

FAILURE PROCESSING IN A SYSTEM FOR DESIGNING COMPLEX ASSEMBLIES

Jean-Claude LATOMBE

IMAG, BP 53X, 38041 GRENOBLE cédex
France

We believe that the capacity to recover intelligently from intermediate failures and to learn from these failures is an essential ingredient of attempting to solve complex real-world problems. Intelligent failure recovery requires the problem-solver not only to diagnose the causes of its failures, but also to circumscribe carefully the propagation of recovery updating. Learning from failures requires that the problem-solver remember the conditions of its failures in such a way that it will not recreate these conditions. In this paper, we describe methods providing such capabilities. We have implemented and experimented with these methods in a rule-based system used for designing complex assemblies such as electromechanical equipment.

1. INTRODUCTION

A traditional difficulty in automating the problem-solving process is that of dealing with the combinatorial explosion of the search space. Another difficulty is the difficulty of handling large amounts of domain-dependent knowledge. In this paper, we present new methods for coping with both these difficulties simultaneously. They combine a search representation based on the distinction between control data and logical data, and procedures working with this representation that facilitate selective recovery and learning from failures. The recovery procedure diagnoses the causes of the failures in order to perform selective backtracking and the representation makes it possible to circumscribe carefully the propagation of recovery updating. This combination turns out to be particularly important in reducing unnecessary operations on knowledge.

We have implemented and experimented with these methods in a rule-based system named TROPIC, a problem-solving tool for designing complex assemblies such as electromechanical equipment[4]. Problems with which this system can deal consists of generating models of devices. TROPIC is not restricted to a particular discipline, and we have used it to design electric power transformers among other applications [5]. This application required us to encode large amounts

of special-purpose knowledge (about 350 rules) related to such fields as electricity, electromagnetism, mechanics and thermodynamics. Only a few aspects of TROPIC are presented here. More detailed descriptions can be found in [5] & [6].

Although we claim originality for the methods to be discussed in this paper, the fundamental issues underlying these methods are not new in Artificial Intelligence. For example, the distinction between control data and logical data has been an important aspect of NOAH[9]; failure diagnosis and learning has been studied in EL/ARS [10]. Relations of TROPIC to these works will be discussed in Section 8.

2. OVERVIEW OF THE PROBLEM-SOLVING MODEL USED BY TROPIC

The problem-solving model used by TROPIC is based on the concurrent execution of a problem-solving process, which we call "task-reductive synthesis" and a deduction process, which we call "fact-deductive analysis" :

1) Task-reductive synthesis consists of assuming facts about a potential device (for example the type of components, the connection between these components and the values of various parameters), while partitioning a given "initial task" into successive subtasks. For example, if the problem is to design a transformer tr1, a step of this process might be to assume that parts of tr1 are a magnetic circuit mcl and an electrical circuit ecl, and to reduce the task "design transformer tr1" into three subtasks, "design magnetic circuit mcl", "design electrical circuit ecl" and "select a cooler for tr1".

This work was carried out under contract
IRIA/SESORI n° 77052

One can regard the facts that are assumed when a task is reduced as a solution to this task at some level of abstraction, and task-reductive synthesis as a process of adding successive detail to a description. The system may have to choose among alternative ways of partitioning the same task, each of them corresponding to different assumptions. For example, the task "select a cooler for tr1" may be reduced to the empty set by assuming that tr1's cooler is air; it may also be reduced to the subtask "design tank tk1" based on the assumption that tr1's cooler is oil, and that it is contained in a tank tk1. As we will see next, some combinations of choices may produce contradictions.

2) Fact-deductive analysis consists of deducing new facts from the assumed facts and of controlling the consistency of the set of both assumed and deduced assertions. Deductions are made by using such kinds of knowledge as constraints and approximations to physical laws. For example, the design constraints "if air is the cooler of a transformer, then the voltage between the terminals of any coil in this transformer must be lower than 3KV; otherwise it must be lower than 5KV" may be used to deduce the fact "the voltage between the terminals of c11 is lower than 5KV" from the facts "the cooler of tr1 is oil", "c11 is a part of tr1" and "c11 is a coil". Contradictions among assumed and deduced facts produce failures, from which the system recovers by resuming task-reductive synthesis at a previous decision point. For example, the facts "the number of turns of c11 is 350", "the voltage per turn of c11 is 18V", "the voltage between the terminals of c11 is lower than 5KV", and "the voltage between the terminals of c11 is equal to the number of turns of c11 times the voltage per turn of c11" are contradictory facts.

