DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

DAI Working Paper No. 125
Date: October 1982

Title: A Retrospective on the "Planning: a joint AI/OR Approach" Project

Authors: Lesley Daniel and Austin Tate

Contents

A retrospective on the "Planning: A Joint AI/OR Approach" Project

Lesley Daniel and Austin Tate

This memo collects together material produced to report the outcome of the Science Research Council (now SERC) grant on "Planning: A Joint AI/OR Approach". It is being produced to respond to requests for further information on the final state of the project (which ended in 1978) and the task domain being tackled. The task domain involved the planning of overall procedure for electricity generation turbines. The application part of the work was carried out in conjunction with the Central Electricity Generating Board OR group. This part would be called a "Generative" Expert System if it was written today. For further detail, the final report should be read in conjunction with DAI memo 25 "Project planning using a hierarchic non-linear planner" by Austin Tate and DAI memo 24 "Planning: modifying non-linear plans" by Lesley Daniel.

Also as an outcome of the research, a proposal was made to SRC in 1976 to extend the hierarchic non-linear planning work and improve the use of meta-planning aids such as "Goal Structure" to construct an "Integrated approach to planning, plan execution and plan monitoring". The proposal was subsequently funded by SRC but the lead researcher had left DAI by that time. The proposal is left intact although pages 19 to 21 of the AI/CR project final report repeat some material. The Task Formalism examples in the proposal reflect an early version. DAI memo 25 reflects the adopted form.

An interesting (now widely recognised) link between AI non-linear planning techniques and the generation of scheduling information for cooperating parallel processers with a message passing system is explored in a short note included in this retrospective.

Further thought on the question answering system used in network (as opposed to tree structured) change levels to a word model are described in the next paper.

The final paper gives the user notes for the NONLIN planner and its critical path method package as it was used for the later stages of the project. These notes are included since they describe user interaction procedures to enable the task formalism descriptions for the domain to be built incrementally by the experts involved.

Final Report on Science Research Council Project

"Planning: a joint AI/OR Approach"

(B/RG/9445.5)[1]

Lesley Daniel[2] and Austin Tate[3]

1 The grant holder was Professor Bernard Meltzer

2 Now at The Open University
Milton Keynes

3 Now at Edinburgh Regional Computing Centre
59 George Square
Edinburgh

PLANNING: A JOINT AI/OR APPROACH

Final Report

1. Introduction

The aim of the project "Planning: a Joint AI/OR Approach" has been to investigate ways of aiding a human user in the process of constructing project networks. To this end, we have developed systems based on AI work on plan generation.

The planning system NOAH developed by Sacerdoti (1975a, 1975b) for the computer based consultant project at Stanford incorporated novel ideas for the representation of a plan as a partially ordered network of actions (a procedural net). This is in contrast to previous work which concentrated on the generation of linear sequences of action, e.g. STRIPS (Fikes and Nilsson, 1971), LAWALY (Siklossy and Deussi, 1973), INTERPLAN (Tate, 1974) etc. Knowledge about a domain is given to NOAH in a language SOUP to explain the decomposition of high level tasks into more detailed lower level actions.

Work at Edinburgh has investigated the use of partially-ordered networks of actions to represent a plan at any stage of development. Such networks are in a suitable form for the use of critical path analysis techniques having only those ordering constraints imposed by the fact that either

an action achieves a condition for a subsequent action, or
an action interferes with an important effect of another action and must be removed outside its range.

We have adopted Sacerdoti's philosophy of hierarchical planning where planning proceeds in stages at progressively greater levels of detail and, at each stage, the current plan is represented as a graph where the nodes represent actions (or goals to be achieved) and the edges ordering relations (links) between them. The graph is refined by expanding each node into a more detailed sub-network of nodes and adjusting the orderings accordingly.

In the first phase of the project, we were concerned with the development of a general hierarchical planning system; our intention has been not only to write programs which perform well in a particular problem domain but to structure the programs so that the various aspects of the planning process can easily be identified. Accordingly the problem was attacked under several headings:

(i) Task Formalism a formalism for defining a hierarchy of actions and goals-

Planner a program which uses the task formalism to generate plans in the hierarchic manner described above.

(iii) Optimization Procedure since we are interested in practical problems considerations of efficiency are important and the planner must produce cost effective plans. Accordingly, we have developed criteria for choosing efficient alternatives.

(iv) Modifying Plans a decision-graph has been implemented to record the relationship between decisions made in generating a plan and allow appropriate modification of the plan to recover from failure.

In the second phase of the project the general planning system was adapted to cope with a practical application to the problem of repeatedly generating plans for the annual overhaul of power-stations.

2. Task Formalism

At the outset of the work the problem of specifying a domain to a problem solver in a hierarchic fashion was recognized as being of primary importance and a uniform and straightforward method of description was sought.

The formalism allows high level definitions of a task to be given; each part of which can be expanded into lower level descriptions and so on down to some arbitrary level which the user of the program requires as output. Each lower level component can be specified in a modular way - not requiring knowledge of the exact form of the other components.

The specification of an action must specify how it may be expanded into more detailed sub-actions (there may be alternative expansions for an action e.g. different methods for installing electrical wiring) and how the constituent actions relate to each other. Rather than explicitly expressing ordering constraints in terms of precedence relations between actions, the task formalism allows specification of the conditions which must hold before an action can start (e.g. the walls of a house must be finished before the roof can be built) and the changes an action makes to the world (i.e. making some conditions true and some false) leaving the planning system to deduce feasible ordering relations. Thus, the task formalism allows individual actions to be specified independantly of other actions in the plan.

The task formalism allows the specification of a hierarchy of actions in terms of schemas (called opschemas) for an expansion which specify:

(i) pattern the pattern of an opschema determines for which actions the expansion is suitable.

(ii) expansion the constituent actions (or goals) are specified as a partial orde

(iii) conditions the conditions which are required by the constituent actions.

(iv) effects the changes which are made to the world model.

3. Planner

The planning system uses a hierarchic specification of the problem domain (expressed in the Task Formalism) to plan at progressively more refined levels of detail. At each level the plan is represented as a graph where the nodes represent actions (or goals) and the edges ordering relations between them. Each node has associated with it:

(i) the node type action, goal, phantom (a goal which has been achieved in another part of the plan).

(ii) pattern the expansions for a node are found by matching its pattern against the patterns of opschemas.

(iii) node context contains the effects of the node.

(iv) parentnode the node was inserted as a result of the expansion of its parentnode.

(v) prenodes a list of nodes linked immediately before this one.
succnodes a list of nodes linked immediately after this one.

Two other data structures are built by the planner:

TOME - the table of multiple effects stores information about facts made true or false at different nodes.

GOST - the goal structure records the conditions on nodes along with the contributors to a condition (those nodes which make the pattern true). The goal structure thus specifies a set of time "ranges" for which the truth of certain conditions must be maintained. The use of goal structure in problem solving was first described by Tate (1974, 1975).

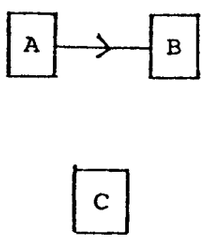
The planning cycle consists of expanding nodes in the network - i.e. replacing a high level action by a subnetwork of more detailed actions - and then looking for interactions with other parts of the plan.

Goal nodes are expanded by first looking to see whether the goal is already achieved by some other action in the plan and, if necessary, adding links to make the goal true at the point required (e.g. in overhauling machinery it is necessary to remove the cover to gain access to a component and it is worthwhile looking to see whether an action to remove the cover has already been included because of a repair to another component, the node is then made a phantom node and a record of this fact kept by adding a phantom condition to the GOST. If the goal node cannot be made a phantom it is expanded, in the same way as an action node, by finding an expansion

opschema whose pattern matches the pattern of the node.

The expansion of a node causes the introduction of new conditions (on the new nodes) and new effects and the planner seeks to ensure that the conditions are satisfied and the effects do not interfere with conditions on other nodes in the network. It may be necessary to add new links to the network to remove an endangered condition out of the range of a node which denies the condition (e.g. in a housebuilding task, it is necessary to have access to brickwork before electrical wiring can be installed, plastering the brickwork denies this access and cannot be done until after the wiring has been installed).

In general, an interaction involves three nodes (A, B and C below).



If action A achieves a condition required by action B and made untrue by action C, an interaction occurs unless C is already a predecessor of A or a successor of B. The interaction can be resolved by adding a link to ensure that C is outside the range of the condition in question and three cases must be considered:

(i) the interaction is totally unconstrained because C is parallel to both A and B - in this case, the interaction can be resolved either by linking C as a predecessor of A or as a successor of B.

the interaction is constrained because C is already linked as a successor of A - the interaction must be resolved by linking C as a successor of B.

(iii) the interaction is constrained because C is already linked as a predecessor of B - the interaction must be resolved by linking C as a predecessor of A.

4. Efficiency Criteria

Since we are interested in practical problems considerations of efficiency are important and the planner must produce cost-effective plans. A preliminary step was to determine criteria for judging the efficiency of alternative plans. Such plans have a set of constituent actions each with cost of execution and duration. A suitable measure of the efficiency of

a plan must trade-off the total cost of the constituent actions against the overall duration of the project. In many cases it is easy either to assign an overhead cost to the duration of the project or a penalty cost to any delay beyond a fixed duration. For example, if the project in question is the overhaul of a power station, the cost of having the station out of action can be measured as the additional cost of generating power by a less efficient alternative. For example, the overhead cost per day may be greater for a modern nuclear power station than for an old coal-fired station. It is very common for building contractors to insert penalty clauses in their contracts whereby they undertake to make penalty payments for delays beyond an agreed completion date.

One possible formulation of the problem of generating an efficient plan is that of choosing a set of activities P which will perform the specified task and minimise the objective function

$$C = \sum_{i \in P} c_i + kT$$

where c_i is the cost of the i^{th} job

k is the overhead cost per day

T is the duration of the project in days

There have been several studies (Crowston & Thompson) which have assumed the choices of alternative actions can be incorporated into a single graph which allows alternatives to have different predecessors and, in previous work (Daniel, 1974) we have considered ways of searching such a graph for a good solution.

We now consider such an approach to be inadequate for the following reasons:

(i) The complexity of carrying along alternatives is too great because of the combinatorial explosion of interactions between different actions.

(ii) The representations (e.g. and/or graphs) proposed for networks with alternatives are inadequate for the complexity of the problem.

An alternative approach to the problem of choosing efficient plans suggests that plans can be generated at different levels of detail and that, at each level in the hierarchy choices be made between different actions and different ways of resolving interactions. Thus no alternatives are kept in the network as it is expanded to the next most detailed level, but efficiency criteria are considered when choices are being made.

5. Modifying Plans

At various stages in the planning process choices are made between alternatives. However well organized the planner, there is always the possibility that a wrong choice will be made resulting in either an infeasible or a very inefficient solution. Recovery from such a failure involves identification of the decision responsible for the failure and appropriate modification of the current plan. In order to facilitate such modification procedures, a separate structure called a "decision graph" has been implemented to record the relationships between decisions made in generating a plan.

Hierarchical planning corresponds to choosing between alternative plans at each level in the hierarchy. Because, at each level, the planner is working on a complete plan, global information is available to direct the choice and, moreover, because high level plans have relatively few actions, it is possible to make a comprehensive investigation of the search space for that particular level. A planner which explored the whole search space at each level of detail and did not back-track between levels would have a complete search space as shown in fig. 5.1.

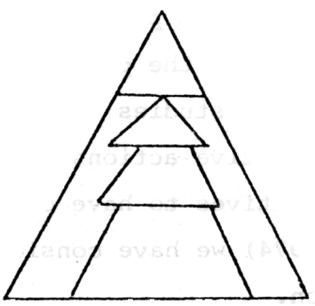


fig. 5.1

The expectations of such a system are:

- It is possible to explore all alternatives at each level of detail in order to yield the 'best' plan according to some criteria.
- The 'best' plan at a particular level will always generate a good plan when examined in more detail - i.e. it is not necessary to back-up to higher level choice points.

In fact the search space at a particular level is often large and, since, high level plans must be inaccurate, the decisions made at these levels sometimes prove to be wrong and must be reversed. Consequently, in designing a search strategy it is necessary to consider how to structure the search within a particular level in order to get a good solution without necessarily searching the whole space and how to recognize when it is necessary to back-up to a decision made at a higher level of detail

The decision-graph based on Hayes' (1975) work on a journey planner allows the system to back-up and undo decisions made at higher levels without throwing away all the work done since the decision was made. Only the affected parts of the network are modified when a decision is undone.

The basic assumption behind the structure of the decision graph is that the decisions made in generating a plan are of two types:

(i) Choice of expansion for a node.

(ii) Choice of links to remove an interaction.

Such choices are interrelated inasmuch as an interaction and consequent link may depend upon a particular choice of expansion. The purpose of the decision graph is to record such relationships. Every node in the plan (net node) points to a node in the decision graph (d-node) corresponding to the expansion which introduced it. Every time an expansion (or phantom) is made, a node is set up in the d-graph with pointers to:

(i) the net node being expanded.

(ii) the parent d-node - i.e. that which introduced the net node being expanded.

(iii) the new net-nodes introduced by the expansion.

(iv) any subsequent expansions of these nodes.

(v) any interaction d-nodes consequent on this expansion.

(vi) any phantom d-nodes corresponding to phantom links using a pattern achieved by the expansion.

(vii) in the case of a phantom - any d-node corresponding to a phantom link required to establish the phantom goal.

Every time a link is introduced to enable the creation of a phantom net-node, a d-node is introduced with pointers to:

(i) the net-node and corresponding d-node made into a phantom.

(ii) the net-node and corresponding d-node achieving the pattern.

Every time a link is introduced to remove an interaction, a d-node is introduced with pointers to:

(i) the link being introduced.

(ii) the netnode and corresponding d-node which needs the condition involved in the interaction.

(iii) the netnode and corresponding d-node which achieves the condition in question.

the netnode and corresponding d-node which deletes the condition in question.

The only way in which a plan can be modified is to undo one of the two kinds of decision involved in the plan - i.e. to remove an expansion or a

link (either phantom link or an interaction link).

Removing a phantom link is very straightforward only requiring that the netnode in question be reinstated as a goal node and that the GOST and decision-graph be modified accordingly. Removing an interaction link involves either choosing an alternative to remove the interaction or undoing an expansion which introduced one of the actions involved in the interaction.

The steps involved in undoing an expansion are:

step 1 Trace down the decision-graph and the netnodes (and their expansions if any) and links dependant on the expansion.

step 2 Remove these nodes and links and replace by the single parent netnode.

step 3 All remaining links which connected to any removed netnodes are now attached to the parent netnode.

step 4 The TOME and GOST are modified for every deleted netnode as follows:

(1) remove all conditions and effects which are needed/achieved by the deleted node and not the replaced node.

(2) remove them as contributors to any condition which is not achieved by the replaced node.

step 5 For all conditions which now have no contributors, undo the expansion which was dependant on it (i.e. phantom nodes replaced by goal nodes).

step 6 Look for interactions arising from the removal of links from the graph.

