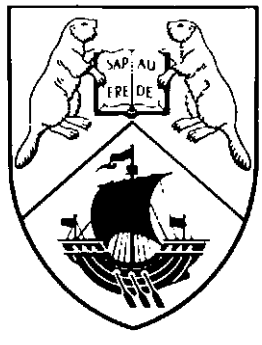
[redacted]

[redacted]

SCHOOL OF COMPUTER SCIENCE



[redacted]

UNIVERSITY OF NEW BRUNSWICK

FREDERICTON, N. B. CANADA E3B 5A3

XpeRTD:
AN EXPERT SYSTEM FOR DIAGNOSING RESISTANCE TEMPERATURE DETECTORS

by

Kirby Ward

TR88-042, June 1988

This is an unaltered version of the author's
M.Sc.(C.S.) Thesis

ABSTRACT

A prototype expert system, XpeRTD, is introduced. XpeRTD diagnoses mechanical faults in resistance temperature detectors (RTDs). RTDs are electrical sensing devices used in the nuclear power industry to monitor reactor activity. It is shown that the expert system approach is best for RTD diagnosis. By implication, it is the best approach for similar problems where there are no exact models or where statistical methods don't apply.

The introduction of an expert system provides an opportunity to discuss the design and implementation of expert systems in general. A particular approach to the difficult task of knowledge engineering is detailed. A FLOPS (Siler and Tucker, 1986) fuzzy logic production system implementation is compared with other expert system "shells". FLOPS' fuzzy logic is compared with other techniques for representing and reasoning from uncertainty.

ACKNOWLEDGEMENTS

My thesis advisor Brad Nickerson was always ready with advice and encouragement. He greatly assisted this work by asking the right questions and providing the right perspective. Along with Bryan Patterson at N.B. Power, Professor Nickerson initially identified the RTD problem as one for which the expert system approach might apply.

The staff at the Point Lepreau nuclear facility were both helpful and friendly. Malcolm Callister supplied the expertise on RTDs and gave up many days to this project. During the engineering process, Malcolm was generous and patient in responding to sometimes naive and persistent questioning.

I am also grateful to family and friends for their moral and other support, especially Ruth Ward and Kyriakos Koilakos, Padjen Zdenko and Kevin Scott.

CONTENTS

ABSTRACT ii

ACKNOWLEDGEMENTS iii

Chapter Page

1. INTRODUCTION 1

2. EXPERT SYSTEMS 4

Knowledge-based Computer Systems 6
Knowledge Representation 9
Natural Languages 9
Predicate Logic 11
Structured Objects 14
Frames. 15
Semantic Networks. 17
Representations in Lisp. 19
Rule-based Expert Systems 20

3. FUZZY LOGIC 31

Probabilistic Reasoning in PROSPECTOR 34
Certainty Factors in MYCIN 49
The Dempster-Shafer Theory of Evidence 58
Fuzzy Logic in FLOPS 59

4. RESISTANCE TEMPERATURE DETECTORS 71

Description 71
RTD Maintenance 78
RTD Performance Data 81
Startup Bias 81
Refuelling Response 85
Derating Response 88
RTD Diagnosis 88
An RTD Expert System 92

5. KNOWLEDGE ENGINEERING OF AN RTD EXPERT SYSTEM . . . 95

Problem Identification 98
The Diagnostic Model 99
The Diagnostic Process 100
Problem Definition 105
Prototype PRO1: Forward Chaining 110

Prototype PRO2: Backward Chaining 114
Rule Definition 118
Interviews with an Expert. 118
Defining Rules by Induction. 120
Prototype PRO3: Parallelism and Contexts . 126
Testing and Debugging 132
Verification 136

6. SYSTEM DESCRIPTION 137

Operating Philosophy 137
Hardware and Software 139
Data Organization 139
System Options 142
Explanation 144
Training 147
Documentation 147
Sample Diagnosis 148

7. CONCLUSION 153

Thesis Summary 153
Further Research 156

Appendix Page

I. FUZZY SETS AND FUZZY LOGIC 160

II. FLOPS COMPARED WITH OTHER EXPERT SHELLS 170

III. FLOPS PROGRAM LISTINGS 176

Prototype PRO1 176
PRO2 205
PRO3 211

IV. SAMPLE RTD DIAGNOSIS 223

REFERENCES 234

LIST OF TABLES

Table		Page
1.	Comparison of Uncertainty Representations	70
2.	RTD temperature-voltage conversion.	75
3.	RTD Mechanical Faults	80
4.	Features: Expert System Shells.	171
5.	FLOPS compared with ART.	173

LIST OF FIGURES

Figure		Page
1.	Semantic network data structure	18
2.	Prolog depth-first search (Cohen, 1985)	25
3.	FLOPS best-first search	29
4.	PROSPECTOR posterior probability P(h|f)	43
5.	Heating transport system (HTS) of Candu reactor	72
6.	HTS single channel showing fuel cells	74
7.	Control monitor display of RTD data	76
8.	Resistance Temperature Detector (RTD)	77
9.	RTD startup response	82
10.	RTD startup bias. Method 2.	85
11.	RTD refuelling response (trend)	87
12.	Knowledge representation for RTD diagnosis	102
13.	RTD REFLEX data base entry, Channel P11	108
14.	Patterson-Callister rules for RTD diagnosis	109
15.	PRO1 forward-chaining rules: decision tree	110
16.	PRO1 data base definitions	112
17.	PRO2 backward-chaining rules: Prolog pseudocode	115
18.	PRO2 backward-chaining rules: decision tree	116
19.	Repertory grid for RTD diagnosis	121
20.	Sample dialogue for program SG (adapted from Shaw and Gaines, 1984)	123

21. PRO3 forward- (FC) and backward-chaining (BC) rules: semantic net 127

22. XpeRTD Expert System 140

23. XpeRTD data and data communication 142

24. XpeRTD screen display. 144

Chapter 1

INTRODUCTION

Expert systems are general problem-solvers. They incorporate the best techniques from more than 20 years' research in artificial intelligence. Their efficiency and applicability were established in the 1970's by systems such as DENDRAL (Lindsay et al., 1981) and MYCIN (Buchanan and Shortliffe, 1984). These early systems introduced a new and practical approach for problems that are too complex for conventional programs. These are problems which have no exact solution or whose solution involves a combinatorial search. MYCIN is an expert system for medical diagnosis, a problem which is not well-defined. There is no general procedure for diagnosis because the same symptoms can indicate more than one cause. DENDRAL is an expert system to determine chemical structure. The DENDRAL problem is well-defined but the number of possible solutions is exponential.

Expert systems have become a general-purpose programming tool used to solve not only "expert" problems, but other problems for which there are no readily available or easily generated algorithms. This thesis looks at one such problem and analyzes the applicability of the expert system approach

to its solution. The problem is typical of an expert system problem because it is not well-defined. It represents a task that is performed by a human expert whose skills depend more on experience than on a formal understanding of the problem domain.

Expert system solutions follow the difficult task of knowledge engineering, i.e., the representation of a large or imprecise body of knowledge as a precise set of facts and rules. The engineering techniques applied to the thesis problem are detailed in Chapter 5. Chapter 6 describes XpeRTD, a prototype expert system which incorporates the solution developed in Chapter 5. Program details and sample output for XpeRTD are listed in the appendices.

