PROJECT PLANNING USING A HIERARCHIC NON-LINEAR PLANNER

by

Austin Tate

D.A.I. Research Report No. 25

August 1976

PROJECT PLANNING USING A HIERARCHIC NON-LINEAR PLANNER

Austin Tate
Department of Artificial Intelligence
University of Edinburgh

abstract

We describe work on a project aimed at producing an interactive program for the construction of project networks (e.g. for house building tasks). To do this we have developed a planner which can form plans represented as a partially ordered network of actions. A formalism (TF) is given for describing a domain in a hierarchic fashion. The representation of plans and the planner (NONLIN) are fully explained. During this work, a general technique was developed for answering queries about a situation when the information about the world is stored as a partially ordered network of alterations made to some initial situation. We give a general procedure for recognizing and correcting for interactions between actions in the network. This is based on an analysis of the goal structure of the problem. The work is compared to that of Sacerdoti (1975a) who pioneered the techniques of planning using plans represented as partially ordered networks of actions.

Contents

PROJECT PLANNING USING A HIERARCHIC NON-LINEAR PLANNER

1. Planning and Task Specification

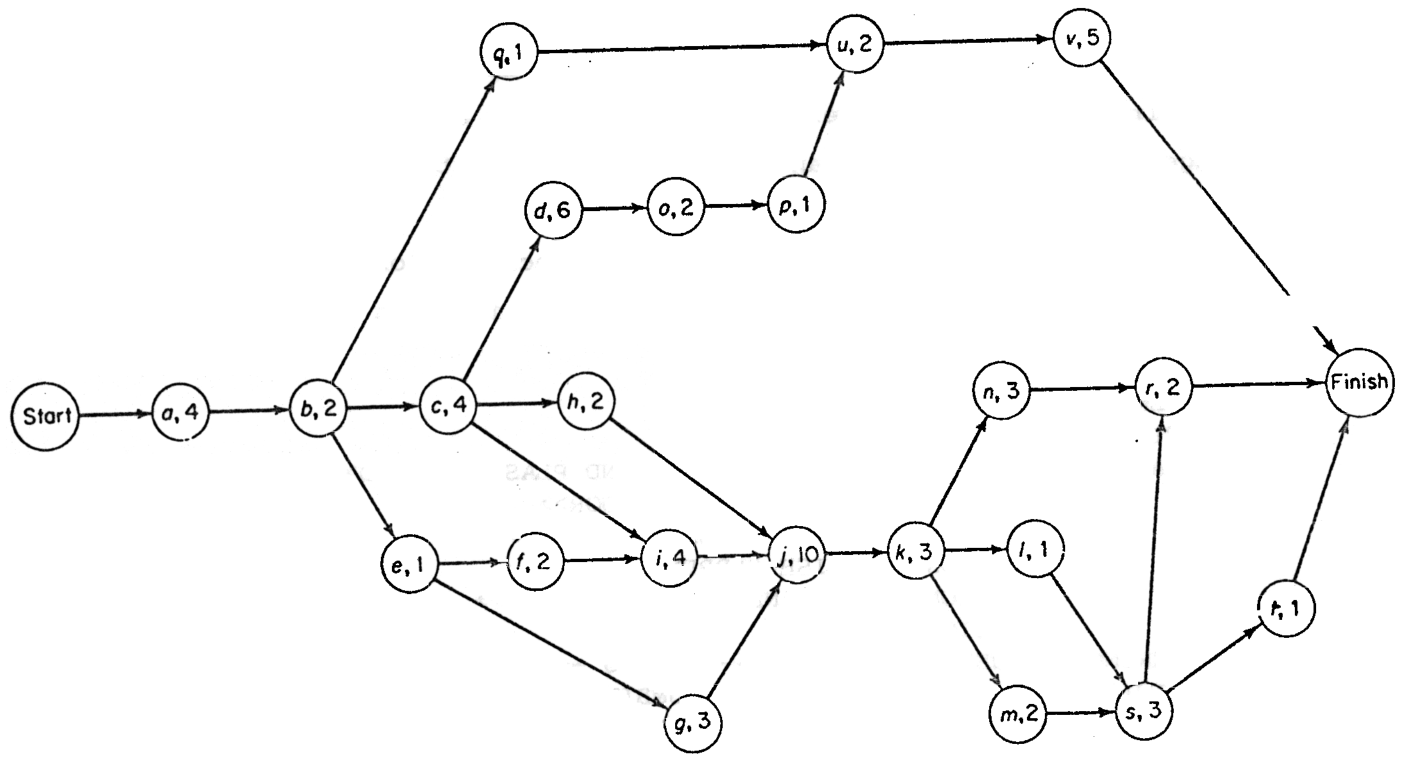
Recent work on the generation of plans has been concerned with producing plans which could be used to transfer the knowledge of an expert or experts in a domain to a novice. Work of this type has been performed on the Computer Based Consultant Project (Hart, 1975) at Stanford Research Institute and on 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 (a 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. As an example of the type of project network upon which optimization techniques can be used, consider a house building task whose component jobs are given in the table below (taken from Wiest and Levy, 1969, which gives an introduction to project planning, PERT and critical path analysis techniques).

	Description	Immediate Predecessors	Time (days)
a	Excavate, Pour Footers		4
b	Pour Concrete Foundations	a	2
c	Erect Frame and Roof	b	4
d	Lay Brickwork	c	6
e	Install Drains	b	1
f	Pour Basement Floor	e	2
g	Install Rough Plumbing	e	3
h	Install Rough Wiring	c	2
i	Install Air Conditioning	c,f	4
j	Fasten Plaster and Plaster Board	g,h,i	10
k	Lay Finished Flooring	j	3
l	Install Kitchen Equipment	k	
m	Install Finished Plumbing	k	2
n	Finish Carpentry	k	3
o	Finish Roofing and Flashing	d	2
p	Fasten Gutters and Downspouts	o	1
q	Lay Storm drains	b	1
	Sand and Varnish Floors	n,s	2
s	Paint	l,m	3
t	Finish Electrical Work	s	1
u	Finish Grading	p,q	2
v	Pour Walks, and Landscape	u	5

The jobs in this table and the orderings given between them define the following project network (of the "Activities on Nodes" type). Each node gives the node letter and the duration of the activity. If two jobs are not ordered (i.e. are in "parallel"), this means that the jobs could be done in any order or simultaneously (depending on some executing strategy).



AON Diagram—House Construction Project

Operational Research has concentrated on the optimization and scheduling problems for some given network. However, the network can be difficult and time-consuming to set up. It may also be 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. Once a network is constructed and is in use on a project, much effort can go into modifications to keep it up to date with actual progress on a task. The Planning: a joint AI/OR approach project has initially been concerned with aiding a user in the process of constructing a project network. To do this, as in the work of Sacerdoti, we have been investigating the use of a partially-ordered network of actions to represent a plan (or project) at any stage of development. 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

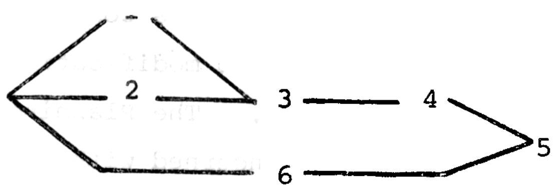
ii) an action interferes with an important effect of another action and must be removed outside its range

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.

An "ACTSCHEMA" from a house building specification in the Task Formalism is given below (the complete listing is given later in section 7).

```
ACTSCHEMA DECOR
   PATTERN <<DECORATE>>
   EXPANSION 1 ACTION <<FASTEN PLASTER AND PLASTER BOARD>>
             2 ACTION <<POUR BASEMENT FLOOR>>
             3 ACTION <<LAY FINISHED FLOORING>>
             4 ACTION <<FINISH CARPENTRY>>
             5 ACTION <<SAND AND VARNISH FLOORS>>
             6 ACTION <<PAINT>>
   ORDERINGS SEQUENCE 2 TO 5  1 --->3   6 --->5
  CONDITIONS UNSUPERVISED <<ROUGH PLUMBING INSTALLED>> AT 1
             UNSUPERVISED <<ROUGH WIRING INSTALLED>> AT 1
             UNSUPERVISED <<AIR CONDITIONING INSTALLED>> AT 1
             UNSUPERVISED <<DRAINS INSTALLED>> AT 2
             UNSUPERVISED <<PLUMBING FINISHED>> AT 6
             SUPERVISED <<PLASTERING FINISHED>> AT 3 FROM 1
             SUPERVISED <<BASEMENT FLOOR LAYED>> AT 3 FROM 2
             SUPERVISED <<FLOORING FINISHED>> AT 4 FROM 3
             SUPERVISED <<CARPENTRY FINISHED>> AT 5 FROM 4
             SUPERVISED <<PAINTED>> AT 5 FROM 6
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>> (6), achieves the SUPERVISED condition <<PAINTED>> on ACTION 5). UNSUPERVISED conditions are made true by other experts (mainly here by an "INSTALL SERVICES" expert). Another condition type, "HOLDS", would say that an ACTSCHEMA containingit should not be used unless the condition was true. There are no effects specified in this ACTSCHEMA as they are all 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 would be difficult to extract from the detail of the plan itself. The use of goal structure to direct search in a problem solver was first investigated in Tate (1975). The goal structure of a plan not only provides information to aid the search of the planner, it contains valuable information for monitoring the execution of the plan and for directing such modifications as are necessary to the plan so that it will achieve its purpose after execution failures. NONLIN and the task formalism (TF) are fully described in this paper.

2. Task Formalism for domain specifications

At the outset of this work the problem of specifying a domain to a problem solver in a hierarchic fashion was recognized as being of primary importance. A uniform and straightforward method of description was sought to allow a domain to be specified hierarchically. We believe the formalism described below gives a powerful descriptive tool. However, we would anticipate some interface being used to give the user greater flexibility.

2.1 Hierarchic Specification

We wish to allow 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. It should be possible for each component at lower levels to be specified in a modular way - not requiring knowledge of the exact form of other components.

For example, consider a house building task, someone at the highest level decides he can build a house if he had (a) built some walls and a roof and had carried out (b) the installation of services and (c) done the fitting out and decoration. The builder may know nothing about how to perform some of the sub-tasks so he asks experts in each field to decide on ways to do these. The experts in turn specify jobs to be done or lower level tasks to be planned by further, more specialized, experts. Eventually we get down to the "shop floor" and tradesmen presumably signal that they can or cannot carry out their appointed task at the appointed time in the overall plan. The planner is left with the task of ensuring that each part can be performed successfully and that the overall purpose of the plan is achieved.

2.2 Conditions for sub-tasks

When sub-tasks are being planned the experts may know that the individual jobs ought to be done in a particular order, or know that several jobs can be done together (in parallel). They know that certain conditions ought to hold before some jobs can proceed. For example, a carpet layer knows that before he does his job the floor boards ought to be layed, even though that isn't part of his job. These conditions are not under the supervision of this expert and are the responsibility of others. They will be termed UNSUPERVISED CONDITIONS.

Experts also know that certain conditions must be made to hold under their supervision before their task can be completed. Again, the carpet fitter knows it is his responsibility to get the carpet to the site, but the details of that task may be sub-contracted. Such conditions will be termed SUPERVISED CONDITIONS. N.B. as we will see these correspond to normal preconditions in means-end analysis driven systems such as STRIPS (Fikes and Nilsson, 1971).

There is a third type of condition which an expert may impose. Conditions may be stated which must hold before this expert can be called into use at all. For example, consider a block stacking expert which knows how to clear blocks by moving a block on top of the block to be cleared to some other place. If a block cannot be found to be on top of the one to be cleared it is no use calling this expert at all. If the conditions were merely stated as a goal to be achieved before the movement of the upper block to somewhere else was done, we could get into a situation where we actually move some block onto the one to be cleared and then move it off again. Such static conditions on the use of a particular expert will be called HOLD CONDITIONS. Hold conditions can be considered to be an extension to the check of relevancy of some operator which imposes them (extra to the actual pattern-directed invocation - i.e., the check that this operator adds some sought for pattern). They can also be used to instantiate variables for some operator in the way that the pattern directed invocation does, or to impose global conditions (i.e. those which should never be achieved).

So we appear to need three different types of conditions:*

(a) Unsupervised conditions

(b) Supervised conditions

(c) Hold conditions

Only conditions of type (b) should cause further expansions to be made to the plan being generated, i.e., allow further experts to be called in to plan to achieve the conditions. This is why they correspond to normal preconditions as specified by any operator in a means-end analysis driven system. The other two types of conditions merely direct the planner when to use the operator. Hold conditions will allow acceptance or rejection of the use of an operator at any given point in a plan.

---

*Of course in any particular domain description certain types of conditions may not be used. In fact another condition type, COMPUTE, is provided to allow the interface of computational processes (see section 2,3.1).

Unsupervised conditions specify ordering constraints on when this operator can appear. In this latter case, if expansions were allowed to achieve the unsupervised conditions we could find that the net contained much redundancy which could be difficult to resolve. It seems better to allocate jobs to the appropriate experts.

2.3 Task Formalism (TF)

I wanted a completely declarative representation of the knowledge of experts (c.f. NOAH's SOUP code) and the effects of actions. The representation is based on the operator schemas provided in STRIPS (Fikes and Nilsson, 1971). The components of the representation are given below. An example will follow in case the BNF gets a bit much (as in the POP-2 manual <...>? means 0 or one example of <...> - i.e. optional, <...>*? means 0 or more examples of <...>).

2.3.1 OPSCHEMAs

(i) OPSCHEMA|ACTSCHEMA introduces the definition of the expert.

(ii) <name> a POP-2 variable through which the opschema can be accessed. Any components can now be defined in any order by writing their codeword. Typing "END" ends the definition. Typing REVIEW <name> allows further components to be added to or defined.

One can also use a <name> ANONYMOUS if subsequent REVIEWing is not needed.

The order of the following components is immaterial. ";" is ignored between components but is needed after a VARS (see vi).

(iii) PATTERN <pattern> An HBASE pattern used for pattern-directed invocation of this opschema.

e.g. expansion of GOAL <pattern> will cause this opschema to be considered. A simple extension to the definition of a pattern component is envisaged to allow one opschema to be used to achieve several goals e.g., PATTERN <patt1> OR <patt2> ...;

(iv) <expansion form>

The expansion this expert suggests to achieve the invocation <pattern>. An expansion is made up of a collection of nodes with ordering information and a collection of conditions which must be true at the points at which certain nodes in the expansion are placed. DUMMY nodes are automatically inserted where necessary to give a unique start and end node to the ordering, or they may be given by a user to enable conditions to be attached at points not represented by a normal node.

