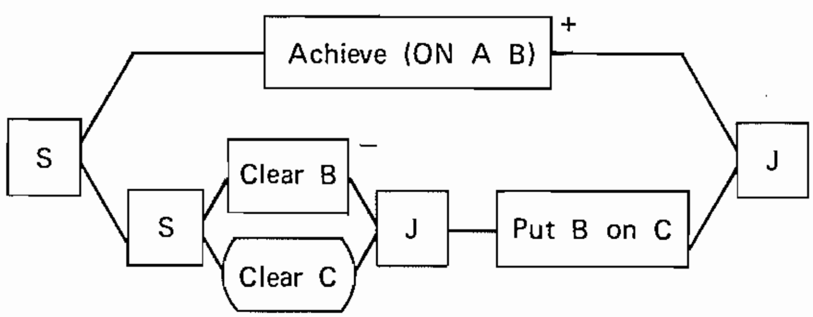
LEVEL 3
(Before Criticism)



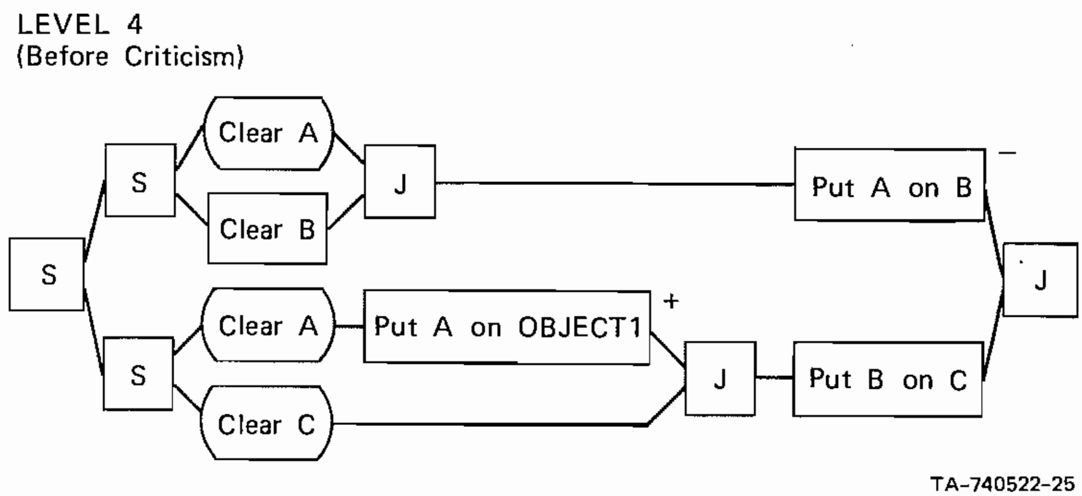
TA-740522-23

Resolve Conflicts notices that one node (-) deletes a precondition for a subsequent subgoal. The precondition in this case is (ON A B), and the subgoal is the initial conjunctive goal. The system therefore alters the PHANTOM goal (+) to become a genuine goal, to be achieved in time for the subsequent subgoal.

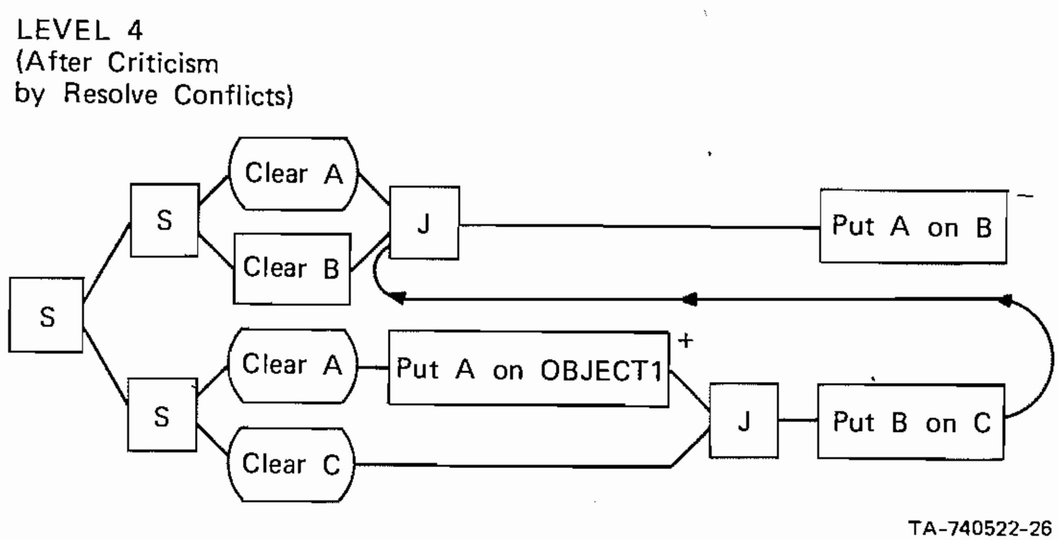
LEVEL 3
(After Criticism)