The remainder of the thesis introduces expert systems and explains the concepts which appear in subsequent discussions. Chapter 2 explains knowledge representation and rule-based systems. Chapter 3 details the representation of uncertainty in expert systems. The problem domain is described in Chapter 4. This chapter characterizes resistance temperature detectors (RTDs) and RTD diagnosis. It also justifies an expert system approach for computer-assisted RTD diagnosis (Section 4.5).

The XpeRTD prototype is implemented in FLOPS (Siler and Tucker, 1986), a rule-based programming language using fuzzy logic. Chapter 2 introduces the FLOPS implementation

(Section 2.3). FLOPS' fuzzy logic is explained in Chapter 3 by comparing it with other approaches for rule-based reasoning, including probabilistic reasoning (Section 3.1). Appendix I discusses fuzzy logic in more detail and provides the definitions and theoretical background on which the logic is based. Appendix II assesses the efficacy of FLOPS compared with other expert system "shells", including a state-of-the-art shell called ART (Inference Corp., 1987).

Chapter 2

EXPERT SYSTEMS

Expert systems are knowledge-based systems. Their design follows research in artificial intelligence (AI) toward duplicating or approximating human problem-solving. Human problem-solvers have the advantage of knowing more than the average computer program. Given this greater knowledge, they can efficiently select among the details of a particular problem, isolate relevant facts and apply the appropriate procedures. Expert systems simulate this approach. They differ from conventional programs to the extent that they are general problem-solvers which incorporate human problem-solving skills.

A conventional program (e.g., Fortran) incorporates a set of procedures (i.e., an algorithm) that applies to every set of facts (i.e., data). The model is a single resource machine whose output depends only on input. In an expert system, different procedures are used with different sets of data. The model is a multiple resource machine which applies the resources appropriate for the input. This model for expert systems is the one used by psychologists to describe human intelligence (Friedman, 1982).

A multi-resource model for human problem-solving was also identified by the designers of MYCIN. In formulating a computer solution for diagnosing diseases of the blood, MYCIN used the specialized knowledge of diagnostic experts in the form of simple rules. Each rule represents a judgement or diagnosis which applies to only one case. MYCIN's rule-based system inherits the diagnosticians' skills by duplicating their judgements in these special cases. Like the human expert, MYCIN performs a diagnosis by applying the one or more rules or procedures which are appropriate for the current case.

The MYCIN example also demonstrates that the solution to an expert problem is inherent in the way that knowledge is represented. In MYCIN, a computer solution was realized when its designers recognized the manner in which the experts perform a diagnosis, i.e. how the experts use patient symptoms and clinical data to diagnose and treat a particular disease. The task of translating expert skills into a computer solution is the main task in expert system design. This difficult process is called knowledge engineering. It involves, for example, identifying and summarizing the rules on which a medical diagnosis is based.

Knowledge engineering involves defining the essential characteristics of a machine representation for knowledge. Knowledge representation has been and continues to be one of

the major focuses of research in expert systems and in the wider field of AI. It provided the basis for the programming language Lisp, a general-purpose language for AI. Standard procedures for knowledge representation are implemented in the design of Prolog and other specialized languages called expert system "shells". Other applications extend these procedures to include a facility for solving problems with uncertainty. These techniques are introduced next.

2.1 KNOWLEDGE-BASED COMPUTER SYSTEMS

Expert systems organize knowledge as a knowledge base of facts and procedures. It includes the information required to simplify complex problems. It provides the techniques for reducing complex tasks to simple problems, or for constraining the problem domain. MYCIN is an example of problem reduction where the complex problem of medical diagnosis is reduced to a set of simple rules. DENDRAL uses a "planning" algorithm to process chemical data and constrain the size of the DENDRAL problem. These are standard techniques but, in expert systems, they are applied differently. In conventional programs, constraints are functions and simpler problems are subroutines. This is an algorithmic or "procedural" approach. It requires knowledge that is exact. Expert systems use constraints which are best guesses. Facts are often changed or substituted to accomodate different procedures. Knowledge is useful even when it is inexact.

Knowledge-based systems accomodate "expertise". This is knowledge that may not contribute directly to a solution but which may be useful in choosing among different solutions. It represents the practical skills which an expert acquires beyond the formal knowledge of his field. Medicine, for example, is more art than science. Medical science can be taught but not the reasoning on which it is based. Medical students, after they learn the formalities, are subjected to long internships where they learn the "practice" of medicine. These practiced skills are not well-defined but they are summarized as "heuristics" or rules of thumb which apply in special cases. Heuristics simplify complex problems by reducing the number of decisions which are required, or by specifying the kind of decision which applies.

DENDRAL uses a heuristic generate-and-test algorithm to reduce the problem of identifying chemical structures. DENDRAL identifies the molecular structure of a compound based on spectrometric data. Spectrometry is a laboratory procedure that analyzes the atomic composition of a test sample. It provides a "spectrum" or spectrogram of mass versus relative abundance, data with characteristic peaks which indicate the presence of atoms or atomic groups.

The DENDRAL problem is too complex for an ordinary program. A group of atoms may exist as one of a large number of combinations and configurations. Each

configuration is a distinct compound. DENDRAL identifies a particular configuration using heuristics to control the generator and to define an efficient test.

Specifically, DENDRAL uses its knowledge of chemistry to generate an ordered set of hypotheses. Candidate solutions are ordered by increasing atomic number (i.e., C, N, O, P, S, ...), so an unlikely configuration, e.g., a sulphur-centered compound, is not considered. DENDRAL then prunes the candidate set using a rule-based procedure. The test procedure first predicts possible solutions based on spectrometric data. It then discards hypotheses that are inconsistent. For example, if the spectrum shows no C = C double carbon bond, then the compound is not a ketone.

DENDRAL depends on knowledge that is specific to the task at hand. It is the knowledge used by a chemist to select cases which are relevant and to eliminate others which are not. Because the selection procedures are task-specific, they don't apply to other domains. Also, because of its specificity, the knowledge is not usually available in a form which is ready to use. How this knowledge is identified and structured is discussed next.

2.2 KNOWLEDGE REPRESENTATION

A computer solution of a problem follows from its representation. Referring to Wirth's (1976) equation,

program = algorithm + data structures,

a computer solution specifies what facts (data) are relevant to the problem and what method (algorithm) is used to derive the solution from the facts. Together, these elements specify what is known about the problem domain and how this knowledge is best represented for processing in a machine format.

Wirth's equation doesn't apply to many AI applications. A problem that is complex or not well-formulated does not divide easily into separate facts and procedures. Other representations are used in AI to formalize the knowledge on which problems and solutions are based. These formalizations are more suited for "intelligent" applications because they represent structures and processes that characterize human thinking and problem-solving.

2.2.1 Natural Languages

Knowledge is most easily represented in English or another natural language. Language is the formalism by which humans understand the world. It is the "natural" representation for communicating and structuring what we know. Based on this observation, AI researchers are attempting to develop

computer systems which "understand" English. It is hoped that, eventually, such systems will respond to English commands and will automatically generate the machine representations for executing these commands (Michalski et al., 1983). This research has served to underline the complexity of human intelligence. It has also produced some general approaches to knowledge representation in expert systems.