A problem is specified as an initial task, a set of synthesis rules to be used for task-reductive synthesis, and a set of analysis rules to be used for fact-deductive analysis. It is solved when the initial task has been reduced to the empty set in such a way that no contradiction derives from the assumptions generated by this reduction.

3. TASK-REDUCTIVE SYNTHESIS

3.1. Form and use of synthesis rules

Knowledge for task-reductive synthesis is represented in TROPIC by means of synthesis rules. The form of these rules is the production:

$$t \Rightarrow \{ \{f_1, \dots, f_m\}, \{t_1, \dots, t_n\} \}$$

where $t, t_1, \dots, t_n$ ($n \geq 0$) are expressions representing tasks and $f_1, \dots, f_m$ ($m \geq 0$) are expressions representing facts. The right-hand side may also be a function that evaluates to the expression $\{ \{f_1, \dots, f_m\}, \{t_1, \dots, t_n\} \}$.

The production uses to contain variables that are instantiated when it is applied.

Given an initial task and a set of synthesis rules, task-reductive synthesis proceeds according to a depth-first search by successive expansion of an AND/OR tree, which we call task-reduction tree. The AND nodes of this tree are labeled with instantiated tasks, and its OR nodes are labeled with instantiated synthesis rules. At every moment during the problem-solving process, this tree represents the partitioning of the initial task that is currently considered by the system.

Initially, the task-reduction tree consists of a single node that is labeled with the initial task. Next, each expansion of the tree is carried out by applying a synthesis rule such that the task in its left-hand side matches the task associated to a pending AND node N of the current tree. The following two operations are then executed:

(1) An OR node N' is added to the tree as a successor of N, and the instantiated synthesis rule is associated with it.

(2) Each task in the right-hand side of the instantiated rule is associated with an AND node that is added to the tree as a successor of N'; if the set of tasks in the rule's right-hand side is empty, then N' is a terminal node.

Simultaneously, the facts contained in the rule's instantiated right-hand side are entered into a factual data base as new assumptions about the device being designed.

The task-reduction tree is terminal when all its pending nodes are terminal nodes. If there is no contradiction in the factual data base, the problem is solved then. TROPIC applies various general-purpose and input domain-specific heuristics to guide task-reductive synthesis [6].

3.2. Examples

These synthesis rules correspond to the informal examples of section 2:

SY1:(design-transformer &tr)

=> (progn

(setq &mc (! mc))

(setq &ec (! ec))

(i&{(part-of &tr &mc),

(=(is-a &mc) magnetic-circuit),

(part-of &tr &ec),

(=(is-a &ec) electrical-circuit)),

{(design-magnetic-circuit &mc),

(design-electrical-circuit &ec),

(select-cooler-for &tr)))))

SY2:(select-cooler-for &tr)

=> (progn

(setq &tk (! tk))

(i&{(=(cooler-of &tr)oil),

(part-of &tr &tk),

(=(is-a &tk) tank)),

{(design-tank &tk))}))

```
SY3:(select-cooler-for &tr)
=> ({(=(cooler-of &tr) air)},nil)
```

[Comment : When rule SY1 is applied, the variable "&tr" (all atoms prefixed by "&" are variables) is first bound to a constant (for example: "trl") by unifying the left-hand side of the rule with an instantiated task ; next, the right-hand side of the rule is evaluated. This evaluation binds the variables "&mc" and "&ec" to constants (for example "mcl" and "ecl") generated by evaluating (!mc) and (!ec) ("!" generates a new atom each time it is evaluated by concatenating its quoted argument and an integer). The evaluation of the TROPIC function "i" then returns the instantiated expression ({(part-of trl mcl),...}, {(design-magnetic-circuit mcl),...}). When rule SY3 is applied, its right-hand side is merely instantiated).

About 100 synthesis rules were necessary to run the transformer example.

3.3. Fundamental aspects of task-reductive synthesis

Task-reductive synthesis explicitly represents the problem-solver's control state (the task-reduction tree) and logical state (the factual data base). At any moment, the logical state contains the facts that are currently believed to be true by the system, while the control state records both the choices on which these beliefs are based and the control dependencies among these choices. This twofold representation may be regarded as a combination of the "state-space" and "problem-reduction" representation[8]. The failure recovery procedure used by TROPIC draws much of its power from the following two possibilities :

1 - When a choice is rejected, the explicit knowledge of the current control state permits the system to circumscribe the extent of the updating necessary. Only that part of the task-reduction tree which is rooted at the rejected choice, and the subsequent assumptions, need to be discarded. Other choices can be kept effective even if they are more recent choices. This possibility may save a lot of effort reintroducing large parts of the solution that were needlessly discarded. It is not provided by the state-space representation, at least in an easy way.