```
<expansion form>::=
   EXPANSION <nodes defn>? <orderings defn>? <conditions defn>?
   | <conditions defn>   If no <nodes defn> given then it only
                         possible to specify conditions AT SELF
                         (e.g. in defining a primitive action)
```

<nodes defn>::= <node form>*?

<node form>::= <nodenumber>? DUMMY

| <nodenumber>? <nodetype> <pattern> <node cost>?

Node numbers are optional for a user's convenience. As the system numbers nodes in the order they are given in <nodes defn>, take care the <node number>s in an <orderings defn> refer the numbers given by the system.

<nodenumber>::= <integer>

<nodetype>::= GOAL|ACTION

<node cost>::= : <number>

Nodes are given zero cost by default. A cost assigned in a schema overrides one assigned by COST (see 2.3.2).

<orderings defn>::=

ORDERINGS <ordering form>*?

We are currently considering a graphics system for the input of the partial ordering on nodes.

<ordering form>::=

<nodenumber> <ordering separator> <nodenumber>

SEQUENCE <nodenumber> TO <nodenumber>

SEQUENCE 2 TO 4 is equivalent to 2 ---> 3 and 3 ---> 4. Redundant links will be tidied up by the system. Ensure <nodenumbers> are in the range of nodes given and refer to the order of nodes as given in the <nodes defn>.

<ordering separator>::=

.ny POP-2 word not a marco>

--->" is used in the examples given in this paper.

<conditions defn>::=

CONDITIONS <condition form>*?

<condition form>::=

AUTO Fill in supervised conditions FROM each GOAL node AT following node automatically for this schema. This overrides the variable AUTOCOND being set to 0. AUTOCOND set to 1 means this is done for each schema anyway.

NONAUTO This overrides the variable AUTOCOND being set to 1 for this schema.

SUPERVISED <not>? <pattern> AT <nodenumber> FROM <contributors>

| UNSUPERVISED <not>? <pattern> AT <AT specification>

HOLDS <not>? <pattern> AT <AT specification>

COMPUTE <not>? [<pattern> = <compute clause>]

HOLD and COMPUTE conditions are tested by the system in the order given, hence variables will be set in the order given. A COMPUTE condition is provided as a temporary way of interfacing to computational facilities or numeric data bases. If the <compute clause> is a pattern, its value is looked up in ALWAYCTXT and matched against the <pattern> before the =. Otherwise the <compute clause> is interpreted as a function call on several arguments (which could be OPSCHEMA variables) which will produce a value to be matched against the <pattern> before the =.

e.g. COMPUTE [<<$*X $*Y $Z >> = POSITION $*BLOCK].

<AT specification>::=

<nodenumber> | SELF

~~Only~~ SELF allowed if there is no <nodes defn>.

<contributors>:

<nodenumber>   [<nodenumber>*]

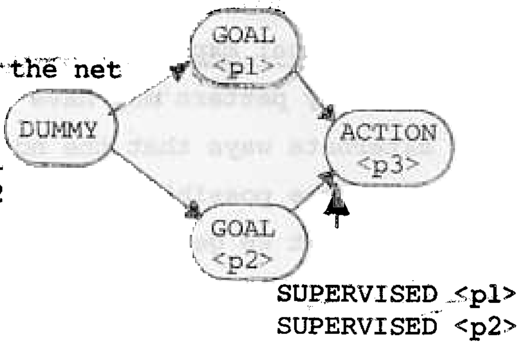
<compute clause>::=

~~<pattern>~~ | <POP-2 function name> <argument>*?

<not>::=   NOT

An expansion therefore defines a collection of nodes with ordering information between them, and specifies any conditions before any node. For example, an expansion written:

```
EXPANSION 1 GOAL <p1>
          2 GOAL <p2>     would define the net
          3 ACTION <p3>
ORDERINGS 1 --->3  2 ---> 3
CONDITIONS SUPERVISED <p1> AT 3 FROM 1
           SUPERVISED <p2> AT 3 FROM 2
```



CONDITIONS AUTO and AUTOCOND

To save repetition of GOAL <pattern> node forms and a corresponding CONDITION SUPERVISED <pattern> condition form on a succeeding node, a variable AUTOCOND may be set true to provide the CONDITION statements automatically for the node following each GOAL <pattern> one. This will occur commonly.

e.g. with AUTOCOND true

```
EXPANSION 1 GOAL <p1>       is the same  EXPANSION 1 GOAL <p1>
          2 GOAL <p2>       as writing             2 GOAL <p2>
          3 ACTION <p3>                            3 ACTION <p3>
ORDERINGS 1 --->3  2 --->3               ORDERINGS 1--->3 2--->3
                                         CONDITIONS SUPERVISED <p1> AT 3 FROM 1
                                                    SUPERVISED <p2> AT 3 FROM 2
```

AUTOCOND set to false may be overridden for any individual schema by writing CONDITIONS AUTO in its <expansion form>. AUTOCOND set to true may be locally overridden by writing CONDITIONS NONAUTO.

Default Expansions in OPSCHEMAs

OPSCHEMAs are given a null expansion by default until the expansion is defined. At run time only ACTION nodes can have a null expansion. GOAL nodes should have some expansion.

GOAL AND ACTION roles in EXPANSIONS

The difference between putting GOAL <pattern> and ACTION <pattern> in an expansion should be mentioned. An expansion of either causes the list of OPSCHEMAs and ACTSCHEMAs to be searched for any whose PATTERN component matches the <pattern> of the node being expanded. However, the GOAL <pattern> node is given an effect which says that <pattern> will be true at this node. This is additional to any MAINEFFECTS defined for a pattern (see 2.3.2). The effect can be used to detect interactions etc. before expansion. Efforts will be made to see if a <pattern>at a goal node is true before ways to expand the <pattern> are tried. The ACTION <pattern>node is not given such an effect and no attempt is made to see if its <pattern> is true at the point in the net.

Conditional Expansion of a pattern

Any pattern may have several opschemas. Each of these may specify alternate ways that the node with the pattern can be expanded. It is therefore possible to specify different hold conditions in an expansion such that we get conditional expansion of the pattern.

(v) EFFECTS <effect form>*? The changes made to the world model after the last component of the expansion has been executed.

```
          <effect form>::=  + <pattern> means true → value (<pattern>)
                            - <pattern> means false → value (<pattern>)
    VARS <var form>*? ;
                <var form>::= <POP-2 variable name> <initial value>
          <initial value>::= UNDEFined variable, can be set to any value.
                            | <HBASE actor>  (restriction on variable)
                            | <any particular value>
```

Variables in the components of an opschema with prefix $* are semi-open variables (Bobrow & Raphael, 1974) and refer to the variable specified in VARS (an alist). See appendix I for details of $* and variable restriction.

(vii) END ends the definition of the opschema.

As mentioned previously the form given above may be considered to be too free in the sense that it does not structure a user's thoughts to let him define a domain in a simple way. Following a suggestion made by Jerry Schwarz I introduced ACTSCHEMA as an alternative header word for an OPSCHEMA. An ACTSCHEMA has the same syntax and components as an OPSCHEMA except that ACTSCHEMA replaces OPSCHEMA at its beginning. They are semantically equivalent. However, it is proposed that a user may consider that OPSCHEMAS are definitions of how to achieve certain GOALS. They should have an EXPANSION component but should not specify the effects of the execution of the expansion. This should be left to ACTSCHEMAs which define the effects of the actions but should not have any expansion component (except, possibly, for adding conditions). This may help non-expert users of the system to code simple domains.

e.g., Consider Sacerdoti's SOUP formulation of the block stacking domain (see Sacerdoti, 1975a) translated to TF (our task formalism). The variable AUTOCOND is set true to automatically insert the SUPERVISED conditions before any node following a GOAL node (see iv above). The layout is for the human reader only and can be in any style a user may prefer.

```
ACTSCHEMA PUTON
  PATTERN <<PUT $*X ON TOP OF $*Y>>
  CONDITIONS HOLDS <<CLEARTOP $*X>> AT SELF
             HOLDS <<CLEARTOP $*Y>> AT SELF
             HOLDS <<ON $*X $*Z>> AT SELF
  EFFECTS + <<ON $*X $*Y>>
          - <<CLEARTOP $*Y>>
          - <<ON $*X $*Z>>
          + <<CLEARTOP $*Z>>
  VARS X UNDEF Y UNDEF Z UNDEF;
END;
```

```
OPSCHEMA MAKEON
  PATTERN <<ON $*X $*Y>>
  EXPANSION 1 GOAL <<CLEARTOP $*X>>
            2 GOAL <<CLEARTOP $*Y>>
            3 ACTION <<PUT $*X ON TOP OF $*Y>>
  ORDERINGS 1 --->3  2 ---> 3
  VARS X UNDEF Y UNDEF;
END;

OPSCHEMA MAKECLEAR
  PATTERN <<CLEARTOP $*X>>
  EXPANSION 1 GOAL <<CLEARTOP $*Y>>
            2 ACTION <<PUT $*Y ON TOP OF $*Z>>
  ORDERINGS 1 ---> 2
  CONDITIONS HOLDS <<ON $*Y $*X>> AT 2
             HOLDS <<CLEARTOP $*Z>> AT 2
  VARS X <:NON TABLE:> Y UNDEF Z <:ET <:NON $*X:> <:NON $*Y:> :>;
END;

ALWAYS <<CLEARTOP TABLE>>;
```

N.B. There are two separate OPSCHEMAs, one of which knows how to clear the top of a block and the other knows how to place one block above another. However, there is only one action involved and the effects of this are described separately. The two OPSCHEMAs act as extra sources of knowledge on how to use the primitive action to accomplish certain tasks.

Now, of course, it is possible to give both an expansion and effects to an OPSCHEMA (or ACTSCHEMA). For example, high level action descriptions can be defined by giving an ACTSCHEMA an expansion component.

e.g. (PUTON X Y) may expand to three lower level actions

(PICKUP X); (GOABOVE Y); (PUTDOWN X ON Y).

and effects may be given at the higher level as well as the lower. The higher level action may refer to effects on different predicates than the low level actions. Similarily, effects can be added to OPSCHEMAs if required. The possibilities are endless. I will not pursue the design criteria for a formalism which may be used as an interface from a user to the OPSCHEMAs which NONLIN uses. HOWEVER, the design of TF will be a major component of the research being undertaken on the "Planning: a joint AI/OR approach" project.

2.3.2. Other Definitions in TF

(a) The 'ALWAYS <pattern>' statements (as used in the block stacking example in section 2.3.1) says that <pattern> is true in any situation.

(b) MAINEFFECTS is used to declare the effects of a particular action which are independent of the choice of an expansion for it. It can also be used to declare logical consequences of a goal pattern being true at some point in the plan. Main effects are those effects always associated with the occurrence of the <pattern> in the plan.