Natural languages emphasize the importance of "context". A single English sentence is complicated by the context or background information which gives it meaning. For a machine to "understand" an English sentence, it must understand the relevant contexts. Defining a single context can often generate an infinite regression.. For example, the sentence

'An expert system is a complex computer program.'

might be used as the basis for a program that instructs users on expert systems. This sentence implies knowledge about computer programs and complex objects. It also implies that there is a relation between 'expert systems' and 'computer programs' that is characterized by some element, such as 'problem', that is essentially 'complex'. The contexts in this case are computer programming, problem-solving and complex (and simple) objects. Each context subsumes a large list of "subtexts" and an uncountable

number of facts. In a practical application, the number of facts can be reduced by restricting the problem domain. This is the approach used in most expert systems.

For most applications, it is more practical to restrict the context to a highly specialized domain. In these cases, there are other representations which are more efficient than natural languages. Even where the context is restricted, an English statement of a problem is unnecessarily complex. A computer language, for example, provides a more concise statement of a problem than its English counterpart, especially the computer language APL.

Among computer languages for AI applications, Lisp (Abelson et al., 1985) and Prolog (Clocksin and Mellish, 1984) are preferred. Lisp and Prolog evolved from AI applications. Their syntax and semantics are designed to most easily represent the structures (symbols) and processes (reasoning) which characterize human intelligence. These structures and their relation to Lisp and Prolog are discussed next.

2.2.2 Predicate Logic

Logic is an early attempt to systematize human reasoning. Logical systems provide a concise way to represent facts about the world. They also provide a set of operations or rules which approximate human reasoning, especially as it

applies to solving problems in a well-specified domain. Logical systems are defined by logical languages that abstract those characteristics of natural languages that are essential to solving problems. Logical languages specify a structure or syntax for representing facts as logical formulas. They include a semantics that specifies the meaning or interpretation of logical sentences in terms of a natural language such as English.

Logic provides different representational schemes depending on the specificity of the language. In propositional logic, an English sentence is represented by a single variable. For example,

P = "An expert system is a complex computer program."

In predicate logic, the same sentence is represented as a structured statement:

P(f(x),y) = is_a(complex(ComputerProgram),ExpertSystem).

For each of these representations, logicians have developed a formal "calculus" that reduces logic theory to a system of formulas, operators and rules. The calculus provides a general format for representing facts and for reasoning from these facts. Propositional and predicate calculus are especially important as logical systems because they are consistent and complete. These characteristics guarantee that the systems are "computable".

In fact, early attempts to create an "artificial intelligence" were efforts to develop an algorithm to incorporate a logical calculus. These efforts produced programs that were inefficient and impractical until the formulation of resolution logic by Robinson (1965). This logic reduces formal logic to a single rule, ie., resolution, and it avoids the computational complexity of other logical languages. Resolution is the basis for the logic programming language Prolog whose development during the 1970s renewed interest in logic as a basis for computation.

Prolog has re-established logic as an efficient, general-purpose schema for intelligent problem-solving. Its efficiency is based on a simple syntax and a single-rule (resolution) interpreter. The simple design hides the level of complexity that Prolog represents. Based on resolution, Prolog inherits all the complexity of a first-order predicate calculus.

For all its proponents, Prolog has as many detractors. Its most serious shortcoming is its obvious dependence on a logical calculus. This explicit use of a logical format entails an approach to problem-solving that is unique among computer languages. In place of Wirth's characterization, Kowalski (1979) defines a computer problem with a formulation that is characteristic of Prolog:

algorithm = logic + control.

In Prolog, program control is provided by the Prolog interpreter. The Prolog programmer need only define the logic of a particular problem in terms of Prolog's facts and rules. This definition, however, is more or less difficult depending on the programmer's familiarity with formal logic. Kowalski (1979), a logician, continues to argue the sufficiency of Prolog but other researchers in AI, including Robinson (1979), propose reasoning schemes that are more "natural".

2.2.3 Structured Objects

Logic doesn't apply when humans don't think logically. Formal logic is based on "deductive" reasoning that analyzes what is already known. Humans also reason "inductively" to synthesize new information. They often reason based on "common sense" and make guesses when they don't have all the facts. As an alternative to formal logic, AI has developed schemas for "natural" deduction. Instead of facts, knowledge is arranged around objects. Objects are "structured" from simpler elements which may represent other objects or events. The simple elements represent characteristics or properties that distinguish a more complex object or a class of objects. This representation follows the "natural" tendency to define concepts in terms of their distinguishing characteristics.

2.2.3.1 Frames.

A logical representation such as

```
is_a(complex(ComputerProgram), ExpertSystem)
```

encompasses a good deal more information. Before it can be used by a computer program, this information must be made more explicit. The characterization of expert systems, for instance, may be clarified by examples. The logical representation above may be expanded to include additional facts:

```
is_a(expert_system, Mycin)
is_a(expert_system, Dendral)
diagnoses(BloodDisease, Mycin)
elucidates(ChemicalStructure, Dendral)
```

A useful fact base would include a rule relating particular systems to expert systems in general:

```
is_a(expert_system, _x) -->
is_a(complex(ComputerProgram),_x)
```

The development of this example demonstrates the limitations of a logical format. It forces a representation in terms of a rigid structure of facts and rules. It assumes that the problem domain is already well understood and is easily characterized as a sequence of facts. It also assumes that the relations expressed by rules are well-defined. Rules are

defined to generate a preferred conclusion. For each rule, the conclusion must be identified and the prerequisite conditions must be specified.

Alternatively, the same information can be structured as a set of objects, each defined by their obvious or relevant characteristics:

```
ExpertSystem:
  IsA:        ComputerProgram
  TypeIs:     Complex
  Examples:   Mycin, Dendral
  UsedFor:    (Examples TypeIs)
  AppliesTo:  (Examples AppliesTo)

Mycin:
  IsA:        ExpertSystem
  TypeIs:     Diagnosis
  AppliesTo:  BloodDiseases
```

An object may be understood as a collection of logical facts which may be combined and manipulated using logical operations. The object may also be understood as a record data structure whose elements may be searched and combined using conventional programming procedures. The format incorporates additional information by grouping related facts. For example, it clarifies different 'types' of objects, a facility not available in first-order logics. It also includes hierarchical or taxonomic information to relate classes and subclasses. Implicit in a hierarchical structure is the property of "inheritance" where objects of a subclass inherit all the properties of the superclasses.

In logic, the same relation must be supplied by explicit rules.

The general format for a structured object is

Object:

Attribute: Value

Attribute: Value

...

This structure is usually called a frame. It was first defined by Marvin Minsky (1975) as a "framework" or data structure applied to a vision problem. For such a problem, a frame may have an attribute 'ViewPoint' to distinguish among the different "frames" or perspectives of a 3-D object.

2.2.3.2 Semantic Networks.

An object-attribute-value relation is the basis for other data structures which emphasize different ways of representing objects. For example, a set of objects which are defined as frames may be conveniently grouped as a frame of frames. A semantic network is a data structure that defines more explicitly the relations among a set of objects. The objects from the previous example are represented by the network in Figure 1 . The network may be understood as a graph with each object or object class represented by a single node. Relations between values and

objects or instances and object classes are represented as labelled arcs. The labels are names of functions which return attribute values, e.g.,

inst_of(Dendral) = ExpertSystem , or

appl(ChemStructure) = Dendral

can be defined from Figure 1 . This figure assumes a hierarchy with arcs directed from top to bottom (left to right) but inverse functions are also implied, e.g., applies_to(Dendral) = ChemStructure.