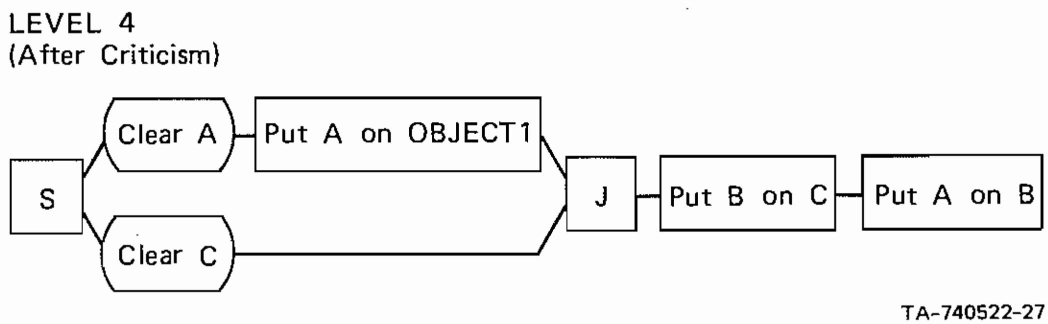
TA-740522-24



Resolve Conflicts notices that (CLEARTOP B) is asserted by one node (+) and deleted by another (-). It therefore reorders the plan.



Eliminate Redundant Preconditions cleans up the plan.



The final plan is: Put A on Object1, Put B on C, Put A on B. The total time to produce the plan was 26 seconds.

VII DISCUSSION

We have seen how a variety of problems which can be represented as conjunctive goals have simple, straightforward solutions in NOAH. There are a number of other problem-solving systems that use alternative approaches to solve similar problems. Among these are Sussman's use of debugging [1,9], Tate's search in a space of "tick lists" [4], and the approach of passing actions backward over a partial plan, which is used by Manna and Waldinger [10] and Warren [5].

The approach presented in this paper is in many ways antithetical to that of Sussman's HACKER. HACKER attacks conjunctive goals by making a "linear" assumption. That is, conjunctive goals are assumed to be independent and additive, and so to achieve the overall goal each conjunct may be achieved in sequence. The system is explicitly aware of this assumption. If the developing plan fails, it can be debugged by comparing the problem that occurred with the known types of problems generated by the assumption of linearity. As bugs are encountered and solved, a collection of critics is developed, each of which notices that a certain type of bug has occurred in a plan.

HACKER does a lot of wasted work. While the problem solver will eventually produce a correct plan, it does so in many cases by iterating through a cycle of building a wrong plan, then applying all known critics to suggest revisions of the plan, then building a new (still potentially wrong) plan.

NOAH makes no rash assumptions, but preserves all the freedom of ordering that is implicit in the statement of a conjunctive goal. It assumes that conjuncts are independent, but the nonlinear representation frees it from worrying about additivity. It applies its critics constructively, to linearize the plan only when necessary. By waiting until it knows the nature of the conjuncts' interactions, NOAH is sure to place actions in the correct order, and thus needs never undo the effects of a false assumption.

Tate's INTERPLAN performs a search for a correct linear ordering by using both debugging and backtracking. INTERPLAN does this not by creating alternative sequences of actions, but rather by examining a tabular representation of the interactions between conjunctive goals. Tate demonstrated that a planner can perform reasoning about plans by dealing with information that is much simpler than the plan itself. This concept has been used extensively by the critics in NOAH, which do much of their analysis on the tables of multiple effects rather than on the plans themselves.

Manna and Waldinger and Warren build linear plans in non-sequential order. They require that the partial plan at every stage be a linear one. However, they allow additions to the plan by insertion of new actions into the body of the plan, rather than restricting new actions to appear at the end. This approach has the advantage of being constructive, in the sense that when the planner adds each step to the plan, it takes into account all the interactions between conjuncts that it knows about. But by forcing the plan to be linear at all intermediate stages, these planners must do unnecessary search with backtracking, or sophisticated plan optimization to find the correct order in which to attack the conjuncts.