More detailed descriptions of the task formalism and planning system (NONLIN) can be found in Tate (1976) while the optimization schema and decision graph are described in Daniel (1977) and Daniel (1978).

6. A Practical Application

6.1 Introduction

In the second phase of the project we sought to apply the ideas developed in the first phase to a practical problem in a particular domain. For a practical application involving the generation of networks of the size usually used for critical path analysis techniques (several hundred activities) it was important to avoid large search spaces. We looked for an application where a large class of networks were required; each member of the class having great similarities with all the others but important differences which required a new network to be generated for each individual project.

The advantages of such a problem domain are

(i There is enough historical data of plans previously generated to allow an action hierarchy to be set up.

The choice of plans to be generated is sufficiently restricted to keep the seach space under control.

(iii) The difference between individual plans is sufficient to make it worthwhile to have a system which aids in their generation.

The chosen application was the annual overhauling of power-stations where each annual overhaul is different from previous years but still has the same overall structure. Another possible application might be the construction of a set of houses of a particular type but where, in each individual case, there are choices to be made between restricted alternatives about such things as type of heating, type of flooring, type of bathroom fixtures, etc.

6.2 The problem domain

Within the large task of power-station overhaul we selected the sub-project of overhauling a turbine as being the size of problem the planning system should cope with. The networks involved usually have more than five hundred activities but the problem is highly structured in the following ways.

(i) A turbine consists of 4 sections: high pressure, intermediate pressure and two low pressure sections.

(ii) Overhauls on each of the four sections can proceed independently of each other except for one alignment check along the whole length of the roter.

(iii) The overhaul of a section comprises the removal of layers of covers and the roter, giving access to various components in the process, and replacement of parts as access is no longer required.

(iv) An individual overhaul varies from previous ones inasmuch as different components may or may not be overhauled and there are different types of overhaul for each component. The choice of which type of overhaul is to be done on each component (or whether it is to be overhauled at all) is totally at the discretion of the user and the planner has merely to work out the consequences of each decision - e.g. ensuring that access is gained.

It seemed natural to define the action hierarchy in terms of three levels giving networks of about 70 nodes, 300 nodes and 750 nodes. When using the task formalism to specify opschemas only those conditions and effects were mentioned which were involved in interactions. These were relatively few at each level of detail since in most cases the ordering within expansions, which was passed down to lower level networks, took care of most potential interactions. In this particular application there

are no choices to be made between alternative expansions (such choices are decisions to be made by the user of the system) and in all cases, interactions can only be removed in one way. Consequently, the process of generating the network is completely deterministic and there is no need to consider search strategies or decision-graphs.

6.3 Satisfying conditions and removing interactions

The task formalism allows several types of condition to be specified, but the two which were useful in this application were

- supervised conditions - these conditions which are achieved within the expansion containing the node for which they are needed - i.e. their contributor is specified by the opschema.
- unsupervised conditions - these conditions which are expected to be achieved by an action in another part of the plan - i.e. their contributors are not specified and must be determined by the planner.

Ensuring that unsupervised conditions are satisfied often involves adding links to the network thus constraining later attempts to remove interactions or satisfy conditions. Consequently, it is important that such conditions be satisfied as soon as possible. In the latest planning system unsupervised conditions are satisfied at every level before proceeding to the next most detailed level.

For the same reason, constrained interactions are removed immediately while those which can be resolved in two ways* are only removed when all nodes at one level have been expanded.

6.4 Achieving Goals

The planning system now distinguished between goal nodes, action nodes and query nodes. Query nodes represent those goals (e.g. <<high pressure section overhauled>>) whose inclusion is at the discretion of the user. Every time a query node is expanded the system interrogates the user as to whether this particular goal should be achieved. If the answer is no, the system behaves as though the goal had already been achieved by making the goal have value true in the initial state. The node-type is then changed to a goal node and the planning system tries to achieve the goal. If the goal is true in the initial state (i.e. the user does not want this task to be done) node will be made into a phantom node - otherwise an expansion will be to include actions to achieve the goal.

---

* In this particular application those interactions which at first appear to be unconstrained become constrained as links are added to remove other interactions. In such cases, they are removed as soon as they become constrained.

In general, an expansion will include several goal nodes which are used to avoid redundancy in cases where several actions share preconditions For example, two different components A and B may be located under the same cover.

If A is to be overhauled then it will not be necessary to remove the cover again for B. However, if A is not included then an action must be introduced to remove the cover for B. Thus, the opschemas for B will have the following expansion:

1 goal <<Remove top cover>>

2 action <<Overhaul component B>>

3 goal <<Replace top cover>>

orderings 1 - - → 2 2 - - → 3

The actions to remove and replace the cover should naturally be included in the same network as the action to overhaul B. In the planning system the two goal patterns are marked as having the same level as that of the pattern currently being expanded. In this way they will both be expanded (i.e. the goal replaced by actions) before the action to overhaul B is expanded.

6.5 The planning cycle

The planner operates at distinct levels (in this case three) and tries to tie up as many loose ends as possible before going on to the next level of detail. Each pattern is marked with a level and only those whose level is equal to the current one will be expanded. For a particular level the cycle is as follows:

step 1 Increment net level to current level.

step 2 Expand a node whose level is equal to current level. If there is none go to step 8.

step 3 If there are any query nodes in the expansion interrogate user and act accordingly.

step 4 Introduce new nodes into network.

step 5 Try to satisfy unsupervised conditions - if any cannot be satisfied keep a note.

step 6 Look for interactions involving nodes in the current expansion. All constrained interactions are removed immediately - unconstrained interactions are stored to be removed when they become constrained or at the end of the cycle.

step 7 go to step 2.

step 8 Remove any outstanding unconstrained interactions*.

step 9 Go to step 1.

6.6 Conclusion

We have set up the action hierarchy for the overhaul of a turbine for a power-station run by the South of Scotland Electricity Board (part of this hierarchy is described in Appendix I) and the planning system operates effectively to produce networks of the type in current use.

As stated above, there is no requirement, in this particular application for the planning system to make any choices since:

- choices between alternative expansions are made by the user - i.e. in specifying the exact type of overhaul that he wishes to have performed.
- all interactions become constrained at each level of detail - either by the time they are identified or by the introduction of links to satisfy unsupervised conditions before they are resolved. (In the current implementation, unconstrained interactions are not resolved until the very end of each level of expansion.)

However, it is reasonable to expect that in very similar applications, (e.g. different types of overhaul or construction tasks) the problem of choices will arise. Limited choices between alternative expansions might arise because there is more than one way to perform a task (one being fast and expensive, the other being slow and cheap) the "best" choice being dependent on the criticality of the task in question - i.e. its relationship to other parts of the project.

Again it is easy to envisage occasions where choices between alternative ways of resolving interactions need to be made by the planning system.

The considerations to be taken into account when making such choices can best be illustrated by an example for a house-building task. When the network shown in Figure 6.1 is examined for interactions the following conditions and effects are discovered.

---

* In general this step may involve alternatives but in this particular application no interactions remain unconstrained at the end of the cycle.

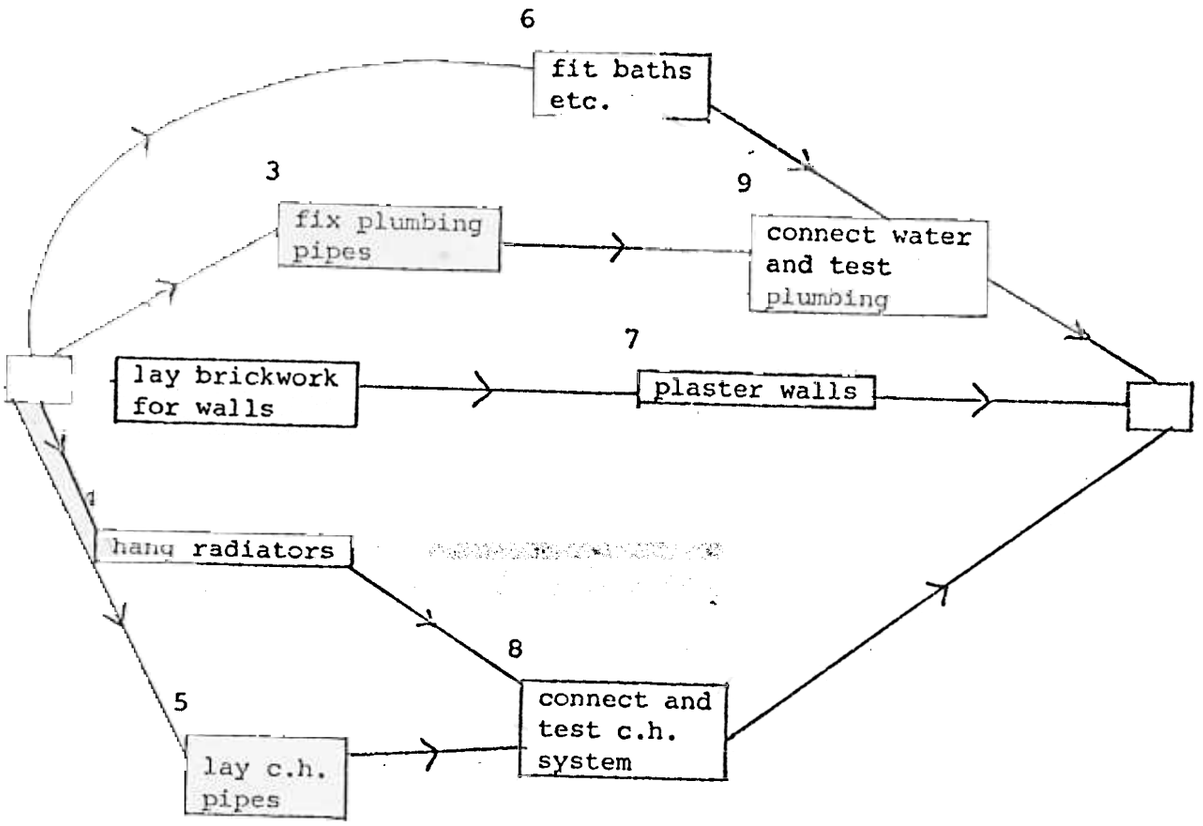


Figure 6.1

Conditions

Supervised condition <brickwork laid> needed at 3
<walls plastered> " at 4
Unsupervised <brickwork bare> " at 3

Effects

add <brickwork laid> by 1
<brickwork bare> by 1
<walls plastered> by 7
delete <brickwork bare> by 7

A sensible planner would satisfy the supervised conditions (which involve no choices) before resolving the interaction between nodes 3 and 7. When the supervised conditions are satisfied the network is as in Figure 6.2.

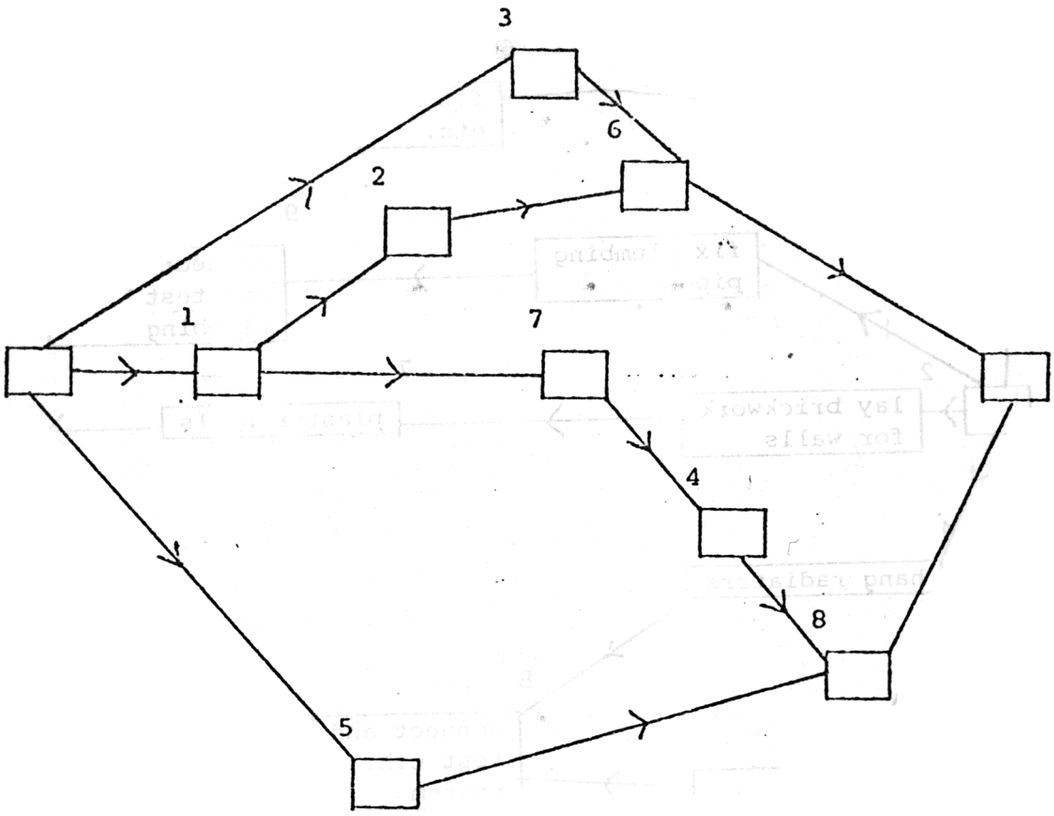


Figure 4

There is now an interaction because the unsupervised condition <brickwork bare> needed by node 3 is deleted by node 7. The interaction can be resolved in two ways:

either, add a link which means 7 can't start until node 3 has finished and 7's predecessors will be delayed

or, link 7 before 3 and expect that the condition must now be established by the addition of a new action into the network.

Thus, we see that the problem of choosing how to resolve the interaction involves trading off possible delays in the project against the cost of introducing additional actions. However, if no radiators were being fitted, the alternative resolution would be more satisfactory.

The planner in use (NONLIN) has the ability to cope with this and the present turbine overall system needs only minor modification to cope with such choices and in another report (Daniel 1978) we describe a search-strategy for such applications.

7. Suggestions for further work

7.1 Alternative applications

We have described in section 6.6 how the problem of choices might arise in an application and it would be worthwhile to modify the present system to handle a problem domain such as house construction where such situations might be expected.

7.2 Specifying Action Hierarchies

In order that the planning program can generate plans at different levels of detail the data base must contain hierarchical descriptions of the actions involved in the task. Such descriptions must specify

1. the conditions which must hold before an action can be performed
2. the effects that an action has on the world, and
3. the way in which actions can be decomposed into more detailed sub-actions.

The current program accepts domain descriptions in a formalism which allows hierarchical descriptions of tasks which incorporate the items above. However, the program usually expects such task descriptions to be pre-defined and has only crude facilities to guide the user through the process of defining such a data base.

There are two aspects on which such a user would require guidance.