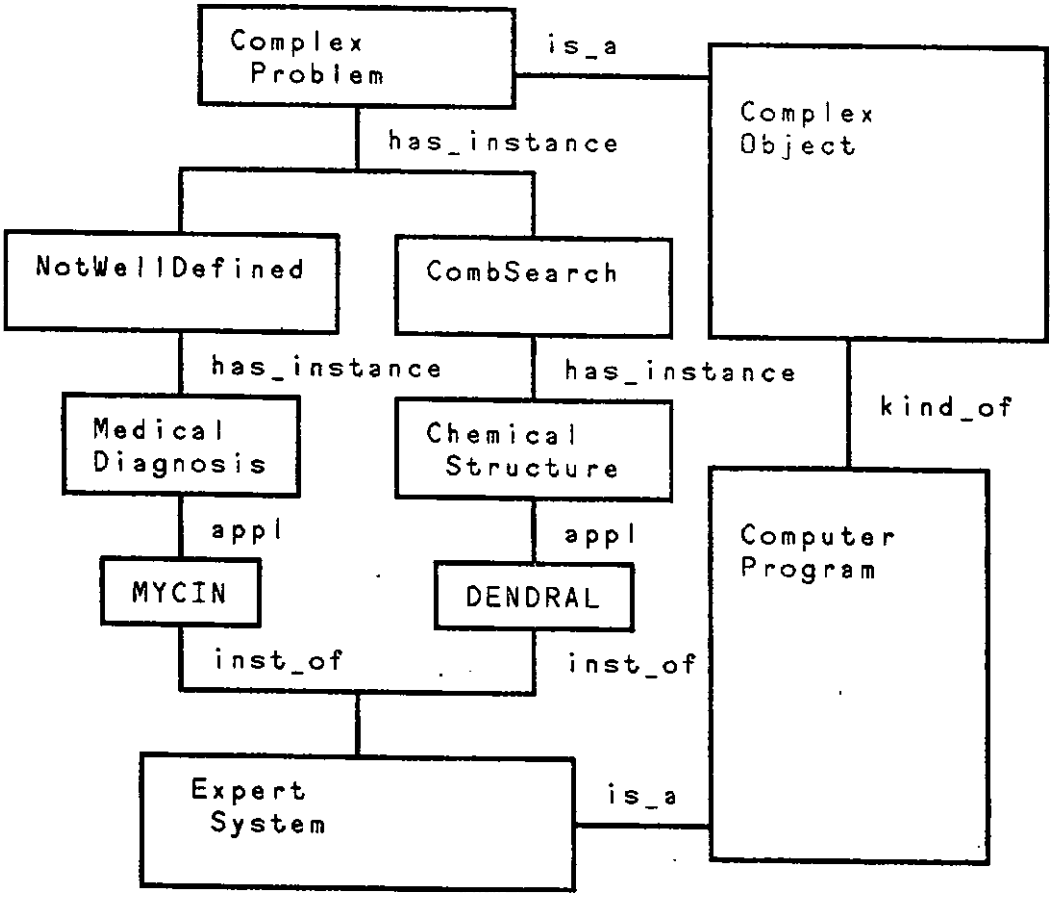


Figure 1: Semantic network data structure

A semantic net, because it has a graphical representation, provides a "natural" way to structure complex data. Similar to frames, it defines the hierarchical relationships among objects. It also defines a hierarchy of object classes and introduces the notions of reflexivity and transitivity among object relations.

2.2.3.3 Representations in Lisp.

All the representations in this section, despite the apparent differences, may be implemented using a single data structure. This structure is the linked list which is the basis for the programming language Lisp. Because of its design as a list processor, Lisp has established itself as the most efficient and convenient language for AI applications. Even logic programs are easily represented in Lisp. In fact, Prolog depends on list structures for its efficient implementation.

Complex data structures, for example the frames defined earlier, translate easily into Lisp lists:

```
ExpertSystem:
  ((IsA ComputerProgram)
   (TypeIs ComplexObject)
   (Examples (Mycin Dendral))
   (UsedFor (apply Examples TypeIs))
   (AppliesTo (apply Examples AppliesTo))
  )
```

```
Mycin:
   ((IsA  ExpertSystem)
    (TypeIs Diagnosis)
    (AppliesTo BloodDiseases)
   )
```

Lisp provides a simple but powerful format for representing and manipulating complex objects. A Lisp list (f x y) is interpreted as either a data structure, such as a frame, or a Lisp procedure (i.e., a logical function of the form f(x,y)). Its simplicity is based on its use of a single structure that is a "natural" representation for both data and procedures. Its power, like Prolog, is based in logic. Lisp incorporates the lambda calculus, a rigorous formulation for a logic of mathematics.

2.3 RULE-BASED EXPERT SYSTEMS

The preceding discussion emphasized the data structures that are appropriate for knowledge representation in a knowledge-based system. Data structures capture the factual or declarative information which defines the objects or events that constitute a knowledge base. A knowledge-based system also depends on procedural information which defines the relations among objects and facts. In a formal logic, for example, procedural information is provided by the rules which operate on logical formulas. In a frame

representation, declarative and procedural knowledge may be combined: an attribute may be defined by a function or procedure which operates on other objects to return the attribute value. A knowledge base may also include heuristics which direct what knowledge (i.e., facts and procedures) is selected or applied. Finally, the knowledge base may be structured and defined with reference to a control procedure which provides the top level for a knowledge-based system.

Rule-based expert systems provide a convenient organization that simplifies the design and development of knowledge-based applications. Rule-based systems are constructed as two separate and independent components:

expert system = inference engine + knowledge base .

A control procedure or "inference engine" provides a top-level strategy that applies to every "rule-based" knowledge base. The system designer need only specify the application by defining a knowledge base in terms of facts and rules.

The "fact base" contains a description of the objects in the problem domain (e.g., Mycin, expert system), their characteristics (e.g., Mycin is applied to medical diagnosis) and the relations which exist among the different objects (e.g., Mycin is an example of an expert system). The fact base may be represented by a predicate calculus, by

object-attribute-value triples, or by multiple-entry data structures, depending on the inference engine.

The "rule base" provides a set of rules or procedures that operate on the fact base. For example, a rule is

```
IF _x IS ExpertSystem
  AND
   (Application OF _x) IS _y
  AND
   (TypeOfProblem OF _y) IS _z
THEN
   (TypeOfApplication OF _x) IS _z
```

The knowledge base may also contain a set of hypotheses. These are simple facts which represent goals to be "proved" or solved by the system (e.g., TypeOfApplication OF Mycin IS MedicalDiagnosis).

The inference engine of a rule-based system operates on the knowledge base to prove (or solve) a particular hypothesis or set of hypotheses. In general terms, the inference engine is a set of procedures which search the knowledge base for existing facts or which apply the rules in the knowledge base to generate new facts. Rule-based systems employ different control procedures (data- or event-driven forward chaining or backward chaining with chronological or non-chronological backtracking) depending

on the application. The inference engine implements a controlled search from an initial state to a final or goal state. The search may move "forward" from the start state or "backward" from the goal state. In general, an efficient search starts from the state with the fewest choices.

The logic programming language Prolog (Clocksin and Mellish, 1984) is implemented as a rule-based system. The Prolog interpreter provides a backward-chaining inference engine that is based on resolution. Prolog's syntax provides a logical format for specifying both facts and rules. Figure 2 shows a simple Prolog program consisting of four rules. A Prolog rule has the general form

```
<conclusion>:-
    <fact>
    | <procedure>    ,
```

which means if <fact> or <procedure> is "true" then <conclusion> is true. A Prolog <fact> is any statement in the data base which is not a rule, for example, 'e' and 'f' in the sample program in Figure 2 . A Prolog <procedure> is another rule or a Prolog command.