2 -The logical state permits the system to consider the combined effects of the individual choices recorded in the control state. Choices, which may not be linked among them by control dependencies, may concur to generate contradictory assertions, so that the maintenance of the logical state's consistency permits the system to take into account logical dependencies among such choices. This possibility to consider transversal interactions among choices is critical because the hypothesis according to which design tasks are

reducible to independent subtasks would be unrealistic. Its is not provided by the problem-reduction representation.

4. FACT-DEDUCTIVE ANALYSIS

4.1. Form and use of analysis rules

Knowledge for fact-deductive analysis is represented in TROPIC by means of analysis rules. The form of these rules is the production

$$\{f_1, \dots, f_m\} \Rightarrow \{f'_1, \dots, f'_n\}$$

where $f_1, \dots, f_m, f'_1, \dots, f'_n$ ($m \geq 0, n \geq 0$) are expressions representing facts. The right-hand side may also be a function that evaluates to the expression $\{f'_1, \dots, f'_n\}$ ($n \geq 0$).

Each analysis rule is subject to pattern-directed invocation. When the facts in its left-hand side match facts in the factual data base, its right-hand side is instantiated or evaluated according to the matching substitution and the resulting facts are entered into the data base. These facts may in turn contribute to the invocation of other analysis rules. Rules with empty left-hand sides are invoked as soon as problem solving starts so that facts in their right-hand sides are premises to the solution. A deduction graph records all the deductions that have resulted from the invocation of analysis rules. When a contradiction is noted, it is used in reverse to determine the set of assumptions that are antecedents of the contradictory facts. During failure recovery, it is used in the forward direction to update the factual data base.

4.2. Examples

These are sample analysis rules :

```
- {(=(is-a &tr) transformer), (=(cooler of &tr) &cf),
  (part-of &tr &cl), (=(is-a &cl) coil)}
=> (progn
    (if (eq &cf air)
        then (i&{(<(voltage &cl) 3e3)})
        else (i&{(<(voltage &cl) 5e3)})))
```

This rule implements: "if air is the cooler of a transformer, then the voltage between the terminals of any coil in this transformer must be lower than 3KV ; otherwise, it must be lower than 5KV".

```
- {(=(is-a &tr) transformer),
  (part-of &tr &tk), (=(is-a &tk) tank)}
=> {(=(temperature-rise-in-oil-of &tr)
    (temp-oil (losses-in &tr)
    (radiating-surface-of &tk)))}
```

This rule encodes a physical law in its right-hand side. Atom "temp-oil" is the name of an input function that computes the temperature rise in the oil based on the losses in the transformer and the radiating surface of the tank.

4.3. Detection of contradictions

A contradiction occurs when some facts in the factual data base cannot be true simultaneously.

Two programs of TROPIC are responsible for detecting contradictions. One is based on heuristic numerical search techniques. We refer the reader to [5] for a description of these programs. Contradictions produce failures from which TROPIC immediately attempts to recover.

5. RECOVERY FROM FAILURES

5.1. Principle

When a failure is noted, the failure recovery procedure used by TROPIC determines the logical support of the contradiction, i.e. the subset of those assumptions from which the contradictory facts were derived. Then, it rejects the choice of a synthesis rule the application of which has produced at least one of these assumptions. Thus, it never tries to reject a choice that is irrelevant to the failure. In addition, only those choices which are subordinated to the rejected choice by control dependencies are immediately discarded. However, there are usually several relevant choices that may be rejected alternatively in order to recover from a failure. The recovery procedure must be able to consider them in a systematic way in order to avoid perpetual iteration through successive failures or omission of possible solutions. Indeed, the choice that is rejected will be replaced later by another choice and it may happen that this new choice leads to another failure from which it is possible to recover by reintroducing the previously rejected choice. But, since this choice already led to failure, its return, if it is not accompanied with other actions, will drive TROPIC into an infinite loop of successive failures. On the other hand, ruling out the possibility of reactivating this choice may result in omitting potential solutions, if there was more than one way to recover from the earlier failure. This difficulty requires that the system remembers the conditions of its failures so that, if a once rejected choice is to be reintroduced, the recovery procedure can previously correct the failure responsible for this choice being rejected in a different way.

The recovery procedure of TROPIC avoids infinite loops through failures and omission of possible solution as follows :

- The conditions of the failures that are encountered are remembered in expressions called failure indicators, which are attached to the choice nodes (AND nodes) corresponding to the rejected choices. If recovering from a new failure is possible by reactivating a previously rejected choice, then the recovery procedure can use the corresponding failure indicator to recover differently from the failure that was responsible for this choice being rejected. It is only after making this change that the once rejected choice is reintroduced. In this way, the system never tries

the same combination of choices more than once.