1. Which effects and conditions need be mentioned for actions described at each level of detail.
2. How to aggregate and describe actions to form a useful hierarchical representation of a domain.

For the small problem domains which traditional planning programs have handled (e.g. block-stacking, pushing objects around a maze of rooms) the specification of actions has been very straightforward. The world is so small with so few objects and locations that the action schemata can describe all the changes made to the world without being too large. Moreover, the action schemata have usually been completed by the programmer - i.e. by an expert in computer science who understood the requirements of the planning program.

Action schemata for a realistic domain with a large number of actions, objects and possible changes must be provided by a number of users rather than a programmer. In general such people will have expertise in the particular task in hand (e.g. overhauling plant, building roads) but no knowledge of the work of the planning program. Each user supplying information to the system must select, from all the possible things which could be said about an action, which are the important conditions and

effects - those which are likely to cause interactions with other parts of a plan used to perform some task in the domain being described.

Such a process is combinatorially explosive inasmuch as it may involve comparing one action with all other possible actions and must be controlled by the program in such a way that the user is guided through the hierarchy of actions in a systematic fashion which makes apparent the important conditions and effects.

If the hierarchy of actions is to enable the planner to plan efficiently then the constituent actions must be organized to meet certain conditions. The planner uses high-level plans, with aggregated actions, to make choices between alternative actions. If such choices are to be sensible then the high-level plans give fairly accurate representations of their more detailed descendants.

Also, each high-level aggregate can, in general, be decomposed into sub-actions in several different ways. If it is to be usefully included in a high-level plan, the aggregate's descriptions must be fairly representative of each of its possible decompositions.

A high-level action is an approximate representation of a group of actions not only because it represents several possible decompositions but also because its description contains only part of the detail of the conditions and effects which apply to its constituents. The point in the hierarchy at which a condition or effect is introduced has serious implications for the efficiency of the planning program inasmuch as it determines when interactions will be detected. If interactions are found too soon the plan will be overconstrained while if they are found too late the high-level plans will not accurately reflect the possible plans at a lower level.

The program must help the user to define hierarchies of actions in which the more detailed actions are grouped into representative aggregates and the descriptions at each level are of appropriate detail.

An important extension of the work reported here would be a program enabling a user to describe actions in the task formalism by prompting him in the following manner:

(i) An expansion of a high-level action into a network of sub-actions should be called for.

(ii) Conditions on the sub-actions should be called for in terms of possible interactions with other activities at the same level of detail.

We expect that the user will think in terms of interactions between actions and will then be prompted by the system to explain such interactions in

terms of conditions and deletions.

(iii) The important effects of actions should become explicit in the same way as conditions.

(iv) We expect that a user will be continually modifying his specifications as the consequences become apparent (e.g. the definition of an unsupervised condition reduces the number of possible plans and the user may wish to increase the choice by making a condition achievable). The program should be able to cope with such modifications and do the appropriate housekeeping.

7.3 Use of intermediate representations of a plan for resource allocation

Hierarchical planning is a way of reducing the search space involved in practical problems by planning at different levels of detail and, in this section, we consider the ways in which the intermediate representations of the network can be used for associated optimization problems.

An important problem associated with network planning is that of scheduling the activities in the plan to satisfy resource constraints. In general, the resource constraints are not taken into account when the plan is generated and the precedence relationships represent technological orderings between actions - e.g. the walls of a house cannot be built until its foundations are laid. In fact the management of most projects has to take into account fixed manpower ceilings and the availability of a limited number of machines or other pieces of equipment. Jobs occurring on parallel paths through the network may compete for the same resources and even though the precedence relationships allow them to be done in parallel resource constraints might force them to be scheduled sequentially.

Scheduling activities to satisfy such resource constraints is such a large combinatorial problem that, for practical applications, only heuristic methods can be used. In general such algorithms correspond to depth first search through the search tree where the heuristics are used to choose the best branch at each stage. In such a program the critical path information gives some measure of how a job fits into the whole plan and allows scheduling for each individual job to take account of some global considerations. However, no account can be taken of the fact that some combinations of jobs can be chosen which use up all resources while others waste resources on certain days and consequently delay the project.

An alternative approach is to use a high-level representation of the plan and find a schedule which satisfies the resource constraints and minimises the overall duration of the project. The high-level plan has relatively few constituent actions, so all schedules can be explored and the "best" schedule can then be refined by scheduling the jobs at the next level

of detail within the constraints imposed by the high-level schedule. If the high-level plan is very inaccurate the resultant schedule will be poor and we might expect that a hierarchical approach will only give good results in cases where the aggregates are fairly homogeneous in terms of resource requirements and critical path data. This hierarchical approach to resource allocation would mesh well with our general planning philosophy.

An interesting use of hierarchical planning is included in the SPADOR program (Pease, 1977) for planning missions from aircraft carriers. The task handled by the program is that of answering a request to plan an individual mission within a background of several previously planned and ongoing activities. The constituent actions of each mission are pre-determined and stored in the data base. However, each mission requires several resources such as aircraft, pilots, technicians and maintenance crews and the problem is to make assignments from the pool of resources which meet the deadline of the particular mission and are compatible with the availabilities of each resource. In this case the requirement is not to optimise the resources needed for the mission but to find a feasible set of assignments.

The program tackles the problem in a hierarchic manner by using several representations of a mission at different levels of detail. At each level of detail only those resources are considered which are required for the whole of an action (or group of actions) corresponding to that level. Planning proceeds by taking a high-level representation of a mission and making assignments for all resources mentioned at that level. Such assignments fix activities within certain time-limits and, consequently, constrain the assignments which can be made when the mission is expanded into more detailed activities

The SPADOR project is concerned with repeatedly planning relatively simple tasks against a background of many activities but such an approach can be extended to large projects where efficiency considerations are more important.

The traditional formulation of the scheduling problem for large projects is of allocating resources from pools (such as different types of manpower) which are continuously available throughout the whole of the duration of the project. A different situation occurs when an individual project must be scheduled against a background of ongoing activities (all making calls on the same pool of resources). In such a case a particular resource is, in general, not available for the whole time period but only for certain intervals. The assignment of a particular piece of equipment to a job within the project corresponds to fixing that job within a time

slot (with corresponding constraints for other jobs within the network).

In the first case a simple depth-first search strategy (without backtracking) is adequate because it always terminates in a feasible (though not necessarily the most efficient) solution. In the second case, feasibility cannot be guaranteed and backtracking cannot be avoided.

We suggest that, for such problems, a hierarchical search strategy is appropriate and Pease's work might be usefully extended in the following ways:

(i) To design a search algorithm which will give efficient assignments of resources (because Pease's tasks are small he has not worried about efficiency).

(ii) To investigate the structure of the hierarchical search space with a view to making intelligent recovery from failure when a set of assignments prove to be infeasible. The hierarchy should facilitate recovering in a more sophisticated way than by simply backtracking to the most recent choice-point. This can be compared with our use of the Decision Graph (Daniel, 1977) to recover from planning failures.

7.4 Monitoring the execution of plans

The most ambitious planning, plan execution and monitoring system to date has been the STRIPS/MACROPS/PLANEX system developed at SRI (Fikes and Nilsson, 1971; Fikes, Hart and Nilsson, 1972; and Fikes, 1971) to enable the robot SHAKEY to perform plans generated by STRIPS. The interface between the planner (STRIPS) and the execution monitor or run-time system (PLANEX) was a triangular table representation of a plan (a MACROP). STRIPS worked in a very simple domain and could generate only very short plans in a relatively long planning time (see Tate, 1974, for a comparison of several problem solvers). Recent advances in planning techniques allow us to suggest a planning and monitoring system of wider scope and greater flexibility.

In NONLIN, the necessary conditions on actions in a plan and the points in the plan where they are achieved are represented in a GOST (goal structure). Goal structure contains information which is more structured than the detail of the plan itself. It has proved á valuable tool for detecting and correcting for interactions in a plan and directing the search of a planner (Tate, 1975; Sacerdoti, 1975b; and Tate, 1976). A GOST also gives the information used from a MACROP by PLANEX and can thus replace it for execution monitoring purposes. The information in a GOST is such that we need only check that an action does not alter a GOST entry for any succeeding action to know that the rest of the plan can be achieved. Only if any condition on a succeeding node

is altered is the plan in jeopardy. Since conditions are attached to actions in a plan, it is possible to specifically ask if each monitorable condition holds before the action is executed.

The use of goal structure as the basis of the plan monitoring scheme has several advantages from the point of view of the run-time system where time and program space are at a premium. Specifically monitoring certain conditions before an action is performed is more practicable in a working environment than asking that an accurate world model be kept after the execution of each action (as is needed for the MACROPS/PLANEX scheme). For a run-time system with no feed-back available from the domain, the condition types available in the goal structure of the current version of NONLIN may be sufficient to monitor the execution of the plan (the system stopping if any condition was violated). For more advanced systems where sensory information from a domain is available to the run-time system which can be used to choose between several contingencies in a plan, further TESTABLE condition types will be necessary together with the associated plan-time facilities to allow conditional branches and rejoins in the plan being generated.

There have been very few attempts to produce plans which can have conditionals in them. One typical approach is that described in Warren (1976) where case analysis on the branches of a conditional is performed and conditional plans branch but do not then rejoin. The structured way in which a conditional would be added to a plan when using a hierarchical representation of a domain should allow the generation of conditional plans which branch and rejoin. We believe our algorithm for question-answering in a partially-ordered network (Tate, 1976) can be modified to cope with networks containing conditionals.

Useful further work would be the provision of a re-planning ability based on the GOST conditions detected as not holding before some action and on the remaining unexecuted part of the plan at that point.

The current plan modification system is designed to cope with unexpected failures during the generation of the plan. If, at some stage during planning, failure occurs the responsible decision can usually be identified. Using information in the Decision Graph, the plan can then be modified by undoing the decision in question and revising the actions affected without changing the remainder of the plan. Some modifications usually take the form of removing a block of actions corresponding to a

particular expansion of a high-level action, undoing any dependant ordering links and replacing the high-level action.

In the case of monitoring the execution of plans the situation is rather different. A failure may occur at some stage during the execution of the detailed plan when some actions have terminated, some are in progress and some are not yet started. The failure is identified as the inability to start some particular job as expected and it then becomes necessary to re-enter the planner in order to correct the plan.

Such a recovery procedure will differ from the existing plan-time modification procedure in the following ways.

(i) In general, the situation can be retrieved either by satisfying some unexpectedly unsatisfied condition and then continuing as before or, in more serious cases, a part of the plan must be removed and replaced by different sub-actions. The recovery program must be able to identify standard types of failure and point to the appropriate remedy.

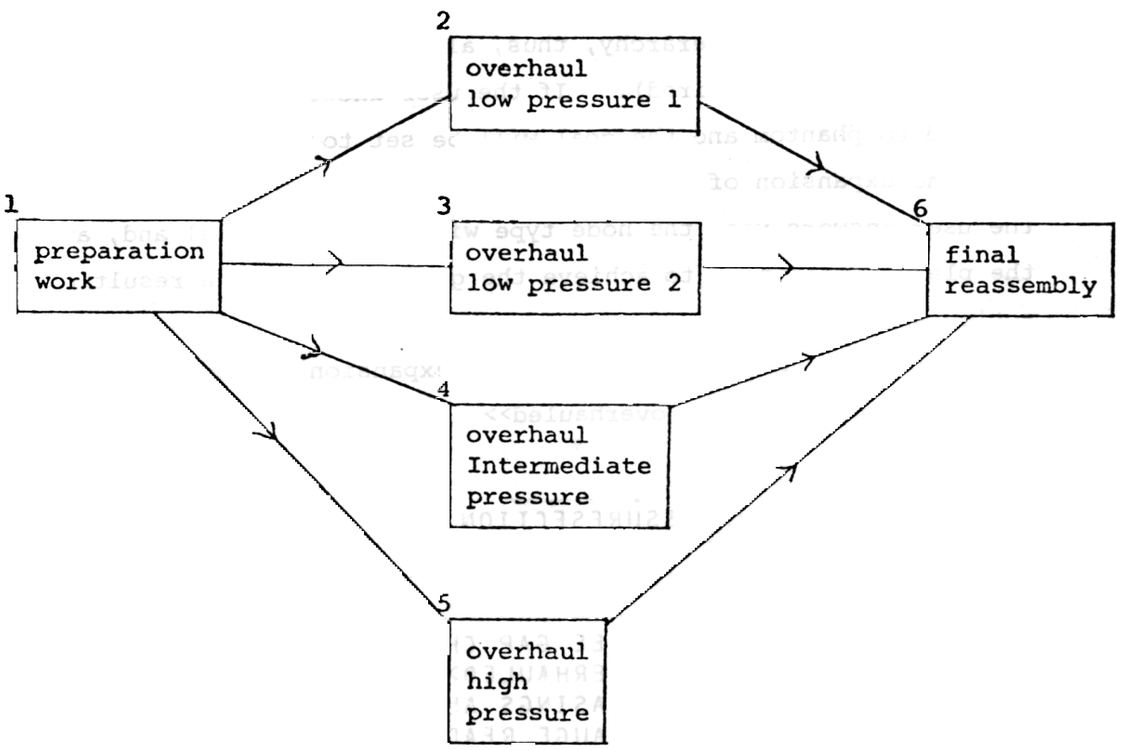
(ii) The replacement of some of the actions in the current plan is a more complicated procedure than the modifications required during the plan generation phase. The plan cannot be modified by simply replacing a group of actions corresponding to a high-level aggregate because, in general, the failure will have occurred part-way through such an aggregate. The recovery process must be able to use the goal structure of the plan to generate new patches which do not correspond to any decomposition in the action hierarchy.

Since a failure in execution is often the result of inadequacy in the definition of the actions in the plan, the appropriate action schema should be offered by the system for review, so that such a situation should not arise again

APPENDIX

In this section we illustrate the representation in the Task Formalism of part of the action hierarchy for turbine overhaul - the overhaul of the high pressure section.

The highest level network for turbine overhaul is shown below.



Such a network can be represented in the Task Formalism as an opschema:

```
opschema turbine
pattern <<turbine overhaul>>
expansion  1. action  <<preparation work>>
           2. query   <<low pressure 1 overhauled>>
           3. query   <<low pressure 2 overhauled>>
           4. query   <<intermediate pressure overhauled>>
           5. query   <<high pressure overhauled>>
           6. action  <<final reassembly>>
orderings 1 --→ 2  1 --→ 3  1 --→4  1 --→5
          2 --→ 6  3 --→ 6  4 --→6  5 --→6
end;
```

A call to the system

PLAN GOAL <<turbine overhaul>>

will cause the planning program to use the opschema described above to set up the initial high level network. The presence of query nodes causes the system to interrogate the user as to whether he wishes the corresponding actions to be performed i.e. does he wish to overhaul the intermediate pressure section in this particular overhaul (note that the query nodes can occur at every level in the hierarchy, thus, allowing refinement of the particular type of overhaul required). If the user answers no, the node type will be changed to phantom and the goal will be set to true in the initial state - thus, no expansion of this high level node will be made at any stage. If the user answers yes, the node type will be set to goal and, at some stage, the planner will seek to achieve the goal either as a result of some other action already in the plan or by introducing new actions.

