SRC TR 87-201

A Method of Fault Diagnosis: Presentation of a Deep-Knowledge System

by

D.T. Chung and M. Modarres

A METHOD OF FAULT DIAGNOSIS: PRESENTATION OF A DEEP-KNOWLEDGE SYSTEM

D. T. Chung

M. Modarres

Department of Chemical and Nuclear Engineering

University of Maryland

College Park, Maryland 20742

A METHOD OF FAULT DIAGNOSIS: PRESENTATION OF A DEEP-KNOWLEDGE SYSTEM

1. INTRODUCTION

The purpose of a fault diagnostic expert system in a process control environment is to aid process operators in detecting and resolving plant failures. Two general approaches have been applied to the development of expert systems; model-based (deep knowledge) and model-free (shallow knowledge).

A survey of shallow knowledge fault detection expert systems has been discussed by Pau [1], and Waterman [2]. Shallow knowledge expert systems use (if-then) type rules as the primary mean of knowledge representation. These rules are formulated based on a large collection of empirical observations or compiled knowledge. Typically, the development steps for a rule-based system can be generalized as: data abstraction, heuristic mapping onto a hierarchy of preenumerated solutions, and refinement within this hierarchy [3].

Performance of these rule-based diagnostic expert systems can become very effective provided that all failure modes have been compiled, and the failures can be sufficiently characterized from measurements and /or observations. In cases where the

failure modes are not well known, these systems are inadequate, and deep knowledge systems are more appropriate.

When confronted with an unfamiliar problem an expert can resort to "first principles". Through in-depth understanding of the problem, an expert can resolve problems that have not been well documented by prior observations [4]. For example, when detecting faults in a process control domain, an expert can diagnose the failure through his fundamental understanding of the principles behind the operations of the plant. In electronics, an expert can isolate the fault by tracing through the functional structure of an electronic circuit [5,6]. The knowledge used by the expert in these situations are referred to as "deep knowledge". There are several diagnostic expert systems using deep knowledge. A number of these systems are presented in the literature [7-11].

In process control, there is a large spectrum of conditions that can lead to failure. Many of these failures are unanticipated and have very low probability of occurrance. As a result, they are not thoroughly characterized and cannot be represented through rule-based shallow knowledge. However, these failure conditions can be diagnosed by using fundamental understanding of the principles behind the operation of the plant.

The purpose of this paper is to present an expert system shell (GOTRES) that utilizes a deep knowledge model which is based on specific plant goals. In this method, the principles of process operations and control are organized in a hierarchial tree structure. This method of modelling knowledge is known as Goal Tree-Success Tree (GTST) [12].

1. The GTST Method

In the GOTRES expert system program, diagnostic procedure is performed using functional relationship represented by a goal tree. The top goal on the tree expresses the system objective. An example of a system objective is: "Prevent Failure of Pump". Goal tree is a hierarchical structure that breaks the objective down to finer goals [13]. Goals in a goal tree are related to its higher level and lower level goals by "WHY" and "HOW" relationships respectively. Specifically, a goal is linked to its parent goal by the relationship "WHY"; while, it is linked to its subgoal by the relationship "HOW".

For a goal to be successful, all of its subgoals must be successful. Each subgoal can, in turn be decomposed to its descendant subgoals. When proceeding down a goal tree, one ultimately reaches goals that no longer can be subdivided reasonably without referencing hardware or plant conditions. At

this point, pertinent hardware or conditions that satisfy the lowest level goal are specified.

The top goal of a goal tree is successful when all of its supporting hardware and conditions are successful. When there is a failed hardware or upset condition in a system, the pertinent goals that are being supported by the failed hardware will also be lost (not achieved), this in turn will lead to the loss of higher level goals. The initiating failure will, therefore, ascend the goal tree to the top most level goal. Conversely, when tracing a failed goal down a goal tree, one ultimately reaches the lowest level hardware or condition whose failure has caused the loss of the top goal.

2. Knowledge Representation in GOTRES

The goal tree and the criteria for testing success of a goal are stored in a data base known as frame representation [13,14]. In this representation each goal in a goal tree is represented by a frame. Each frame, in turn, is represented by an embedded list that contain all attributes associated with a goal. For example, Figure 1 shows such attributes as Goal-Name, Goal-Parent, Subgoal, Success-Requirement, and Heuristic Rules that justify achievement of the goal.

The GOTRES program is an expert system shell. It is capable of inquiring information from the user. It does so by setting up this data into a frame-based representation, and store the information in a LISP file. The goal tree itself does not appear as a tree structure in the knowledge base. Rather, the relational attributes, "parent" and "subgoal" values represent the tree structure and logic. For instance, the top goal in a goal tree can be identified by its parent goal (which is "NIL"),and its immediate subgoals (second level goals). While, the second level goals will have the top goal as their "parent" value.

3. Problem Solving

The GOTRES diagnoses process is performed using the frame representations of the GTST, and system observations. The frame representation encompasses all the attributes of the goals, including the relational knowledge that is specified in the GTST model. The system observations are instrument readings and whatever observable variables that are used when evaluating the success-requirement of goals (such as system behavior and plant conditions). Starting from a top level lost goal, the GOTRES diagnostic process tests and finds path(s) of failed goals, and follows the path(s) down to the initiating cause of failure. When

a goal is successful (achieved), then all of its children subgoals and supporting hardware are considered successful. When a goal is not successful (failed), then there is at least one failed goal on each level of its subgoals, including the supporting hardware and conditions level. In checking goals on any level of the tree, the first encountered failed goal is traced downward. The downward tracing would only check the failed goal's lower level goals (ie., its immediate subgoals). The process continues down the goal tree to reach the actual failed system hardware or upset condition. This process of tracing only failed goals permits GOTRES to quickly reach the cause of a failure.