- The recovery procedure avoids omitting potential solutions by operating chronologically. It never rejects a choice such that rejecting a more recent choice would enable TROPIC to recover from the current failure without recreating the conditions of an earlier failure.

5.2. A simple expository example

We now describe a possible behavior of TROPIC on the oversimplified problem defined by Table 1, "A" being the initial task. For the sake of simplicity, we represent tasks by capital letters and facts by lower-case letters. Each fact has the form "a" or "-a" (negation of "a"), so that a contradiction can only occur between a fact and its negation. In addition, we omit synthesis rules providing alternative ways of reducing tasks A,B,C,..., because they will not be used in the particular solution considered below ; similarly, the analysis rules that will never be invoked are now shown. Three failures will be encountered before a solution is ultimately produced.

Suppose that TROPIC applies synthesis rules <1>, <2>, <5>, <3>, <8>, <9>, <15>, <16>, <4>, <11>, <13>, <17>, <18> and <6> successively. Then, the task-reduction tree is that of figure 1. For the sake of readability, the assumed facts have been associated with the corresponding OR nodes of the tree. The oriented dashed line that is superposed on the tree connects the expanded AND nodes in the chronological order of their expansion. We call it the decision path and it is generated during task-reductive synthesis.

At this moment, the system notes the first failure, since fact -e, which is contradictory with assumption e, is deduced (invocation of analysis rules <1> and <2>'). An expression, called the diagnostic expression, is then initialized to $\{e, h, c_1, f, \text{nil}\}$, where $\{e, h, c_1, f\}$ is the logical support of the contradiction (cf. § 5.1.). The general form of this expression is (A, T) , where A is a set of assumed facts and T is a set of tasks. It is always such that the conjunction of the conditions "each element of A is an assumption in the factual data base" and "each element of T is a task labeling an AND node of the task-reduction tree" is a sufficient condition for having encountered a failure.

Then, the system traces back the decision path until it meets an AND node N that satisfies the following two criteria :

C1 : An assumption or a task contained in the set A or T of the diagnostic expression (A, T) has been generated when N was expanded.

C2 : There exists an alternative synthesis rule for expanding N, the application of which would not result in generating this assumption or this task.

N is called the recovery node. In the present case it is the node labeled by task F, since the reduction of F by rule <7> would not generate assumption f. The expression $((\{e, h, c_1\}, \text{nil}), (\{f\}, \text{nil}))$ is then drawn from the diagnostic expression and it is associated with N as a failure indicator ϕ_1 . The general form of this expression is $((A_1, T_1), (A_2, T_2))$, where A_1 and A_2 are sets of assumed or assumable facts, and T_1 and T_2 are sets of tasks. It is a conditional statement saying that expanding N with a synthesis rule that generates each assumption in A_2 and each task in T_2 always leads to failure if each element of A_1 is an assumption in the current factual data base and if each element of T_1 is a task labeling an AND node of the current task-reduction tree. Expressions (A_1, T_1) and (A_2, T_2) are respectively called the antecedent and the consequent of the failure indicator. When the conditions "each element of A_1 is an assumption in the data factual data base" and "each element of T_1 is a task labeling an AND node of the task-reduction tree" are true because of the expansion of AND nodes that are all predecessors of N in the decision path, the failure indicator is ON; otherwise, it is OFF. All failure indicators recorded by TROPIC are ON at the time of their creation; it is in particular the case of ϕ_1 .

The choice of synthesis rule <6> is now rejected, so that fact f is no longer assumed and analysis rules <1> and <2> are desinvoked. Then, task-reductive synthesis is resumed by applying synthesis rule <7>. At this moment, the task-reduction tree is that of figure 2 but the system notes the second failure since fact-b is deduced (invocation of analysis rules <3>, <4> and <5>). It sets the diagnostic expression to $((\{b, g, f'_2, c_2, f'_1\}, \text{nil}), \text{nil})$ and it traces back the decision path:

- The node labeled by F satisfies both criteria C1 and C2 stated above since the application of rule <6> would not generate assumptions f'_1 and f'_2 . But failure indicator ϕ_1 , which is currently ON, shows that reselecting rule <6> would inevitably recreate the conditions of an earlier failure. Therefore, this node is not chosen for recovery node. Instead, the system executes the following two operations:

- . it draws the expression $((\{b, g, c_2\}, \text{nil}), (\{f'_1, f'_2\}, \text{nil}))$ from the current diagnostic expression and it associates that expression with the node labeled by F as a failure indicator ϕ_2 .