A Prolog program is goal-driven. A "goal" is a fact in the form of a question (query) or hypothesis (theorem) which is entered by the user to initiate program execution. A Prolog program applies a depth-first search of the knowledge

base and tries to match a goal with either a fact in the fact base or with the conclusion to a rule (Figure 2). If more than one rule is "fireable", i.e., more than one rule matches the query or hypothesis, Prolog applies the rule which appears first in the knowledge base. If the matching process subsequently fails, then Prolog "backtracks" to the next rule which matches the current goal.

FLOPS (Siler and Tucker, 1986) is a rule-based programming language using a different implementation. FLOPS is based on the OPS family of languages (Brownston et al., 1986) and is a "production system" that uses a semantic interpretation of rules. Rules are "productions" of the form

conditions --> actions

where the facts or conditions on the left-hand side (LHS) are "reduced" to the actions on the right-hand side (RHS).

FLOPS facts are structured objects of the form

```
%
%  This is a Prolog program,
%  a database of facts and rules:
%

a:- b, c, d.    % rule1
a:- e, f.       % rule2
b:- f.          % rule3
e.              % fact1
f.              % fact2
a:- f.          % rule4

%
%  When user enters query or goal...

:- a, e.

%  ...Prolog initiates a depth-first search:
%
```

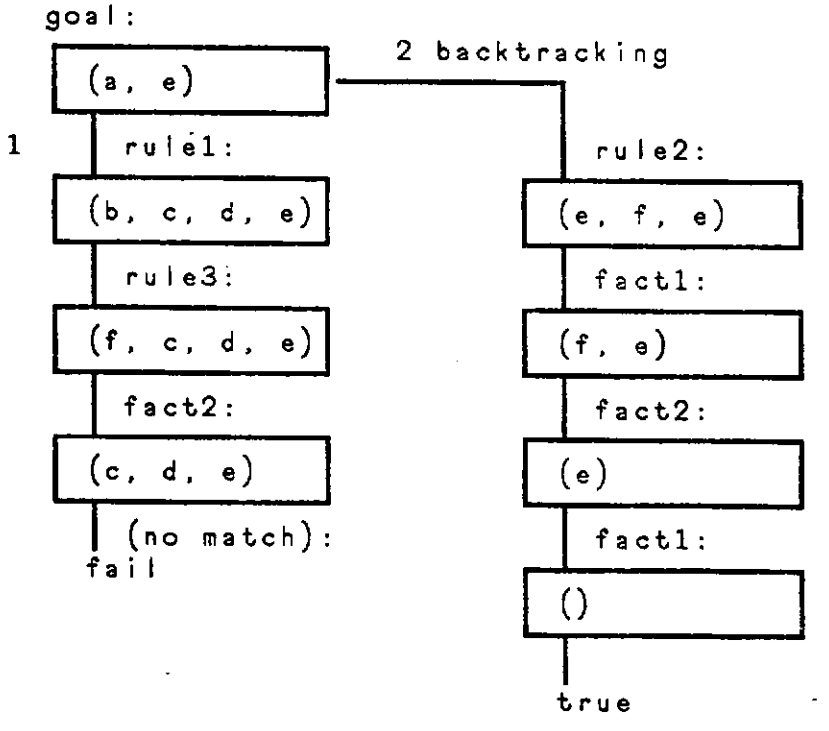


Figure 2: Prolog depth-first search (Cohen, 1985).

```
(object_name
   (attribute_1  value_1  cf_1)
   (attribute_2  value_2  cf_2)
   ...) ;
```

for example,

```
(rtd
   (id "P11" 1000)
   (clamp "yes" 1000)
   (spare "" 1000)
   (connector "" 1000)
   (work "Replace_connector" 600)
)
```

A FLOPS rule is

```
(trend ^id <id> ^bias.high <CF>)
(rtd ^id <id>  ^connector "")
  -->
  modify 2 ^work "Replace_connector" ;
```

The LHS of a FLOPS rule consists of "patterns" which match facts in the FLOPS fact base. In the example, <id> is a variable which would be "instantiated" with the value "P11". The RHS of a FLOPS rule contains FLOPS actions or commands (e.g., 'modify', 'delete') which are executed when the rule is "fired".

FLOPS rules are data-driven. Rules are selected for firing if their LHSs match facts in the fact base. If more than one rule is fireable following a match, FLOPS orders the rules using a conflict resolution algorithm which selects the "best" rule. FLOPS applies a "means-end analysis" (MEA) to resolve conflicting rules and to select a best hypothesis. MEA is a standard resolution algorithm which considers the recency of the facts in the fact base and the number of rule conditions. All things being equal, it makes an arbitrary selection. In FLOPS, after MEA is applied, the best rule is fired and others are saved on a stack or agenda to be used when FLOPS backtracks.

FLOPS implements a forward-chaining search. It derives conclusions or hypotheses from facts. FLOPS backtracking is "non-chronological" since the stacked rules are continually ordered by the conflict resolution algorithm. FLOPS backtracks to the next "best" rule, not just the most recent.

Figure 3 shows the FLOPS' implementation of the program of Figure 2 . A FLOPS program is initialized from facts in the fact base. FLOPS first matches facts and rules. Rules are placed on the agenda if their LHSs are matched. The agenda is then ordered so that the "best" hypothesis is proved first. In Figure 3, rule2 is "better" than rule3 or rule4 because it is more "specific", i.e., it has more LHS

conditions. After a rule is fired, FLOPS repeats the matching procedure, then it re-initializes the agenda. In the example, after three rules are fired, the agenda is cleared and the FLOPS' program is complete.

Prolog and FLOPS are general-purpose programming languages that accomodate a rule-based format. They use control mechanisms which are standard procedures and which provide a top level for most rule-based expert systems. These standard procedures are also incorporated in packaged software known as expert system "shells".

Expert system shells provide interactive software that allows the user to design an expert system by specifying only the facts and rules of a particular problem domain. The inference engine which operates on the knowledge base is hidden from the user, although the user may be able to specify the type of reasoning to be applied. He may choose between a forward chaining search which applies the rules to the fact base to generate a hypothesis or conclusion, or a backward chaining search which attempts to prove a hypothesis or answer a query by matching rules and facts.

An expert system shell is especially useful for prototyping an expert system. It provides a convenient scheme for "engineering" a knowledge base. The division of knowledge into facts and rules provides a scheme that is easily modified. Facts and rules may be added or deleted

```
:
:  This is a FLOPS program,
:  represented as a knowledge base of facts and rules:
:

(b, c, d) --> a ;     : rule1
(e, f) --> a ;        : rule2
(f) --> b ;           : rule3
e ;                   : fact1
f ;                   : fact2
(f) --> a ;           : rule4

:
:  FLOPS initiates a "best-first" search when the user enters
:  a RUN command...
:
```

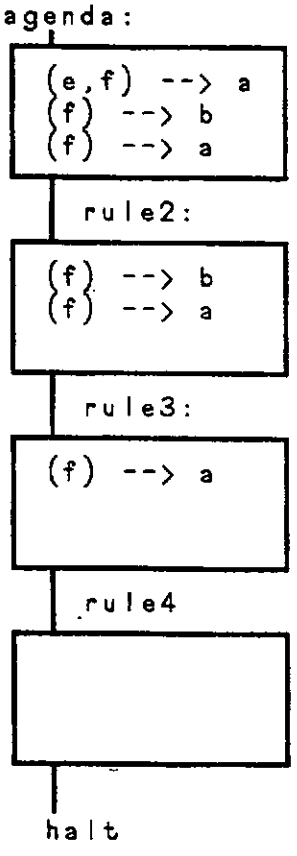


Figure 3: FLOPS best-first search.

without impact on other levels of the system. Once the knowledge base is complete, the expert system may be further tailored for a particular application by redesigning top-level procedures in Lisp or another general-purpose language.