Below are some opschemas for the expansion of the goal node <<high pressure section overhauled>>

```
OPSCHEMA OVERHAULHIGHPRESSURESECTION
PATTERN <<HIGH PRESSURE OVERHAULED>>
EXPANSION 1 QUERY <<TOP AND BOTTOM LEADS FITTED>>
          2 QUERY <<TOP GLANDS AND DIAPHRAMS OVERHAULED>>
          3 QUERY <<JOINT FREE GAP CHECKED>>
          4 QUERY <<ROTOR OVERHAULED>>
          5 QUERY <<BOTTOM CASINGS AND BOTTOM BEARINGS OVERHAULED>>
          6 QUERY <<BRIDGE GAUGE READINGS TAKEN>>
          7 ACTION <<FINISHED>>
ORDERINGS 1--->2 1--->3 1--->4 1--->5 2--->7 3--->7 4--->7 5--->7
CONDITIONS UNSUPERVISED <<TOP CASING REPLACED>> AT 7
          UNSUPERVISED <<ROTOR REPLACED>> AT 7
END;
OPSCHEMA FITTOPANDBOTTOMLEADS
PATTERN <<TOP AND BOTTOM LEADS FITTED>>
EXPANSION 1 GOAL <<TOP CASING REMOVED>>
          2 ACTION<<FIT TOP AND BOTTOM LEADS>>
          3 GOAL <<TOP CASING REPLACED>>
ORDERINGS 1--->2 2--->3
CONDITIONS SUPERVISED <<TOP CASINGS REMOVED>> AT 2 FROM 1
END;
```

```
OPSCHEMA OVERHAULTOPGLANDSANDDIAPHRAMS
PATTERN <<TOP GLANDS AND DIAPHRAMS OVERHAULED>>
EXPANSION 1 GOAL <<TOP CASING REMOVED>>
          2 ACTION <<OVERHAUL TOP GLANDS AND DIAPHRAMS>>
          3 GOAL <<TOP CASING REPLACED>>
ORDERINGS 1--->2 2--->3
CONDITIONS SUPERVISED <<TOP CASINGS REMOVED>> AT 2 FROM 1
END;
```

```
OPSCHEMA CHECKJOINTFREEGAPS
PATTERN <<JOINT FREE GAPS CHECKED>>
EXPANSION 1 GOAL <<TOP CASING REMOVED>>
          2 ACTION <<CHECK JOINT FREE GAPS>>
          3 GOAL <<TOP CASING REPLACED>>
ORDERINGS 1--->2 2--->3
CONDITIONS SUPERVISED <<TOP CASING REMOVED >> AT 2 FROM 1
END;

OPSCHEMA OVERHAULROTOR
PATTERN <<ROTOR OVERHAULED>>
EXPANSION 1 GOAL <<SHAFT REMOVED>>
          2 ACTION <<OVERHAUL ROTOR>>
          3 GOAL <<SHAFT REPLACED>>
ORDERINGS 1--->2 2--->3
CONDITIONS SUPERVISED <<SHAFT REMOVED>> AT 2 FROM 1
           UNSUPERVISED<<TOP CASING REMOVED>> AT 1
                "         "     "      "     AT 2
                "         "     "      "     AT 3
END;

OPSCHEMA OVERHAULBOTTOMCASINGSANDBOTTOMBEARINGS
PATTERN <<BOTTOM CASINGS AND BOTTOM BEARINGS OVERHAULED>>
EXPANSION 1 GOAL <<SHAFT REMOVED>>
            2 ACTION <<OVERHAUL BOTTOM CASINGS AND BOTTOM BEARINGS>>
            3 GOAL <<SHAFT REPLACED>>
ORDERINGS 1--->2 2--->3
CONDITIONS SUPERVISED <<SHAFT REMOVED>> AT 2 FROM 1
           UNSUPERVISED<<TOP CASING REMOVED>> AT 1
                "         "     "      "     AT 2
                "         "     "      "     AT 3
END;

OPSCHEMA TAKEBRIDGEGAUGEREADINGS
PATTERN <<BRIDGE GAUGE READINGS TAKEN>>
EXPANSION 1 ACTION <<TAKE BRIDGE GAUGE READINGS>>
CONDITIONS UNSUPERVISED <<SHAFT REPLACED>> AT 1
END;

OPSCHEMA REMOVETOPCASINGS
PATTERN <<TOP CASING REMOVED>>
EXPANSION  1 ACTION <<REMOVE TOP CASINGS>>
END;

OPSCHEMA REMOVESHAFT
PATTERN <<SHAFT REMOVED>>
EXPANSION 1 GOAL <<TOP CASING REMOVED>>
          2 ACTION <<REMOVE SHAFT AND CHECK LEADS>>
ORDERINGS 1--->2
CONDITIONS SUPERVISED <<TOP CASING REMOVED>> AT 2 FROM 1
END;

OPSCHEMA REPLACESHAFTANDCHECKALIGNMENTANDCLEARANCES
PATTERN <<SHAFT REPLACED>>
EXPANSION  1 ACTION << REPLACE SHAFT AND FIT LEADS>>
```

```
        2 ACTION <<ALIGNMENT CHECK>>
          3 ACTION <<CLEARANCE CHECK>>
ORDERINGS 1--->2 2--->3
CONDITIONS UNSUPERVISED <<JOINT FREE GAPS CHECKED>> AT 1
END;

OPSCHEMA REPLACETOPCASING
PATTERN <<TOP CASING REPLACED>>
EXPANSION 1 ACTION <<REPLACE TOP CASING>>
END;

OPSCHEMA OVERHAULTOPCOVER
PATTERN <<OVERHAUL TOP GLANDS AND DIAPHRAMS>>
EXPANSION 1 QUERY <<OUTER CASING CLEANED AND CHECKED FOR DISTORTION>>
          2 QUERY <<OUTER DIAPHRAMS OVERHAULED>>
          3 QUERY <<OUTER GLANDS OVERHAULED>>
          4 QUERY <<INNER GLANDS OVERHAULED>>
          5 QUERY <<INNER DIAPHRAMS OVERHAULED>>
          6 QUERY <<INNER CASING CLEANED AND CHECKED FOR DISTORTION>>
END;

OPSCHEMA FITTINGBOTHLEADS
PATTERN <<FIT TOP AND BOTTOM LEADS>>
EXPANSION 1 ACTION  <<FIT BOTTOM LEADS>>
          2 ACTION <<FIT TOP LEADS>>
ORDERINGS 1--->2
END;

OPSCHEMA CLEANOUTERCOVER
PATTERN <<OUTER CASING CLEANED AND CHECKED FOR DISTORTION>>
EXPANSION 1 GOAL <<OUTER CASING TURNED OVER>>
          2 ACTION <<CLEAN AND CHECK OUTER COVER>>
          3 GOAL <<OUTER CASING TURNED BACK>>
ORDERING 1--->2 2--->3
CONDITIONS SUPERVISED <<OUTER CASING TURNED OVER>> AT 2 FROM 1
END;


OPSCHEMA OVERHAULOUTERDIAPHRAMS
PATTERN <<OUTER DIAPHRAMS OVERHAULED>>
EXPANSION 1 GOAL <<OUTER CASING TURNED OVER>>
          2 ACTION <<OVERHAUL OUTER DIAPHRAMS>>
          3 GOAL <<OUTER CASING TURNED BACK>>
ORDERINGS 1--->2
          2--->3
CONDITIONS SUPERVISED <<OUTER CASING TURNED OVER>> AT 2 FROM 1
END;


OPSCHEMA OVERHAULOUTERGLANDS
PATTERN <<OUTER GLANDS OVERHAULED>>
EXPANSION 1 GOAL <<OUTER CASING TURNED OVER>>
          2 ACTION <<OVERHAUL OUTER GLANDS>>
          3 GOAL <<OUTER CASING TURNED BACK>>
ORDERINGS 1--->2 2--->3
CONDITIONS SUPERVISED <<OUTER CASING TURNED OVER>> AT 2 FROM 1
END;

OPSCHEMA CLEANINNERCOVER
```

```
 ATTERN <<INNER CASING CLEANED AND CHECKED FOR  DISTORTION>>
EXPANSION 1 GOAL <<INNER CASING TURNED OVER>>
          2 ACTION <<CLEAN AND CHECK INNER COVER>>
          3 GOAL <<INNER CASING TURNED BACK>>
ORDERINGS 1--->2 2--->3
CONDITIONS SUPERVISED <<INNER CASING TURNED OVER>> AT 2 FROM 1
END;


OPSCHEMA OVERHAULINNERDIAPHRAMS
PATTERN <<INNER DIAPHRAMS OVERHAULED>>
EXPANSION 1 GOAL <<INNER CASING TURNED OVER>>
          2 ACTION <<OVERHAUL INNER DIAPHRAMS>>
          3 GOAL <<INNER CASING TURNED BACK>>
ORDERINGS 1--->2 2--->3
CONDITIONS SUPERVISED <<INNER CASING TURNED OVER>>
    AT 2 FROM 1
END;


OPSCHEMA OVERHAULINNERGLANDS
PATTERN <<INNER GLANDS OVERHAULED>>
EXPANSION 1 GOAL <<INNER CASING TURNED OVER>>
           2 ACTION <<OVERHAUL GLANDS>>
           3 GOAL <<INNER CASING TURNED BACK>>
ORDERINGS 1--->2 2--->3
CONDITIONS SUPERVISED <<INNER CASING TURNED OVER>> AT 2 FROM 1
END;

OPSCHEMA TURNOUTER
PATTERN <<OUTER CASING TURNED OVER>>
EXPANSION 1 ACTION <<TURN OVER OUTER CASING>>
END;
OPSCHEMA TURNOUTERBACK

PATTERN <<OUTER CASING TURNED BACK>>
EXPANSION 1 ACTION <<TURN BACK OUTER CASING>>
END;

OPSCHEMA TURNINNER
PATTERN <<INNER CASING TURNED OVER>>
EXPANSION 1 ACTION <<TURN OVER INNER CASING>>
END;

OPSCHEMA TURNINNERBACK
PATTERN <<INNER CASING TURNED BACK>>
EXPANSION 1 ACTION <<TURN BACK INNER CASING>>
END;
OPSCHEMA CHECKLEADSANDREMOVESHAFT
PATTERN
         <<REMOVE SHAFT AND CHECK LEADS>>
EXPANSION 1 ACTION <<CHECK LEADS AND REMOVE SHAFT>>
          2 ACTION <<TAKE AXIAL CLEARANCES>>
ORDERINGS 1--->2
END;

OPSCHEMA OVERHAULBOTTOMBEARINGSANDCASINGS
PATTERN <<OVERHAUL BOTTOM BEARINGS AND BOTTOM CASINGS>>
EXPANSION 1 QUERY <<BOTTOM CASING GLANDS OVERHAULED>>
          2 QUERY <<BOTTOM CASING DIAPHRAMS OVERHAULED>>
```

```
          3 QUERY <<CASING JOINTS AND NOZZLE RINGS OVERHAULED>>
          4 ACTION <<NDT 1&2 BEARINGS BOTTOM>>
END;

OPSCHEMA BOTTOMCASINGGLANDS
PATTERN
      <<BOTTOM CASING GLANDS OVERHAULED>>
EXPANSION 1 ACTION <<OVERHAUL BOTTOM CASING GLANDS>>
END;

OPSCHEMA BOTTOMDIAPHRAMS
PATTERN <<BOTTOM CASING DIAPHRAMS OVERHAULED>>
EXPANSION 1 ACTION <<OVERHAUL BOTTOM CASING DIAPHRAMS>>
END;

OPSCHEMA NOZZLERINGS
PATTERN <<CASING JOINTS AND NOZZLE RINGS OVERHAULED>>
EXPANSION 1 ACTION <<OVERHAUL CASING JOINTS AND NOZZLE RINGS>>
END;

OPSCHEMA ROTOROVERHAUL
PATTERN <<OVERHAUL ROTOR>>
EXPANSION 1 ACTION <<CHECK ROTOR>>
          2 QUERY <<JOURNALS AND GLAND LABRYNTHS EXAMINED>>
          3 QUERY <<ROTOR REBLADED>>
          4 QUERY <<WORKS PROGRAMME COMPLETED>>
ORDERINGS 1--->2 1--->3 3--->4
END;

OPSCHEMA JOURNALS
PATTERN <<JOURNALS AND GLAND LABRYNTHS EXAMINED>>
EXPANSION 1 ACTION <<EXAMINE JOURNALS AND GLAND LABRYNTHS>>
END;
OPSCHEMA REBLADE
PATTERN <<ROTOR REBLADED>>
EXPANSION 1 ACTION <<REBLADE ROTOR>>
END;


OPSCHEMA WORKS
PATTERN <<WORKS PROGRAMME COMPLETED>>
EXPANSION 1 ACTION <<SEND FOR WORKS PROGRAMME>>
END;



OPSCHEMA REMOVALOFTOPCASING
PATTERN <<REMOVE TOP CASINGS>>
EXPANSION 1 ACTION <<REMOVE TOP OUTER CASING>>
          2 ACTION <<REMOVE TOP INNER CASING>>
           3 ACTION <<BGR 1&2 BEARINGS TOP>>
ORDERINGS 1--->2 2--->3
END;


OPSCHEMA JOINTFREEGAPS
PATTERN <<CHECK JOINT FREE GAPS>>
EXPANSION 1 ACTION <<CHECK JOINT FREE GAPS INNER CASING>>
           2 ACTION <<CHECK JOINT FREE GAPS OUTER CASING>>
ORDERINGS 1--->2
```

```
 CONDITIONS UNSUPERVISED <<INNER CASING WORK FINISHED>> AT 1
            UNSUPERVISED <<OUTER CASING WORK FINISHED >>AT 2
 END;

 OPSCHEMA CLEARANCES
 PATTERN <<CLEARANCE CHECK>>
 EXPANSION 2 ACTION <<CHECK AXIAL CLEARANCES AND BOTTOM LEADS>>
           2 ACTION <<CHECK INNER OUTER CLEARANCES AND TOP LEADS>>
 END;


 OPSCHEMA REFITTOP
 PATTERN <<REPLACE TOP CASING>>
 EXPANSION 1 ACTION <<REPLACE TOP INNER CASING>>
           2 ACTION <<REPLACE TOP OUTER CASING>>
 END;

 MAINEFFECTS
 <<REMOVE TOP CASING>>    + <<TOP CASING REMOVED>>
                           - <<TOP CASING REPLACED>>
                            END
 MAINEFFECTS
 <<REMOVE SHAFT AND CHECK LEADS>>  + <<SHAFT REMOVED>>
                                     - <<SHAFT REPLACED>>
                                    END
 MAINEFFECTS
 <<REPLACE SHAFT AND FIT LEADS>>    + <<SHAFT REPLACED>>
                                           - <<SHAFT REMOVED>>
                                    END
 MAINEFFECTS
 <<REPLACE TOP CASING>>                      + <<TOP CASING REPLACED>>
                                             - <<TOP CASING REMOVED>>
                                  END
 MAINEFFECTS
 <<TURN OVER OUTER CASING>>                 + <<OUTER CASING TURNED OVER>>
                                             <<OUTER CASING TURNED BACK>>
                                   END
 MAINEFFECTS
 <<TURN BACK OUTER CASING>>                            + <<OUTER CASING TURNED BACK>>
                              + <<OUTER CASING WORK FINISHED>>
                                - <<OUTER CASING TURNED OVER>>
                                          END
 MAINEFFECTS
 <<TURN BACK INNER CASING>>                + <<INNER CASING TURNED BACK>>
                                          + <<INNER CASING WORK FINISHED>>
                                            <<INNER CASING TURNED OVER>>



                                    END
MAINEFFECTS
<<TURN OVER INNER CASING>>          + <<INNER CASING TURNED OVER>>
                                    - <<INNER CASING TURNED BACK>>
                                END
 MAINEFFECTSDONE@.PRSTRING;
PRIMITIVE
<<FINISHED>> ;
```

Attaching a level to every pattern in the action hierarchy helps the planning program control the order in which nodes are expanded. The opschemas contain expansions, orderings and conditions on constituent nodes The effects of actions are specified using MAINEFFECTS which declares the effects always associated with the occurrence of the <pattern> in the plan irrespective of any choice between expansions.