- . it sets the diagnostic expression to $((\{b, g, c_2, e, h, c_1\}, \{F\}), \text{nil})$, by combining the antecedents of ϕ_1 and ϕ_2 . Indeed, in order to recover from the current failure, TROPIC may now either reject a choice responsible for one of the assumptions b, g and c_2 (so that a cause of the current failure will be removed), or

reject a choice responsible for one of the assumptions e, h and c_1 (so that it will be possible to reselect rule <6>), or reject the choice responsible for the generation of task F (so that the need for assuming either f, or f'_1 and f'_2 , will be removed).

- Nodes labeled by Q, P, J, I, D, N and M do not verify criterion C1. None of them is chosen for recovery node. Thus, the system passes over seven backtrackable nodes that would have been successively considered by a blind chronological procedure.

- The node labeled by H satisfies both criteria C1 and C2, since applying rule <10> would not generate assumption h. No ON failure indicator being associated with this node, TROPIC chooses it for recovery node and associates the failure indicator $\phi_3 = ((\{b, g, c_2, e, c_1\}, \{F\}), (\{h\}, \text{nil}))$ with it. The recovery node and the node labeled by F, the expansion of which made the current failure effective, belong to two different AND paths. This means that the corresponding two choices are not logically independent.

The system now updates the current solutions. first, it rejects the choice of rule <9>, so that fact h is no longer assumed. Because they are subordinated to this choice by control dependencies, rules <15> and <16> are also discarded. Next, TROPIC notes that failure indicator ϕ_1 has become OFF (because of the removal of assumption h), while failure indicator ϕ_2 remains ON. The consequent of ϕ_2 shows that the choice of rule <7> cannot be maintained which means that the current failure has not been remedied yet. However, since reselecting rule <6> has just been made possible (ϕ_1 being OFF), TROPIC can now reject the choice of rule <7>, which was a choice responsible for this failure. After rejecting this choice, the task-reduction tree is that of figure 3 (note the updated decision path). Several parts of the current solution remain unchanged although they were produced after the recovery node was expanded.

Task-reductive synthesis is resumed again: rules <10> and <6> (because of ϕ_2) are successively applied (cf. figure 4). This leads the system to note the third failure because fact -g is deduced (invocation of analysis rules <6'>, <7'> and <8'>). The diagnostic expression is set to $((\{g, d, i, f, h'_1\}, \text{nil}), \text{nil})$ and the decision path is traced back:

- The node labeled by F satisfies both criteria C1 and C2, but because of ϕ_2 , which is currently ON, it is not chosen for recovery node. The failure indicator $\phi_4 = ((\{g, d, i, h'_1\}, \text{nil}), (\{f\}, \text{nil}))$ is associated with this node and the diagnostic expression is set to $((\{g, d, i, h'_1, b, c_2\}, \{F\}), \text{nil})$.
- The node labeled by H satisfies both criteria

C1 and C2, but because of ϕ_3 , which is currently ON, it is not chosen for recovery node. The failure indicator $\phi_5 = ((\{g,d,i,b,c_2\},\{F\}),(\{h'\},\text{nil}))$ is associated with this node and the diagnostic expression is set to $(\{g,d,i,b,c_2,e,c_1\},\{F,H\})$. Note that this expression combines conditions of the first two failures and conditions of the current failure.

- Nodes labeled by Q,P and J do not satisfy criterion C1. None of them is chosen for recovery node.
- The node labeled by I satisfies both criteria C1 and C2. No ON failure indicator being associated with this node, it is chosen for recovery node and the failure indicator $\phi_6 = ((\{g,d,b,c_2,e,c_1\},\{F,H\}),(\{i\},\text{nil}))$ is associated with it.

TROPIC rejects the choice of rule <11>, so that fact i is no longer assumed. Then, it examines the failure indicators that have been associated with successors of the node labeled by I in the decision path. Because fact i is no longer assumed, failure indicators ϕ'_4 and ϕ_5 are now OFF, so that the system recognizes that it has recovered from the current failure. Thus, no other choice is rejected.

Task-reductive synthesis is resumed by applying rule <12> and rule <14>. At this moment, the task-reduction tree is terminal (cf. figure 5). Since no contradiction is detected, the problem is solved.

This example shows that, each time a failure is recovered from, an ON failure indicator ϕ is associated with the recovery node N. This indicator prevents the system from reselecting the rejected synthesis rule as long as it remains ON. Since ϕ may be turned OFF only if an AND node, which preceded N in the decision path when N was chosen for recovery node, is chosen for recovery node in turn, the same combination of choices cannot be tried more than once, and therefore perpetual iteration is impossible. On the other hand, the diagnostic expression always determines a sufficient condition for encountering a failure. Thus, by tracing back the decision path, the recovery procedure always chooses for recovery node the most recently expanded AND node that may enable the system to resume task-reductive synthesis without re-creating the conditions of an earlier failure. In this way, omission of potential solutions is impossible.