Chapter 3

FUZZY LOGIC

Most computer languages use a logic which is limited for AI applications. Built-in decision structures (i.e., while, if..then statements) are Boolean: they evaluate to either true or false. Human reasoning is not so "categorical". It is often based on beliefs or rules which aren't so certain. It makes decisions which are more or less true depending on the evidence that supports these decisions. Uncertainty is characteristic of much of the knowledge used in expert systems, especially the judgemental knowledge provided by human experts. Expert systems incorporate decision procedures based on general logics to represent uncertain data and to reason based on uncertain evidence.

Rule-based expert systems are theoretically based on ordinary Boolean logic, i.e., rules and facts are true or false. In two-valued systems, rules are limited to categorical statements, e.g., definitions such as

if x is a mammal

then x has hair, or p --> q .

Semantically, this rule means that the proposition q:'x has hair' is true if p:'x is a mammal' is true. In a rule-based

system, a proposition p is true if it appears as a fact in the fact base. Generally, this semantics is implemented by an inference engine which matches facts and rules. If a match is successful, then the conclusion q is recorded or asserted in the fact base. The "inference" on which this implementation is based is the logical inference modus ponens, represented by

```
p --> q
p
-------
q                                    (3.1)
```

In other words, the proposition q is asserted whenever there is a rule p --> q and proposition p is a fact in the fact base.

In many applications, knowledge isn't so certain. In diagnosis, for example, the rule

```
if 1) engine won't turn over and
   2) headlights are dim,
then
   battery is discharged
```

is a "rule of thumb" whose conclusion, though believable or probable, may not be correct. Conceivably, effects 1-2 may have a cause which is different than a discharged battery. Besides uncertainty in the rule itself, there may be

uncertainty or ambiguity about the fact 'headlights are dim'. In a system of uncertain rules, inference (3.1) has the form

```
p --> q    CF1
p          CF2
----------------------   ,                    (3.2)
q          CF3
```

where, for each proposition, there is an associated certainty factor (CF) that measures the certainty or belief that the proposition is "true". In this case, "truth" becomes a function of CF values. CFs may be represented by actual measures such as probabilities. Probability theory, then, provides combination rules to evaluate the relation CF3 = f(CF1,CF2) in inference (3.2). Alternatively, CFs may be implicit. For example, Prolog applies inference (3.2) by choosing the first rule or fact that matches the goal q. Rules and facts in a Prolog program are ordered so those which most likely "prove" q are selected first. This means, if q is a predicate P(X) which contains variables X = x_1, x_2, ..., x_n, then X is instantiated with the most likely values.

In either case, the implementation of a rule-based system is complicated when uncertainty is introduced. The simple inference algorithm (3.1) no longer applies. For inference (3.2), there is no single function which specifies how CF

values are assigned and evaluated (Gaines, 1976; Zadeh, 1988). As a result, expert systems employ techniques which are designed for particular applications but are restricted for others.

Expert systems apply at least four different mechanisms for constructing, testing and modifying sets of beliefs. PROSPECTOR (Duda et al., 1978), an expert system for mineral exploration, uses a probabilistic-based approach. MYCIN employs an "ad hoc" mechanism based on certainty factors. Other systems use the Dempster-Shafer theory of evidence (Shafer, 1976) or fuzzy logic (Zadeh, 1965) which are generalizations from probability theory and multi-valued logic. These methods are detailed in the following sections. The discussion compares the different techniques for reasoning with uncertainty and, eventually, justifies using fuzzy logic in the XpeRTD system of the thesis.

3.1 PROBABILISTIC REASONING IN PROSPECTOR

Probability theory is a systematic and general approach for reasoning from uncertainty. It provides both a representation and a logic for rule-based systems using uncertain data. For example, the "truth" of a proposition

p: 'x is A'

is defined by a probability distribution P(p) or P(A) depending on the value of the random variable x in a set A.

For each value x, P(p): A --> [0,1] specifies a number between 0 and 1 which is the probability or "belief" that the proposition p is true. Also, the "truth" of a statement which combines propositions is evaluated using the combination rules for probabilities. That is, the probability P(p) for a statement p: 'if e then h' depends on the probabilities P(e) and P(h) of the constituent propositions e and h.

Probability theory (Duda et al., 1978) defines inference (3.2) by

```
e --> h   P(h|e)
e         P(e|f)
---------------
h         P(h|f) = P(h|e) * P(e|f)
                 + P(h|¬e) * (1 - P(e|f)) .
```

(3.3)

This definition is the basis of the expert system PROSPECTOR (Duda et al., 1978). Following inference (3.3), a rule r: e --> h is applied if the rule premise e matches facts f in the fact base, or if h is the goal (i.e., top-level) hypothesis. The probability attached to a rule r is the conditional probability P(h|e). That is, the current or "prior" probability P(h) of the rule hypothesis h is changed to P(h|e) if evidence e is observed with certainty P(e) = 1. Inference (3.3) recognizes the possibility that P(e) is less

than certain. Since evidence e is evaluated in terms of facts f, then P(e) is conditioned by the certainty P(f) of f, i.e., P(e) = P(e|f). In this case, the final outcome P(h|f) of inference (3.3) approaches P(h|e) depending on the degree to which f confirms e, i.e., P(h|f) also depends on P(e|f).

PROSPECTOR is a rule-based expert system for identifying mineral deposits from geological evidence. PROSPECTOR uses inference (3.3) to compute the probability P(h|f) that a mineral exists based on the prior probability P(h) that the mineral is present and on the geological characteristics of the exploration site. The PROSPECTOR algorithm is summarized in the following discussion based on Duda et al. (1978), Reboh (1981) and Buchanan and Duda (1983).

The conditional probability P(h|e) of a rule r: e --> h in inference (3.3) is defined by Bayes' rule:

$$P(h|e) = L * P(h) \quad ; \text{ or,}$$
$$P(h|\neg e) = \neg L * P(h) \quad ; \qquad (3.4)$$

where the likelihood ratios L and ¬L are defined as

$$L = \frac{P(e|h)}{P(e|\neg h)} \quad ; \text{ and,} \qquad \neg L = \frac{P(e|\neg h)}{P(e|h)} \quad . \qquad (3.5)$$

In PROSPECTOR, the ratios L and ¬L are subjective estimates provided by expert geologists and recorded with the rules in PROSPECTOR's rule base.