After completing a given trace through the tree and finding failed hardware or upset conditions, GOTRES further checks the GTST knowledge base for other possible lost goals. This upward process is referred to as "up-checking", and proceeds upward from the site of a diagnosed failure in the lowest level to higher levels of the tree. The function of up-checking is to check goals that have been previously skipped. These are goals on the same level as the previously encountered failed goals that had been skipped because of locating a failed goal. In the up-checking procedure, GOTRES first checks the remaining goals on the same branch and level as the lowest level failed goal. If there are no other failed goals, the up-checking proceeds one

level higher in the tree. There, GOTRES checks the goals on the same branch and level as the second to the last failed goal. GOTRES program keeps track of all goals that have been checked and their respective results. If the up-checking process reveals a new failed goal, GOTRES will again trace downward to find the initiating cause of the failure (that is failed hardware or conditions). When the cause of failure is found, GOTRES will resume up-checking again. The process of tracing down the tree and up-checking insures that all causes of failure will be detected. This diagnostic process is completed only when the up checking reaches the top level goal. A schematic of this process is provided in Figure 2. Section 4 explains, in more detail, how this problem solving method is programmed and implemented in GOTRES.

The selection of this diagnostic process as opposed to using other search processes such as breadth first search will depend on the shape of the GTST structure. GOTRES approach is ideal for performing diagnostics in GTSTs that are broad (see Fig. 3a). It is recognized, however, that the GOTRES approach will not be the most efficient for other possible structures. For a GTST having more goal levels than breadth at any cross section of the tree (see Fig. 3b), breadth-first or other methodsof search could be faster. For large trees, a portion of which is narrow and deep (see Fig. 3c), a combination of breadth

and depth search could provide a quicker inspection.

4. Programming Structure

GOTRES is written in LISP. The LISP programming language is selected because of its special features; namely, its support of: list structure, procedure concepts, recursion, and logic primitives [16]. The diagnostic process in GOTRES expert program is written as procedures. A pop-up menu is provided in GOTRES, which provides the ability to perform diagnosis or enter new knowledge. The diagnostics process is initiated when the procedure name "Perform-Diagnostics" is returned by the user. This procedure in the LISP program is defined as follows:

```
(DEFUN Perform-Diagnostics
       (SEND *TERMINALIO* :CLEARSCREEN)
       (EXAM))
```

The first step in this procedure clears away the pop-up menu. Next, it calls up the "EXAM". Neither of these procedures here takes any argument.

The "EXAM" procedure is the executive procedure in GOTRES. In the flow structure of the GOTRES program, "EXAM" serves as

the staring procedure (see Fig. 4). The defined operation of this procedure includes initiating the global lists used in GOTRES to empty lists, and calls on the procedure "CHECK". The procedure is defined as follows:

```
(DEFUN EXAM
      (PROGN (SETF *FAILWARE* NIL)
             (SETF *FAIL-GOALS* NIL)
             (SETF *SUCCESS-GOALS* NIL)
             (SETF *FAIL-RULES* NIL)
             (CHECK GOAL1)))
```

Global lists are those that can be accessed from any procedure in the entire program. Four global lists are defined in GOTRES they are:

FAIL-GOALS

SUCCESS-GOALS

FAIL-RULES

FAILWARE

These lists are used to store diagnostic information. During the diagnostic process, GOTRES checks goals and records the results by "APPEND"ing the names of each checked goal into the perspective lists. Likewise, rules that have failed and items that have initiated the lowest failed goals, are registered into

their lists. Through these lists, the diagnostic process can recognize goals that have been checked; and the user can receive advice on what has failed and what rules were used in reaching a conclusion.

"CHECK" (see Fig. 5) is the only procedure called by "EXAM". "CHECK" calls on several procedures to examine whether a goal is successful. These procedures use the success requirements in a goal's frame. The requirements can be either instrument readings required to be in certain ranges or questions made based on success requirement for a goal that can be answered by the users.

The procedure "TEST" performs the required instrument reading tests by comparing the current values against ranges of value that indicate goal success (see Fig. 6). If the results indicate that a goal is successful then the goal name is added into the *SUCCESS-GOALS* list; and "CHECK" is called on to check the other goals on the same level of the goal tree branch. If the goal is not successful then it is added to the *FAIL-GOALS* list; and a procedure, "CHECK-END", is called to perform depth search. The depth search terminates at the lowest level in the goal tree where the items that have initiated that failure path are identified.

"CHECK-END" is used to determine if the lowest level subgoal has been reached (see Fig. 7). One indication of a lowest level goal is the subgoal value in a goal's frame. The lowest level goal will have a subgoal value of "NIL". If the examining process has not reached the lowest level, then the subgoals of the failed goal are extracted from the goal's frame, and "CHECK" is called to examine them. "CHECK-END" finds whether the lowest level has been reached. If it has, in which case, it adds the failed item to the *FAILWARE* list, and calls on "UP-CHECK".

As the name implies, "UP-CHECK" carries the checking process upward through the goal tree (see Fig. 8). It checks the remaining goals that are on the same level and in the same branch as the previously known failed goal. To do this, it uses the procedure "CHECK". This begins at the lowest level using the last failed goal as its starting point. The procedure "ON-SAME-LEVEL" is used to collect a list of the goals on the same level. Then the procedure "EVALUATE" inspect each element of the list (see Fig. 9).

"EVALUATE" inspects goals that have not been examine by the "CHECK" procedure. Unchecked goals are not members of either *FAIL-GOALS* or *SUCCESS-GOALS* lists. If a goal is not successful, "CHECK" will again trace down the goal tree and find

the initiating failed item. Then, "CHECK-END" will again call "UP-CHECK", which will take the most current failed goal as the new starting point for further search.

