FAULT FINDER
A PC BASED Expert System

W. Elliott, Staff Engineer
HARRIS Air Traffic Control
Systems Division

Dr. M. Schneider
Computer Science Department
Florida Institute of Technology

ABSTRACT

The FAULT FINDER Expert System implements fault isolation decisions for any target system or equipment that can be modeled by lowest replaceable units (hereafter called LRUs). The term "Target System" will be used to refer to the system being fault isolated. The Fault Finder expert system fault isolates the target system's LRUs. This expert system utilizes a data base to represent each LRU, a status interface to obtain LRU status, and a knowledge base to store the rules of fault isolation for the target system. The expert system has multiple "learning" capabilities in the data base, the knowledge base and the inference procedure. Another aspect of learning which influences the structure of the knowledge base is that each rule has parameters associated with it to store the information learned as a result of user feedback and the inference process. The certainty or possibility associated with the conclusion of each rule is adjusted as the system runs and gains experience. The inference procedure uses fuzzy logic for premise matching certainty, and combining of premise certainties for the rule firing certainty. This expert system brings together for the first time a fault isolation system with unique knowledge representation, inference processing, fuzzy logic, and multiple learning capabilities in one design. Also presented are issues of knowledge structure, and possible types of fault isolation knowledge.

INTRODUCTION

The FAULT FINDER is an expert system designed to implement fault isolation decisions for a target system using fuzzy logic [1][5] and learning techniques. One of the problems of today's systems is the ever more complex knowledge, experience and training needed to operate, maintain and service these systems. Most systems have fault isolation software and sensor hardware that find faults in a fixed, non-adaptable way. This fixed logic is built into the system before field experience is gained. Each new system that is designed must have custom logic that will fault isolate its components and topology. Thus, engineers must reinvent each new system's logic for fault isolation [2].

The FAULT FINDER Expert System implements fault isolation decisions for any target system or equipment that can be modeled by lowest replaceable units. The FAULT FINDER system is configurable to any target system's components and topology. The target system expert enters the topology in the data base and enters the fault isolation rules in the knowledge base.

The FAULT FINDER prototype is written in TURBO PASCAL [4] using a PC/AT compatible. TURBO Pascal [4] has numerous performance advantages in the PC environment. TURBO Pascal [4] (Ver. 4.0) allows separate code units which extends the code size limit, and by using dynamic memory the data limit can be extended to 64K bytes per data structure. This allowed the goal of a least 250 rules to be achieved.

Design Overview

The FAULT FINDER prototype implementation is designed to be a technicians helper and is based stand alone. This can be on any small computer such as a PC/AT. The only interface is through the Man Machine Interface (MMI) of the CRT. The technician may use the system as a consultant, and a library to store knowledge learned through experience.

An optional Status Interface allows the FAULT FINDER to request real time status as part of the inference process. In a full scale real time implementation, the FAULT FINDER may be set up to monitor the health of the target system or test assemblies as a final step in a production process.

Top Level Design

The system is divided into the Main Control module, the LRU Editor module and Data Base, the Rule Editor module and Knowledge Base, the Explanation module, the Real Time Status interface, and the Inference Procedure built around the expert system shell. When designing the knowledge base and database structures and the access environment the following objectives were set.

1. The knowledge base needed to accomodate at least 250 rules and knowledge base must be retained when the execution of the FAULT FINDER is terminated, and restored when the FAULT FINDER is started.

2. The knowledge base must be named (target system name) to allow multiple knowledge bases to be stored and retrieved.

3. The LRU database must be linked to the knowledge base by the same name.

4. The system response time must be fast when a question is asked (fault isolation time).

5. The data base and the knowledge base must be editable, and reports must be available to review data and rules and must contain data structures to support learning.

The user interface for the FAULT FINDER is menu driven. The system is started by typing "FAULT" on the PC. The startup menu asks for the keyword or name of the target system to be fault isolated. This keyword is used to retrieve, or create for the first time the data and knowledge base files for the target system. Next the Fault Finder retrieves the password and is ready for verification. When the correct password is entered the main menu appears as shown in figure 1.

The LRU data, Status data and Knowledge (rules) may be entered in any order. However, experience has shown that the best or most effective order is to enter the LRUs first, followed by the items for the status interface, then the rules. In this way the terms or arguments referred to in the rules are established first.

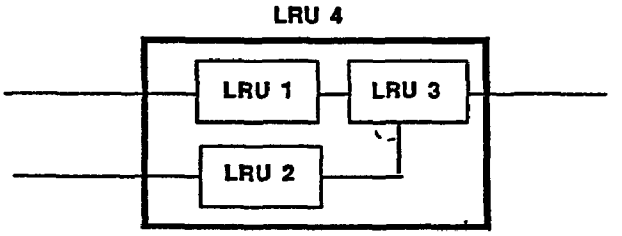
The expert system shell which utilizes fuzzy set theory and fuzzy logic [5] was designed and written for the FAULT FINDER expert system. However the shell itself is general and could be used for any other expert system application. The inference procedure is described in a later section.