We can now consider how the planning system would use this action hierarchy to expand the goal node <<high pressure section overhauled>> (it is reasonable to ignore the expansion corresponding to low and intermediate pressures since the interactions between them are very limited.

The system will match the pattern of the goal node against that of the opschema overhaul high pressure section. The goal node in the zeroth level network will be replaced by six query nodes and an action node in the first level network and the user will be interrogated to determine whether the query nodes should become phantoms or goal nodes (i.e. whether the corresponding actions should be performed or not).

No effects are associated with any of the new nodes but two new unsupervised conditions are introduced by the expansion. Before moving to the second level network the planning program will seek to satisfy the unsupervised conditions - in this case they cannot be satisfied until the second level network.

In moving from first to second level networks the system replaces the goal node <<top and bottom leads fitted>> by three nodes

goal <<top casing removed>>

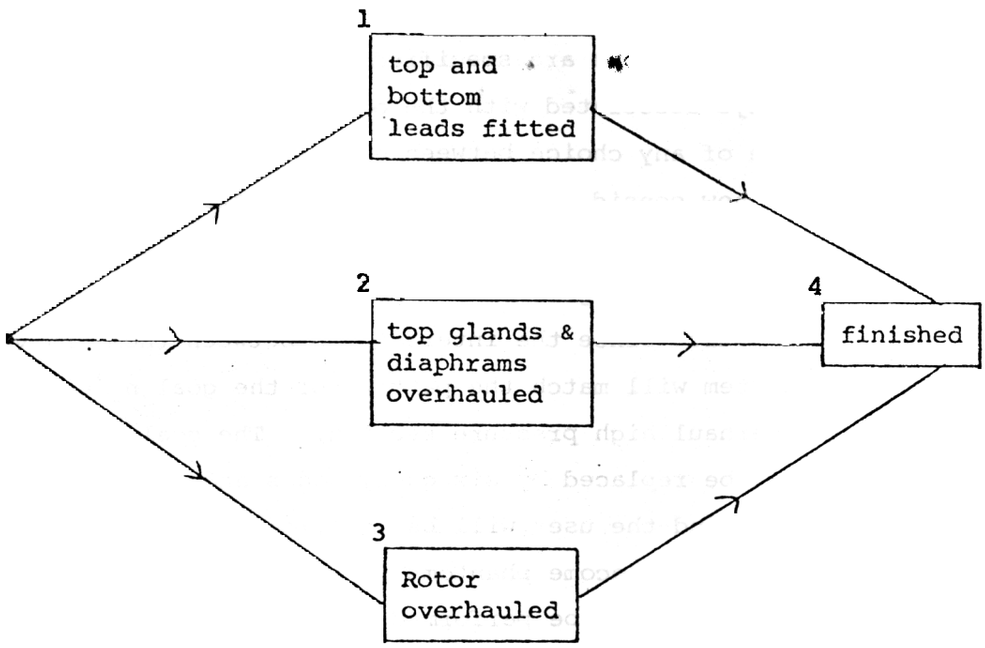
action <<fit top and bottom leads>>

goal <<top casing replaced>>.

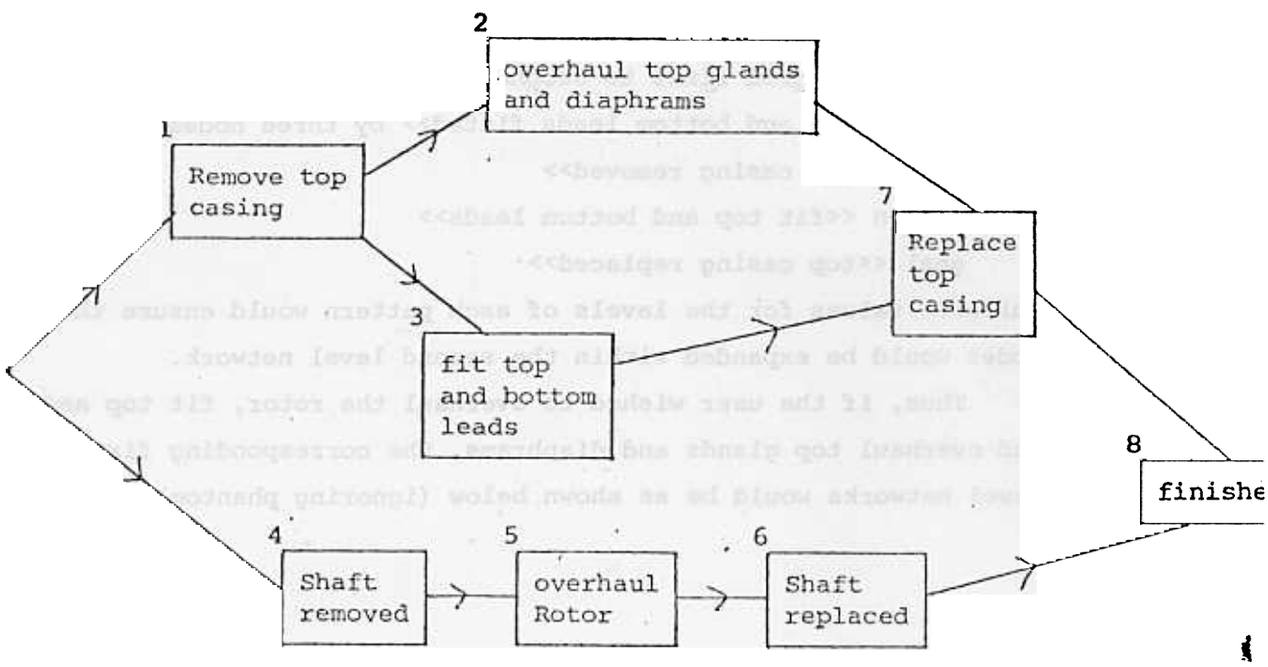
Suitable values for the levels of each pattern would ensure that the two goal nodes would be expanded within the second level network.

Thus, if the user wished to overhaul the rotor, fit top and bottom leads and overhaul top glands and diaphrams, the corresponding first and second level networks would be as shown below (ignoring phantom nodes).

first level



second level



At the second level the unsupervised condition

<<top casing replaced>> can be satisfied

The new unsupervised conditions <<top casing removed>> required by nodes 4, 5, and 6 are made true by node 1 and untrue by node 7. The

planner will remove the interactions and satisfy the condition by linking 1 before 4 and 6 before 7. The interactions will not be removed until they have become constrained by the satisfaction of the unsupervised conditions.

References

Crowson, W.B. and Thompson. "Decision CPM: A method for simultaneously Planning Scheduling and Controlling Projects", Operations Research 15, pp407-426.

Daniel, L.M. (1974) "And/or graphs and critical Paths", IFIP '74.

Daniel, L.M. (1977) "Planning: Mofifying non-linear plans", DAI Working Paper: 24, Department of Artificial Intelligence, University of Edinburgh.

Daniel, L.M. (1978) "Search Strategies for hierarchical Planning", forthcoming DAI Memo.

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.

Pease, M. (1977) "ACSI: Automated Command System", SRI Technical Report 13, Stanford Research Institute.

Sacerdoti, E.D. (1975a) "The non-linear nature of Plans", Advance Papers of 4th International Joint Conference on Artificial Intelligence, Tibilisi, U.S.S.R.

8. Sacerdoti, E.D. (1975b) "A Structure for plans and behaviour", SRI Technical Note 109, A.I. Center, Stanford Research Institute.

9. Siklossy, L. and Deussi, J. (1973) "An efficient robot planner which generates its own procedures", Advance papers of 3rd International Joint Conference on Artificial Intelligence, Stanford, California.

10. Tate, A. (1974) "INTERPLAN: A plan generation system which can deal with interactions between goals", MIRU Memo MIP-R-109, University of Edinburgh.

11. Tate, A. (1975) "The use of goal structure to direct search in a problem solver", Ph.D. thesis, University of Edinburgh.

12. Tate, A. (1976) "NONLIN: A hierarchic non-linear Planner", DAI Memo 25, University of Edinburgh.

An Integrated Approach to Planning, Plan Execution and Plan Monitoring

Austin Tate

Scientific Proposal to Science Research Council 26th July, 1976

We propose to develop a system capable of automatically forming plans and monitoring their execution in the domain of computer-controlled assembly. We will specify a formalism to enable a domain to be described in a straightforward fashion. To ensure generality and to make this research tractable, we will evaluate the system in two distinct domains both of which are the subject of current research in this Department. The research will involve the provision of a conditional planning facility and a run-time plan monitor which in its later stages should cope with sensory information from a domain.

1. Description of work proposed

We wish to allow a task to be given to a programmable robot assembly system in a straightforward way, preferably by giving a description of the finished object. Current languages under development for the generation of control programs for such robots rely on being explicitly given a fairly detailed sequence of the operations to be performed. We believe that recent work on the representation of problem domains and the generation of plans could usefully be employed to ease the burden on the programmer of a robot.

Most early research on planning was divorced from the actual use of the plans generated. However, recent work has concentrated on the benefits that a planning ability can give within specialized systems. For example, planning is used in the Computer Based Consultant for instructing a novice mechanical engineer on some task (Hart, 1975); in an automatic programming system (Manna and Waldinger, 1974); in a mission scheduler for an aircraft carrier (Fikes, 1975); to aid in understanding natural language (Schank and Abelson, 1975); and on an electronics trouble shooting system at MIT. A planning ability frees a user of the system from specifying sequences of the primitive operations available and allows the automatic choice of actions most suited to an overall task. Some recent planners, used as components of larger systems, have focussed on the need to represent a complex problem domain hierarchically, i.e. at many different levels of detail (Sacerdoti, 1975a; Tate, 1976, paper attached). Such planners work at a symbolic rather than a geometric level and thus cannot be used to generate plans containing the detailed actions necessary to perform assembly operations.

In this work we propose the development of a hierarchic task specification language and the extension of a planner able to generate plans in a domain represented in this language. This will enable control programs to be generated for a robot manipulator from quite high level descriptions of a task. We will consider the execution and monitoring of the plans generated on a particular device. The use of execution-time sensory information will be studied.

We want the task formalism and planner to be uniformly applicable to a wide range of domains. To achieve this we propose a common approach to two problems currently being tackled in distinct ways.

a) The development of an interactive program to enable a project network (as used for critical path analysis) to be generated in a straightforward and correct fashion, e.g. for a house building task.

b) The generation of instructions to assemble some object on a computer controlled robot manipulator.

Recent work on problem a) by us has led to a planning system NONLIN which is fully described in the paper attached (Tate, 1976). The planner can be given a hierarchic description of a domain (i.e. the problem domain can be specified at several levels of detail). Such domains are at present restricted to those which can be represented symbolically. Computational processes (such as a geometric modelling package) cannot be interfaced to allow the level of detail to be low enough for that required to perform assembly planning. NONLIN can generate a plan to perform a task in such a domain where the plan is represented as a partially-ordered network of actions. During planning a record is kept of the conditions necessary for any action and the points in the plan where the conditions are achieved (this record is called the goal structure of the plan). Problem b) has usually been tackled by specifying trajectories for manipulators and basic manipulator activities. No planning capability has been provided.

We aim to establish that

a) the planner NONLIN can be extended to cope with task descriptions which can include low level manipulator movements.

b) the use of sensory information from a domain can be planned for by the use of special TESTABLE conditions on actions in the plan. This will involve the provision of conditional planning facilities.

c) The goal structure of a plan, extended to incorporate b) above, can be used to monitor the execution of the plan and aid in re-planning when necessary.

We thus hope to show that an integrated approach to planning at several different levels and execution monitoring of a plan is possible. The program of work could not be undertaken with the resources asked for if not for the fact that simultaneous work is being undertaken by two projects at Edinburgh on the two problems mentioned.

a) the project "Planning: a joint AI/OR approach" under the direction of Professor B. Meltzer is nearing the end of its first year (Daniel and Tate, 1976). The work on NONLIN is a result of this first year effort (Tate, 1976, paper attached).

b) the project "a computer aided assembly system" is about to begin under the direction of Mr. R.J. Popplestone (Popplestone, 1976).

The close co-operation we have with the planning group will continue and this will enable the evaluation of the use of the planner on, for example, the house building task they are considering.

The work on computer aided assembly will make available a run-time mini-computer system and software to enable the evaluation of the integrated approach on problem b).

The cross-fertilization of ideas between the two different approaches being tackled by the attempt on this project to develop an integrated planning approach to both tasks could be very fruitful. We have chosen the two tasks for the evaluation of our approach because of the availability of local expertise and continuing work on these. A specific commitment to an integrated approach to the two different tasks will avoid the common pitfall encountered on many projects of a lack of generality in a final system.

2. Relation to previous and current work

2.1 Planning and Task Specification

Recent work on the generation of plans has been concerned with producing plans which could be used to transfer the knowledge of an expert or experts in a domain to a novice. Work of this type has been performed on the Computer Based Consultant Project (Hart, 1975) at Stanford Research Institute and on an Electronics Troubleshooting System at MIT. Experts in some field provide a collection of job descriptions each of which may require the knowledge of other experts to break down sub-tasks. The system builds up a hierarchy of jobs which can be used to generate plans at various levels of detail. An apprentice or novice using the system is interactively directed through a task at a level appropriate to his skill by the system asking him to perform tasks at a higher level first. If the higher level tasks are beyond the apprentice, the expert knowledge encoded in the system is used to choose some way to break down a task. The planner ensures that all the plan can still be performed successfully when these more detailed steps are added. Accommodation of further detail into the plan may cause re-ordering of part of the plan. The lower level tasks are then given to the apprentice who again can signify his ability to perform them or not.