Each time "CHECK" is called, it will check the goals that have not been tested. If there is a failed goal, it will trace the failed goals down the goal tree to the lowest level. Whenever "CHECK-END" finds that the lowest level failed goal has been reached, it will remember the initiating items (hardware, conditions, etc.) that caused the failure and calls "UP-CHECK". "UP-CHECK" will check the remaining goals. If there are no other failed goal on that branch level, the procedure will go up to the next higher level of the goal tree. Finally, the diagnosis for failure is completed when "UP-CHECK" reaches the top goal.

REFERENCES

[1] L. F. Pau, "Survey of Expert Systems for Fault Detection, Test Generation and Maintenance", Expert Systems, Vol.1, April 1986. pp100-111.

[2] D. A. Waterman, A Guide to Expert Systems, (Addison-Wesley),1986

[3] J. S. Kowalik, Knowledge Based Problem Solving, (Prentice-Hall), 1986.

[4] B. Chandrasekaran, S. Mittal, "Deep Versus Compiled Knowledge Approaches to Diagnostic Problem Solving", Proceeding of AAAI-82,1982, pp.349-354.

[5] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M. Shirley, S. Polit, "Diagnosis Based On Description of Structure And Function", Proceeding of AAAI-82, 1982, pp.137-142.

[6] W. Hamscher, "Using Structural and Functional Information In Diagnostic Design", Proceeding AAAI-83, 1983, pp.152-156.

[7] R. Milne, "Fault Diagnosis Through Responsibility", Proc. 9th IJCAI, 1985, pp.423-425

[8] T. J. Laffey, W. A. Perkin, T. A. Nguyen, "Reasoning About Fault Diagnosis with LES", IEEE Expert, Vol.1, Spring 1986. pp.13-20.

[9] R. Davis, "Reasoning From First Principles In Electronic Trouble shooting", Int. J. Man-Machine Studies, 1983, Vol.19,pp.402-423.

[10] M. R. Genesereth, "Diagnosis Using Hierarchical Design Methods", Proc. AAAI-82, August 1982, pp.278-283.

[11] M. A. Kramer, B. L. Palowitch Jr., "Expert System and Knowledge Based Approaches To Process Malfunction Diagnosis", AICHE National Meeting, Chicago, Nov. 1985.

[12] T. Cadman, M. Modarres, "A Method of Alarm System Analysis In Process Plants With The Aid of An Expert Computer System", presented for publication to Computer & Chemical Engineering Journal,Sept. 1985.

[13] R. N. Hunt, M. Modarres, M. Roush, "Application of Goal Trees to Evaluation of The Impact of Information Upon Plant Availability", Proc. ANS/ENS Topic Meeting on Probabilistic Safety Methods and Applications, San Francisco, Ca, Feb. 1985.

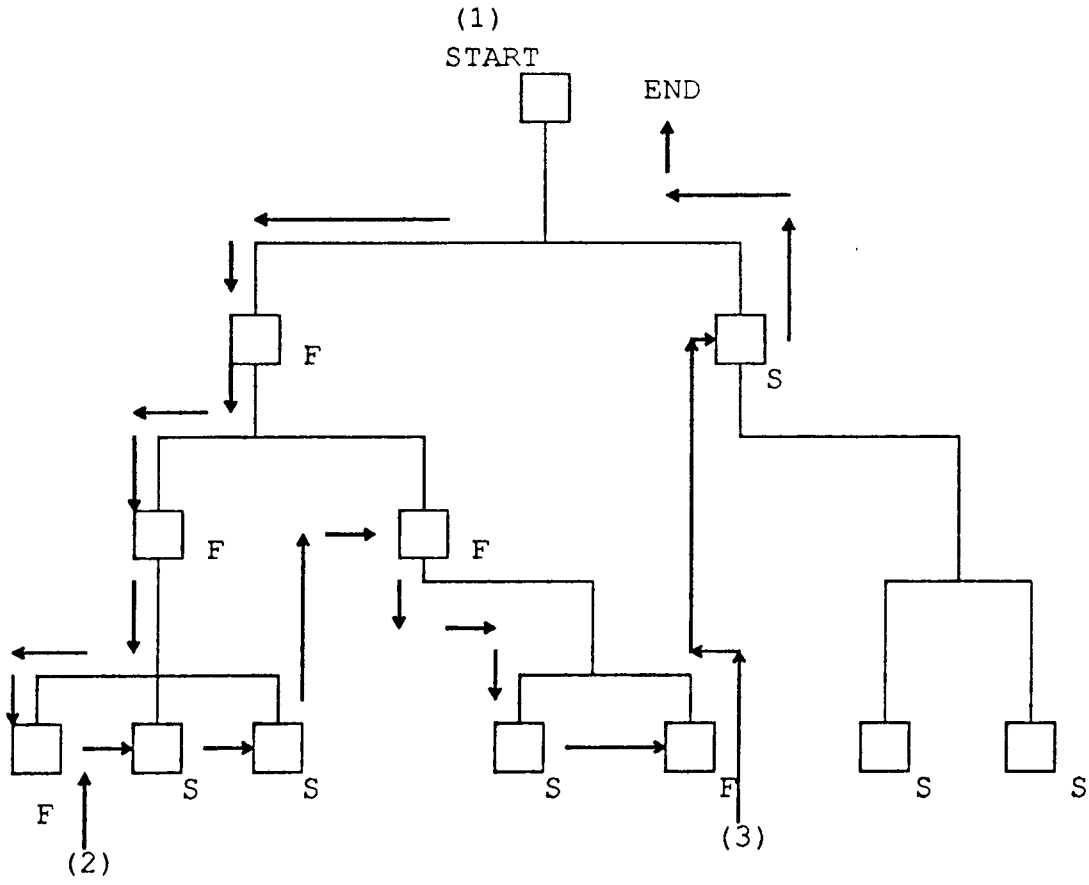
[14] P.H. Winston, Artificial Intelligence (Addison-Wesley),Reading, Massachusetts, 1983.