Data Base Structure

The Data Base design centers around the LRU concept. Any level of component in the target system may be designated an LRU. Also LRUs may contain other LRUs. Example 1 shows a set of LRUs. In this example three LRUs make up a larger LRU boundary. The knowledge base contains the rules for the interrelationships of LRUs.

Choosing the best LRU boundaries is somewhat of an art, but for most system designs it is usually at the printed circuit card or assembly level where clear functional and replaceable boundaries exist.

Example 1: A Set of LRUs and their connectivity.



Set of LRUs

The basic structure of the database is represented as follows:

```
LRU_State_Type =(Operational,Degraded,Failed,
        Offline_Standby,Out_of_Service);

LRU_Record_Type = Record
        LRU_Name  : String[25];
        LRU_Number: String[25];
        LRU_State : LRU_State_Type;
        Failures  : integer; {0 to Maxint}
        Fail_Rate : Real; { 0 to 1 }
        Test_Avail: Boolean;
        Test_Conf : real; { 0 to 1 }
        Test_State: LRU_State_Type;
        End; {record}

DataBase : Array[1..Number_of_LRUs] of
        LRU_Record_type;
```

One of the learning functions of the FAULT FINDER uses the LRU failure rates stored in the LRU database (Failures). This function uses a fuzzy set (defined below), and produces a number range [0..1] (Fail_Rate) that is used by the inference procedure when evaluating the conclusion certainty.

The function is as follows:

let **A** be a fuzzy set "failed LRUs",and **Failures** be a member of **A**

also let **Max_Fail** be the largest LRU failure in the data base

Then:

```
Fail_Rate
    := 0,                    if the LRU has not failed
    := Failures/Max_Fail,    if 0< Failures< Max_Fail
    := 1,                    if Failures = Max_Fail
```

```
Fault Isolation Expert System  -  B. Elliott, M. Schneider
Main Menu      -  F A U L T   F I N D E R    -  Version:  1004, 1989

    Select one of the following options and enter <cr>:
        0  - Rule Editor, Max # rules allowed = 250 current rule cnt = 43
        1  - Start Fault Isolation with Blackboard Data
        2  - Explanation Module
        3  - Status Interface,    No. Status Elements = 17
        4  - Review the Blackboard Data
        5  - LRU Editor, Max # of LRUs= 50 Current LRU cnt= 22
        6  - Enter Data to be placed onto the Blackboard
        7  - Clear the Blackboard
 Target System: commsys                  8  - Exit The Program
```

figure 1.

Example 2: In this example the FAULT FINDER had 5 LRUs defined in the LRU data base. After a conclusion was reached the learning function described above was allowed to run. For this example the conclusion was that LRU# 1 was failed. This is the LRU data base before the learning function is run.

LRU#	Failures	Fail Rate
1	5	0.71
2	1	0.14
3	0	0.00
4	7	1.00
5	2	0.29

The first thing that is done is the number of failures for LRU# 1 is increased by one, then the function is applied to each LRU record in the data base. The result is shown below.

LRU#	Failures	Fail Rate
1	6	0.85
2	1	0.14
3	0	0.00
4	7	1.00
5	2	0.29

THE LRU EDITOR

The LRU Editor is accessed by selecting choice 5 on the main menu. The user may enter LRU records into the LRU Data Base, print the LRU Data Base report, delete LRU records, or exit.

Status Interface

The Status interface allows the Fault Finder to access real time status from the target system. Each status element has a name and a type of status, boolean or real counter. The structure of the status database is represented as follows:

```
Status_record = record
          Elm_Used  : Boolean;
          Elm_Name  : ident;
          Elm_type  : integer;{ 0 = cnt,
                                1 = boolean}
          Fault     : boolean;
          Fail_cnt  : Real; { 0 to intmax }
          End; {record}

Status_type = Array [1..Elm_Max] of Status_record;
```

The Status interface database Editor is accessed by selecting choice 3 on the main menu. This menu allows the user to enter Status records, print a Status Data Base report, delete Status records, modify status records, or move the status to the blackboard for inference process access.

Knowledge Base Structure

The Knowledge Base is constructed of production rules of the form: IF <premise> THEN <conclusion>. The premise and conclusion use the following grammar:

[the]KW1[of[the]KW2]is[not][modifier]{[KW3][number] [number to number]}

where KW1,KW2 and KW3 are keywords,
[ ] means optional and { } means one choice is required,
number may be a real.

let p be the premise of a rule and c be the conclusion, then using the BNF notation we have:

<p> = p | p or p | p and p | not p | (p)
<c> = c | c and c | not c | (c)

It should be noted that conclusions have different BNF forms than premises. This is because of the nature of a conclusion. If an "or" were allowed in a conclusion, how would the inference process decide which clause to use? The "or" in a conclusion is ambiguous, and therefor is not allowed.