```
MAINEFFECTS <pattern>
  VARS <var form>*? ;  (not needed if no variables in the <pattern>)
  <effect form>*?
END;
```

where <var form> is defined in 2.3.1 (vi) and <effect form> in 2.3.1 (v)

(c) PRIMITIVE <pattern>* ; declares each of the <pattern>s given to have null expansions. This saves writing an OPSCHEMA for each and is also more efficient for the planner. Of course, as mentioned previously, only actions can be declared PRIMITIVE.

2 further options are allowed in PRIMITIVE.

i) PRIMITIVE <pattern> with EFFECT <effect form>;
or PRIMITIVE <pattern> with EFFECTS <effect form>*;
is equivalent to writing PRIMITIVE <pattern>; and giving MAINEFFECT <pattern> <effect form>* END;
This appears to be a commonly useful facility.

ii) any <pattern> in a primitive declaration (after any effects if these are given) can be assigned a cost (duration etc.) by following it by a <node cost> (e.g. :12). This may be easier to write than a COST definition.

(d) LEVELS <level defn>* ;

<level defn>::= <pattern> <number>

LEVELS is used to define the hierarchy of <pattern>s which can be expanded during planning. The planner tries to expand <pattern>s which are highest in this defined hierarchy first. A lower number indicates a pattern higher in the hierarchy. Any <pattern> not given a level is assumed to have level 0 and is thus expanded first. It is not essential to give levels for planning to be successful, however, it may make it more efficient. It should be possible to build up the level list automatically in due course.

(e) COST <cost defn>* ;

<cost defn>::= <pattern> <number>

COST is used to define node costs or durations for any pattern. The cost of any node is stored in the NODECOST component and can be used by critical path or other algorithms either during or after planning. It is not used in the current version of the program (August 1976). Costs may also be given in a PRIMITIVE declaration (see (c) above) or in the expansion of any schema (by following a node definition by : number). A cost assigned within a schema overrides one assigned by the COST form. Nodes by default are given zero cost.

3. Representation of the Network

For reasons which I hope will become clear as the planner is described the network is represented as follows:

- a collection of nodes kept in a flexible strip ALLNODES of current length MAXNODES containing NUMNODES nodes. The strip is increased as necessary in increments of INCRNODES. Nodes are referred to via their subscript in ALLNODES e.g. NODE(4). Lists of PRENODES and SUCCNODES for each node are kept and refer to the appropriate ALLNODES subscripts. By convention, NODE(1) is the start node of the network, and NODE(2) is the end node. Access to these is needed for the calculations of critical path length.
- Each node has associated with it

nodenum	its subscript in ALLNODES
nodetype	GOAL, ACTION, PHANTOM, PLANHEAD or DUMMY
pattern	used to seek an expansion
nodectxt	a context containing the effects of this node. The partially ordered network of contexts is defined by the nodectxts and the prenodes and succnodes links.
expansion expanconds	together these 2 components hold the network and conditions for <u>an</u> expansion of this node (these would be set after a choice of expansion.
parentnode	the node was inserted as a result of the expansion of its parentnode
prenodes	a list of nodes linked immediately before this one
succnodes	" " after "
nodecost	a cost for the execution of this node (0 by default).
nodevars	a list of variables which may occur in the expansion or expanconds together with their values (an alist)
nodemark	a marker used to discover the relation of this node to any other in the network "BEFORE", "NODE", "AFTER" or 0 (for parallel node) with respect to a node held in the variable NETMARKED. It is also used to help re-locate a chosen expansion being inserted into ALLNODES (see section 5.5)

The network thus described gives the ordering constraints between the nodes in the network. 2 other structures are used, a TOME and a GOST.

- <u>Table of Multiple Effects (TOME)</u>

  As in Sacerdoti (1975a) this provides a simple means of determining the

values given to any pattern at any node quickly.
It is used to detect interactions.
Its entries are HBASE items

<< <pattern> <nodenum> >> and have value <value>.

- Goal Structure (GOST)

any conditions on any node is stored in GOST together with a list of "contributors". Contributors are nodes, any one of which could make this condition hold. See section on the QA system for how the contributors for any pattern are found.

A GOST entry is

<< <condtype> <pattern> <value> <nodenum> >>
and has value <list of contributors>

<condtypes> currently dealt with are those outlined in section 2.2, SUPERVISED, UNSUPERVISED and HOLDS. A system generated <condtype> PHANTOM can also appear. The Goal Structure therefore allows the purposes of any particular effect at any node to be determined (if it has any). This allows interactions to be detected and allows corrections (suggested linearizations) to be sensitive to the important effects of nodes (those with purposes). Unimportant effects are therefore ignored.

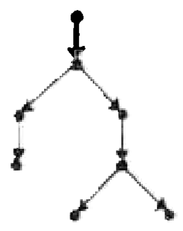
Once the interacting effects of nodes are determined and the goal structure is available simple linearizations can be suggested to remove the interactions (as in INTERPLAN - Tate, 1975a and 1975b). Two special nodenums are used in the GOST, these are 0 for ALWAYCTXT as a contributor and -1 for INITCTXT as a contributor (these are needed because of the distributed world model representation - see section 4 on QA system).

Note that TOME and GOST entries as well as the pointers between nodes are in terms of node numbers (entries in ALLNODES). This convention allows a simple scheme to be used to copy a network when needed for a choice point and allows expansions to be made with the minimum of effort. The entries are also arranged so that efficient lookup via HBASE SUPITEMS of a pattern is possible.

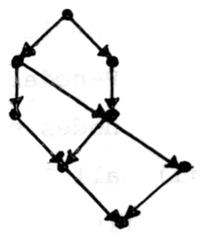
a) P on its own is a NODECTXT entry
b) P in the 1st place is a TOME entry
c) P in 2nd place is a GOST entry

4. Question answering in a partially ordered network of contexts

Current data base systems which provide a context mechanism, e.g. CONNIVER (McDermott and Sussman, 1972) and HBASE (Barrow, 1975), provide efficient facilities for storing a changing data base by remembering the alterations made to an initial situation. However, they only provide facilities for the determination of the value of a pattern with respect to a given context in a fully ordered tree of contexts. The nature of non-linear planning makes it very desirable that we store only the changes made to a world model to allow the determination of the value of patterns with respect to a given context in a partially ordered network of contexts. In such a network we define the order in time of 2 contexts to be specified if there is a chain of time ordering links between them. In the diagram below x → y means x before y.



A tree of contexts



A partially ordered network of contexts

In a tree of contexts there is a strict time sequence along a single context path so answers are fully determinate. In the partially-ordered network answers will depend on the nodes in parallel with a particular node as well as the answer got by retracing back through a network. This answer, got by retracing through the network, will itself vary as nodes are linked earlier in the network. This is the reason why a full world model should not be kept for each context and why it is best to store only the changes to the world model at each point.

The questions asked of the QA system will be of two kinds:-

(a) Does P have value V at node N in the current network? It could have value definitely V, definitely not V, UNKNOWN because it is both V and a value other than V on parallel branches or undefined because it is not mentioned anywhere.

(b) What links would have to be added to the network to make P have a certain value if it did not have this value in the given network?

The QA system will recognize its function by a parameter "NONLINK" for (a) or "LINK" for (b). The system finds 4 lists of nodes in the network and uses these to give a truth result for the request. The lists contain the information needed to suggest links if this is required.

<u>defns</u> - P-node is a node which gives statement P a value.

<u>PV-node</u> " " " " " " " " " V.

<u>PNOTV-node</u> other than V.

a <u>critical node</u> for (P,N) is a node which, in a possible linearization, gives a value to statement P for the last time before node N. (i.e. could be made to be the nearest predecessor of N which gives a value to P in some legal linearization).

<u>N.B.</u> The critical nodes for (P,N) are

i) the last P-node on each incoming branch of N (ignore P-nodes which are also predecessors of any other critical nodes since there may be redundant links in the network).

ii) all P-nodes which are in parallel with N.

The system finds lists	VL	critical	PV-nodes which are predecessors of N
	VNOTL	"	PNOTV-nodes "
	PARVL		PV-nodes which are in parallel with N
	PARVNOTL		PNOTV-nodes " "

QA (P,V,N, "LINK" of "NOLINK") first attempts to see if P definitely has value V at node N or definitely has not.

P definitely has value V at node N if there is at least one critical PV-node before node N, and there are no critical PNOTV-nodes.

P definitely does not have value V at node N if there is at least one critical PNOTV-node and there are no critical PV-nodes.

If neither of these 2 definite cases arises then it may be possible to <u>make</u> P have value V at node N by making suitable links in the network. This is only done if "LINK" is signalled to QA. If "NOLINK" and there are critical PV-nodes but no critical PNOTV-node then return "undef" (i.e. P has value V at some node in parallel with N and is not given an opposite value anywhere) otherwise return "UNKNOWN". To make P have value V at node N it is necessary to have at least one critical PV-node <u>linked in</u> before node N and to <u>link out</u> all critical PNOTV-nodes (both parallel to and predecessors of N), so that they cannot be placed between at least one critical PV-node and N. The system will suggest a possibility list of different sets of compatible links to achieve the above conditions. The process used to suggest compatible links for this scheme is very closely based on the process which corrects for interactions in a network. The common linking procedure used is described later (see section 5.6). It is given the 4 lists, VL, PARVL, VNOTL and PARVNOTL.

Linking restrictions

It is legal to make a new link between 2 nodes

(a) if they are not linked in the opposite order.

(b) every node which achieves a pattern as a condition for some other node (its purpose) continues to do so.

Redundant links may be removed or left in. The planner to be described later removes redundant links to ease the printing of the networks.

The notion of contributors to a condition

It is possible that an answer to some question "does P have value V at node N" is answered positively and that there be more than one critical PV-node. In this case it is possible to remember that any of the critical PV-nodes could make the pattern P have value V at node N. We keep the list of critical PV-nodes as a set of contributors to any condition satisfied in this way. The system can allow the set of contributors to be reduced (e.g. by not correcting for certain interactions etc.) so long as at least one contributor remains for any condition. This facility is used extensively in the planner to prevent choices of linearization earlier than necessary.

A distributed world model

The QA system used by NONLIN, besides providing the facilities outlined above for storing the effects of nodes in the network and retrieving the value of any pattern in this network with respect to any node, also keeps two data bases.

There is a data base called ALWAYCTXT which holds the values of any patterns given as unchangeable or global facts e.g. <<CLEARTOP TABLE>> is always true.

There is a data base called INITCTXT which is set by the system to be a copy of the world situation the user asks a plan to be found in. Any items in ALWAYCTXT are removed from INITCTXT to keep them disjoint.

As in NOAH (Sacerdoti, 1975b, pages 20-22), whenever the value of a statement is modified in the plan it is removed from INITCTXT and put in the context held in the first node of the plan (a special PLANHEAD node). This means that statements in ALWAYCTCT, INITCTXT and the network itself are all disjoint and an answer for any query can be obtained efficiently by going to ALWAYCTXT and INITCTXT before interrogating the network (under the assumption that the number of patterns whose values are altered in a plan is small compared to the total number of facts known to the system).

<u>Getting a list of all patterns which match P with value V at node N</u>

The scheme outlined above is only satisfactory for finding whether a certain fully instantiated pattern has value V at node N in a network of contexts. It is sometimes necessary to get a possibility list of patterns which match some given pattern and which have value V at node N. We have provided a simple facility to do this in the case where "NOLINK" is required. The possibility list is made up of three parts

i) fetch all items matching P with value V from ALWAYCTXT

ii) fetch all items matching P with value V from INITCTXT

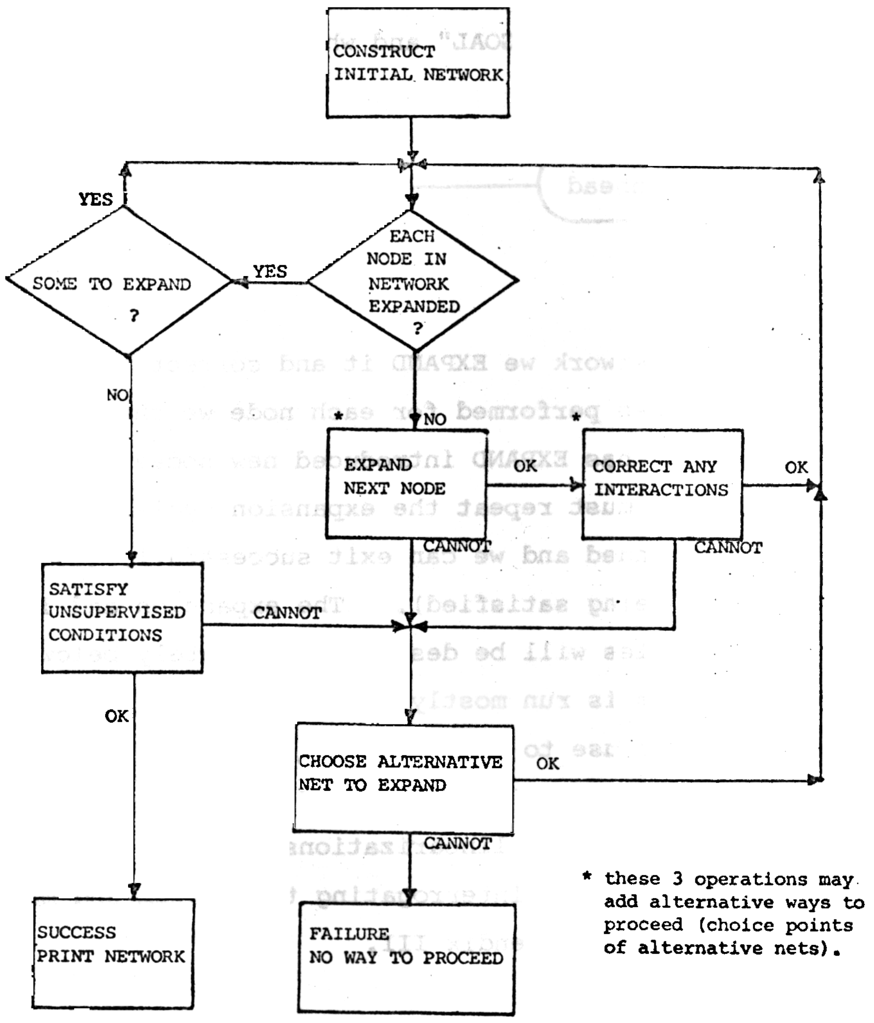
iii) the table of multiple effects (TOME) keeps entries for any pattern given a value at any node in the network. We get any TOME entry which gives value V to a pattern matching that required. The TOME entry gives us the node number at which the pattern is given the value. If this node is time linked <u>before</u> N, it is possible that the pattern may still have value V at N. This is checked using the algorithm defined for fully instantiated patterns.

<u>Note</u> I am grateful to Gottfried Eder for discussions during the design of this algorithm. He suggested the terminology of "critical nodes", "linking in" and "linking out".

5. NONLIN: The Planner

NONLIN is an operational program which performs all the functions specified in this report. It is written in POP-2 (Burstall, Collins and Popplestone, 1971) and uses the HBASE data base system (Barrow, 1975) with additions (see Appendix I). The program runs under the POP-2 interpreter on the Edinburgh DEC10 under time-sharing. With the associated OPSCHEMA construction programs and output routines it occupies 8K words of store.

5.1 Top Level Control


CONSTRUCT INITIAL NETWORK
EACH NODE IN NETWORK EXPANDED ?
YES
SOME TO EXPAND ?
YES
NO
NO
EXPAND NEXT NODE
OK
CORRECT ANY INTERACTIONS
OK
CANNOT
CANNOT
SATISFY UNSUPERVISED CONDITIONS
CANNOT
OK
CHOOSE ALTERNATIVE NET TO EXPAND
OK
CANNOT
SUCCESS PRINT NETWORK
FAILURE NO WAY TO PROCEED
*
*
* these 3 operations may add alternative ways to proceed (choice points of alternative nets).

5.2 Construct the Initial Network

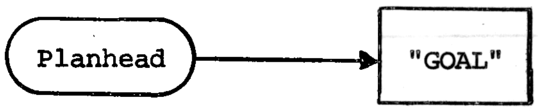
Typing PLAN <expansion form>; to NONLIN constructs an initial network to start planning. <expansion form> has all the flexibility allowed in any expansion of an OPSCHEMA (it may contain ACTION nodes, ORDERINGS and CONDITIONS as well as the more normal GOAL nodes).

e.g. with AUTOCOND set true we could write

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

for 2 parallel goals (note that the goals were not given the optional node numbers as no orderings etc. were to be specified).

The network constructed consists of the special PLANHEAD node followed by a goal node whose pattern is "GOAL" and whose expansion is the <expansion form> given.



5.3 The Basic Cycle

For each node in a network we EXPAND it and correct any INTERACTIONS.* After this process has been performed for each node we check if a variable SOMETOEXPAND is set (i.e. has EXPAND introduced new nodes into the net). If the variable is set we must repeat the expansion cycle. If not then the network is fully expanded and we can exit successfully (subject to any UNSUPERVISED conditions being satisfied). The expansion and interaction correction (linking) modules will be described separately below.

At present the system is run mostly "stand-alone" without user interaction. However, for the use to which it will be put in the Planning: a joint AI/OR approach project, it is envisaged that a user will aid in the choice of nodes to expand and linearizations to make etc. Several primitives which may be used for interrogating the system and making directions to it are given in Appendix III.

5.4 Or-choice Mechanism

While expanding a node alternative operator choices and alternative instantiation choices may be found. These are added onto a list of choice points. Also when correcting for any interaction several linearizations may be suggested, again the alternatives are remembered as choice points.

* This is rather simplistic. In fact only nodes above a certain LEVEL in a defined hierarchy of patterns are expanded.

Whenever an expansion cannot be made or an interaction cannot be corrected an alternative is chosen from the choice points and planning is resumed from that point. It is also possible to generate further solutions after a successful exit from NONLIN by repeatedly typing REPLAN.

At present choices are kept in a very simple fashion by ordering them according to a numeric heuristic on a list. This will be inadequate in a system operating in a domain with many operator choices and in which complex interactions can occur. At present we are investigating the use of a decision graph which will remember the dependancy of certain nodes and links on earlier choices (Daniel, 1976). Selective removal of nodes or links in the graph related to a failure will then be possible. Such a scheme was used by Hayes (1975) for a journey planning system which could deal with execution failures while travelling.

5.5 Expanding a Node

A node is expanded to find more detail of how a task can be performed or a goal achieved. There are certain nodes in the network which should not be expanded because they are present only to correctly define the net. These nodes are the PLANHEAD node and any DUMMY node. The former is inserted to allow facts used in the plan from the initial planning situation to be stored for question-answering. The DUMMY nodes are inserted to allow conditions to be attached unambiguously to some point in the plan (by a user or the system). These two types of nodes are not expandable and are merely returned unaltered by the expansion function.

Thus 3 types of nodes are considered expandable, GOAL nodes, PHANTOM nodes and ACTION nodes.

1) GOAL nodes A goal node is assumed to be present to state that the pattern of the node should be true at the node. There are three ways this could be achieved.

a) If the pattern was already true at that point.

b) If we could introduce links into the network to make the pattern true at that point.

c) If we could make an expansion of the node which would make the pattern be true.

In cases (a) and (b), the GOAL node is returned with a new type "PHANTOM". A GOST entry with <condtype> "PHANTOM" is made to show the contributors which make the pattern true at the node. Links will have been put in the network as a result of (b).

In case (c), it is necessary to find an expansion for the pattern and replace the goal node by the expansion in the network. A list of OPSCHEMAS (or ACTSCHEMAS) which can be used to expand any pattern is kept in a list ACHLIST (built up by the system as the schemas are input). One is chosen and alternative ways to expand the pattern or alternative instantiations in the expansion chosen are kept as choice points. The process used to link an expansion into the network is described later in this section.

2) PHANTOM nodes

If the pattern of a phantom node is still true at the node (there are still contributors which make it true) the node is returned unaltered. However, if the network has been altered such that the pattern of the PHANTOM node does not now hold, the node type is altered to "GOAL" and the expansion process for GOAL nodes is used.

3) ACTION nodes An action node is assumed to be present as a command to do something. No attempt is therefore made to see if its pattern is true or can be made true (by linking) as cases (a) and (b) for goal nodes. However, case (c) is performed exactly as for GOAL nodes. An expansion of the pattern is sought and if there is one it replaces the action node in the network.

Null Expansion

An action node is allowed to have a null expansion. This indicates to the system that the action can be considered primitive and it should not be replaced in the network or expanded further. The expansion function returns action nodes which have a null expansion unaltered.

An expansion with conditions only

In the special case of an action node whose expansion consists only of a set of conditions (i.e. no further action or goal nodes are given), the expansion is treated as being null and the conditions are merely added to the node being expanded and this is returned otherwise unaltered.

Replacing a node by an expansion

To fit an expansion of a node into the network in place of an existing (higher level) node we should perform the following 6 functions.

1) Prenodes links of the existing node are attached to the front of the expansion. All prenodes are modified to point to the front of the expansion.
2) Succnodes links of the existing node are attached to the last node of the expansion. All succnodes are modified to point to the last node of the expansion.
3) Any conditions on the higher level node are made into conditions on the first node of the expansion. The contributors to any condition from the higher level are inherited.
4) Any effects of the higher level node are attached as effects of the last node of the expansion. Higher level effects will have been achieved when the whole expansion is completed.