[15] P. H. Winston and B. K. Horn; LISP (Second Edition) Addison-Wesley, Reading, Massachusetts, 1984.

[16] Chadwick, Michael and Hannah, John; Expert Systems For Microcomputers, Sigma Press, 1987.p42

```
 (GOAL-1 (GOAL-PATH-NAME (VALUE 1))
     (GOAL-NAME (VALUE "Failure of Pumping System Prevented"))
     (GOAL-PARENT (VALUE NIL))
     (SUB-GOAL (VALUE GOAL-2 GOAL-3 GOAL-4 GOAL-5))
     (SUCCESS-REQ (VALUE))
     (GOAL-HARDWARE (VALUE ))
     (RULES (rule1 "If the system parameters are in limit,
then failure is prevented."
 "Are all other parameters (other thatn Q and H) in the limits?"
 "Does prime mover power match water power in the vicinity of
BEP?")))
```

Figure 1. Frame representation of the attributes of a goal using embedded lists.



1. The search process traces down a cause of failure.

2. The process up-checks and traces down a second cause of failure.

3. The process returns to up-checking and terminates at the top level goal.

Figure 2. The flow of the search process used in GOTRES.

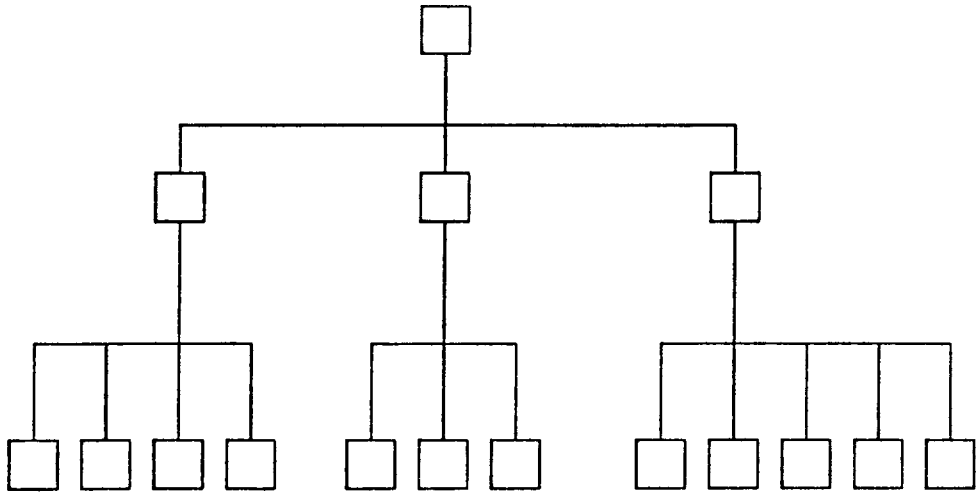


Figure 3a. An example of a broad tree

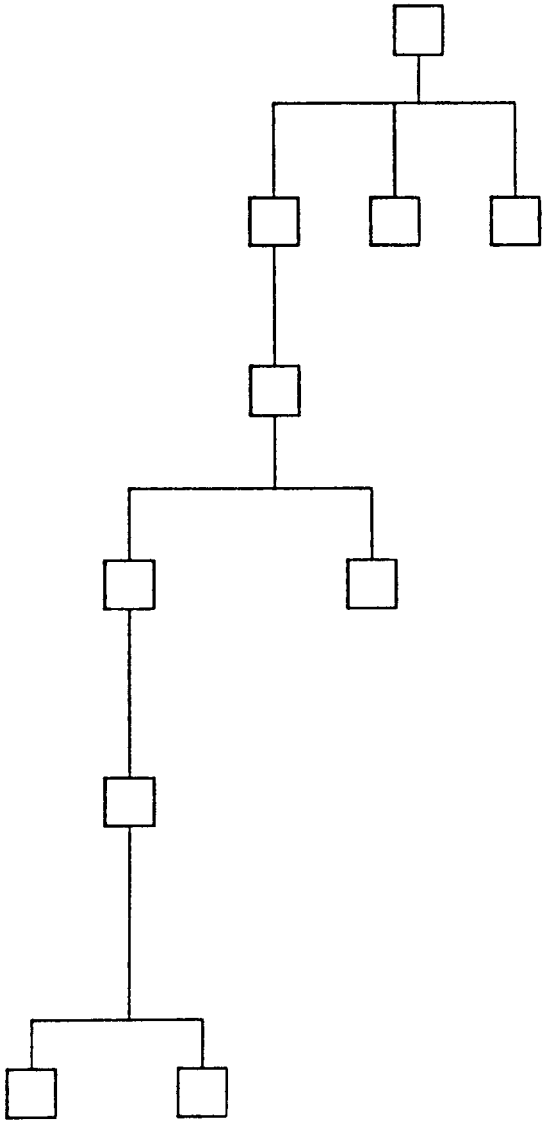


Figure 3b. An example of a tree that has more levels than breadth.