Each rule has an expert user assigned certainty value range [0..1] that describes the user's belief in the truthfulness of the conclusion at the time the rule was entered. This user assigned number is not changed by the system, but is used in the learning function with other parameters to produce the final certainty in the conclusion.

The knowledge of fault isolation can be divided into the major areas of internal LRU design knowledge and external LRU or target system knowledge [6]. Internal knowledge is design knowledge of the LRU details. This is the detail design knowledge that only the design engineer may know of how each component is interrelated within the LRU. External knowledge is interface knowledge of the LRU, without knowing the internal design implementation. The knowledge base of the FAULT FINDER [7] is composed of rules of "system knowledge" or "shallow knowledge". An example of system knowledge is, "If two inputs are present and no output is present then the LRU is faulted". Most technicians or support engineers servicing system sites do not have the deep knowledge of how each card (LRU) works or fails. Only the designer may know this. But this deep knowledge is not needed to fault isolate and replace the failed LRU. Shallow knowledge is not 100% correct in all cases, but a substantial amount of system maintainence can be done with shallow knowledge.

Each conclusion is assigned a real number, range [0..1], derived in the inference process, which represents the possibility that this conclusion is correct.

As shown in figure 2 the knowledge base is made up of three major data structure types; the Rule_Pointer_Array, the Rule_Record, and the Clause_Record. The Rule_Pointer_Array points to rule records that are dynamically allocated as needed using the 'NEW' function of Pascal. The array pointers are set to nil unless a valid rule is pointed to. The Premise_idx and Conclusion_idx both point to a Clause_Record which in turn may point to another Clause_Record. The last Clause_Record in a chain has a zero forward index. The grammar defined previously allowed the Premise and Conclusions to have the forms:

<p> = p | p or p | p and p | not p | (p)
<c> = c | c and c | not c | (c)

These forms are supported by stringing the clauses together using the Relation structures to store the type of linkage between the clauses.

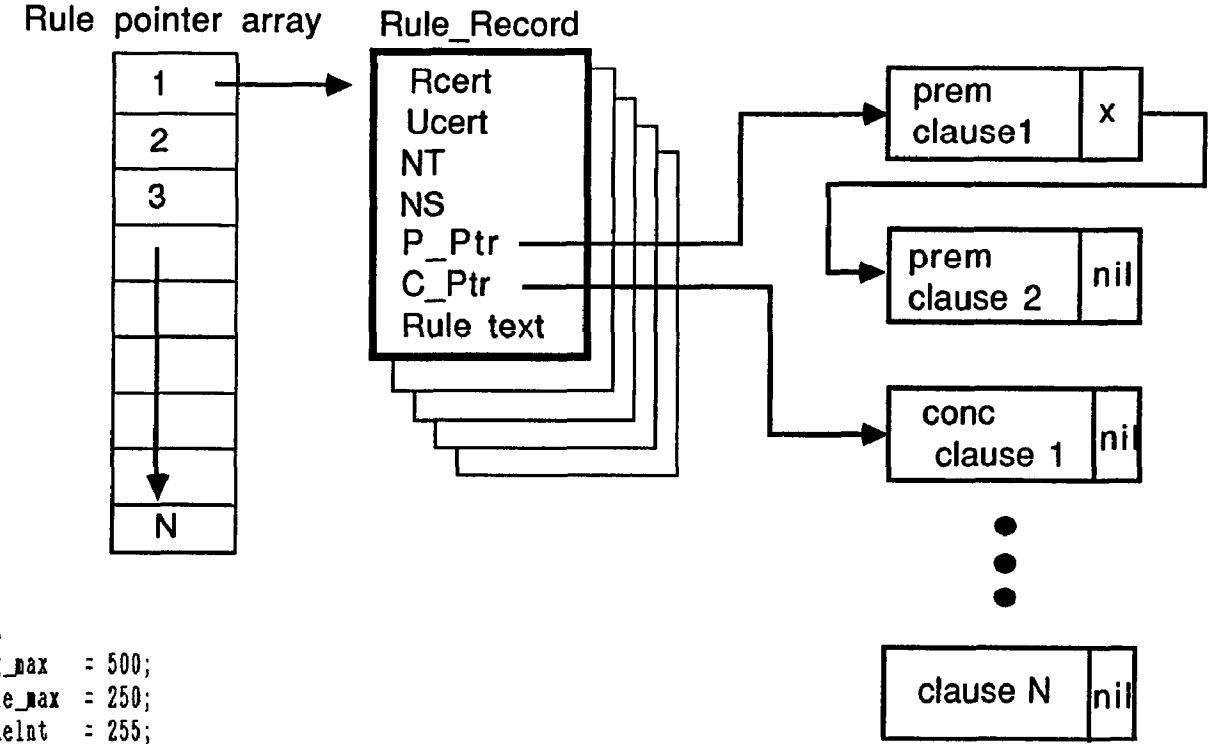
The Knowledge Base Editor is accessed from the main menu by choice 0. The editor prompts for the rule certainty. If a rule has a syntax error, the editor protects the knowledge base from this rule by deleting it. The knowledge base and the bit matrix [5] are initialized when the editor is exited. The bit matrix is a data structure used by the inference process in the direct chaining of rules. The bit matrix is built to store information concerning the interrelationships of rules. The bit matrix (BM) is a N by N matrix where N = the number of rules, and is defined as follows:

BM [I,J] = **1** if rule I may participate in the firing of rule **J**,
0 otherwise

The bit matrix is initialized to all zeros. Next each rule is compared to all other rules in the knowledge base and the matrix is updated with the results of the comparison. If a rule has all zeros in its row then the rule is a concluding rule (a rule that fires no other rules). If a rule has all zeros in its column then the rule is fired by no other rule and requires user entered data to be fired. As shown below each modifier has a lower and an upper fuzzy value. In the grammar defined previously the modifier may be a single number, or a range "number to number".

Exactly	1.00	1.00
Almost	0.97	0.99
More_or_less	0.90	1.10
Nearly	0.95	0.99
Approximately	0.85	1.15
Around	0.85	1.15
About	0.85	1.15
Somewhat	0.80	1.20
Slightly	0.75	1.25
Barely	0.65	0.95

FAULT FINDER Knowledge Base




```
const
  Arg_max   = 500;
  Rule_max  = 250;
  linelnt   = 255;
  BlkBd_Max = 50;

type
  ident     = string[25];
  Relation_type = (null, znot, zor, zand,
                   open_Parn, Close_Parn, Clause);
  Statement_type = array[0..20] of Relation_type;

  rule_record_type = record
                     Ucert    : real;
                     Rcert    : real;
                     Fcert    : real;
                     NT       : integer;
                     NS       : integer;
                     Prem_idx : integer;
                     Prem_ord : Statement_type;
                     Conc_idx : integer;
                     Conc_ord : Statement_type;
                     Srule    : String[linelnt];
                     Conc_ptr : integer;
                     end;

  Rule_ptr = ^rule_record_type;
  rule_array = array [1..Rule_max] of rule_ptr;
  BlkBd_array = array [1..BlkBD_max] of rule_ptr;
  rule_state_type = (premise,conclusion);
```

```
S_Arg_type = record
             Arg_used      : boolean;
             KW1           : ident;
             KW2           : ident;
             Int_not       : boolean;
             Adj           : integer;
             KW3           : ident;
             Num_val1      : Real;
             Num_val2      : Real;
             Num1          : Real;
             Num2          : Real;
             forward_idx   : integer;
             end; { RECORD }

Args_type     = array [1..Arg_max] of S_Arg_type;
Arg_ptr_type  = ^args_type;
```

When the modifier is applied to KW3 the Fault Finder ignores the modifier. To see why this was done consider the following examples:

Example 3: The user types in a rule as follows:

If the fan is not active and the power is not failed then the source of the fault is almost LRU2

The adjective almost cannot be applied to LRU2 in a meaningful way.

Example 4: Now consider the following rule.

If the input is more_or_less 35 then LRU2 is not faulted.

The adjective more_or_less can be applied to 35 in a meaningful way. Using fuzzy matching [1] the number 35 is replaced with an interval defined as follows:

Lower fuzzy value * 35 to Upper fuzzy value * 35,

using the values of the fuzzy modifiers we have,

(0.90 * 35) to (1.10 * 35) = interval 31.5 to 38.5

As shown in example 3, the modifier should be ignored when a KW3 is entered. The modifier in example 4 however, may be applied to the number in a meaningful way. The values of the lower and upper modifier fuzzy values were selected based on the opinion of the author, and other values may be suitable for other applications.

KNOWLEDGE BASE LEARNING FUNCTION

The Knowledge Base learning function uses four parameters associated with each rule. These parameters are the expert's confidence in the conclusion (Ucert), the number of times the rule was tried (NT), the number of times the rule was successful (NS) and the output of a fuzzy function (Rcert). The fuzzy function uses the above parameters to calculate the possibility that the conclusion is correct as follows:

Rcert := Rcert - 0.5 * (Rcert - NS/NT)

The Rcert is initialized to the user certainty (Ucert) when the rule is first entered. The 0.5 factor is applied to cut the rate of change in the learning process. This factor helps stop large oscillations but allows multiple changes, through experience, to accumulate in a direction.

Example 5:
let Rcert = Ucert = 0.9 for the first run with a success, then

NS = 1, and NT = 1
Rcert := 0.9 - 0.5 * (0.9 - 1/1)
Rcert := 0.95

Next, a run with a failure
NS = 1, and NT = 2
Rcert := 0.95 - 0.5 * (0.95 - 1/2)
Rcert := 0.725

Then, another run with a failure
NS = 1, and NT = 3
Rcert := 0.725 - 0.5 * (0.725 - 1/3)
Rcert := 0.529

The learning functions are applied to the knowledge and data bases only after the user enters a 'yes or no' answer as to the conclusion's verified correctness.