A sample PROSPECTOR rule is

```
quartz_sulfide_veinlets are abundant &
not(alteration_halos are apparent)
  --> (320, 0.2)
  alteration is_favorable_for potassium_zone    ;
```

(3.6)

```
or,
if evidence(e)
  then (to degree L, ¬L)
   hypothesis(h)    .
```

That is, evidence e confirms the hypothesis h to some degree L. If evidence e is not present, then h is confirmed to some degree ¬L. In the sample rule, L = 320 and ¬L = 0.2 indicate that the evidence is "encouraging" but it is not conclusive, i.e., a geological formation (alteration) with abundant quartz but no halos is certainly characteristic of a potassic geological zone. On the other hand, the absence of this evidence is not sufficient to rule out the same conclusion. In general, the value L indicates the degree of support for the hypothesis h if evidence e is present, and ¬L indicates the degree of support if e is absent.

For convenience, the likelihood ratios (3.5) are polled as odds, defined by

$$O(h) = \frac{P(h)}{1 - P(h)} \quad ;$$

then Bayes' rule (4) becomes

$$O(h|e) = L * O(h) \quad ; \text{ or,}$$

$$O(h|\neg e) = \neg L * O(h) \quad .$$

With this interpretation, the weights in rule (3.6) estimate the odds $O(h|e)$ that a hypothesis h is true when the evidence e is true, and the odds $O(h|\neg e)$ that h is true when e is false. For example, in rule (3.6), $L = 320$ and $\neg L = 0.2$ when $O(h|e) = 320/1$ and $O(h|\neg e) = 1/5$. This interpretation not only simplies the estimating procedure, but also accounts for inconsistencies which may result when the values L and $\neg L$ are arbitrarily assigned (Zadeh, 1988).

In the remaining discussion, without loss of generality, Bayes' rule is represented as rule (3.4). This rule "computes" the effect of evidence e on the probability P(h) of h from the values L and $\neg L$. It also restricts L and $\neg L$ so both are positive (or zero). Further, if $L > 1$, then e is more favorable for h than $\neg h$. If $L < 1$, then e is more favorable for $\neg h$. If $L = 1$, then e and h are statistically

independent, i.e., e has no effect on h. If L = 0, then h is false when e is true, or h is true when ¬e is true. Generally, Bayes' rule (3.4) constrains L and ¬L to only three cases:

1. L > 1 and ¬L < 1 ; or,
2. L < 1 and ¬L > 1 ; or,
3. L = ¬L = 1 .

Subjective estimates don't always agree with these constraints. In practice, it is not unusual for an expert to decide that observing evidence e is important, but its absence has absolutley no significance (Reboh, 1981). In this case, ¬L = 1 but L ≠ 1. In PROSPECTOR, the algorithm for inference (3.3) is modified to accomodate these inconsistencies. Also, since Bayes' rule (3.4) assumes P(e) = 1 or P(e) = 0, this rule must be modified for the general case where P(e) is uncertain, i.e., P(e) in (0,1). Uncertainty in evidence e is represented in inference (3.3) as the conditional probability P(e|f). The term P(e|f) is explained next.

Uncertain evidence. In PROSPECTOR, each geological fact f is recorded in the fact base with a probability P(f). A rule r: e --> h is applied if evidence e matches facts f. The certainty P(e) of e depends on the certainty P(f) of facts f. This dependence is represented by the conditional probability P(e|f). It is evaluated as a logical combination of simple facts. That is, if e = f is a simple fact, then

$P(e|f) = P(f)$. If $e = f1 \ \& \ f2$ is a logical combination of two or more facts, then $P(e|f)$ is computed using fuzzy set operators (Zadeh, 1965). These operators are defined recursively by

$$\begin{aligned} P(f1 \ \& \ f2) &= \min[P(f1), P(f2)] \quad , \\ P(f1 \text{ or } f2) &= \max[P(f1), P(f2)] \quad , \\ P(\neg f) &= 1 - P(f) \quad . \end{aligned} \tag{3.7}$$

For example, a PROSPECTOR rule is

```
     (e1: plagioclase is_altered_to albite , or
      e2: plagioclase is_altered_to sericite ) ; and
not (e3: plagioclase is_altered_to epidote)
  -->
  plagioclase_alteration
    is_favorable_for barren_core_zone    .
```

Premise $e = ([e1 \text{ or } e2] \ \& \ (\neg e3))$ is evaluated by

$$P(e|f) = \min(\max[P(f1), P(f2)], \ 1 - P(f3)) \ .$$

In other words, the certainty of evidence e depends on the individual weights of observations f_i. Formally, the operators min() and max() extend the usual probabilty functions to a probabilistic logic. Dubois and Prade (1980) show these operators are the only operators that reduce to a Boolean logic under certainty (Appendix I). In a multivalued logic, the usual operations of addition and multiplication lose the properties of idempotency and distributivity. That is, +() and *() fail to guarantee the completeness of a rule-based system.

Inference (3.3) may now be defined as a computation on P(h|e) and P(e|f) given by

$$P(h|f) = P(h|e) * P(e|f) + P(h|\neg e) * (1 - P(e|f)) \quad . \tag{3.8}$$

P(h|e) is calculated by Bayes' rule (3.4) and P(e|f) is evaluated by the logical operators (3.7). Intuitively, equation (3.8) accounts for the uncertainty in evidence e, the nature of the match between evidence e and facts f in the fact base, and the degree to which evidence e is favorable or unfavorable for the hypothesis h. Actually, equation (3.8) is valid only when P(e|f) = 1 or P(e|f) = 0. In practice, equation (3.8) is inconsistent since P(h|e) and P(h|¬e) are based on estimates which are subjective and arbitrary (Buchanan and Duda, 1983). In other words, the computed value P(h|f) will often not agree with the expert's estimate for P(h|f). That is, inference (3.3) will return a conclusion with a certainty that is higher or lower than expected.

For the general case, when P(e|f) is less than certain, PROSPECTOR computes P(h|f) by interpolating among three cases for P(h|f). In other words, P(h|f) is interpreted as a piecewise linear function of P(e|f) as shown in Figure 4 . If P(e|f) = 1 or P(e|f) = 0, then Bayes' rule (3.4) applies and P(h|f) = P(h|e) or P(h|f) = P(h|¬e). If P(e|f) = P(e), then P(h|f) = P(h) where P(e) and P(h) are prior

probabilities recorded in the PROSPECTOR knowledge base. Otherwise, $P(h|f)$ is some value in $(0,1)$ and is computed by the function in Figure 4 . This function is defined by the following algorithm (Reboh, 1981):

Post_probability;

begin

if $0 \leq P(e|f) < P(e)$ then

$$P(h|f) = P(h|\neg e) + \frac{P(h) - P(h|\neg e)}{P(e)} * P(e|f)$$

else if $P(e) \leq P(e|f) \leq 1$ then

$$P(h|f) = P(h) + \frac{P(h|e) - P(h)}{1 - P(e)} * [P(e|f) - P(e)]$$

end ;

(3.9)

In practice, algorithm (3.9) guarantees that the results of inference (3.3) are consistent, even when the expert's estimates for L and $\neg L$ are inconsistent. For example, if $L > 1$ and $\neg L = 1$, then $P(h|f)$ is unchanged when $P(e|f) < P(e)$, but $P(h|f)$ increases with $P(e|f)$ when $P(e|f) > P(e)$.