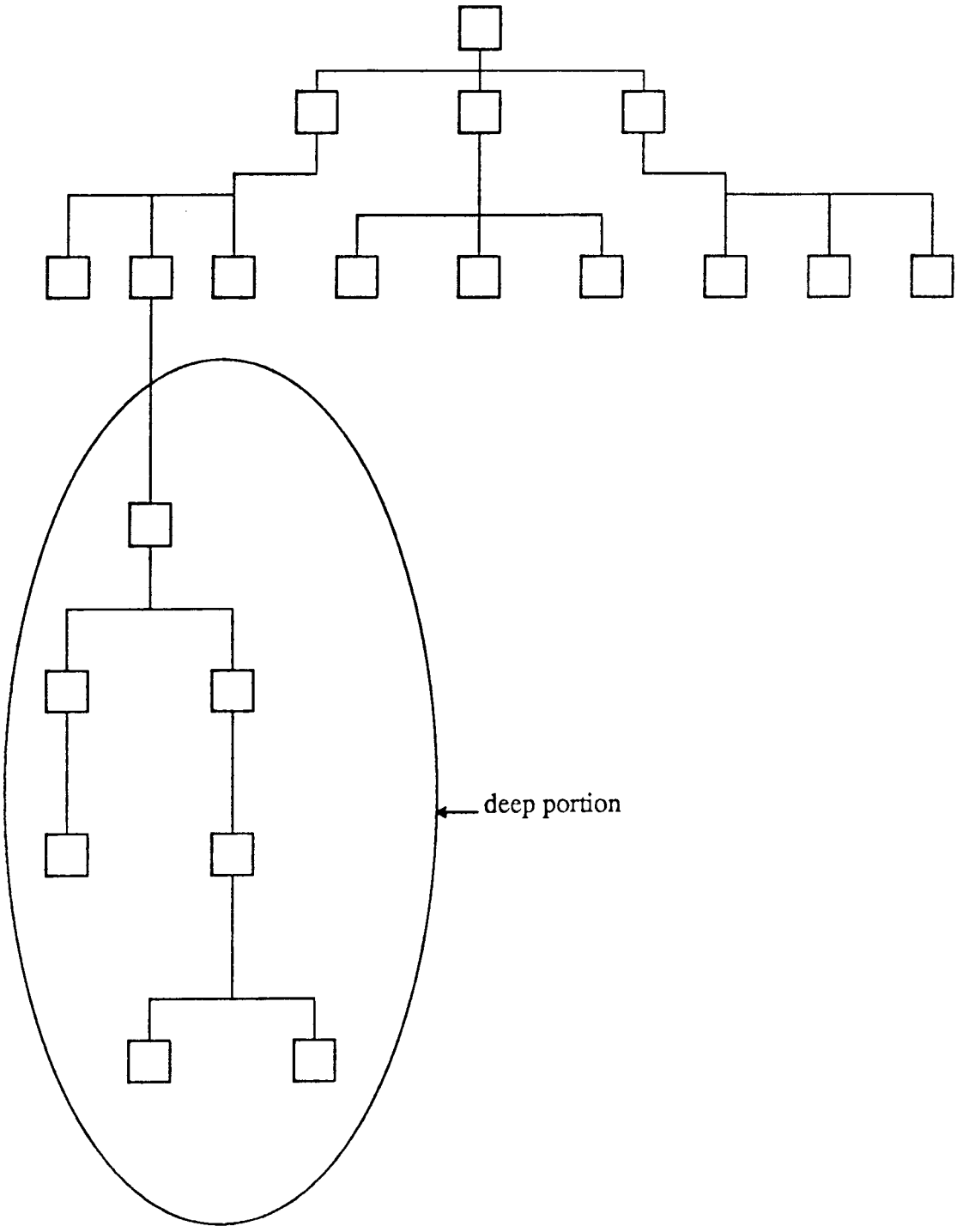


Figure 3c. An example of a generally broad tree which has a deep portion.

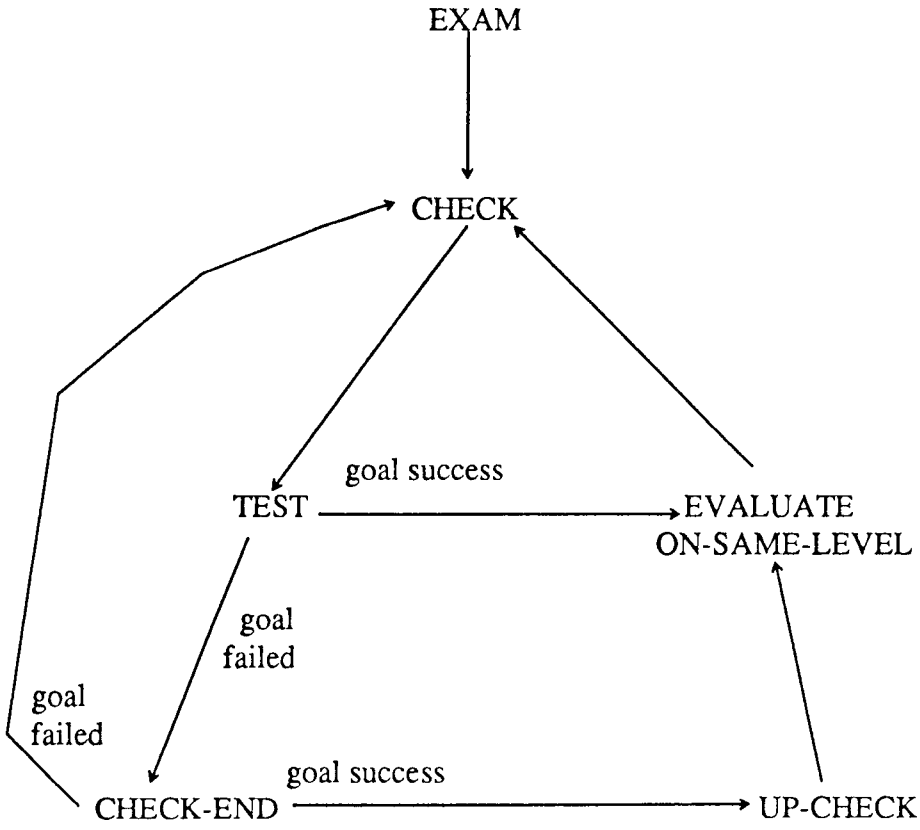


Figure 4. The control structure of GOTRES expert shell.

```
(DEFUN CHECK (GOAL)
        (COND   ((NULL GOAL) 'T)
                ((TEST GOAL) (EVALUATE (ON-SAME-LEVEL GOAL)))
                ((RULE-TEST  GOAL) (EVALUATE (ON-SAME-LEVEL GOAL)))
                ((CHECK-END GOAL) 'NIL)
                 (('T  (CHECK (EVAL (CAR (SUB-GOAL GOAL))))))))
;If there are no more goals, then T is returned.
;There are two level of evaluation for each goal.
;If a goal fails TEST, then it will be evaluated by RULE-TEST.
;If a goal is successful, then goals on the same level
;are evaluated.
;If a goal is not successful, then it's subgoals are evaluated.
```