6. LEARNING FROM FAILURES

Failure indicators are efficient means of recording failure conditions because they are connected to the choice nodes of the task-reduction tree. However they do not completely prevent the system from re-achieving the conditions of earlier failures. Indeed, due to intervening failures, an ON

failure indicator may be turned to OFF, so that the conditions determined by its consequent may then be achieved by the application of a synthesis rule; if it happens later that the conditions in its antecedent are also achieved, then the occurrence of a failure is inevitable.

OFF failure indicators are not needed to avoid infinite loops and omission of possible solutions and they also have little chance to become ON again. Thus, TROPIC discards them, so that updating the ON/OFF status of failure indicators merely consists of erasing the failure indicators that are no longer ON. However, the information they contain is not systematically forgotten: when a failure indicator $((A_1, T_1), (A_2, T_2))$ is discarded, the system may decide to generate the failure expression $(A_1 \cup A_2, T_1 \cup T_2)$ that determines a context in which failure is unavoidable. Failure expressions are stored separately from the task-reduction tree. When expanding an AND node N, TROPIC eliminates from consideration every rule the application of which would reestablish the conditions recorded either in an ON failure indicator associated with N, or in a failure expression.

In order to keep the list of failure expressions a reasonable size, TROPIC converts an OFF failure indicator into a failure expression only if it combines the conditions of several failures. In addition, it reconverts failure expressions into ON failure indicator whenever possible: let (A, T) be a failure expression, and let call A_1 and T_1 the subsets of A and T determining conditions that are already achieved before the expansion of an AND node N; if there exists a synthesis rule applicable to N, the application of which would achieve all the conditions in $A_2 = A - A_1$ and in $T_2 = T - T_1$, then not only does TROPIC not apply this rule, it also reconverts the failure expression into an ON failure indicator $((A_1, T_1), (A_2, T_2))$ associated with N.

Failure indicators and failure expressions provide means of a good compromise for recording failure conditions. ON failure indicators avoid infinite loops and omission of potential solutions and they can be used efficiently. Failure expressions enable the system to convert into a usable form information contained in OFF failure indicators. They are less efficiently used, but their number may be arbitrarily limited since they have no effect on the completeness of the problem-solver.

7. RELATION TO OTHER WORKS

Other systems use representations that distinguish between control and logical data, but they do not exploit this distinction in the same way nor for the same purpose as TROPIC. One of them is NOAH [9], a program for generating plans of actions at varying levels of detail. NOAH's control state is a hierarchical net of actions that combines the

decomposition of an action into successively more detailed subactions and the partial orderings of the plans at each level of detail. All actions bring their effects into a model of the world and logical interdependences among actions are considered in this model. When a conflicting interdependence is noted in a plan, correction is attempted only by enforcing ordering of the partially ordered actions. Research on NOAH was deliberately restricted to non-search aspects of problem solving and consequently the system includes no provision for backtracking.

Another related system is NASL [7], a program for designing electronic circuits. NASL's control state is a net of design tasks similar to NOAH's net of actions. As in TROPIC, the goal of the system is to produce the model of a device. NASL includes no backtracking procedure and Mc Dermott proposes an alternative scheme according to which correcting a mistake is treated as a task like another, drawn from the description of the mistake. However, the recovery procedure based on this scheme was not implemented.

Hayes [3] describes a representation for robot plans that consists of two interlinked data structures representing the subgoal structure of a plan and the logical relationships of the decisions taken in constructing the plan. Hayes states that "the representation enables a decision to be remade without remaking chronologically subsequent but logically independent decisions". Nevertheless, the improved backtracking procedure was not implemented.

Selective backtracking has been investigated in EL/ARS [10] a system for analyzing electrical circuits. The recovery procedure implemented in this system has been generalized by Doyle [2]. It has no explicit representation of the problem-solver's control state in the sense of Section 3.3. Subsequently the conditions of the encountered failures are recorded into NOGOOD lists that are very similar to the TROPIC's failure expressions. In order to avoid perpetual looping, these lists must all be examined before making a choice. A possible improvement is de-emphasizing some conditions in the NOGOOD lists in order to produce conditional expressions which are similar to TROPIC's failure indicators, except they are not connected to a control structure. De-emphasis reduces the number of NOGOODs to be considered at any moment, but is difficult to implement because it has no sound domain-independent basis.