5) Any nodes for which the higher level node contributed towards some condition (the purposes of the higher level node), take the last node of the expansion as the contributor (because of (4)).

6) TOME entries refer to the node number at which a pattern is given a value. TOME entries for the higher level node should refer to the last node of the expansion (because of (4)).

In order to facilitate a simple update of the network when an expansion is made, all nodes in an expansion except the last node are inserted onto the end of the ALLNODES flexible strip of nodes in the network (incrementing NUMNODES), then the last node of the expansion replaces the higher level node entry in ALLNODES. The last node thus inherits the node number of the higher level node.

Since PRENODES and SUCCNODES links, TOME and GOST entries are all in terms of node numbers, operations (2), (4), (5), and (6) above are automatic, since the numbers refer to the correct node. Only operations (1) and (3) need be performed. After this is done, the MAINEFFECTS (see section 2.3.2) of any pattern are made and true is given as the value of the pattern of any GOAL node.

Is this simple approach sufficient?

We have had a great deal of discussion to try to decide whether the simple process of linking an expansion to the net in place of a node is sufficient. Schemes have been tried of apportioning the effects of a higher level node between the nodes of an expansion; of linking prenodes links to nodes of the expansion which repeat a condition of the higher level node; and linking succnodes to nodes of the expansion which repeat effects of the higher level node. The scheme presented for the replacement of a node by an expansion (as Sacerdoti, 1975a) assumes two things.

a) a condition on a node means that this condition only need hold before the first node in any expansion.

b) the effects of a node are those achieved by executing some expansion of the node (i.e. can only be assumed to hold after the last node of an expansion is performed).

This seems to be a natural assumption to make when describing a task hierarchically and should simplify any problem a user may otherwise have had.

Relocatable Expansions

In order to ease the insertion of an expansion into the network, each expansion schema in an operator schema is kept as a minature network of nodes in the same form as the main network being generated by the planner and as a list of conditions on these nodes. Prenodes, succnodes and conditions all are given with node numbers relative to the strip representing this minature network. The first and last nodes in the strip are the first and last nodes respectively in the expansion. To insert an expansion in the main network we use the current size of the network (NUMNODES) to give a relocator to be added to all node numbers referred to in the prenodes, succnodes and condition entries of the expansion. Since the last node of an expansion actually replaces the node being expanded, this is treated separately.

Since expansion schemas are stored as network fragments it should be possible to use the planner to build up small networks from more primitive jobs in order that the network can be saved as a schema and used as a component of larger tasks (the use of small, well worked out components is common in network planning). This is one area in which further effort must be made to develop the user interaction facilities.

5.6 Linking process for the network

There are two occasions on which it is necessary to suggest links in the network.

a) To linearize the net to remove a list of interactions.

b) To make a statement have a particular value at some node (as a result of a request to "LINK" to the QA module - see section 4).

The same process is used in both of the above tasks.

We start with a SUGGESTed set of networks. Initially this would only be the original (input) network, but later it would be set on each iteration to be a NEWSUGGESTed set of networks defined below. We also have a list of interaction records (INTERACT) each of which gives a node N at which a statement P has a value V and a list of nodes $N^1, \ldots, N^i, \ldots N^\ell$ at which statement P has a value other than V (say $\bar{V}$) and which contribute to a set of purposes $PN^i_1 \ldots PN^i_n$ such that P can have value $\bar{V}$ for part of the time for which P could have value V at node N. Some of the N^i may give P a value $\bar{V}$ and be in parallel with node N and interact for this reason alone (i.e. they may have no purpose for P).

We must ensure that we suggest appropriate linearizations to make P have the appropriate value for any point in the network so that the purposes are achieved. We have a set of "ranges" for which P must have value $\bar{V}$ (the range may be a single node at which P has value $\bar{V}$ if it has no purpose). We also find a set of "ranges" for which P must have value V (from the single node N to any nodes ($PN_1, \ldots PN_m$) to which N "contributes" pattern P to achieve a condition). All that is needed is to ensure that none of the "$\bar{V}$ ranges" overlap with any "V range". The algorithm below performs this task. It also takes into account two facts.

a) only one contributor to any purpose is necessary. This allows the number of ranges to be reduced.

b) if the type of condition is a PHANTOM we can remove all contributors if necessary as this will merely make the node with the PHANTOM conditions into a goal node.

The process produces all networks based on the initial set SUGGEST for which links can be made to remove all interactions in the list INTERACT.

Given a list SUGGEST of networks and a list INTERACT of interact record.

Iterations

1) For each member of the list INTERACT choose an interact record which gives one node (say N) at which a statement is given a value V and which has purposes $PN_1, \ldots PN_m$ and a set of interacting nodes $N^1, \ldots N^i, \ldots N^\ell$ each of which has purposes $PN^i_1 \ldots PN^i_n$

2) for each N choose a member of the set of interacting nodes $N^1, \ldots N^i, \ldots N^\ell$ (say N^i - OPP in the program).
N and N^i may establish P with value V and $\bar{V}$ respectively for some other nodes (the purposes if any are given in the GOST - see section 3) $PN_1, \ldots PN_j, \ldots PN_m$ and $PN^i_1, \ldots PN^i_k, \ldots PN^i_n$.
If there are no PN_j (i.e. m = 0) or no PN^i_k (i.e. n = 0), the single nodes N or N^i respectively are used to define the "ranges" below.

3) for each PN_j we define a VRANGE (a period for which P must have value V) from N to PN_j.

4) for each PN^i_k we define a $\overline{\text{VRANGE}}$ (a period for which P must have value $\bar{V}$) from N^i to PN^i_k.

$$\begin{tikzcd} N \arrow[r, "VRANGE"] & PN & \text{or} & N & \text{if no } PN_j \\ N^i \arrow[r, "\overline{VRANGE}"'] & PN^i_k & \text{or} & N^i & \text{if no } PN^i_k \end{tikzcd}$$

5) for each network currently in SUGGEST we must try to remove any overlap of these two interacting ranges or remove the N or N^i as "contributors" to PN_j or PN^i_k respectively.
** We must suggest putting the VRANGE both (a) before and (b) after the $\overline{\text{VRANGE}}$. This is in fact an elegant generalization of the interaction correction procedure first suggested in Tate (1974 and 1975b). This merely involves putting a link from the end of one block to the beginning of the other in both cases. This can only be done if a link does not exist in the opposite direction.

Alternatives — for each alternative (a) and (b) above we suggest the following.

(1) (a) & (b) With neither N nor N^i removed as contributors to PN_j and PN^i.
(2) (a) & (b) With N^i removed as a contributor to PN^i_k but N not so removed
(3) (a) & (b) With N removed as a contributor to PN_j but N^i not so removed
(4) An unaltered net (no extra links) is suggested with N removed as a contributor to PN_j and N^i removed as a contributor to PN^i_k

Any legal net produced is added to the list NEWSUGGEST which after dealing with all nets currently in SUGGEST is used to reset SUGGEST for the next iteration.

**When choosing between the alternatives we should prefer orderings which do not put a statement with a particular value after some node which has an Unsupervised condition for that statement and value.

Any legal suggestion is added to the list NEWSUGGEST. Note that if ever both VRANGE and $\overline{\text{VRANGE}}$ becomes a single node no interaction is present and the net can be left as it stands (alternative 4). Only one unaltered net should be produced for any single VRANGE/$\overline{\text{VRANGE}}$ ordering attempt. Some of the alternatives (1) (a) → (3) (b) may produce such an unaltered net so redundancy must be checked. It is anticipated that many of the orderings attempted will fail because of incompatible linking in the networks. This will be especially so when there are several interact records or many purposes for nodes.

<u>Use of the Linking Process</u>

The function as described can correct for any number of interactions in any suggested set of networks. It is thus possible to write the planner so that it performs interaction correction at any time the designer wishes. Different schemes of correcting when one interaction is found, when an expansion of a node is made, or after all nodes in the current net have been expanded once can thus be experimented with.

In NONLIN the linking process is used for the two purposes mentioned at the beginning of this section.

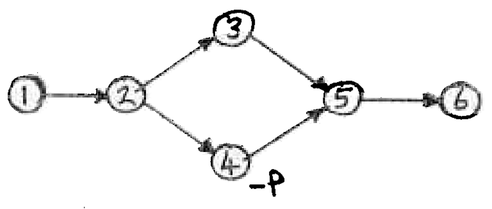
(a) <u>to linearize the net to remove interactions</u>

As effects are added to nodes in the net they are also recorded in the TOME (whose entries are of the form << <pattern> <nodenum> >> with a value). When any TOME entry is made, a check is made to see if:-

i) any parallel node has an opposite value for the pattern (in TOME)

ii) any node has a condition for the pattern with an opposite value, and has a contributor such that part of the range for the pattern to have an opposite value is in parallel with the node just given the effect. (found from GOST)

e.g. P with value false ("-") being entered at node (4)



+ P at (3) interacts

+ P at (1) and a purpose at (2) does not

+ P at (5) and a purpose at (6) does not

+ P at any other node which has a purpose at some later node interacts

Interact records (as described earlier) are found for each potential interaction and put in the list INTERACT. If several effects are added which cause interactions all the interact records are added to the list INTERACT.

The linking process described is then entered with a single suggested net (the current network) and the list INTERACT.

===n INTERACTIONS $\left\{\begin{matrix}\text{m LINEARIZATIONS}\\ \text{NO WAY TO REMOVE}\end{matrix}\right.$ is printed to signify that n interaction records are being handled and m linearization were suggested to correct for them or no ways were found to correct for them.

(b) <u>to make a statement have a particular value at some node</u>

When the QA system was asked if P had a value V at node N and failed to prove that this was true, if the system was given a parameter "LINK" it may be able to return four lists (VL, PARVL, VNOTL and PARVNOTL) of nodes which can be used to suggest ways of <u>making</u> P have value V at node N. See the section on the QA system for the justification for the "link-to-make-true" procedure. That section also gives the definition ofthe critical PV-nodes (VL and PARVL) and the critical PNOTV-nodes (VNOTL and PARVNOTL).

The procedure is to "link-in" one critical PV-node before node N if such a node does not already exist (if VL is not null then there is already one "linked-in" critical PV-node). This linking may suggest one or more networks and we must now "link-out" all critical PNOTV-nodes from between a linked-in critical PV-node and N. So for each network suggested we construct interact records in the list INTERACT for each critical PNOTV-node and call the linking process described in this section.

<u>N.B.</u> it is only necessary to "link-in" one critical PV-node before N since linking more would merely add extra ordering constraints to a network without contributing to any purpose. This would then make "linking-out" harder too.

5.7 Conditions and their use in the planner

An expansion specifies various nodes, their time-order linking and the conditions which must hold before each node. Basically a condition on a node is inserted in the GOST when that node is added to the network. The GOST entry for the condition stores the "contributors", or nodes, any one of which would be sufficient to make the condition hold. Conditions have different types as explained in the section on domain description (section 2.2). The differences between the handling of the different types is given below.

There is a rule for deciding which type a condition should have if it is given two different types at different levels of description.

SUPERVISED	Fill in immediately. Must have one contributor from within the expansion just made.
↑	
HOLDS	Fill in immediately. Can have several contributors.
↑	
UNSUPERVISED	Fill in by the time the plan is finished.

Higher types replace lower ones. A PHANTOM condition type can only be added to an unexpanded node, and it has a pattern which is the pattern of the node itself. Since it is impossible to have a condition on a node (i.e. a pattern must hold before the node) which is the pattern of the node itself, PHANTOM need not be included in the priority ordering. PHANTOM is essentially for system use only. Compute conditions are evaluated at the time an expansion is chosen and do not appear in a GOST.

Hold and Compute

Hold and compute conditions are gathered together at the beginning of the list of conditions of the OPSCHEMA in the order they were written. This first part of the list of conditions is used as a further check on the applicability of an OPSCHEMA after the pattern directed invocation through the OPPATTERN. Each hold condition must have some instance at the point in the plan where the expansion is to be made. So a hold condition must hold at the point in the plan immediately before an expansion and continue to hold up to the node where it is a condition. If a hold condition is not true when an OPSCHEMA is chosen the OPSCHEMA is considered inapplicable. Compute conditions are also evaluated when an OPSCHEMA is chosen. If they do not produce a successful result, the OPSCHEMA is considered inapplicable.

Unsupervised

An unsupervised condition need not be made to hold until the plan is completed in all other details. It specifies a final ordering constraint on the plan. Alternative linearizations may be chosen with regard to the later ability to set Unsupervised conditions.

Supervised

A supervised condition is always put as a goal node explicitly in the net and the goal node is made the only contributor to the satisfaction of a supervised condition on a following node. The goal node is inserted explicitly to allow it to be expanded out if the goal pattern is not true (Supervised conditions are the only ones which can cause expansions). If it is true a condition of type PHANTOM can be considered to "achieve" the goal node.

Phantom

If a goal node has a pattern which is true at the point required the goal node is marked as a PHANTOM (its node type is altered), and an entry is put in the GOST which is like a condition on the goal node of condtype PHANTOM. If necessary to remove an interaction, the contributors which could establish the condition can be removed. The node then reverts to a goal node and may be expanded normally.

5.8 Useful properties of the data structures representing the network

Besides the actual plan representation available after a problem is solved, the GOST context provides much valuable information.,

1) Plan Optimization and Execution

   The GOST provides details of the contributors to any condition. In an optimized plan it is only necessary to have one contributor to each condition at a node. Any actions in the plan which do not contribute to any condition can then be removed. It may also be possible to make an action redundant by "linking in" a parallel node to achieve some condition, thus making the previous contributor redundant.

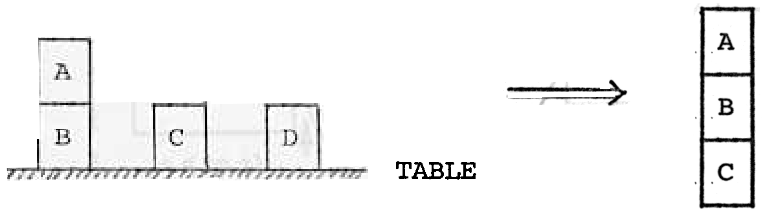
   Sophisticated execution monitoring and run-time plan alteration strategies such as those outlined in Sacerdoti (1975b) can be supported by the representation of the network and are greatly aided by the explicit provision of the GOST structure. Such schemes are currently being investigated.

2) Plan Generalization and re-use

   The GOST also specifies conditions on nodes which were satisfied from the ALWAYCTXT (contributor [0]) and from INITCTXT (contributor [-1] or [1] if used from the INITCTXT and modified in the plan). We can thus get the set of conditions which must be true of any world model for the plan to be applicable and achieve its purpose. Symbolic alteration of the plan using the goal structure to link variables between actions can lead to generalized or re-instantiated plans. A complete plan can be saved as an OPSCHEMA since the expansion of an OPSCHEMA is kept as a network of nodes and as a list of conditions. It may thus be possible to generalize and re-use plans as was done by Sussman (1973).

6. An Example Problem - Block Stacking

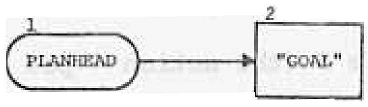
Taking the block stacking domain given as three operator schemas in section 2.3.1 (on the task formalism), I will describe the action of NONLIN on the problem of stacking Block A on B and stacking B on C. In the following <<put x on top of y>> will be abbreviated (put x on y), <<cleartop x>> will be abbreviated (cl x), <<on x y>> will be abbreviated (on x y).



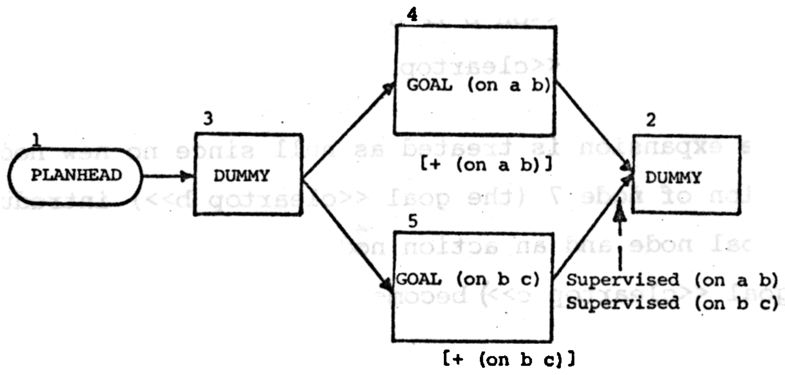
PLAN GOAL <<ON A B>>

GOAL <<ON B C>>;

creates the initial net below (nodes are numbered):-

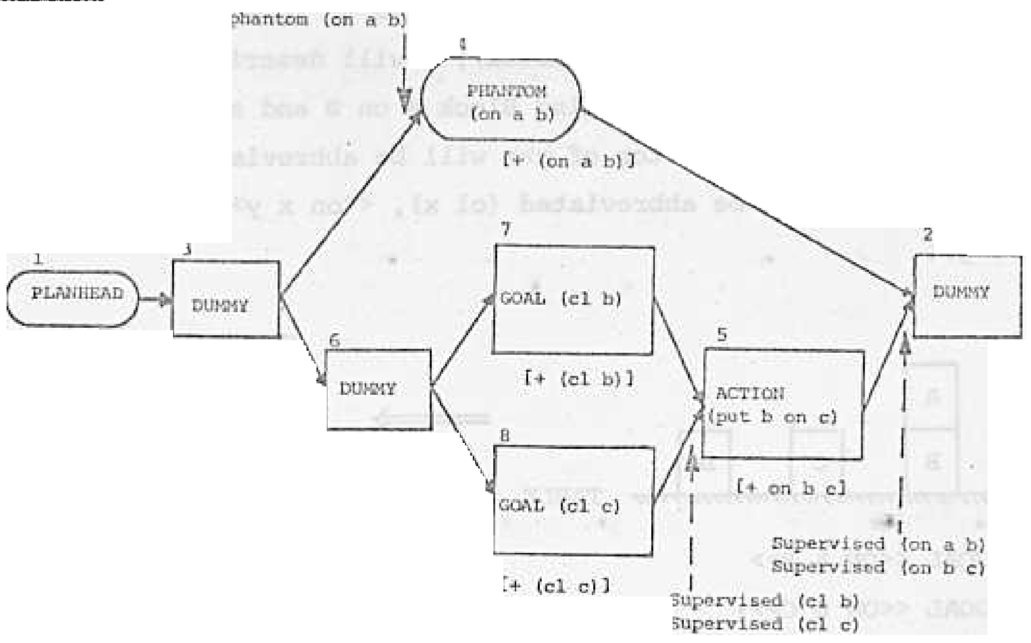


Expansion 1 (Using the expansion provided by the PLAN statement)



No interactions are introduced so we expand each node again.

Expansion 2



No interactions are introduced so we expand each node again.

Expansion 3

The expansion of node 5 (the action <<put b on top of c>> node) introduces further conditions at the node:

holds <<cleartop b>> is superseded by the existing condition supervised <<cleartop b>>

holds <<cleartop c>> is superseded by the existing condition supervised <<cleartop c>>

holds <<on b table>>

and further effects - <<cleartop c>>

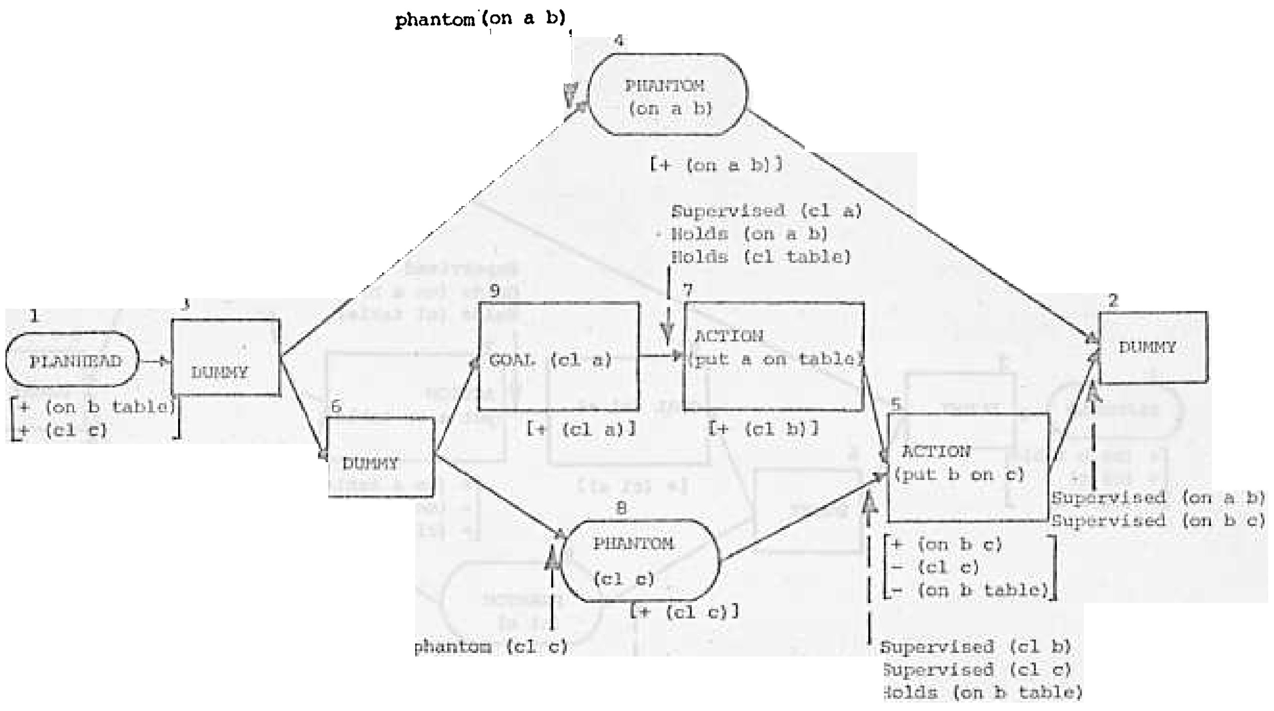
- <<on b table>>

+ <<cleartop table>> is not asserted as it is ALWAYS true

however the expansion is treated as null since no new nodes are added.

The expansion of node 7 (the goal <<cleartop b>>) introduces 2 new nodes, a goal node and an action node.

Node 8 (goal <<cleartop c>>) becomes a PHANTOM node.



No interactions are introduced so we expand each node again.

<u>Expansion 4</u>

Expanding node 7 (the action <<put a on top of table>> node) introduces new conditions

holds <<cleartop a>> superseded by the supervised condition <<cleartop a>>

holds <<cleartop table>> is already present

holds <<on a b>> is already present

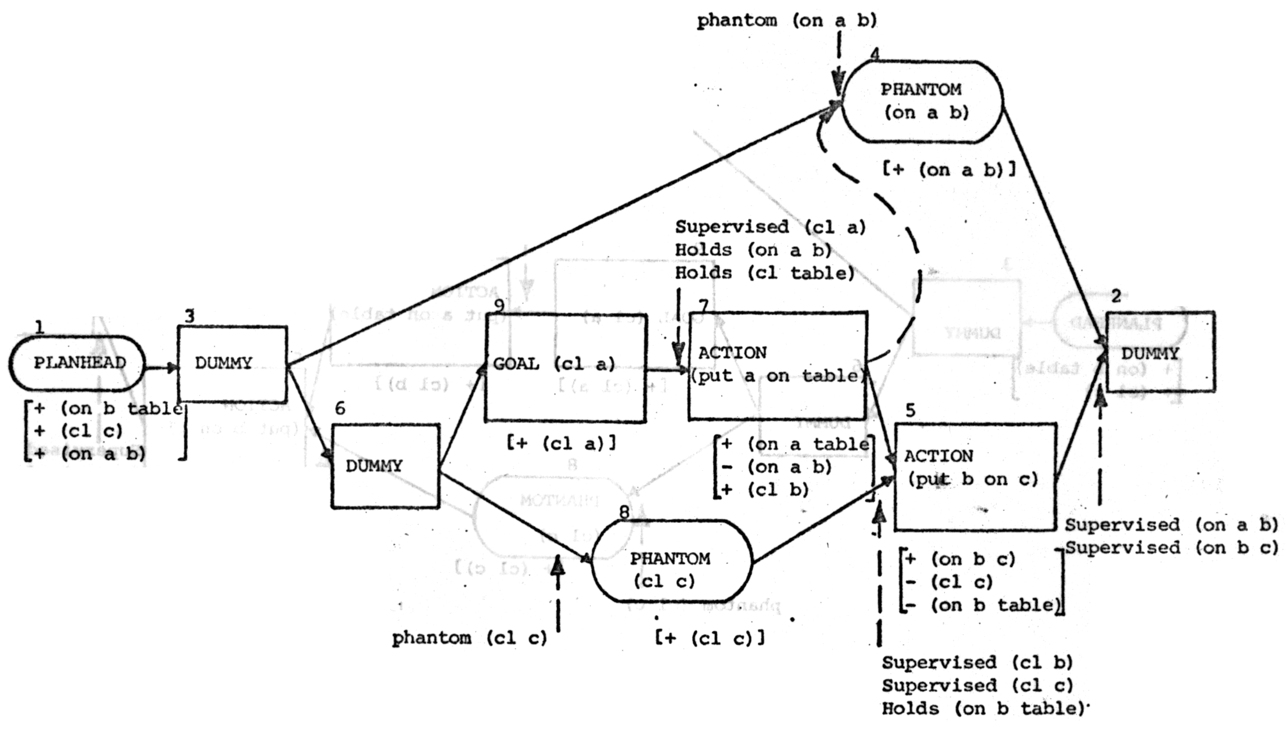
new effects are also introduced - <<cleartop table>> cannot be made as ALWAYS <<cleartop table>>

- <<on a b>>

- <<on a table>>

No new nodes are introduced so the expansion is treated as null.

The - <<on a b>> effect at node 7 introduces an interaction with + <<on a b>> at node 4.

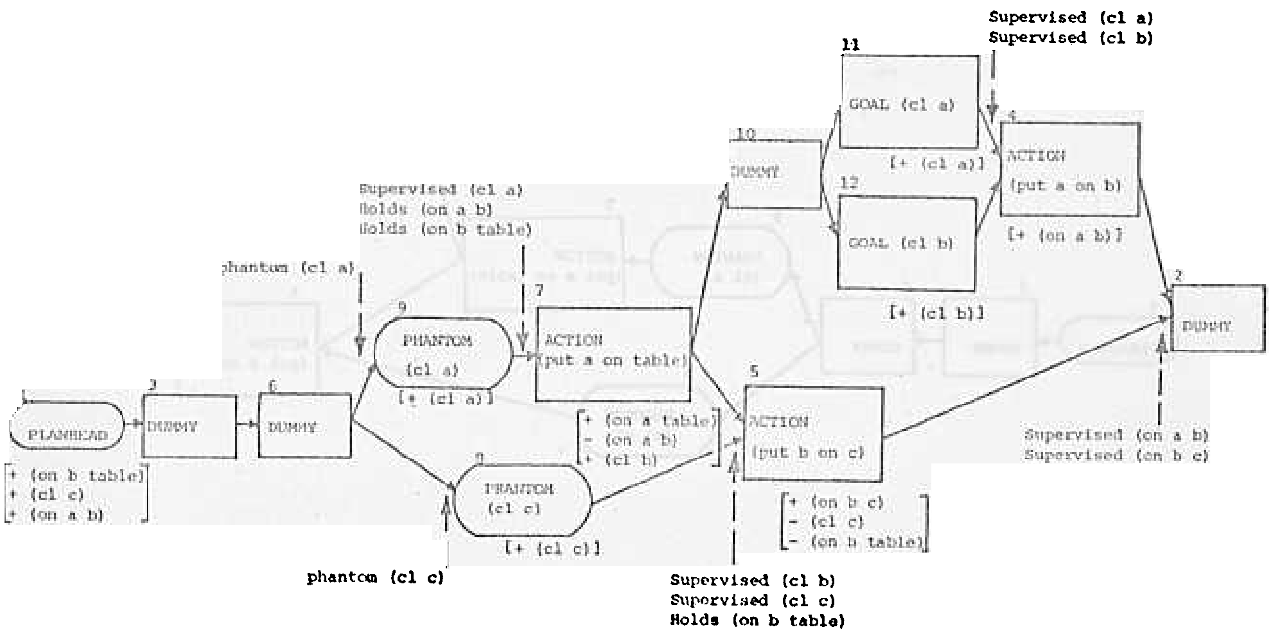


We have a node 4 with an effect + <<on a b>> needed at node 2 and a node 7 with an effect - <<on a b>> which has no purpose. Also at present + <<on a b>> at node 1 is used as a contributor to the phantom node 4. To correct for this interaction it is necessary to ensure that there is no point in the plan at which - <<on a b>> is in parallel with points at which + <<on a b>> must hold.

The interaction corrector can suggest only one possibility. Remove + <<on a b>> at node 1 as a contributor to the phantom node 4 and link 7 --> 4. (7 cannot be put after 2 as a link exists in the opposite direction).

Expansion 5

Node 4, now a goal node, expands out to 2 goal nodes and an action node.



No interactions are introduced so we expand each node again.

Expansion 6

Only nodes 4, 11 and 12 are now expandable. 11 and 12 are merely converted to phantom nodes, node 4 (the action <<put a on b>> node) adds further conditions:

holds <<cleartop b>> superseded by the supervised condition

" <<cleartop c>> " " " " "

<<on a table>>

extra effects are added:

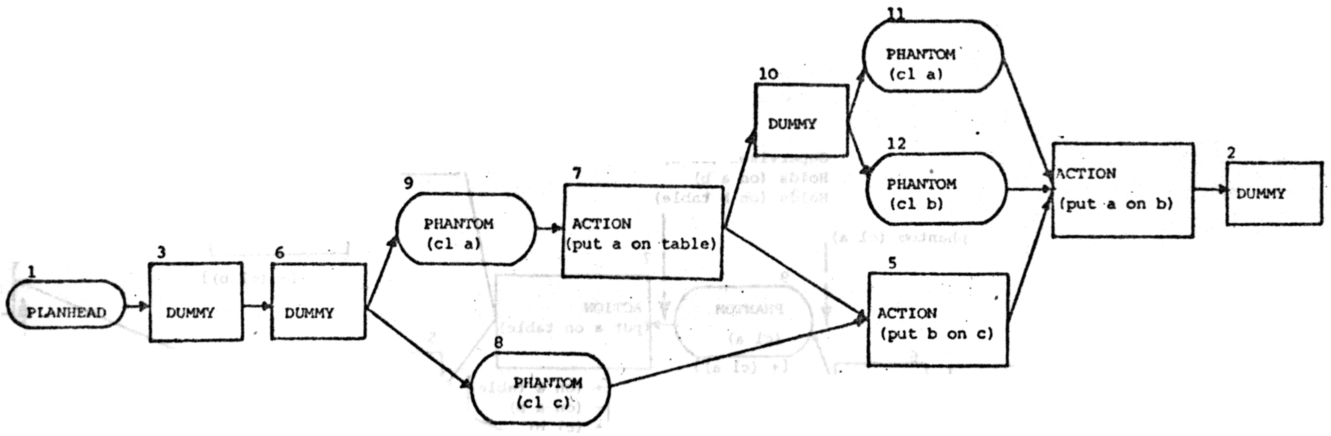
<<cleartop b>>

<<on a table>>

+ <<cleartop table>> is ignored

Now the - <<cleartop b>> effect at node 4 which has no purpose interacts with an effect + <<cleartop b>> at node 7 which must be true at node 5.

Part of range 7 → 5 is in parallel with node 4. The interaction corrector can suggest only one linearization of putting 5 --> 4. The redundant link 5 → 2 is removed. The final plan is shown below without effects or conditions.



The trace of NONLIN on this problem is:-

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

LEVEL O
LEVEL O
LEVEL O
     +++CHOICE ADDED HOLDS
LEVEL O
       = 1 INTERACTION 1 LINEARIZATIONS
LEVEL O
LEVEL O
         1 INTERACTION 1 LINEARIZATIONS

NONLIN 2 TERMINATED.   CPU TIME = 7.342 SECS

1  PLANHEAD   NIL          [ 3]
2  DUMMY      [ 4]         NIL
3  DUMMY      [ 1]         [ 6]
4  ACTION     [ 5 12 11]   [ 2]       <<PUT A ON TOP OF B>>
5  ACTION     [8 7]        [ 4]       <<PUT B ON TOP OF C>>
6  DUMMY      [ 3]         [8 9]
7  ACTION     [ 9]         [lo 5]     <<PUT A ON TOP OF TABLE>>
8  PHANTOM    [ 6]         [ 5]       <<CLEARTOP C>>
9  PHANTOM    [ 6]         [ 7]       <<CLEARTOP A>>
lo DUMMY      [ 7]         [12 11]
11 PHANTOM    [ lo]        [ 4]       <<CLEARTOP A>>
12 PHANTOM    [ lo]        [ 4]       <<CLEARTOP B>>
```