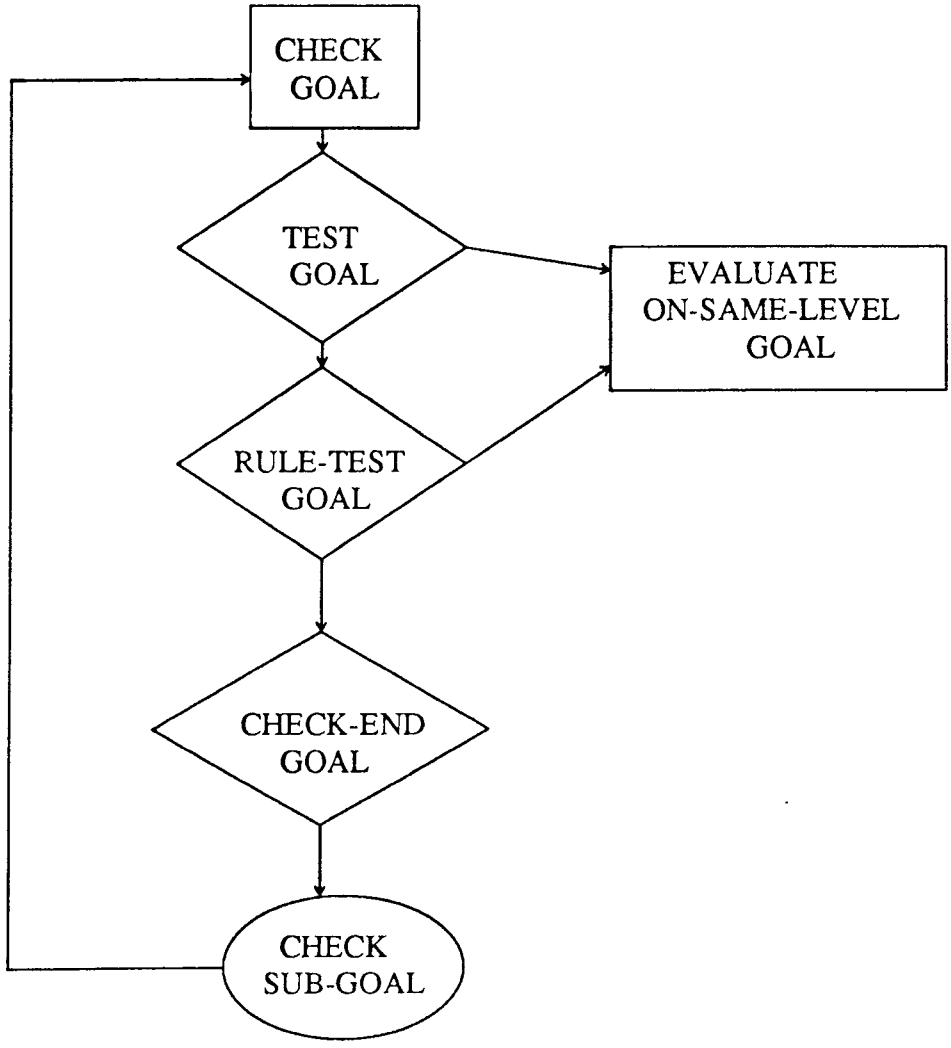


Figure 5. Definition of the procedure "CHECK", and the flowchart for "CHECK".

```
(DEFUN TEST (GOAL)
    (SETF GOAL-INSTRUMENT-LIST (INSTRUMENT GOAL))
     (COND ((NULL GOAL-INSTRUMENT-LIST) NIL)
           ((EVERY CHECK-INSTRUMENT GOAL-INSTRUMENT-LIST)
            (REMEMBER-SUCCESS GOAL))
          (T 'NIL)))
```