The Blackboard

The blackboard menu is used to enter data for the fault finder to use in the inference process. The blackboard editor is entered from the main menu by choice 6. Blackboard data is entered in the same basic format as rule premises. The input data consists of statements as to the state of the target system, the interfaces, or other data that is target system dependent. The structure of the blackboard is very similar to the structure of the knowledge base. This allows the inference process to be more efficient in matching input data, premises, and conclusions.

Inference Procedure

The inference engine is responsible for inferring conclusions from a given knowledge base and user entered data. The process of inferencing involves matching the knowledge base with the data and producing conclusions [5]. The inference procedure uses fuzzy matching [1] for comparing input data, premises and conclusions and may be summerized in the following steps.

1. The conclusion of a rule becomes true if and only if the matching process generates results which are above or equal to a threshold of 0.6.

2. If the conclusion becomes true, then we "fire" that rule and place its conclusion on the blackboard, along with its firing certainty.

3. We repeat the process until no more rules can be fired.

There are two steps in finding the degree in which two ranges are matched. First it is necessary to find out the possible range of the variable in question and then utilize the matching procedure. When matching clauses, two cases must be considered:

1. A clause contains the keyword "NOT" and
2. A clause does not contains the keyword "NOT".

In the first case, the word "NOT" forces two intervals [7] to be created. In the case where the clause does not contain the keyword "NOT", there is one continuous interval. Therefore, when two clauses are compared, it must be assumed that there may be four intervals to consider.

The overall operation of the FAULT FINDER is shown in figure 3. The user defines the LRUs in the LRU data base, defines the status items in the status interface and the rules of the target system in the knowledge base. Next the user enters input data onto the blackboard, and/or requests status to be moved to the blackboard, and starts the inference procedure. The inference procedure produces one or more conclusions with decreasing certainty. The conclusion with the highest certainty is printed as "the conclusion", but the user through the use of the explanation module may review all of the conclusions produced. The user then tries the suggested conclusion, and provides feedback to the FAULT FINDER for the learning functions.

The inference procedure process flow for the FAULT FINDER is shown in figure 4. The inference procedure is based on direct chaining [1] which uses the bit matrix, the blackboard, a list of fired rules and a list of fired conclusions. The modifiers and internal not's are applied at the time of input. The starting point of the inference procedure uses the rules in the knowledge base, and the data on the blackboard.

The inference procedure starts by matching each rule's premises against the blackboard data. At the start of each rule matching process a match matrix of fuzzy values for each clause in the premise is initialized to 0.5. The reason for this is if a clause is not on the blackboard there is a 0.5 certainty that it was a correct assumption. Boolean matching is used for KW1, KW2, KW3, and the internal NOT if KW3 is a word, fuzzy matching [1] is used for all numbers and intervals. In the case of a number the internal not was applied to the interval at the time of input. If a match is greater than 0.6 then the following function is used to update the match matrix.

```
Match_Matrix( Clause# ) =
      (match_value + BBdata.Fcert) / 2
```

The final matching value for a premise is achieved by combining the individual clauses of a premise using fuzzy ands, ors, and nots. If the match value is greater than 0.6 then the rule is fired. When a rule is fired the blackboard must be updated. The inference procedure blackboard updating process uses the Ucert and Rcert values for each rule and a premise matching certainty [1] as it fires a rule to accumulate a final firing certainty (Fcert) for that rule. The fuzzy function for handling firing certainty for the blackboard update is as follows:

```
Temp_result =
   (Result + Rule(x).Ucert + Rule(x).Rcert) / 3;
Rule(x).Fcert = Temp_result;
```

This value is stored in the knowledge base with the rule for later reference. If the rule ends up being the best concluding rule this composite conclusion certainty (Fcert) is the final conclusion certainty. The blackboard is updated by copying the fired rules' conclusion to the blackboard and the following values:

```
BB.Ucert = Rule(x).Rcert;
BB.Fcert = Temp_Result;
```

The final composite conclusion certainty (Fcert) and the LRU Fail_Rate (Fail_Rate) values, from the LRU data base, are used in calculating the FAULT FINDER's final conclusion certainty (FFcert). After the inference process has terminated, one or multiple concluding rules may have been fired. The explanation module will allow the user to access the secondary and subsequent conclusions if needed. These subsequent conclusions may contain additional faults with a lower certainty, as compared to the primary conclusion. The conclusion to be suggested first and its certainty could be calculated by just averaging the Fcert and Fail_Rate values. Another approach is to define classes of conclusion certainty using the Fcert and Fail_Rate values. If we assign points to ranges of certainty then a fuzzy function can be constructed as follows:

Define the following certainty ranges for fail_rate values: High, Med., Low

where

0.7 <= High <= 1.0, and High Certainty = 3 points
0.3 <= Med. < 0.7, and Med. Certainty = 2 points
0.0 <= Low < 0.3, and Low Certainty = 1 point

and, define the following ranges for rule firing certainty: High, Med., Low

where

0.9 <= High <= 1.0, and High Certainty = 3 points
0.7 <= Med. < 0.9, and Med. Certainty = 2 points
0.6 <= Low < 0.7, and Low Certainty = 1 point