After the plan is completed the GOST (goal structure) is:-
(entries are << <condtype> <pattern> <value> <node condition required at> >>
and the value of the entry is a list of contributors to that condition).

```
CONTEXT 35
<< PHANTOM     <<CLEARTOP B>>       1 12>>    [ 7]
<< PHANTOM     <<CLEARTOP A>>       1 11>>    [ 9]
<< HOLDS       <<ON A TABLE>>       1  4>>    [ 7]
<< PHANTOM     <<CLEARTOP A>>       1  9>>    [ 1]
<< SUPERVIS    <<CLEARTOP A>>       1  4>>    [11]
<< SUPERVIS    <<CLEARTOP B>>       1  4>>    [12]
<< PHANTOM     <<CLEARTOP C>>       1  8>>    [ 1]
<< SUPERVIS    <<CLEARTOP A>>       1  7>>    [ 9]
<< HOLDS       <<CLEARTOP TABLE>>   1  7>>    [ 0]
<< HOLDS       <<ON A B>>           1  7>>    [ 1]
<< HOLDS       <<ON B TABLE>>       1  5>>    [ 1]
<< SUPERVIS    <<CLEARTOP B>>       1  5>>    [ 7]
<< SUPERVIS    <<CLEARTOP C>>       1  5>>    [ 8]
<< SUPERVIS    <<ON A B>>           1  2>>    [ 4]
<< SUPERVIS    <<ON B C>>           1  2>>    [ 5]
```