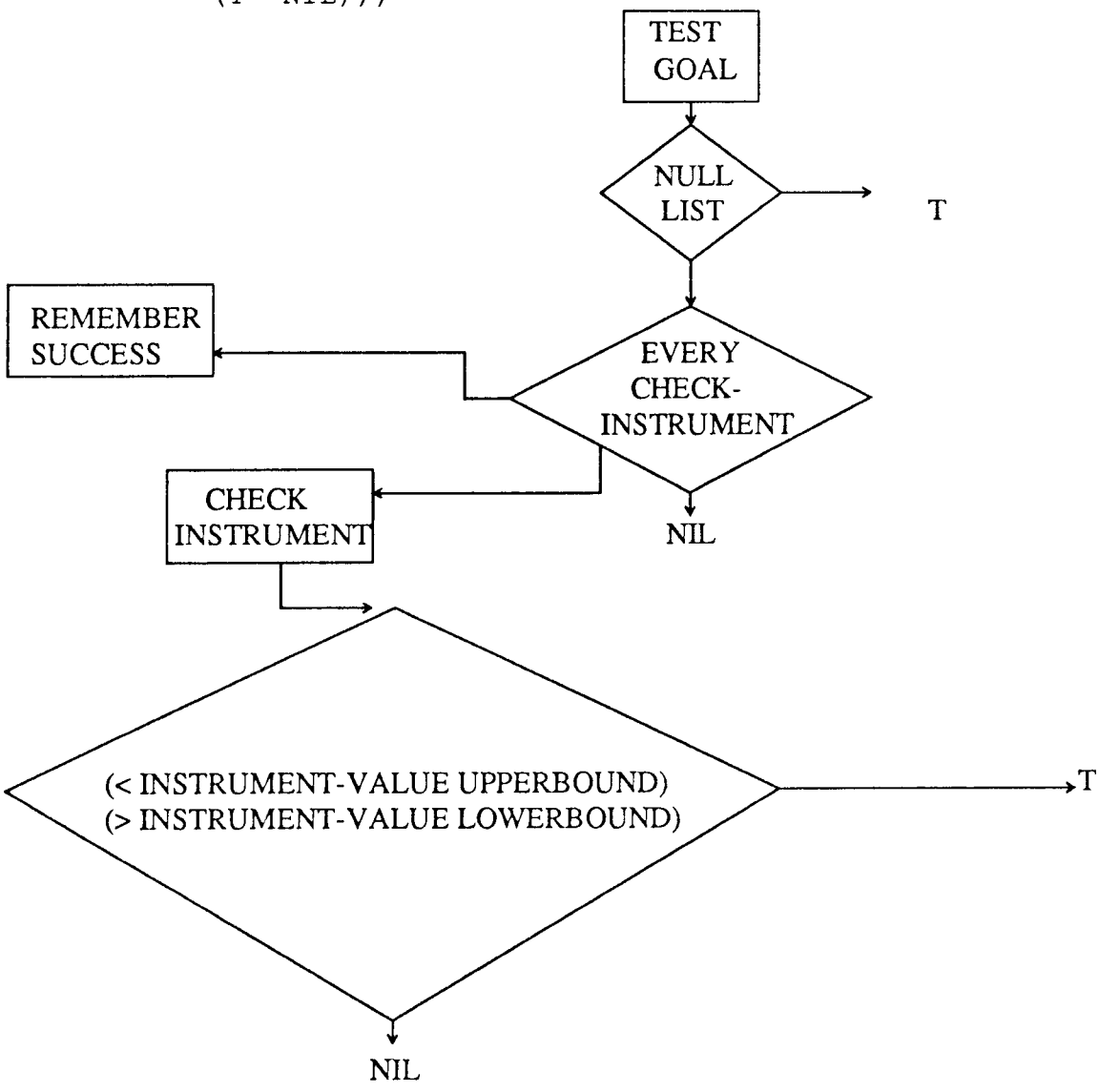


Figure 6. Definition of the procedure "TEST" and its flowchart.

```
(DEFUN CHECK-END (GOAL)
              (SETQ DEPENDENTWARE (HARDWARE GOAL))
              (COND ((NULL DEPENDENTWARE) NIL)
               ((REMEMBER-FAIL-ITEM GOAL)(UP-CHECK GOAL))))
;If ther are no dependentware, then it is not the lowest level.
;If it is the lowest level, then remember fail items and
;"UP-CHECK"
```

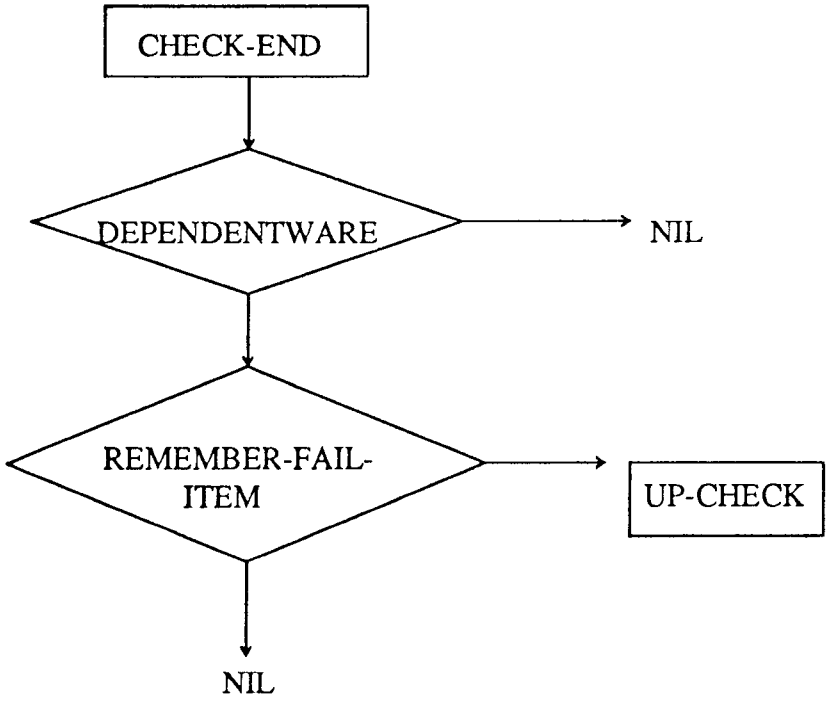


Figure 7. Definition of the procedure "CHECK-END" and its flowchart.

```
(DEFUN UP-CHECK (GOAL)
              (COND ((CHECK-TOP GOAL) 'T)
              ((EVALUATE (ON-SAME-LEVEL GOAL))
                           (UP-CHECK (EVAL (CAR (PARENT GOAL)))))))

;if the top goal has not been reach, them evaluate the goal
;that are on the same level as this goal.  If they all test
;successfully, then check if their parent goal is the top goal.
```