Now a table of combinations can be constructed:

Fcert	Fail Rate	Points	Class
High	High	6	1
High	Med.	5	2
High	Low	4	5
Med.	High	5	3
Med.	Med.	4	4
Med.	Low	3	7
Low	High	4	6
Low	Med.	3	8
Low	Low	2	9

Table 1

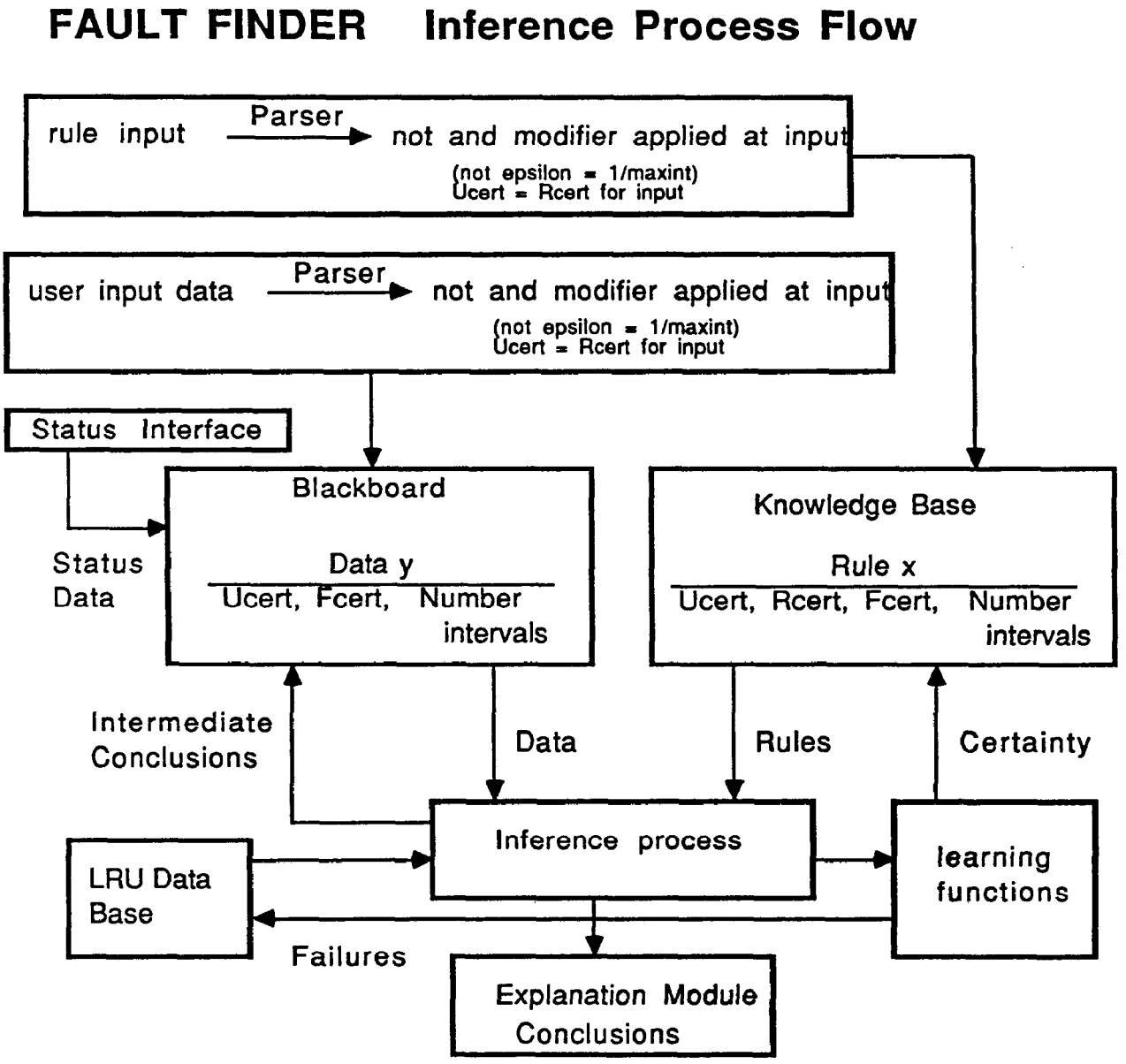

FAULT FINDER Inference Process Flow
rule input
Parser
not and modifier applied at input
(not epsilon = 1/maxint)
Ucert = Rcert for input
user input data
Parser
not and modifier applied at input
(not epsilon = 1/maxint)
Ucert = Rcert for input
Status Interface
Status Data
Blackboard
Data y
Ucert, Fcert, Number intervals
Knowledge Base
Rule x
Ucert, Rcert, Fcert, Number intervals
Intermediate Conclusions
Data
Rules
Certainty
Inference process
LRU Data Base
learning functions
Failures
Explanation Module Conclusions

figure 3.