The Computer Based Consultant is a system intended to guide an apprentice mechanical engineer through various tasks at a workbench. Typical of the tasks is the assembly of an air compressor. The planning system used in this project, NOAH (Nets of Action Hierarchies), was developed by Sacerdoti (1975a, 1975b). It incorporates novel ideas for the representation of a plan as a partially-ordered network of actions (procedural net). This is in contrast to most previous work on planning which concentrates on the generation of linear sequences of actions, e.g. STRIPS (Fikes and Nilsson, 1971), LAWALY (Siklossy and Dreussi, 1973), INTERPLAN (Tate, 1974), etc. Knowledge about a domain is given to NOAH by writing code in a language SOUP (Semantics of User Programs) to explain how to decompose any task to lower level tasks.

Work at Edinburgh, on a project "Planning: a joint AI/OR approach" (Daniel and Tate, 1976), is concerned with the problem of large scale project planning and the development of an interactive program which guides a user through the entire planning process. For project planning, a network must be set up so that critical path analysis and other optimization

procedures can be used to decide where resources should be directed to most efficiently achieve a task. However, the network typically is difficult and time-consuming to set up. It is also difficult to ensure that the ordering constraints on tasks are in their least constrained form. This is essential to allow the optimization to achieve the best results.

As in the work of Sacerdoti, we have been investigating the use of a partially-ordered network of actions to represent a plan at any stage. Such networks are in a suitable form for the use of critical path analysis techniques. Any ordering in the network results from the fact that either

i) an action achieves a condition for a subsequent action

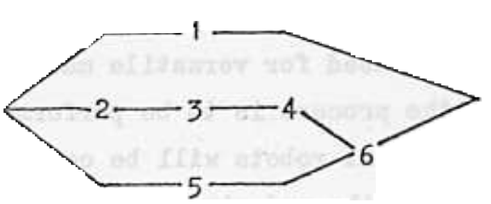
ii) an action interferes with an important effect of another action and must be removed outside its range.

A formalism (TF) has been specified to enable a task to be described in a hierarchic fashion. Task descriptions can be written independently of their use at higher levels. Thus experts at a higher level, middle-management and tradesmen, can each describe their tasks independently and in their own terminology.

A simplified "ACTSCHEMA" from a house building specification in the Task Formalism is given below.

```
ACTSCHEMA DECOR
  PATTERN DECORATE
  EXPANSION [AND  CONDITION UNSUPERVISED <<ROUGH PLUMBING INSTALLED>>
                  CONDITION UNSUPERVISED <<ROUGH WIRING INSTALLED>>
1               ACTION <<FASTEN PLASTER BOARD AND PLASTER>>
                [ [AND [   CONDITION UNSUPERVISED <<DRAINS INSTALLED>>
2                        ACTION <<POUR BASEMENT FLOOR>>
3                        ACTION <<LAY FINISHED FLOOR>>
                           CONDITION SUPERVISED <<FLOOR FINISHED>>
4                        ACTION <<FINISHED CARPENTRY>> ]
                         CONDITION UNSUPERVISED <<PLUMBING FINISHED>>
5                    ACTION PAINT]
                  CONDITION SUPERVISED <<CARPENTRY FINISHED>>
                  CONDITION SUPERVISED PAINTED
6               ACTION <<SAND AND VARNISH FLOOR>> ]]
  EFFECTS + DECORATED
END;
```

The partial ordering on the actions is



The DECOR schema specifies a partial ordering on 6 actions which together will achieve the "DECORATE" task. SUPERVISED conditions are made true within the expansion of the task (e.g. the ACTION PAINT (5), achieves the SUPERVISED condition PAINTED on ACTION 6). UNSUPERVISED conditions are made true by other experts (mainly here by an "INSTALL SERVICES" expert). Another condition type, "USEWHEN", would say that an ACTSCHEMA containing it should not be used unless the condition was true. There is only one effect of this ACTSCHEMA (+ DECORATED) as the other effects are defined by the lower level actions.

A planner, NONLIN, has been specified and is implemented in an experimental version which can take task descriptions given in the formalism. It generates a plan at progressively greater levels of detail and can handle interactions between the components to produce a plan as a partially-ordered network of actions. The algorithms employed in the planner have been designed so that over-linearization is avoided where possible and all choice points are kept for later analysis or re-planning. A simple clear representation of the goal structure (GOST) of a plan is kept (the conditions on nodes of the network together with the points where the conditions are achieved). An example of a GOST entry during a house building task might be

<<SUPERVISED <<SCAFFOLDING ERECTED>> TRUE 6>> with value [4].

This would mean that <<SCAFFOLDING ERECTED>> had to be true at node 6 and was made true at node 4 (nodes in a network are numbered). The GOST specifies a set of "ranges" for which patterns have a certain value. Goal structure provides information about a plan which is much simpler than the detail of the plan itself. The use of goal structure to direct search in a problem solver was first investigated in Tate (1975). NONLIN and the task formalism (TF) are fully described in the paper attached (Tate, 1976).

2.2 Automation Languages

Recently there has been effort to produce a suitable language for instructing industrial robots (such as the UNIMATE, AMF VERSATRAN, etc.) to perform some production process (especially assembly). This is especially important for batch production which is characterised by short runs. There is a need for versatile machines, which can be readily programmed, if the process is to be performed automatically. It seems likely that industrial robots will be controlled by mini-computers which may make use of tactile and simple visual information (Industrial Robot

Technology, 1976). Knowledge about the nature of the tasks being performed will be needed to produce programs for these controllers. Since the life expectancy of some industrial robots can be as long as 40,000 working hours (Engelberger, 1976 for a UNIMATE) the reprogramming may have to be done many times in the device's life.

AL, (A language for automation - Finkel et al, 1975) has been developed at Stanford University as a language for the specification of assembly instructions to two manipulators and for the use of sensory information in the plan to be performed. In the system presently in use under AL, manipulations are mostly described in terms of trajectories and primitive manipulator motions. Advanced versions of AL and the work to be carried out at Edinburgh on a computer-aided assembly system will allow a higher level of description. A sequence of goal states will be given each of which will be specified by relations between components (e.g. that 2 faces are "AGAINST" each other). Expert knowledge encoded in the system will be used to generate the necessary trajectories and manipulator motions to achieve each goal state mainly by an analysis of the geometric constraints on the specified goal positions. The form of language used to describe this knowledge and the method for describing how relations between bodies can be translated to machine motions is still a matter for discussion. To aid in this task a mathematical process can be used to generate the position of individual bodies when body descriptions are available and a set of spatial relations given (Ambler and Popplestone, 1975).

The work on the SRI Computer Based Consultant project is also in the domain of assemblies of mechanical parts though the final output of the system is to a human apprentice and planning is thus at a relatively high level. During the formation of an assembly program AL needs to generate a low level model of the state of the world at any point in the plan. A representation for the plan such that both high level and low level planning can be performed would be investigated as part of the proposed project. A formalism adequate to meet the needs of representing high and low level tasks in a uniform way would also be developed.

During automatic assembly under mini-computer control the run-time overheads of computation and size of program must be minimized. The program must incorporate most of the knowledge about how to deal with small errors of positioning, correction of arm trajectories and so on. A good

run-time monitor must ensure that errors do not get out of the bounds which can be handled by the program. The planning model generated during the formation of run-time code by AL provides a set of expected values for variables etc. which are given to the run-time system and which allow the monitoring to take place.

Our suggestion is that the goal structure of a plan generated by the planner NONLIN (Tate, 1976 - paper attached) provides just the information necessary to monitor the execution of the plan and thus is the information which should be passed to the run-time system and used by it. For a manipulator which assembles by dead-reckoning, the condition types available in the goal structure of the current version of NONLIN may be sufficient to monitor the execution of a plan (the system stopping if any condition was violated). For more advanced systems where sensory information from a domain is available to the run-time system, further TESTABLE condition types will be necessary together with the associated plan-time facilities to allow conditional branches and rejoins in the plan being generated.

3. The use of goal structure for planning, plan execution and plan monitoring

The most ambitious planning, plan execution and monitoring system to date has been the STRIPS/MACROPS/PLANEX system developed at SRI (Fikes and Nilsson, 1971; Fikes, Hart and Nilsson, 1972; and Fikes, 1971) to enable the robot SHAKEY to perform plans generated by STRIPS. The interface between the planner (STRIPS) and the execution monitor or run-time system (PLANEX) was a triangular table representation of a plan (a MACROP). STRIPS worked in a very simple domain and could generate only very short plans in a relatively long planning time (see Tate, 1974, for a comparison of several problem solvers). Recent advances in planning techniques and robot devices allow us to suggest a planning, execution and monitor system of wider scope and greater flexibility.

In NONLIN, the necessary conditions on actions in a plan and the points in the plan where they are achieved are represented in a GOST (goal structure). Goal structure contains information which is more structured than the detail of the plan itself. It has proved a valuable tool for detecting and correcting for interactions in a plan and directing the search of a planner (Tate, 1975b; Sacerdoti, 1975b; and Tate, 1976). A GOST also gives the information used from a MACROP by PLANEX and can thus replace it for execution monitoring purposes. The information in a GOST is such that we need only check that an action does not alter a GOST entry for any succeeding action to know that the rest of the plan can be achieved. Only if any condition on a succeeding node is altered is the plan in jeopardy. Since conditions are attached to actions in a plan, it is possible to specifically ask if each monitorable condition holds before the action is executed.

The use of goal structure as the basis of the plan monitoring scheme has several advantages from the point of view of the run-time system where time and program space are at a premium. Specifically monitoring certain conditions before an action is performed is more practicable in a working assembly system than asking that an accurate world model be kept after the execution of each action (as is needed for the MACROPS/PLANEX scheme). Such a process should allow for the use of information from sensors when a conditional planning facility is available. The test and branch conditions will be represented to the run-time monitor through the goal structure. We hope to be able to provide a re-planning ability based on the GOST conditions detected as not holding before some action and on the remaining unexecuted part of the plan at that point.

4. Relevant work at Edinburgh

As mentioned previously, we have chosen two tasks to explore our integrated approach because of current work at the Department of Artificial Intelligence (AI) at Edinburgh University on these tasks. In order that we can make clear the scope of the proposed project, we will first briefly describe and mention the limitations of this other work.

4.1 Project Planning

The work on the project "Planning: a joint AI/OR approach" (Daniel and Tate, 1976) has been described in section 2.1.

A formalism and planner have been specified to allow hierarchic description of problem domains and the generation of plans in those domains. The lowest job level described may be actions such as "lay foundations" or "install kitchen equipment" (see example in section 2.1). All planning is at a symbolic level and constraints on the ordering of actions in a plan arise through the achievement or destruction of symbolic statements about any time point in the plan (e.g. "floor boards layed" may destroy some required condition "under floor access clear"). It is not possible to form plans whose goals are specified from some computation, e.g. "hand at (x,y,z)" where (x,y,z) is a transformation of the position of an object computed by a geometric modelling package. Extension of the planner and task formalism to cope with lower level or non-symbolic domains will not be undertaken on the AI/OR project.

Work currently in progress on the planning project is aimed at producing a system in which choices of operator, choices of instantiation for variables or the insertion of action ordering links in the network are represented in a "decision graph" which can be used to select alternative approaches to be taken by the planner (Daniel, 1976). Such a scheme may be based on the process used by Hayes (1975) for a travel planning system. Later work will allow the actions in a network to be assigned "costs" which can be used for critical path analysis optimization.

Since the planner and task formalism to be used as a basis of the proposed project is in fact used at present for the above work, we expect that close contact with the planning group will ensure that the planner and formalism in use remains compatible with the need to extend both to cope with the lower level tasks. When they become available, the incorporation of the decision graph and cost analysis components will be desirable for the ability to deal with complex low level tasks. In particular, an assembly program produced for some batch production process must be as

efficient as possible since it may be repeated many times. Plan-time effort is worth expending to produce a good program. The assignment of costs in terms of time or energy used (depending on the application) to actions and the ability to generate low cost solutions will therefore be important in the proposed work.

Automatic Assembly

The project "a computer aided assembly system" (Popplestone, 1976) aims to construct and evaluate a language for specifying assembly operations. The complete system will be given spatial relationships which certain bodies must have to one another and will be required to generate a program for a run-time system which will perform the task on the existing Edinburgh hand/eye device. The form of language to be used for specifying assembly operations is still to be decided, and we would hope that the "linking" nature of this proposed work would have some influence on this.

Part of the work will involve the provision of a facility to locate the position of any feature of a body from a specified set of symbolic relations between the bodies in a world (e.g. AGAINST and FITS). This will be an extension of the work described in Ambler and Popplestone (1975). I will refer to this later as a "position finder". A run-time system will be developed to control the primitive manipulations available and an interface specified for its use. Methods of interfacing sophisticated sensory feedback into an assembly operation will be explored. Later in the project, an investigation of the use of the space occupancy of bodies to modify certain programs (e.g. to avoid hand/object collisions) will be conducted.

A previous assembly system for the Edinburgh hand/eye device (Ambler et al, 1975) relied for the actual final assembly of an object upon a sequence of pre-programmed primitive manipulator motions together with two "fitting" operations which used force sensing. The system had a sophisticated module to locate objects, recognize them and put them in specified "storage" positions. After this phase the system merely ran through the sequence of primitive operations to assemble the object presuming the components were in these fixed positions. No planning was involved. The new assembly system will rely on a human instructor to specify a sequence in which the assembly operations are to be performed. It is not

within the scope of that work to consider how to determine the sequence automatically.

Since this is precisely the task for which NOAH (used in the Computer Based Consultant) and NONLIN were designed, we have an opportunity here to bridge the gap between current planning and automation language design work. The facility for providing specifications of assembly operations through the task formalism (say as a library of primitives) and allowing the planner to find the appropriate ways to combine them to achieve some goal should reduce the amount of specification necessary for any particular task.

5. Programme of work

5.1 Part 1

The first part of the work will be mainly concerned with the provision of facilities known to be needed for the generation and use of lower level plans. To some extent these will be developed with "dummy" modules representing facilities to be experimented with in Part 2. There will be 4 aims in the first part.

1. Computations

Current planning systems do not meet the needs of low level (manipulator) planning because they work symbolically and the interface of computational systems such as a "position finder" is not possible. We will provide facilities to allow plan-time and run-time computations to be performed to instantiate or modify a plan. In order to develop the planning system independently of progress on the assembly project a simple "position finder" will be used at first (possibly using a data base of explicitly asserted positions). We wish to incorporate the use of computations in such a way that the task formalism can be used to describe them naturally and independently of the level of the task given (e.g. it would also be possible to use computations to provide calculations of quantities of materials for house building tasks, etc.).