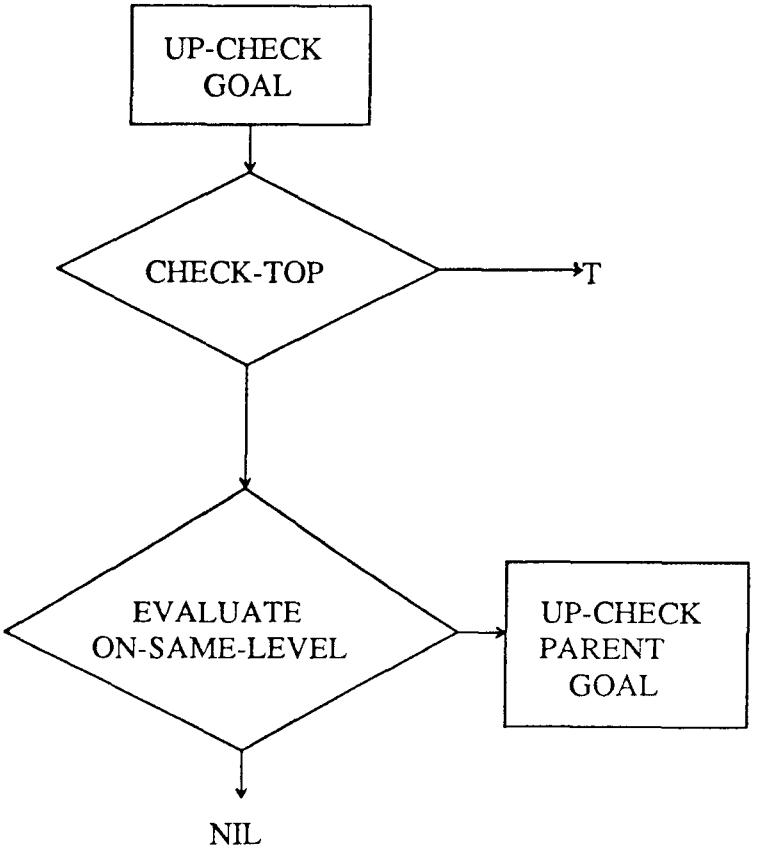


Figure 8. The definition of the procedure "UP-CHECK" and its flowchart.

```
(DEFUN EVALUATE (GOAL-LST)
               (COND ((NULL GOAL-LIST)  'T)
               ((OR (MEMBER1 (CAR GOAL-LIST)  *SUCCESS-GOALS*)
                    (MEMBER1 (CAR GOAL-LIST)  *FAIL-GOALS))
                             (EVALUATE (CDR GOAL-LIST)))
                             ('T (CHECK (EVAL (CAR GOAL-LIST))))) )
;if the goal list is exhausted without finding any failed goal,
;then true is returned.
;if the first atom of the list is a member of either success
;list or failed list, then it has been
;checked.  In which case, recursion is used to evaluate the
;remaining atoms of goal list.
;if an atom is not a member of either list, then it has not
;been checked.  In which case,
;it is passed to the procedure "CHECK".
```

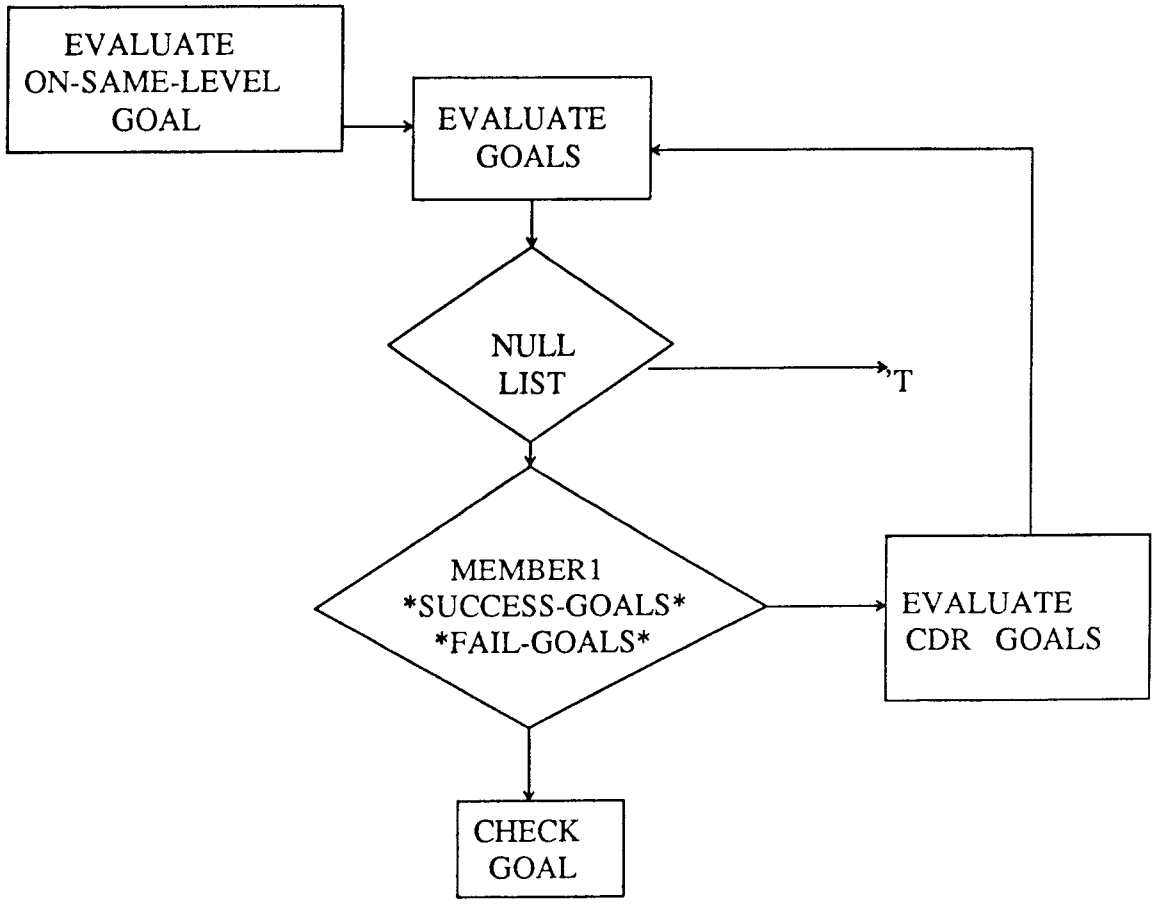


Figure 9. Definition of the procedure "EVALUATE" and its flowchart.