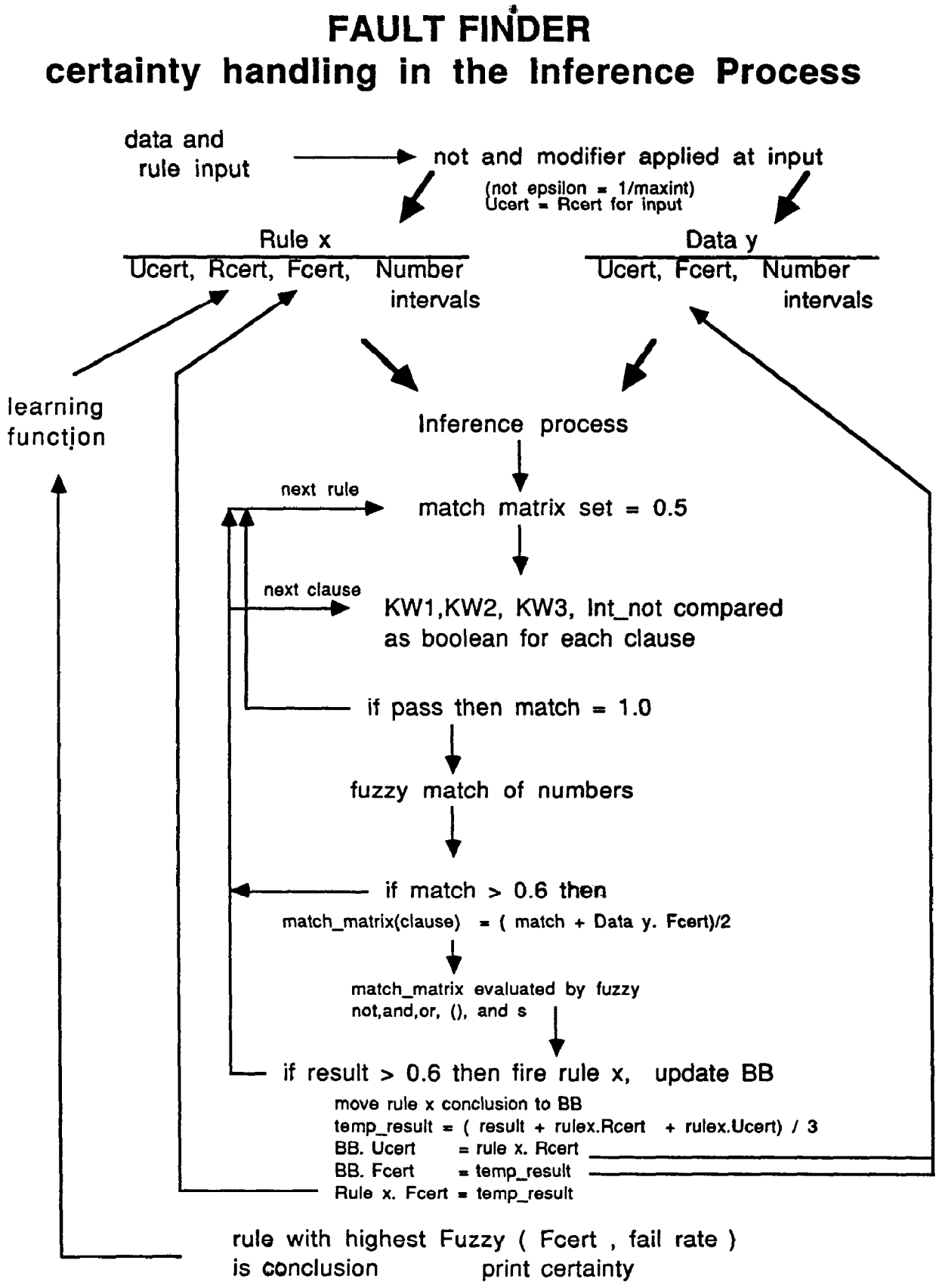

FAULT FINDER
certainty handling in the Inference Process
data and
rule input
not and modifier applied at input
(not epsilon = 1/maxint)
Ucert = Rcert for input
Rule x
Ucert, Rcert, Fcert, Number intervals
Data y
Ucert, Fcert, Number intervals
learning function
Inference process
next rule
match matrix set = 0.5
next clause
KW1,KW2, KW3, Int_not compared as boolean for each clause
if pass then match = 1.0
fuzzy match of numbers
if match > 0.6 then
match_matrix(clause) = (match + Data y. Fcert)/2
match_matrix evaluated by fuzzy not,and,or, (), and s
if result > 0.6 then fire rule x, update BB
move rule x conclusion to BB
temp_result = (result + rulex.Rcert + rulex.Ucert) / 3
BB. Ucert = rule x. Rcert
BB. Fcert = temp_result
Rule x. Fcert = temp_result
rule with highest Fuzzy (Fcert , fail rate)
is conclusion
print certainty

figure 4.