2. Conditional Planning

Condition types will be added to allow execution time testing of a world and plan structures designed to allow conditional branching on such tests. Conditional plans will be necessary in low level domains to cope with, for example, the several stable positions a component may be presented in by a feeding device. There have been very few attempts to produce plans which can have conditionals in them. One typical approach is that described in Warren (1976) where case analysis on the branches of a conditional is performed and conditional plans branch but do not then rejoin. The structured way in which a conditional would be added to a plan when using a hierarchical representation of a domain should allow the generation of conditional plans which branch and rejoin. We believe our algorithm for question-answering in a partially-ordered network (Tate, 1976, paper attached) can be modified to cope with networks containing conditionals.

3. Run-time System

We will begin work on a run-time system to take a plan from the planner and execute it on the Edinburgh hand/eye device. Account should be taken of the mechanical inaccuracies of the device. Initially, the run-time system will be based on the code available in the present manipulation package for the robot.

4. Project Planning

Co-operative efforts may ensure that a broadly similar planner is used for the project planning as will be required for this proposed work (the planner currently used for the project planning is NONLIN). The capability of adding the "decision graph" and cost analysis components as they are developed in the planning project should be preserved.

At the end of the first phase of the work we will have hierarchically specified a collection of assembly operations such that the planner can be given commands at a high level and used to generate code for a run-time system on the Edinburgh hand/eye device. We would hope at this point to be able to generate and use some plans which did not require the use of sensory information (i.e. assembled by dead-reckoning).

5.2 Part 2

Planning systems have not in general concerned themselves with the requirements of executing and monitoring a plan after its generation (with the notable exception of the STRIPS/MACROPS/PLANEX system). We thus anticipate that part two of the work, together with the provision in part one of conditional planning facilities, will provide the greatest challenge and require the most effort. This will be to evaluate the use of the integrated approach in the domain of automatic assembly. At first we will choose a task such as the assembly of a model car from a kit of parts since such assemblies can be done at present in a "blind" fashion (Ambler et al, 1975). Within the framework of exploring this problem there will be 3 aims in part two.

1. Use of goal structure for monitoring the execution of a plan

We aim to show that the goal structure of a plan (a GOST generated by NONLIN) can be used as the basis of an execution monitor. The run-time system will therefore be extended to perform monitoring using goal structure. Any monitoring error would probably be treated as a

stop condition at first though we expect to be able to add a re-planning capability in due course.

2. Executing Conditional Plans

We will experiment with TESTABLE conditions and conditional plan parts in order to use simple sensory information (e.g. force on the hand) to guide assembly. As more sophisticated sensory facilities are developed on the assembly project, we hope to interface to these. We anticipate that the goal structure of a plan can represent the use of run-time conditionals and can thus be used to monitor the execution of plans containing them.

3. Using Computations

An interface to computational processes such as the "position finder" described earlier and to numeric data bases will be made through the facilities provided in part one. Some standardization of the data bases used between the projects would help in this task and efforts should be made to achieve this (it is not the case at present). Since geometric modelling is one of the most complex computational tasks to which we will interface, we must be prepared to make such simplifications as are necessary to enable 1 and 2 above to be achieved.

The project will thus investigate the use of a planner and run-time system in two differing domains and find a task formalism adequate to cope with the needs of representing tasks in those domains. In particular we hope to demonstrate that the system can generate plans with the detail required for their execution on a computer controlled assembly device. We will investigate the generation and use of conditional plans.

6. An example of the task formalism in use

We give here an example of a piece of code which is in a crudely extended version of the Task Formalism (TF) currently accepted by NONLIN. This is part of a problem description given to a modified form of NONLIN to clarify some of the ideas in this proposal. The Formalism has been extended to provide a weak "COMPUTE" facility. The syntax here would not be preserved in the system proposed and work is already in progress to separate the action parts of an expansion from the specification of conditions and their ranges. The code describes a way to pick up a block.

1. It ensures the hand is empty.
2. It approaches the block through an approach position <<X Y HIGHZ>> (where HIGHZ = Z + 20 cm).
3. It sets the fingers 5 cm wider than the width of the block (the block width is found through the value of BLOCKWIDTH in a numeric data base).
4. It lowers the hand to a position computed by the function POSITION (<block to be picked up>) which returns a vector <<X Y Z>>. Real position finders are more complex.
5. It closes the fingers to grip the block.
6. It moves the hand (now holding the block) up to the approach position <<X Y HIGHZ>>.

$*X is a variable X. The variable may be given a restriction in th "VARS" statement. Facilities exist for giving a partial-ordering on actions in an EXPANSION as well as the fully ordered sequence given here (an example of this is given in section 2.1).

```
OPSCHEMA PICKUP
  PATTERN <<HELD $*B1>>
1 EXPANSION GOAL EMPTYHAND
              CONDITION USEWHEN <<CLEARTOP $*B1>>
              CONDITION USEWHEN <<ON $*B1 $*B2>>
              CONDITION USEWHEN <<OVER $*B3>>
              CONDITION COMPUTE [<< $*X $*Y $*Z>> = POSITION $*B1] use of a simple
              CONDITION COMPUTE [$*HIGHZ = + $*Z 20]          computational facilit;
2           ACTION <<MOVEHAND << $*X $*Y $*HIGHZ>> >>
              CONDITION COMPUTE [$*WIDTH = BLOCKWIDTH] use of a numeric data base.
              CONDITION COMPUTE [$*FINGERWIDTH = + $*WIDTH 5]
3           ACTION <<SETFINGERS $*FINGERWIDTH>>
4           ACTION <<MOVEHAND << $*X $*Y $*Z>> >>
5           ACTION <<SETFINGERS $*WIDTH>>
6           ACTION <<MOVEHAND << $*X $*Y $*HIGHZ>> >>
  EFFECTS - EMPTYHAND
          + <<HELD $*B1>>
          - <<OVER $*B3>>
          + <<OVER $*B2>>
  VARS B1 UNDEF B2 UNDEF B3 UNDEF X UNDEF Y UNDEF Z UNDEF HIGHZ UNDEF
       WIDTH UNDEF FINGERWIDTH UNDEF:
END;
```

PICKUP is a relatively low level job specification in the hierarchical description of the domain. The rest of the package gives ways to achieve EMPTYHAND etc. and gives higher level actions (e.g. <<PUT X ON TOP OF Y>> which uses PICKUP as a component). Lower level descriptions of MOVEHAND and SETFINGERS would be needed for a practical system as would a more general approach position mechanism, etc.

7. Wider Justification

1. Hierarchic Task Specification

Robot devices for performing mass assembly and finishing processes are becoming widely used in industry (Industrial Robot Technology, 1976). The development of programmable robots (controlled by a mini-computer) has extended the applicability of these devices to the economically important field of batch production and special purpose "one-off" processes. Since such devices may need to be re-programmed many times during their working life, the generation of control programs should be straightforward. The specification should preferably be done at the level of someone describing a finished article or job. The planning of an assembly using a hierarchic planner and task formalism will enable complex tasks to be dealt with at high levels first and expanded to detail in a structured way. It will allow a natural interface to libraries of detailed pre-programmed sub-tasks (e.g. provided by the manufacturer of a versatile industrial robot).

2. Using Alternatives

In the domain of automatic assembly, a large computer may be used to compile a control program to be repeatedly executed on some device. It is worth expending compile-time effort to produce a good program. A system such as we will develop can be given alternative ways to perform assembly operations and then the planner allowed to select those appropriate for some overall task. This is a capability not being considered in current research on automatic assembly languages. The capability of attaching costs to actions in a plan will allow us to generate cost-effective control programs.

3. Executing and Monitoring Plans

We consider it important that we will be using a run-time system to execute plans since much recent work on planning has been divorced from the actual use of plans on a real device. The systems have, therefore, tended to become rather stereotyped and deal with problems which have difficult features but which are intrinsically simple (e.g. block stacking problems). The only comparable work on a planner and run-time plan monitor was the STRIPS/MACROPS/PLANEX system developed at SRI in 1971-2. Recent advances in planning and in the capabilities of computer-controlled devices make it desirable to construct an up-to-date system.

4. Conditional Planning

Our research is scientifically interesting as we will be investigating the generation of conditional plans (which has much in common with automatic programming) and the use of such plans to allow sensory information from a domain to be coped with. The use of sensors will be vital for versatile assembly robots.

References

Ambler, A.P., Barrow, H.G., Brown, C.M., Burstall, R.M. and Popplestone, R.J. (1975) A versatile system for computer-controlled assembly. Artificial Intelligence, 6, 129-156.

Ambler, A.P. and Popplestone, R.J. (1975) Inferring the positions of bodies from specified spatial relationships. Artificial Intelligence, 6, 157-175.

Daniel, L. (1976) Forthcoming DAI Memo, University of Edinburgh.

Daniel, L. and Tate, A. (1976) Planning: a joint AI/OR approach. AISB Newsletter, 23.

Engelberger, J.F. (1976) Performance evaluation of industrial robots. Advance papers of 3rd Conference on Industrial Robot Technology and 6th International Symposium on Industrial Robots, Nottingham, U.K.

Fikes, R.E. (1971) Monitored execution of robot plans produced by STRIPS. Proceedings of IFIP Congress 1971, Ljublijana, Yugoslavia.

Fikes, R.E. (1975) Automatic planning from a frames point of view. Theoretical Issues in Natural Language Processing, Cambridge, Mass.

Fikes, R.E., Hart, P.E. and Nilsson, N.J. (1972) Learning and executing generalized robot plans. Artificial Intelligence, 3, 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, 189-208.

Finkel, R., Taylor, R., Bolles, R., Paul, R. and Feldman, J. (1975) An overview of AL, a programming system for automation. Advance papers of 4th International Joint Conference on Artificial Intelligence, Tbilisi, U.S.S.R., pp. 758-765.

Hart, P.E. (1975) Progress on a computer based consultant. Advance papers of 4th International Joint Conference on Artificial Intelligence, Tbilisi, U.S.S.R.

Hayes, P.J. (1975) A representation for robot plans. Advance papers of 4th International Joint Conference on Artificial Intelligence, Tbilisi, U.S.S.R.

Industrial Robot Technology (1976) Advance papers of 3rd Conference on Industrial Robot Technology and 6th International Symposium on Industrial Robots, Nottingham, U.K.

Manna, Z. and Waldinger, R. (1974) Knowledge and reasoning in program synthesis. Technical Note 98, AI Center, Stanford Research Institute. Also in Artificial Intelligence, 6, 175-208.

Popplestone, R.J. (1976) A computer aided assembly system. SRC grant proposal (G/RA/18764).

Rosen, C.A. and Nitzen, D. (1975) Some developments in programmable automation. Technical Note 100, AI Center, Stanford Research Institute.

Sacerdoti, E.D. (1975a) The non-linear nature of plans. Advance papers of 4th International Joint Conference on Artificial Intelligence, Tbilisi, U.S.S.R.

Sacerdoti, E.D. (1975b) A structure for plans and behaviour. Technical Note 109, AI Center, Stanford Research Institute.

Schank, R.C. and Abelson, R.P. (1975) Scripts, plans and knowledge. Advance papers of 4th International Joint Conference on Artificial Intelligence, Tbilisi, U.S.S.R.

Siklossy, L. and Dreussi, J. (1973) An efficient robot planner which generates its own procedures. Advance papers of 3rd International Joint Conference on Artificial Intelligence, Stanford, California.

Tate, A. (1974) INTERPLAN: a plan generation system which can deal with interactions between goals. MIRU Memo MIP-R-109, University of Edinburgh.

Tate, A. (1975a) Interacting goals and their use. Advance papers of 4th International Joint Conference on Artificial Intelligence, Tbilisi, U.S.S.R.

Tate, A. (1975b) The use of goal structure to direct search in a problem solver. Ph.D. thesis, University of Edinburgh.

Tate, A. (1976) NONLIN: a hierarchic non-linear planner. Forthcoming DAI Memo, University of Edinburgh.

Warren, D.H.D. (1976) Generating conditional plans and programs. Advance papers of the Summer Conference on Artificial Intelligence and Simulation of Behaviour (AISB), Edinburgh.

Using TF to describe parallel processes and NONLIN as a compiler to establish a partial order on their execution to achieve some goal.

Work in Artificial Intelligence on generating project networks by the use of hierarchic non-linear planning has led to a formalism for describing task domains (Task Formalism - TF) and a hierarchic non-linear planner (NONLIN) which can use TF descriptions to plan to achieve a given goal. In discussion with Owen Evans and others at the ICL Stevenage Research Centre during the Summer of 1976 it became clear that TF could provide the basis of a language for specifying parallel processes and the communication channels between them, and that the planner NONLIN could then act as a scheduler and compiler to ensure that the processes achieved a required goal. This note briefly describes the tie up between TF forms and their interpretation as a process description language.

The main component of a TF description is a schema which describes the sub-processes necessary to achieve the purpose of the overall process. The sub-processes are mentioned explicitly by name (as ACTION nodes of the expansion) or by pattern (leading to non-determinism - as GOAL nodes of the expansion - or in the next NONLIN as ACHIEVE conditions). A partial order may be imposed on these sub-processes.

Inter process communication is facilitated by way of messages broadcast by a terminating process (at a particular time point changes are made on a world model via the effects of the schema - the machine state is altered) and by the conditions on processes. Effects are defined as a pattern (an extension of the notion of a variable) being given a value by the process. Complex messages can be sent by this scheme. Conditions on processes are present for a variety of reasons but all conditions must be satisfied before a process can run. They thus act as triggering monitors on the world model (machine-state) for this process to start. The condition types available describe to the scheduler (the planner NONLIN) how it should go about satisfying the activating conditions for each schema (or process). Supervised conditions describe conditions to be made true by a particular sub-process of the process stating it, defining a protected range for the statement. Unsupervised conditions describe conditions to be made true by an outside concurrent or earlier process, i.e. a monitor on the messages broadcast by other processes. Usewhen conditions describe environmental information about when it is useful or meaningful to invoke the process. These may check the global machine's state (global world model).

Sub-processes and a partial order on them are provided by the expansion of the schema. These describe how the purpose of the high-level process may be achieved in terms of lower-level processes.

Time slices for processes can be allocated using the time estimates which may be provided in TF schemas on sub-processes. The critical path information available allows estimates of processor usage and job length etc to be made by the scheduler when it is dealing with several separate jobs.

Austin Tate
22nd June 1977

A net marking algorithm for Question Answering in a partially ordered network of nodes

Austin Tate 21 June '77

Tate (1976) describes a question answering process for partially ordered networks of nodes. The process relies on having the nodes in the network marked with respect to the node at which a query is made (say node N) and using these marks to find a set of "critical" P-nodes for node N. Critical P-nodes are those which, in a possible linearization of the network, could establish P with some value which could be maintained up to node N. Given such a marked net, a very simple process can be used to determine the answer for any query, or, if the answer is indeterminate, to suggest links in the net which would make the query be satisfied (if this is required).

Presented below is a net marking algorithm which does not require the marks at a node to be cleared each time the net is to be marked, and which, for each marking,

a) traverses each link before node N at least once and at most twice
b) traverses each link after node N exactly once
c) does not traverse any link in parallel with node N.

This is an improvement over previous schemes in use.