SYNTHESIS RULES

- <1> A $\Rightarrow$ ({a},{B,C,D})
- <2> B $\Rightarrow$ ({b},{E,F})
- <3> C $\Rightarrow$ ({c₁,c₂},{G,H})
- <4> D $\Rightarrow$ ({d},{I,J})
- <5> E $\Rightarrow$ ({e},nil)

- <6> F $\Rightarrow$ ({f},{K})
- <7> F $\Rightarrow$ ({f'₁,f'₂},{L})
- <8> G $\Rightarrow$ ({g},nil)
- <9> H $\Rightarrow$ ({h},{M,N})
- <10> H $\Rightarrow$ ({h'₁,h'₂},nil)
- <11> I $\Rightarrow$ ({i},nil)
- <12> I $\Rightarrow$ ({i'},nil)
- <13> J $\Rightarrow$ ({j},{P,Q})
- <14> K $\Rightarrow$ ({k},nil)
- <15> M $\Rightarrow$ ({m},nil)
- <16> N $\Rightarrow$ ({n},nil)
- <17> P $\Rightarrow$ ({p},nil)
- <18> Q $\Rightarrow$ ({q},nil)

ANALYSIS RULES

- <1'> {f,c₁} $\Rightarrow$ {-r}
- <2'> {-r,h} $\Rightarrow$ {-e}
- <3'> {f'₁,c₂} $\Rightarrow$ {s}
- <4'> {f'₂,g} $\Rightarrow$ {t}
- <5'> {s,t} $\Rightarrow$ {-b}
- <6'> {h'₁,f} $\Rightarrow$ {u,-v}
- <7'> {u,i} $\Rightarrow$ {w}
- <8'> {-v,w,d} $\Rightarrow$ {-g}

Table 1

REFERENCES

- [1] R.DAVIS, B.G.BUCHANAN(1977):"Metal-level knowledge : Overview and applications", IJCAI 77, Cambridge, 1977.
- [2] J.DOYLE(1977) : "Truth-maintenance systems for problem solving", Master thesis, MIT, Cambridge, 1977.
- [3] P.HAYES(1975):"A representation for robot plans", IJCAI 75, Tbilisi, 1975.
- [4] J.C.LATOMBE(1976):"Artificial Intelligence in computer-aided design: the TROPIC system" SRI AIC, Technical note 125, Menlo Park, January 1976.
- [5] J.C.LATOMBE(1977):"Une application de l'Intelligence Artificielle à la conception assistée par ordinateur (TROPIC)", Thèse d'Etat, Grenoble 1977 (in French).
- [6] J.C.LATOMBE(1979):"Problem-solving methods in a system for designing complex assemblies" Rapport de recherche IMAG, Grenoble, 1979 (in preparation).
- [7] D.V.McDERMOTT(1977):"Flexibility and efficiency in a computer program for designing circuits", AI Lab. TR 402, MIT, Cambridge, June 77.
- [8] N.J.NILSSON(1971):"Problem-solving methods in Artificial Intelligence", McGraw-Hill, 1971.
- [9] E.D.SACERDOTI(1977):"A structure for plans and behavior", American Elsevier Publishing Company, 1977.
- [10] R.STALLMAN, G.J.SUSSMAN(1976):"Forward Reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis", MIT AI Lab, Memo 380, Cambridge, September 1976.