After the plan is completed the TOME (table of multiple effects) is:-
(entries are << <pattern> <node at which pattern is given a value> >>
and the value of the entry is the value given to the pattern at the node)

```
CONTEXT 34
<< <<CLEARTOP B>>     4>>    0
<< <<ON A TABLE>>     4>>    0
<<<<<CLEARTOP B>>    12>>    1
<< <<CLEARTOP A>>    11>>    1
<< <<ON A TABLE>>     7>>    1
<< <<ON A B>>         7>>    0
<< <<CLEARTOP A>>     9>>    1
<< <<CLEARTOP A>>     1>>    1
<< <<CLEARTOP C>>     5>>    0
<< <<ON B TABLE>>     5>>    0
<< <<ON B TABLE>>     1>>    1
<< <<CLEARTOP C>>     8>>    1
<< <<CLEARTOP C>>     1>>    1
<< <<CLEARTOP B>>     7>>    1
<< <<ON B C>>         5>>    1
<< <<ON A B>>         4>>    1
<< <<ON A B>>         1>>    1
```

7. House Building

We have chosen the construction of project networks for house building tasks as an example domain for the evaluation of our approach. An example of a project network for a simple house building task is given in section 1. We have only just begun experiments using the Task Formalism (TF) and the planner NONLIN to generate project networks from a hierarchic description of the sub-tasks in such domains. However, there are several ways in which such domains are simpler than general robot problem solving (typically represented by our example in the block stacking domain - see section 6).