VIII FURTHER WORK

This paper deals with a deceptively simple idea: a plan may have the structure of a partial ordering, even though its execution might have to be structured as a total ordering. The planning system described here is primitive and incomplete, and a more complete one will be required to fully explore the implications of this representation of plans. The system does not now deal with disjunctive subgoals (for example, to paint the ceiling, get paint and either a ladder or a table). The simplicity of the critics' analysis of the tables of multiple effects breaks down for disjunctive situations. It appears that a search process in a space of alternative tables, such as Tate uses for conjunctive goals, would enable the system to deal with disjuncts.

The current system also fails to deal with what might be termed "non-linearizable interactions." These are interactions between subgoals where no simple ordering of the actions that achieve each subgoal independently will achieve the overall goal. An example of this type of interaction arises in the problem of exchanging the contents of two registers. There is no linear order in which the steps of putting the contents of R1 in R2, and the contents of R2 in R1, will produce a valid plan. The problem solver must suggest the creative step of putting the contents of one register in temporary storage.

This is not as difficult as it appears. An analysis of the table of multiple effects can reveal the non-linearizable nature of the interaction. An analysis of the computations that resulted in the multiple effects can suggest how to make each interacting subplan innocuous to the others by inserting the appropriate actions in each subplan. Inserting these new actions is the creative step alluded to in the preceding paragraph.

The most serious deficiency in the current system is its lack of awareness about the auxiliary computations specified in the procedural semantics (the SOUP code) of a task domain. The procedural net representation lets the system be aware of the goals and subgoals that the planner has decided to tackle, but it does not preserve any information about the computation that resulted in those decisions. In some cases, a reordering of subgoals might alter the state in which one of these computations would be carried out. Then the computation might produce different results. There are two ways in which the deficiency could be dealt with. One approach would be to restrict the complexity of the SOUP code that specifies the actions of a task domain.

However, if NOAH is to be effective in truly complex domains, SOUP must have all the richness of a PLANNER-like language [11], and the system must be aware of this new type of interaction. This can be done by allowing the entries in the tables of multiple effects to specify a computation as well as a simple expression. The computation, evaluated at the time a critic is analyzing interactions, would reflect the effects of the currently postulated order of subgoals.

The solutions to the problems raised in this section will surely alter the particular problem-solving strategies that were adequate for the simple examples discussed in this paper. The problems do not threaten the usefulness of the nonlinear representation for plans, and may in fact be best solved by relying on such a representation.

ACKNOWLEDGEMENTS

The ideas presented in this paper have been stimulated and sharpened by discussions with Richard Waldinger, Richard Fikes, Nils Nilsson, and Austin Tate.

REFERENCES

1. G. J. Sussman, "A Computational Model of Skill Acquisition," Technical Note AI TR-297, Artificial Intelligence Laboratory, MIT, Cambridge, Massachusetts (August 1973).

2. R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2, pp. 189-208 (1971).

3. E. D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," Artificial Intelligence, Vol. 5, No. 2, pp. 115-135 (1974).

4. A. Tate, "INTERPLAN: A Plan Generation System which can deal with Interactions between Goals," Memorandum MIP-R-109, Machine Intelligence Research Unit, University of Edinburgh (December 1974).

5. D. H. D. Warren, "WARPLAN: A System for Generating Plans," Memorandum No. 76, Department of Computational Logic, University of Edinburgh (June 1974).

6. P. E. Hart, "Progress on a Computer-Based Consultant," submitted to IJCAI4.

7. E. D. Sacerdoti, "Solving Problems and Monitoring Execution using a Highly Structured Representation for Actions," forthcoming Ph.D. thesis, Stanford University.

8. R. Reboh and E. D. Sacerdoti, "A Preliminary QLISP Manual," Technical Note 81, Artificial Intelligence Center, SRI, Menlo Park, California (August 1973).

9. G. J. Sussman, "The Virtuous Nature of Bugs," Proc. AISB Summer Conference (July 1974).

10. Z. Manna and R. Waldinger, "Knowledge and Reasoning in Program Synthesis," Technical Note 98, Artificial Intelligence Center, SRI, Menlo Park, California (November 1974).

11. D. G. Bobrow and B. Raphael, "New Programming Languages for Artificial Intelligence", Computer Surveys, Vol. 6, No. 3 (September 1974).