Mark Variables and the values they take

3 integer variables are kept
BEFORE initialized to -2
NODE initialized to -1
AFTER initialized to 0

Each new node added to the network has 3 mark variables

A mark to say node is before node N

B mark to say node is before a critical node for P at node N and possibly a 3rd mark variable

C mark to say node is a P-node.

All mark variables are initialized to 0 when a node is first put in the network. Before marking a network, the mark variables BEFORE, NODE, and AFTER are each incremented by 3 giving unique integers which will be recognized as the position of a node in the net with respect to node N whatever the present marks are at each node (avoiding mark clearing) Due to the initial values, on first use the marks will be 1, 2 and 3.

The marking algorithm

1. An optional first phase is to mark the C mark variables of each P-node. Whether it is necessary depends upon the efficiency of determining if a node is a P-node.
   Get TOME (Table of Multiple Effects) entries for the statement the query is to be made for (say P). This is a single lookup in NOAH and NONLIN. Mark the C mark variable of each node on the list of entries. The mark could be the integer in variable NODE to save mark clearing again.

2. Initialize the critical nodes list to NIL.

3. "Before" marking:
   traverse the links before node N (along PRENODES links) depth-first.
   At each node:
   Is variable A marked with integer in BEFORE?
   YES - terminate this branch of depth-first search.
   NO
   - mark variable A with the integer in variable BEFORE.
   - Is node a P-node? using mark variable C to find out or some other means.
   - YES -
     - critical P-node possibly found. Add to critical nodes list.
     - Mark the A and B mark variables of all nodes before this node. Use depth-first marking, terminating any branch on which there is a node with mark variable B already equal to the value in variable BEFORE. Do not look for P-nodes while marking the A and B marks of these nodes.
     - When this is done, continue normal depth-first marking looking for P-nodes at the point where the possible critical P-node was found.
   - NO - Continue depth first search and marking.

4. "After" marking:
   Simple depth first marking of each node's A mark variable for nodes after node N (along SUCCNODES links) with mark in variable AFTER.

5. "Node" marking:
   Mark A mark variable of Node N with integer in variable NODE

6. Scan down the list of possible critical nodes found. Remove any node whose B mark variable is set to the integer in BEFORE. These nodes have been found to be before some other critical P-node and thus are not genuine critical P-nodes themselves. The remaining list contains the critical P-nodes before node N.

7. Get the TOME (Table of Multiple Effects) entries for P and extract any node which has an A mark which is not the same as any value in BEFORE, NODE or AFTER (in fact can say any node whose A mark is less than the value in variable BEFORE since all 3 variables are always larger than any old mark in the network). This phase gives the critical P-nodes in parallel with node N.

After the process, we have a marked network and the 2 sets of critical P-nodes necessary for question-answering by the scheme presented in Tate (1976).

Further queries at the same node (N)

If links have not been altered in a network, advantage may be taken of the fact that the A mark variables of a net marked with respect to a node N will not alter if a query for a different statement P is made at the same node. This will commonly be necessary in order to establish the truth of several conditions at a node etc. It is then only necessary to mark the B marks of all nodes before any P-node which is itself before node N (found by looking at the P-nodes' A marks). Again a unique number can be used for the mark (by incrementing the value of variable BEFORE by 3 and using this value). The critical nodes for the new statement P at node N can then easily be found by checking the B marks of the P-nodes which were before node N to see if any has had its B mark marked (then it is not a critical P-node). The parallel critical nodes are found in a similar way to step 7 above.

Reference

Tate, A. (1976) Project Planning using a hierarchic non-linear planner. DAI Research Memo No. 25

HOW TO USE THE NONLIN PLANNING SYSTEM

AUSTIN TATE 19 NOV 1976

---

1. Introduction

This note describes how to use the NONLIN planning system on the Edinburgh DEC 10 POP-2 system. NONLIN is a hierarchic non-linear planner based on the NOAH planner developed at SRI by Sacerdoti (1975). NONLIN was developed as a vehicle for research on the "Planning: a joint AI/OR approach" SRC project (Daniel and Tate, 1976). It forms the first attempt at an interactive program which can aid a user in the construction of project networks suitable for the application of Operational Research optimization strategies. However, it is not specialized to any particular task domain and can be used as a general planning system. The user communicates to the system through a declarative Task Formalism (TF) and through other interactive facilities. NONLIN and a full BNF description of TF are documented in DAI Research Memo 25 (Tate, 1976). This report should be consulted along with comments in the source listings as documentation of the program. This present note also serves to indicate additions to the facilities provided in the current implementation. New facilities not documented in memo 25 will be indicated as such here.

The POP-2 source code is available in the file
POP:NONLIN.POP. A package of critical path routines is in
POP:NONLIN.CPM. This text is used in the help facilities and is in
POP:NONLIN.TXT. These 3 files together with sample data in TF
(see later) are backed up on DECTAPE 0092G with the same file names.

N.B. POP:(filename) is a POP-2 library file (filename) in PPN 140,141.

2. To run NONLIN

```
R POP2
(message printed by POP-2 system)
COMPILE(LIBRARY([LIB NONLIN]));
```

The planner will print out messages as it compiles the HBASE data base system (Barrow,1975). HBASE primitives are available for use independently if needed by a user. The NONLIN system asks if you require HELP information. If you type H followed by a (carriage return) it will print out this document. Otherwise just type (carriage return).
The system then asks if you wish to compile the critical path routines (type Y or N followed by a (carriage return)). The planner is then available.

HBASE occupies about 4k words and NONLIN a further 9k words of store on the DEC 10. NONLIN has been used to generate project networks up to about 150 activities (this taking about 180 seconds CPU) for simple house building domains.

3. Modes of use of NONLIN

The system can be used in 2 different ways, Stand-alone or interactively.

3.1 Stand-alone use

TF code can be typed directly to the system or can be compiled from a file. If all relevant information for a task is given beforehand and then planning requests typed at the system, no extra information will be sought by NONLIN and it will produce the answers (if possible) without user interaction.

E.g., if a file for a house building domain is in file HOUSE1.POP and it gives all relevant information to plan to do the ACTION ((BUILD HOUSE)) we could

```
COMPILE([HOUSE1.POP]);
PLAN ACTION ((BUILD HOUSE));
```

The system will then expand the ACTION using the TF forms given and will correct for any interactions found. The project network will then be printed as a table of the nodes in the network with format

(node no.) (node type) (previous node nos) (successor node nos) (node pattern)

Some TF domain descriptions which can be used in stand alone fashion are given on DECTAPE 0092G.

HOUSE1.POP — a house building domain which can be used to PLAN ACTION ((BUILD HOUSE)); It generates a network with 29 nodes.

HOUSE2.POP — As above for a project network of 48 nodes.

HOUSEC.POP — HOUSE1 domain with durations on activities - must have the critical path package available.

NLBLOCK.POP — Sacerdoti (1975) block stacking domain. Can give goals of ((ON x y)) and (CLEARTOP x)). E.g., PLAN GOAL ((ON A B)) GOAL ((ON B C));

NLROBOT.POP — A block stacking domain with a lower level of detail to generate actions such as ((MOVEHAND ((x y z)))). Useful as an example of how to use COMPUTE conditions. However, I have not had time to debug this package so it only works for certain patterns.

See Tate (1976) for how to give your own problem in TF and the general form of PLAN statements. You may find the above examples of use in this also. Since memo 25 was written the PLAN statement has been extended to allow PROTECT (pattern) ... ; to be written after the GOAL or ACTION statements.
E.g., PLAN GOAL ((ON A B)) PROTECT ((ON A B)) ((CLEARTOP E));

If a user wishes to delete the TF description to describe a new domain he should first type
NEWDOMAIN

3.2 Interactive Use

Facilities have been added since memo 25 to allow a top-down approach to the generation of the TF descriptions. If the full details of a domain are not available when some task is given to the planner, the system will prompt a user to provide extra information as it is found to be needed. The facilities in the present planner are only a weak form of the interactive facilities it is hoped will be provided in a more complete system after the "planning: a joint AI/OR approach" work is completed.

The facilities provided allows a task to be planned completely top-down, the system prompting when lower level detail is needed, down to some level which a user indicates can be considered primitive. More usually, lower level detail will be provided as a library of preplanned tasks or primitives and only the top level task must be broken down.

When a schema cannot be found to expand some pattern, the system will ask if you wish to

1. ABORT type 0

2. CARRY ON type 1
   Fail to expand the pattern and hence choose an alternative in the search space of the planner.

3. GIVE EXTRA TF FORMS type 2
   Give ACTSCHEMAs, OPSCHEMAs, PRIMITIVE or PRIMITIVE .. WITH EFFECTS .. declarations only. The system will go into POP-2 ready mode to enable this to be done. When you have given the information the system can use, type GOON.

To aid in such interactive use, the system gives diagnostic information if it fails on any sub-task and considers alternatives. One error not described in memo 25 is

UNSUPERVISED CONDITION NOT SATISFIED AT NODE n (pattern)

This error could indicate an error in describing a condition (check spelling) or indicate that the task which should have established this condition failed to advertise that fact. It may also be that the system has linearized the network in such a way that the condition could not be satisfied. For that reason after giving the error report the planner thus goes on to consider alternatives.

After a top-down approach to generating a network, it may be useful to read back the TF description so that it may be reused. This can be done for each (schema) in the list ALLFNS by:

```
OPEXPANSION((schema)) -> ALLNODES; LENGTH(ALLNODES) -> NUMNODES;
PRINTNET();                          to print the expansion.
GENPR(OPPATTERN((schema())); to set its PATTERN
APPLIST(OPCONDITIONS((schema)),GENPR); to print the conditions.
```

The list of PRIMITIVES can be printed from the list PRIMLIST.

4. Getting further solutions

If a task is successfully solved, it may be possible to get other solutions from the choice points generated by the search

REPLAN reenters the planner to get further solutions. It can be done until no further solutions are given.

5. Diagnostics and other interactive facilities.

Detected errors at TF definition time and at PLAN time are described in Tate (1976). Also in that paper is a list of functions which can be used to examine a project network after an error or during user interaction.

e.g., PRINTNET(); prints the project network.

Various switches can be set to give extra diagnostics and trace information, e.g., BUGEXPAND. See Tate (1976) for these switches and their effects.

Since memo 25 appeared a function to print the critical path data at any stage of network construction has been provided.

PRCPDATA(ALLNODES) gives information for each node in the network in the form

⟨node no.⟩ ⟨node pattern⟩ ⟨duration⟩ ⟨early finish⟩ ⟨late finish⟩

Note that ⟨slack⟩ = ⟨late finish⟩ - ⟨early finish⟩ = 0 for critical nodes.

References

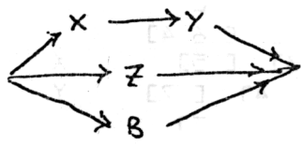
Barrow, H.G. (1975) HBASE: POP-2 library documentation DAI.

Daniel, L and Tate, A (1976) Planning: a joint AI/OR approach
AISB newsletter

Sacerdoti, E.D. (1975) The nonlinear nature of plans IJCAI-75

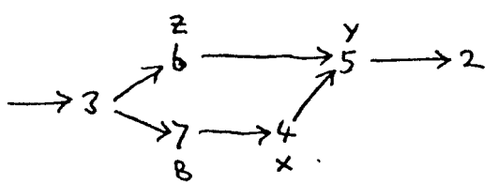
Tate, A. (1976) Project planning using a hierarchic non-linear planner
DAI research memo no. 25.

```
: COMPILE([NONLIN.POP]):
[BASE] READY.
[BASE ACTOR] READY.
[BASE CONTXT] READY.

ACTOR RESTRICTION ON VARIABLES. $*. INST AND NICE FACILITIES AVAILABLE.

NONLIN PLANNER READY          NOVEMBER 1976
*****************************************

THIS IS A NON-LINEAR PLANNING SYSTEM WHICH ACCEPTS DOMAIN DESCRIPTIONS
WRITTEN IN TASK FORMALISM (TF).  THE SYSTEM IS DOCUMENTED IN D.A.I MEMO
NUMBER 25. IF YOU WISH FURTHER HELP INFORMATION TYPE "H" NOW:

CRITICAL PATH PACKAGE REQUIRED (Y OR N): Y
: PRIMITIVES    X WITH EFFECT + P
:               Y
:               Z WITH EFFECT + Q;
: PLAN ACTION A;

LEVEL  0
AN EXPANSION IS NOT GIVEN FOR A AT NODE 1
DO YOU WISH TO ABORT(0), CARRY ON(1) OR ADD EXTRA SCHEMAS(2): 2

READY
:: ACTSCHEMA ANONYMOUS
     PATTERN    A
     EXPANSION  1 ACTION X
                2 ACTION Y
                3 ACTION Z
                4 ACTION B
     ORDERINGS 1 ---) 2
::   CONDITIONS SUPERVISED P AT 2 FROM 1
::              UNSUPERVISED Q AT 2
:: END;
:: GOON
```




```
LEVEL  0
AN EXPANSION IS NOT GIVEN FOR B AT NODE 7
DO YOU WISH TO ABORT(0), CARRY ON(1) OR ADD EXTRA SCHEMAS(2): 2

READY
:: PRIMITIVE B WITH EFFECT - P;
:: GOON

     === 1 INTERACTION    2 LINEARIZATIONS
     +++CHOICE ADDED ALTLINEAR
TRY TO SATISFY UNSUPERVISED CONDITIONS
     (-) LINK TO MAKE  Q HAVE VALUE 1 AT NODE 5
     === 0 INTERACTION    1 LINEARIZATIONS

NONLIN 2 TERMINATED.  CPU TIME =  1.119 SECS
```

```
    PLANHEAD    NIL  [ 3]
    DUMMY       [ 5]  NIL
    DUMMY       [ 1]  [ 7 6]
 4  ACTION      [ 7]  [ 5]       X
 5  ACTION      [ 6 4]  [ 2]     Y
 6  ACTION      [ 3]  [ 5]       Z
    ACTION      [ 3]  [ 4]       B
```



```
    ONE

CONTEXT 17

 <<  P  7 >>      0
     Q  6 >>      1
     P  4 >>      1

: PROST

CONTEXT 18

     SUPERVIS  P     5          [  4]
     UNSUPERV  Q     5          [  6]

: REPLAN

     ---CHOICE TAKEN ALTLINEAR
TRY TO SATISFY UNSUPERVISED CONDITIONS
     (-) LINK TO MAKE  Q HAVE VALUE 1 AT NODE 5
     === 0 INTERACTION    1 LINEARIZATIONS

NONLIN 2 TERMINATED.  CPU TIME =  0.222 SECS


 1  PLANHEAD    NIL  [ 3]
    DUMMY       [ 7]  NIL
    DUMMY       [ 1]  [ 6 4]
 4  ACTION      [ 3]  [ 5]       X
 5  ACTION      [ 6 4]  [ 7]     Y
 6  ACTION      [ 3]  [ 5]       Z
 7  ACTION      [ 5]  [ 2]       B

: REPLAN

NO WAY TO PROCEED
setpop
:
```