a) The expansion of any task will typically not refer to many variables. That is, tasks and sub-tasks are mainly fully instantiated.

b) Strong orderings can be given on a few related tasks, e.g. the sub-tasks for a carpet layer to perform are strongly ordered.

c) There are very few interactions in the domain. Housebuilding is mainly constructive and few things achieved are later undone. The sort of interactions which will be present will relate to construction techniques such as the use of scaffolding and to the use of negative conditions (e.g. NOT <<FLOORBOARDS LAID>>). In general it may only be necessary to specify supervised conditions for those statements on which interactions may occur (this is not done in the example to follow).

d) There will be a large number of simple primitive jobs. So some simple method of describing them is essential (e.g. PRIMITIVE; PRIMITIVE WITH EFFECT; and MAINEFFECTS can be used).

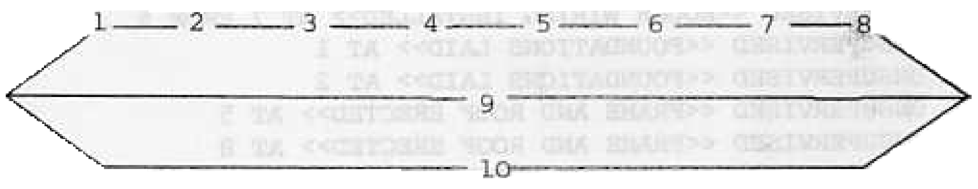
e) There may be many sub-tasks which have only one method of being performed This will apply especially to lower level tasks. Alternative selection during planning may therefore be relatively infrequent.

f) It is becoming clear that there is much redundancy in the ACTSCHEMA specifications. ORDERINGS are mostly determined by the condition ranges, and condition types can be deduced. These observations will be borne in mind when modifications to TF are made.

Currently we are trying to gain experience in writing simple house building tasks to formulate a set of rules which can be used to guide someone describing a domain in TF. We hope these rules may form the basis of an interactive system which can be used for project network construction.

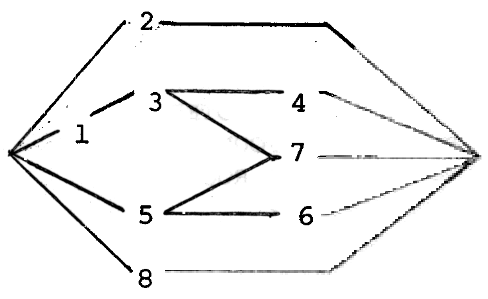
The house building project given as a table of jobs and a project network in section 1 was used as a first example. The top level description is given by a BUILDER who controls the job. He has eight sub-tasks directly under his control as a builder and specifies a strong (completely linear) order on them since all but the last satisfies some conditions for another. The builder sub-contracts the installation of services and the decoration. The TF description for this simple domain is given below. There are no interactions in the domain since there are no negative effects and there are no alternative methods of performing any task. Planning is therefore very simple on this task and reduces to a scheduling problem (the project network for this 22 job task, as shown in section 1, was generated in 20 seconds).

```
ACTSCHEMA BUILD
  PATTERN <<BUILD HOUSE>>
  EXPANSION 1 ACTION <<EXCAVATE, POUR FOOTERS>>
            2 ACTION <<POUR CONCRETE FOUNDATIONS>>
            3 ACTION <<ERECT FRAME AND ROOF>>
            4 ACTION <<LAY BRICKWORK>>
            5 ACTION <<FINISH ROOFING AND FLASHING>>
            6 ACTION <<FASTEN GUTTERS AND DOWNSPOUTS>>
            7 ACTION <<FINISH GRADING>>
            8 ACTION <<POUR WALKS, LANDSCAPE>>
            9 ACTION <<INSTALL SERVICES>>
           10 ACTION <<DECORATE>>
  ORDERINGS SEQUENCE 1 TO 8
```



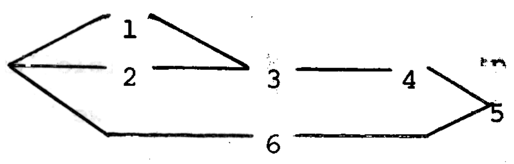
```
  CONDITIONS SUPERVISED <<FOOTERS POURED>>  AT 2 FROM 1
             SUPERVISED <<FOUNDATIONS LAID>>  AT 3 FROM 2
             SUPERVISED <<FRAME AND ROOF ERECTED>> AT 4 FROM 3
             SUPERVISED <<BRICKWORK DONE>> AT 5 FROM 4
             SUPERVISED <<ROOFING FINISHED>> AT 6 FROM 5
             SUPERVISED <<GUTTERS ETC FASTENED>> AT 7 FROM 6
             UNSUPERVISED <<STORM DRAINS LAID>> AT 7
             SUPERVISED <<GRADING DONE>> AT 8 FROM 7
END;
```

```
ACTSCHEMA SERVICE
  PATTERN <<INSTALL SERVICES>>
  EXPANSION 1 ACTION <<INSTALL DRAINS>>
            2 ACTION <<LAY STORM DRAINS>>
            3 ACTION <<INSTALL ROUGH PLUMBING>>
            4 ACTION <<INSTALL FINISHED PLUMBING>>
            5 ACTION <<INSTALL ROUGH WIRING>>
            6 ACTION <<FINISH ELECTRICAL WORK>>
            7 ACTION <<INSTALL KITCHEN EQUIPMENT>>
            8 ACTION <<INSTALL AIR CONDITIONING>>
  ORDERINGS 1 ---> 3  3 ---> 4  5 ---> 6  3 ---> 7  5 ---> 7
```



```
  CONDITIONS SUPERVISED <<DRAINS INSTALLED>> AT 3 FROM 1
             SUPERVISED <<ROUGH PLUMBING INSTALLED>> AT 4 FROM 3
             SUPERVISED <<ROUGH WIRING INSTALLED>>  AT 6 FROM 5
             SUPERVISED <<ROUGH PLUMBING INSTALLED>> AT 7 FROM 3
             SUPERVISED <<ROUGH WIRING INSTALLED>> AT 7 FROM 5
             UNSUPERVISED <<FOUNDATIONS LAID>> AT 1
             UNSUPERVISED <<FOUNDATIONS LAID>> AT 2
             UNSUPERVISED <<FRAME AND ROOF ERECTED>> AT 5
             UNSUPERVISED <<FRAME AND ROOF ERECTED>> AT 8
             UNSUPERVISED <<BASEMENT FLOOR LAID>> AT 8
             UNSUPERVISED <<FLOORING FINISHED>> AT 4
             UNSUPERVISED <<FLOORING FINISHED>> AT 7
             UNSUPERVISED <<PAINTED>> AT 6
END;
```

```
ACTSCHEMA DECOR
  PATTERN <<DECORATE>>
  EXPANSION 1 ACTION <<FASTEN PLASTER AND PLASTER BOARD>>
            2 ACTION <<POUR BASEMENT FLOOR>>
            3 ACTION <<LAY FINISHED FLOORING>>
            4 ACTION <<FINISH CARPENTRY>>
            5 ACTION <<SAND AND VARNISH FLOORS>>
            6 ACTION <<PAINT>>
  ORDERINGS SEQUENCE 2 TO 5 1 ---> 3  6 ---> 5
```



```
  CONDITIONS UNSUPERVISED <<ROUGH PLUMBING INSTALLED>> AT 1
             UNSUPERVISED <<ROUGH WIRING INSTALLED>> AT 1
             UNSUPERVISED <<AIR CONDITIONING INSTALLED>> AT 1
             UNSUPERVISED <<DRAINS INSTALLED>> AT 2
             UNSUPERVISED <<PLUMBING FINISHED>> AT 6
             UNSUPERVISED <<KITCHEN EQUIPMENT INSTALLED>> AT 6
             SUPERVISED <<PLASTERING FINISHED>> AT 3 FROM 1
             SUPERVISED <<BASEMENT FLOOR LAID>> AT 3 FROM 2
             SUPERVISED <<FLOORING FINISHED>> AT 4 FROM 3
             SUPERVISED <<CARPENTRY FINISHED>> AT 5 FROM 4
             SUPERVISED <<PAINTED>> AT 5 FROM 6
END;


PRIMITIVE
   <<EXCAVATE, POUR FOOTERS>>              WITH EFFECT + <<FOOTERS POURED>>
   <<POUR CONCRETE FOUNDATIONS>>           WITH EFFECT + <<FOUNDATIONS LAID>>
   <<ERECT FRAME AND ROOF>>                WITH EFFECT + <<FRAME AND ROOF ERECTED>>
   <<LAY BRICKWORK>>                       WITH EFFECT + <<BRICKWORK DONE>>
   <<FINISH ROOFING AND FLASHING>>         WITH EFFECT + <<ROOFING FINISHED>>
   <<FASTEN GUTTERS AND DOWNSPOUTS>>       WITH EFFECT + <<GUTTERS ETC FASTENED>>
   <<FINISH GRADING>>                      WITH EFFECT + <<GRADING DONE>>
   <<POUR WALKS, LANDSCAPE>>               WITH EFFECT + <<LANDSCAPING DONE>>
   <<INSTALL DRAINS>>                      WITH EFFECT + <<DRAINS INSTALLED>>
   <<LAY STORM DRAINS>>                    WITH EFFECT + <<STORM DRAINS LAID>>
   <<INSTALL ROUGH PLUMBING>>              WITH EFFECT + <<ROUGH PLUMBING INSTALLED>>
   <<INSTALL FINISHED PLUMBING>>           WITH EFFECT + <<PLUMBING FINISHED>>
   <<INSTALL ROUGH WIRING>>                WITH EFFECT + <<ROUGH WIRING INSTALLED>>
   <<FINISH ELECTRICAL WORK>>              WITH EFFECT + <<ELECTRICAL WORK FINISHED>>
   <<INSTALL KITCHEN EQUIPMENT>>           WITH EFFECT + <<KITCHEN EQUIPMENT INSTALLED>>
   <<INSTALL AIR CONDITIONING>>            WITH EFFECT + <<AIR CONDITIONING INSTALLED>>
   <<FASTEN PLASTER AND PLASTER BOARD>>    WITH EFFECT + <<PLASTERING FINISHED>>
   <<POUR BASEMENT FLOOR>>                 WITH EFFECT + <<BASEMENT FLOOR LAID>>
   <<LAY FINISHED FLOORING>>               WITH EFFECT + <<FLOORING FINISHED>>
   <<FINISH CARPENTRY>>                    WITH EFFECT + <<CARPENTRY FINISHED>>
   <<SAND AND VARNISH FLOORS>>             WITH EFFECT + <<FLOORS VARNISHED>>
   <<PAINT>>                               WITH EFFECT + <<PAINTED>>
```

8. Summary and relation to NOAH

NOAH (Sacerdoti, 1975a) was the first attempt to produce a non-linear planner which took a hierarchic specification of a domain. It was designed as part of the Computer Based Consultant project at the Stanford Research Institute (Hart, 1975). The present NONLIN system is a development of that work. However, we have sought to improve over NOAH in several ways.

Task Formalism (TF)

TF is intended to give a powerful and flexible language to a user to describe a domain but is also the basis of what we hope will become a simple and clear formalism to enable a group of people to co-operatively describe a task to the system with the planner's aid. We feel TF has advantages over the relatively unstructured (though still powerful) SOUP code which NOAH requires as input. We hope to gain clarity and efficiency through the use of different condition types.

All alternatives kept

The planner NONLIN generates and keeps for future use any alternatives at choice points. Alternative operator choices, instantiation choices and linearizations are kept. These may be used either on the failure of some approach the planner tries or to generate more than one solution. Alternatives were not kept in NOAH.

Goal Structure

We have provided a powerful aid to planning and subsequent plan execution by giving a summary of the "goal structure" of the plan in a simple form. A GOST is kept which remembers the conditions on any node and the set of possible contributors to those conditions. The explicit provision of the goal structure independently of the chronological links in the plan allows simple schemes to be used to detect and correct for any interactions introduced and, as mentioned elsewhere, to allow plan monitoring during execution. Fine detail of the purposes of individual effects of nodes can be discriminated in a GOST. This is preferable to saying that the whole node (hence all its effects) achieves some purpose of a succeeding node. Unimportant (from the goal structure point of view) interactions can be ignored with the scheme presented here.