If the two or more conclusions fall in the same class then a weighted average is used to make a choice as follows:

let

w(FFcert) =

b_1 * Fcert, w(Fail_Rate) = b_2 * Fail_Rate

and $b_1 + b_2 = 2$.

Then: FFcert = (w (Fcert) + w (Fail_Rate)) / 2

Where a b_1 of 1.1 weights the Fcert at 110% and a b_2 of 0.9 weights the Fail_Rate at 90%.

Example 6: In this example the inference procedure produced two conclusions with the following values:

Conclusion 1 => $\mathbf{Fcert_1} = 0.90$,
$\mathbf{Fail\_Rate_1} = 0.50$

Conclusion 2 => $\mathbf{Fcert_2} = 0.75$,
$\mathbf{Fail\_Rate_2} = 0.90$

Using Table 1, Conclusion 1 has 5 points and conclusion 2 has 5 points, but conclusion 1 is in class 2, and conclusion 2 is in class 3. So conclusion 1 is tried first.

Example 7: In this example the inference procedure produced two conclusions with the following values:

Conclusion 1 => $\mathbf{Fcert_1} = 0.75$,
$\mathbf{Fail\_Rate_1} = 0.25$

Conclusion 2 => $\mathbf{Fcert_2} = 0.80$,
$\mathbf{Fail\_Rate_2} = 0.20$

Using Table 1, Conclusion 1 has 3 points and conclusion 2 has 3 points, also conclusion 1 and 2 both are in class 7, so the weighted average must be used.

$FFcert_1 = (1.1 * 0.75 + 0.9 * 0.25) / 2 = 0.525$

$FFcert_2 = (1.1 * 0.80 + 0.9 * 0.20) / 2 = 0.53$

So conclusion 2 will be tried first.

Conclusion

We have seen that in the PC based version of the FAULT FINDER, maintenance personnel may easily use the system without changing existing maintenance plans or procedures. The system can be used as a consultant, and a library to store knowledge learned through experience. New rules may be added by maintenance personnel. The system also learns from fault data and improves its fault finding performance.

When the real time status interface is added in the PC based version of the FAULT FINDER, the target system status is now available to the inference procedure. The system can still be used as a consultant, and a library to store knowledge learned through experience. But now the Fault Finder can run in a real time mode logging conclusion to a printer for example. New rules may still be added by maintenance personnel, and the system still learns from fault data and improves its fault finding performance. If the Fault Finder was rehosted within the target system it could become part of the fault isolation software for that application and could be tailored to the system's requirements.

The FAULT FINDER PC based system with its learning process helps the designer and manufacturer be more productive, and helps the end user have a system with a higher availability number, with lower costs for maintenance.

References

[1] M. Schneider, D. Clark and A. Kandel."On the Matching Process in Fuzzy Expert Systems",1989.

[2] D.A. Rowan. "AI Enhances On-Line Fault Diagnosis", InTech, May 1988, Page 52.

[3] HARRIS Electronic Systems Sector Melbourne, Florida 32901

[4] TURBO Pascal is a trademark of Borland International, Inc.

[5] M. Schneider and A. Kandel."Cooperative Fuzzy Expert Systems - Their Design and Dpplications in Intelligent Recognition", 1988, Verlag TUV Rheimland.

[6] Conor Clancy. "Qualitative Reasoning in Electronic Fault Diagnosis", Electronic Engineering, Nov. 1987, Page 141.

[7] Schneider, M., Shnaider, E., and Kandel, A. (1989). Application of the Negation Operator in Fuzzy Production Rules, Accepted for publication in the International Magazine for Fuzzy Sets and Systems.

[8] Harmon, P. and King, D. 'Expert Systems' John Wiley & Sons, Inc. 1985.

[9] Zadeh, L. A. 'From circuit theory to system theory, Proc. Institute of Radio Engineers, 50, 856-865. 1962.

[10] Zadeh, L.A. 'Fuzzy Sets', Inf. and control, 8, 338-353, 1965.

[11] Barr A. and Feigenbaum, E. A. The Handbook of Artificial Intelligence. volume 1, William Kaufmann, 1981.

[12] Bonnet, A., Haton, J. P., and Truong-Ngoc, J. M., Expert Systems, Prentice Hall, 1988.

[13] Gevarter, W. B. Expert Systems: Limited But Powerful IEEE Spectrum. August, 1983.

[14] Hayes-Roth, F., Waterman, D.A. and Lenat, D.B Building expert systems. Addison-Wesley Publishing, 1983

[15] Liebowitz, J. Introduction to Expert System Mitchell Publishing, Inc. (1988).

[16] Luger, G.F., and Stubblefield, W. A. Artificial Intelligence and the Design of Expert Systems. Benjamin Cummings, 1989.

[17] Waterman, D.A. A Guide to Expert Systems. Addison-Wesley Publishing, 1985.

[18] Weiss, S.M. and Kulikowski, C. A. Designing Expert Systems.Rowman and Allanheld, 1984.