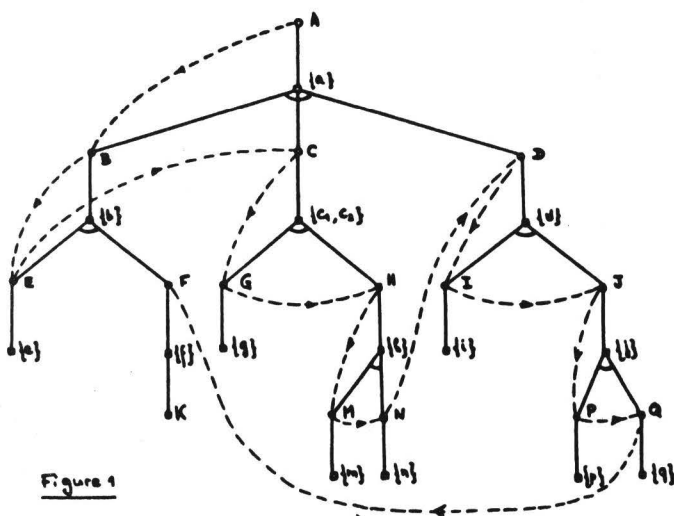


Figure 1

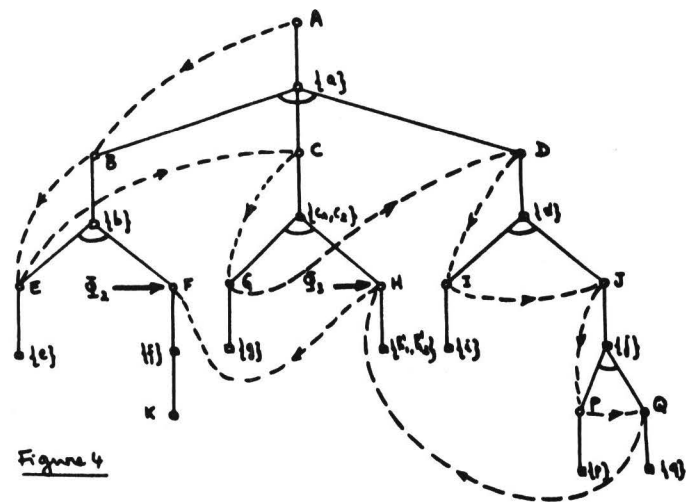


Figure 4

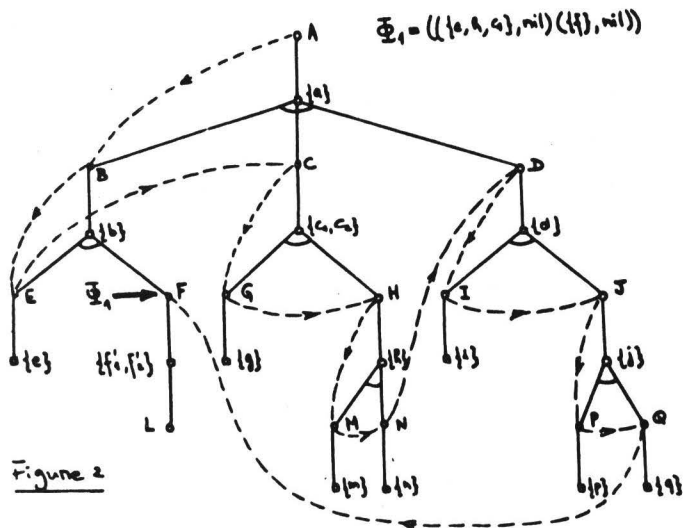


Figure 2

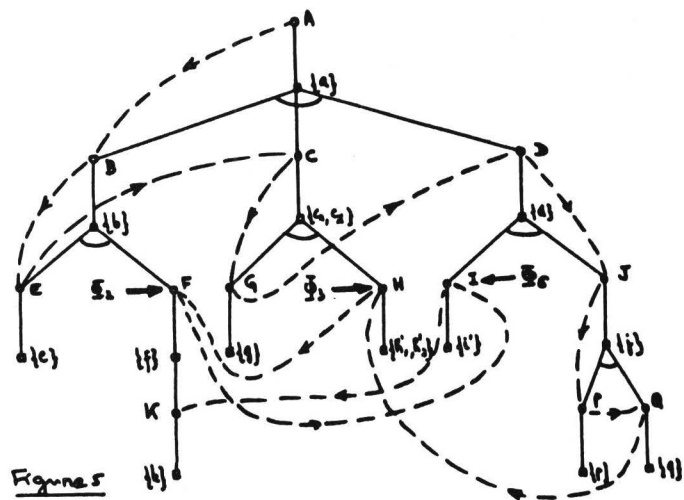


Figure 5

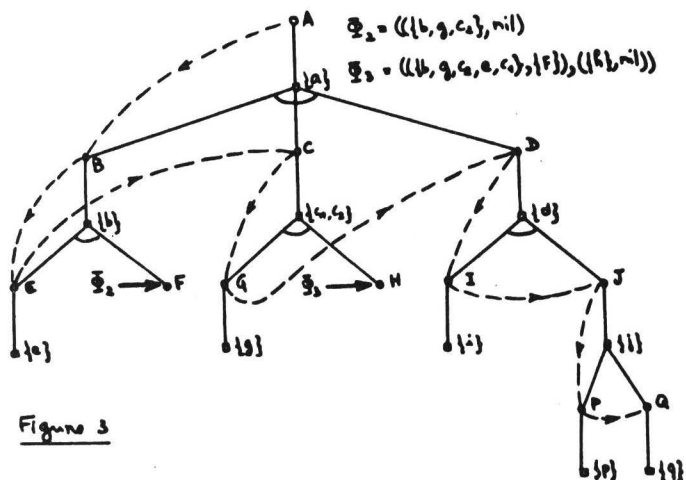


Figure 3