Interactions and Linking

The limitations in NOAH on the linearizations which could be suggested when an interaction was introduced (see Tate, 1975b) have been removed. When goal structure is available it is very easy to see which linearizations may be suggested. Two other particular difficulties of NOAH mentioned in

Tate (1975b), that of the inability to deal with some "double interactions" and beneficial side effects on parallel branches, are also removed in NONLIN (the latter because of the ability to "link-to-make-true" in the QA system). A general interaction correction procedure has been described to perform the above tasks.

QA in a partially ordered network

A scheme is presented for answering questions in a partially ordered network where nodes contain statements about a world situation. This scheme could be of use to other AI workers who find that they cannot represent the context structure of a data base as a simple tree.

Work in Progress

The main work in progress is the writing of small house building domains in TF. Section 7 gives an example and outlines the aims we have in these experiments. These aims can be summarized as

1) To find some simple rules for describing domains hierarchically in order to enable the system to guide someone writing in TF.
2) To aid in the development of error messages and debugging aids for TF.
3) To use the planner to generate project networks from a TF description of a task and to provide the necessary planning techniques to ensure this can be done efficiently.

At present, choices of schema to expand a pattern are made automatically and in a simple way by the planner. Eventually we wish to make choices using three sources of information.

1) The user will interact with the system to make certain choices.
2) Critical path information will be used to constrain choices.
3) A "decision graph" will be constructed by the system (Daniel, 1976) which will record the logical dependancies of links inserted in the network, or added to the network. As mentioned earlier this will be based on some work on a journey planning system which could replan on execution failure (Hayes, 1975).

Work has been performed by Eder (1976) on a non-linear planner which uses program computed priorities to select the order in which nodes in the network are expanded. This work has also begun the investigation of the use of Generalized Networks to represent a non-linear plan. These networks are not constrained to being described solely in terms of binary links between nodes. Statements such as not between (x,y,z) and don't mix (w,x,y,z), as well as general evaluable constraints can be stated for a network. This can reduce the number of linearizations generated as interactions are found and corrected for.

Effort is still needed to provide a smooth user interface to TF. The house building work should enable us to get a better idea of the forms needed for such tasks. In particular we wish to provide graphic input of the orderings on nodes in the expansion of a schema. Graphic output of the project network produced will be vital for any but the smallest networks.

Currently, we are also using TF and NONLIN for description of very simple robot assembly tasks. The COMPUTE conditions described in section 2.3.1 were provided as a necessary facility for this domain. We hope to have the opportunity to pursue the assembly planning in the near future. This work will involve the provision of conditional planning facilities in order that sensory information from the execution of some task can be used to guide assembly.

Recently, it has become clear that a formalism for hierarchically specifying a domain and a planner able to generate plans for tasks in that domain may have wider application than the sort of planning tasks considered in this paper. In particular, the formalism may provide the basis for a hierarchical language to communicate algorithms based upon parallel processes, and the planner may form the basis of a scheduler to decide on lower level processor allocation in such a system.

APPENDIX 1: The Data Base System used by NONLIN

NONLIN uses the HBASE data base system (Barrow, 1975) which is a POP-2 package providing a semantic net like data base of patterns (e.g. <<AT BALL HERE>>) each of which may be given values in some context. Retrieval functions are provided to access the data base. A retrieval may specify a class of patterns using a "wild card" (==) or an "actor" (e.g., <:NON TABLE:>). For the facilities needed in NONLIN and other work we have found it desirable to provide a few additional features in HBASE. These are described below.

Bobrow and Raphael (1974) list the types of variables available in recent programming languages for Artificial Intelligence as:-

1) Open:	an open variable will match any item assigning item to the variable.
2) Closed:	a closed variable will match only an item equal to the previously assigned value of the variable.
3) Semiopen:	a semiopen variable will match any item if it has not previously been given a value; it will match the previously assigned value when it has one. Thus, it acts as an open variable the first time it is encountered, and a closed variable after that. Again, assignment to the variable takes place when the variable first matches an item.
4) Restricted:	a restricted variable will match items that satisfy a set of restrictions which may be specified for that variable, e.g., only match elements of a particular domain. A matching item is assigned to the variable.
5) Macro:	a macro variable has its value substituted in the pattern before a match takes place, thus allowing indirect reference to variables, etc.

The HBASE data base package (Barrow, 1975) provides variables of

type 1)	Open	e.g., $>X.
type 2)	Closed	e.g., $$X.
type 5)	Macro	by using evaluation brackets for a pattern e.g., </ ... />.

This note describes how variables of type 3 (semiopen) and type 4 (restricted) are provided.

I.1 $* - a semiopen variable for HBASE

HBASE provides 2 variable actors: one to assign to a variable ($>) and another to read or use a variable ($$). We have provided a semiopen variable with prefix $* (actor <:GIVEN <variable>:>).

Implementation

Obviously the success of $*X match is assured with VALOF("X")=UNDEF. However, if this occurs in a more global match, e.g., MATCH(<<AT $*X $*Y>>. <<AT ROBOT HERE>>) we must take care. Matches are left to right in HBASE.

"AT" matches "AT"

$*X matches "ROBOT" and assigns "ROBOT" to VALOF("X") if VALOF("X")=UNDEF

But, $*Y may have a value "THERE", so $*Y will not match "HERE".

The whole match fails. We must reset the changed VALOF("X") back to UNDEF. A mechanism is needed to remember which variables are altered so that they can be reset if the whole match fails. For efficiency, we just remember these on a single top level list of "set" variables (SETVARS) at the outer call of MATCH and not in local list for each recursive call of MATCH. The normal MATCH routine is saved as SMATCH and the top level MATCH function is redefined.

VARVAL is used as a lookup function for the value of a variable, rather than VALOF, so that a user may redefine the lookup to search for a value of a variable in a local ALIST etc. By default, VARVAL would be VALOF. A version of $* which looks up the value of a variable in a local ALIST has been in use since early 1974 in INTERPLAN (Tate, 1974) and is used in NONLIN.

Preset Values for $* Variables

Variables used in $* mode must be preset before a match either to UNDEF or to some definite value.

Caution: POP-2 keeps the old value of any global variable when using a variable local to a function.

I.2 Restricted Variables

Variables can be in one of 3 states.

Open:	with value UNDEF. can receive any value.
Restricted:	with value an actor. can receive any value which matches the actor restriction.
Closed:	has some value (not UNDEF or an actor). only matches the actual value

The point made in section I.1 about having to remember set variables in case the top level match fails applies here. However, in the previous case only the variable set had to be remembered as after failure all variables set ($*) are returned to value UNDEF. In the case of variables with actor restrictions, these must also be remembered to enable the resetting to take place. The list SETVARS thus holds the UNDEF or value

of the actor restriction as well as the actual variable name (i.e.,
[(value) (variable name) (value) (variable name) ...]). The $> variable checks any actor restriction before assignment of a suitable value, but is not reset on failure. This corresponds to the use of $> without restriction where top level match failures do not reset any $> variables set earlier.

Note: $* variables must be initialized to UNDEF or an actor restriction before use. A macro RESTRICT is provided to aid in this presetting.

```
e.g., RESTRICT W <:NON FLOOR:> X <:ONEOF A B C:>
                Y <:NON $*X:>   Z UNDEF;
```

I.3 Instantiable Actors

Since we allow actors to be given as restrictions to variables, it may be that we try to instantiate some variable (e.g., for HBASE retrieval functions) while it has such an actor restriction. In HBASE most actors cannot respond to an attempt to be instantiated. However, some can (e.g., $$ the closed variable and $*). We have thus provided a facility to mark actors as instantiable. An attempt to instantiate an actor which is not so marked will produce a copy of the actor with each of its components instantiated (e.g., if X=3 then an attempt to instantiate <:NON $*X:> will produce <:NON 3:>:which will then behave correctly in any HBASE search function).

I.4 Nice Macros

Several macros have been provided to enable initial data bases to be constructed, data bases printed, etc. These include:

```
ASSERT <pl> <p2> ... ;  gives value "true" to <Pl>, <p2>, etc.
DENY   <pl> <p2> ... ;    "     "  "false" "   "     "   "
UNBIND <pl> <p2> ... ;    "     "  "undef" "   "     "   "
PRCONTEXT prints all patterns which have been assigned values in CUCTXT
          together with their values
```

APPENDIX II: Error Messages and Warnings

Errors in TF descriptions

<item> INCORRECT FORM FOR OPSCHEMA

an unrecognized keyword is given or form does not have correct syntax (especially note that VARS ends with ;)

AT CLAUSE OF CONDITION NOT SPECIFIED

FROM CLAUSE NOT SPECIFIED FOR A SUPERVISED CONDITION

FROM CLAUSE CANNOT BE GIVEN FOR A CONDITION OTHER THAN SUPERVISED

<item> INCORRECT FORM FOR MAINEFFECTS

again, as for OPSCHEMA, an unrecognized keyword is given or a form does not have correct syntax.

Plan-time Errors

<type> NULL EXPANSION ON ILLEGAL TYPE

an expansion must be given for a goal node. A null expansion is only allowed (it is the default value) for an action node.

<condtype> REPLACED BY <condtype>

This is a warning that a <condtype> is being replaced by one of higher priority. This is unusual in a hierarchical description where <condtype> normally gets weaker lower in the hierarchy. Check the task description.

<variable name> VARIABLE NON EXISTANT

Access attempted to a variable not declared in the OPSCHEMA in use. Check that VARS statement of an OPSCHEMA contains <variable name>.

[<pattern> = <compute clause>] COMPUTATION UNDEFINED

A computation should always return an answer - not UNDEF.

NO WAY TO PROCEED

An exhaustive search on the problem has failed to generate any solution (or further solutions if REPLAN was called).

APPENDIX III: Interrogation routines and network operations

PRINTNET(	print the current network by printing for each node its nodenum, nodetype, pattern, prenodes and succnodes. We intend to write some graphics routines to display and read in nets in future versions of the system.
PRTOME	prints the table of multiple effects.
PRGOST	prints the goal structure statements.
PRCON(<context>);	prints any HBASE context using PRCONTEXT (see Appendix I).
NODE(<nodenum>);	returns the node held in the <nodenum> subscript of ALLNODES.
PRNODECTXT(<nodenum>);	prints all statements with either a true (1) or false (0) value which can be found from NODE (<nodenum>) - using QAALL.
<nodenum> --> nodenum>;	inserts a link from the 1st node to the second in the network. Redundant links are removed.
BEFORE(<nodenum>, <nodenum>)	returns true if the 1st node is before the second.

There are also several tracing and interrupt switches

BUGEXPAND	causes a message to be printed out as each node is expanded EXPAND <node type> <pattern>
INTERINT	causes an interrupt on each entry to the interaction correction function.
CONDINT	causes an interrupt whenever new conditions are added to a node.

After an interrupt, typing "GOON" resumes the planning. The system may be interrogated and the net altered if desired during an interrupt.

While COMPUTE conditions are being evaluated a variable CURRNODE holds the node being expanded. This may be useful for the interrogation of the network by a COMPUTE function.

ACKNOWLEDGEMENTS

This work has formed part of the project "Planning: a Joint AI/OR Approach" supervised by Professor B. Meltzer at the Department of Artificial Intelligence at the University of Edinburgh. The work was supported by the Science Research Council (Grant No. B/RG/94455).

The work has progressed rapidly and developed through discussions with my co-worker on the project, Lesley Daniel. Gottfried Eder was responsible for some of the terminology used in the network question answerer. Jerry Schwarz read my first descriptions of the task formalism and his comments helped me form the present version of TF. I would like to thank Peggy Avison for transforming my handwritten scrawl into a memo.

References

Barrow, H.G. (1975). HBASE POP-2 library documentation, Department of Artificial Intelligence, University of Edinburgh.

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

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

Daniel, L. (1976). Project planning: modifying non-linear plans. Forthcoming in DAI Memo.

Daniel, L. and Tate, A. (1976). Planning: A joint AI/OR approach, AISB Newsletter, No. 23.

Eder, G. (1976). A system for cautious 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.

Hart, P.E. (1975). Progress on a Computer Based Consultant. Advance papers of 4th International Joint Conference on Artificial Intelligence (IJCAI4), Tbilisi, USSR. pp. 831-841.

Hayes, P.J. (1975). A representation for robot plans. Advance papers of 4th International Joint Conference on Artificial Intelligence (IJCAI4), Tbilisi, USSR. pp. 181-188.

McDermott, D.V. & Sussman, G.J. (1972). The CONNIVER Reference Manual, MIT AI Lab., Memo No. 259.

Sacerdoti, E.D., (1975a). The non-linear nature of plans. Advance papers of 4th International Joint Conference on Artificial Intelligence (IJCAI4), Tbilisi, USSR, pp. 206-214.

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

Siklossy, L. and Dreussi, J. (1973). An efficient robot planner which generates its own procedures. Advance papers on 3rd International Joint Conference on Artificial Intelligence, (IJCAI3), Stanford, U.S.A.

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

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

Tate, A. (1975a). Using goal structure to direct search in a problem solver. Ph.D. thesis, Machine Intelligence Research Unit, University of Edinburgh.

Tate, A. (1975b). Interacting goals and their use. Advance papers of 4th International Joint Conference on Artificial Intelligence (IJCAI4), Tbilisi, USSR. pp. 215-218.

Wiest, J.D. and Levy, F.K. (1969). A management guide to PERT/CPM, Prentice-Hall, New Jersey, USA.