Combining rules. A rule-based system also requires a procedure for combining rules of the form r_i: e_i --> h, i.e.,

e_1 --> h, e_2 --> h, ..., e_n --> h ,

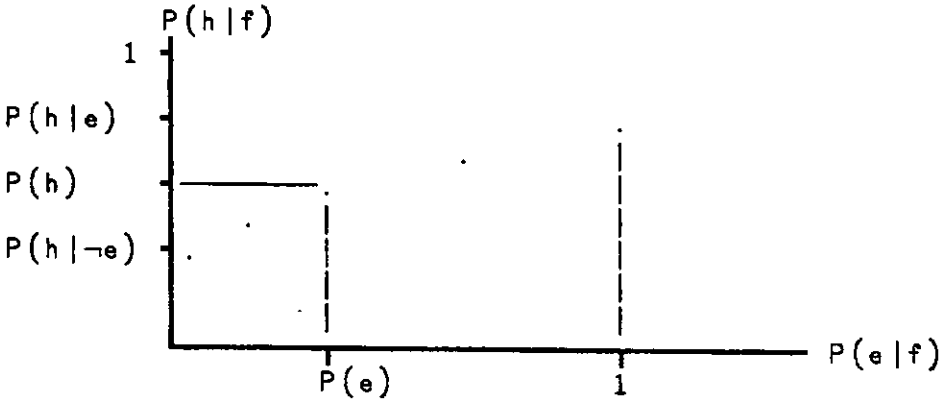


Figure 4: PROSPECTOR posterior probability P(h|f)

where more than one rule bears on the same hypothesis. In this case, Bayes' rule (3.4) may be generalized by defining

$$L = \frac{P(e_1 \;\&\; e_2 \;\&\; \ldots \;\&\; e_n \mid h)}{P(e_1 \;\&\; e_2 \;\&\; \ldots \;\&\; e_n \mid \neg h)} \qquad (3.10)$$

Since evaluation of equation (3.10) involves a combinatorial computation, PROSPECTOR assumes that the different evidence e_i for a hypothesis are <u>statistically independent</u>, i.e.,

$$P(e_1 \;\&\; e_2 \;\&\; \ldots \;\&\; e_n \mid h) = P(e_1|h) * P(e_2|h) * \ldots * P(e_n|h) \;.$$

Then equation (3.10) factors into

$$L = L_1 * L_2 * \ldots * L_n \;. \qquad (3.11)$$

In other words, each rule r_i: e_i --> h contributes a factor L_i to the final outcome of P(h). The effect of

several rules on h, then, may be computed by evaluating each rule separately and adding the result to the current value P(h). In PROSPECTOR, this computation is performed by forward chaining when facts f are changed in the fact base, and by backward chaining when h is a goal hypothesis. In the last case, PROSPECTOR "proves" the goal hypothesis h by applying the rule which has the greatest effect on P(h). This implementation is outlined next. The outline summarizes the PROSPECTOR algorithm and may serve to clarify the preceding discussion.

Implementation. PROSPECTOR employs both forward and backward chaining. The top-level hypothesis can change, depending on the information in the fact base, but the current goal is always the hypothesis which is the most certain, i.e., it has the greatest P(h) among the set of hypotheses h. PROSPECTOR applies backward chaining to prove a goal hypothesis. It determines the best proof (i.e, the best rules) using a heuristic function based on equation (3.8). This heuristic provides an efficient rule-based algorithm that avoids goal searches with a low payoff. PROSPECTOR applies forward chaining to update probabilities P(h_i|f) whenever facts f are added to the fact base and f bears on h_i.

The PROSPECTOR inference engine is summarized (and greatly simplified) by the following algorithm (Duda et al., 1978; Hayes-Roth et al., 1984):

```
Step 1. Executive;
          begin
           Ask_user(f) ;
           if Match(f, e) then
             Propagate(f) ;
             Score(h_i) ;
             top = Best(h_i) ;
             Prove(top)
           end ;
Step 2. Prove(top);
          begin
           if Low(Score(e_i --> top)) then
             Assert(top, P(top))
           else begin
                 rule = Best(e_i --> top) ;
                 list = Premise(rule) ;
                 while Not_empty(list) do
                   begin
                    s = Next(list) ;
                    if Not_fact(s) then
                    Prove(s)
                   end
                end
          end ;
```

In Step 1, a PROSPECTOR program is initialized by facts f which are volunteered by the user. Facts f are recorded with probabilities P(f_i) and these probabilities are propagated through the fact base. A top-level hypothesis is then selected from among the rule conclusions h_i. Among all h_i, PROSPECTOR selects the hypothesis having the highest current P(h_i).

In Step 2, if the goal hypothesis depends on evidence which is not in the fact base, then PROSPECTOR searches other rules (i.e., applies Step 2 recursively) to "prove" or evaluate this evidence. When the evidence is complete, the certainty P(h) = P(top) of the goal hypothesis is evaluated as P(h|f) using algorithm (3.9). In Step 2, if there is more than one rule r_i: e_i --> top, then PROSPECTOR scores the different rules r_i according to the function

$$\text{score}(r_i) = \left|\log \frac{L_i}{L'}\right| P(e_i|f) + \left|\log \frac{L'}{\neg L_i}\right| (1 - P(e_i|f)). \tag{3.12}$$

In effect, function (3.12) orders the rules r_i depending on their contribution to P(h|f) = P(top|f). Specifically, this function computes the utility or expected change in |log P(h|f)|, assuming that evidence e_i will be found true with probability P(e_i|f) or false with probability 1 - P(e_i|f). When e_i is unlikely a priori, function (3.12) produces a high score for rules r_i: e_i --> h that have small values ¬L_i. As evidence e_i becomes more likely, function (3.12)

favors rules r_i with large values L_i. Low scores are assigned when the truth or falsity of all the evidence e_i becomes well-established and there is little gain in trying to prove other rules r_i.

The argument L' in equation (3.12) is the "effective" likelihood of P(h|f) and is given by

$$L' = \frac{P(h|f)}{P(h)} \quad . \qquad (3.13)$$

where P(h|f) in (3.13) is evaluated using algorithm (3.9). Unlike the constants L_i and ¬L_i, L' in equation (3.12) varies in value as information is accumulated, starting initially at 1 (i.e., indifference) and approaching either L_i if evidence e_i is true or ¬L_i if e_i is false. In other words, the score for each rule r_i: e_i --> h decreases as evidence e_i for hypothesis h is accumulated.

Generally, the PROSPECTOR algorithm performs a "best-first" search. The scoring function (3.12) provides a heuristic which evaluates all possible choices for each decision and selects the optimum. Compared with similar algorithms, PROSPECTOR provides a solution that is portable. It is easily adapted for another expert system problem, i.e., another set of facts and rules. Also, during execution, PROSPECTOR accomodates changes or additions to

the fact base, and it propagates these changes through the knowledge base. At the same time, it avoids the complex computation that is usually associated with probabilistic algorithms (Shafer and Logan, 1987). This efficiency, however, is achieved with modifications which both complicate and limit the design of rule-based systems.

Probability theory is a general calculus for evidential reasoning. Its methods can be modified and applied to a variety of problems. Modifications, however, are not straightforward, as the PROSPECTOR example demonstrates. More importantly, they are not made without introducing restrictions which limit the applicability of probabilistic techniques.

The PROSPECTOR scheme simplifies the computation of Bayes' rule using an assumption which, depending on the application, may be unreasonable or inconsistent. PROSPECTOR assumes that facts in the fact base are conditionally independent, i.e., it assumes equation (3.11). This assumption guarantees an efficient algorithm but, at the same time, it restricts the rule base to a rigid format.

3.2 CERTAINTY FACTORS IN MYCIN

The expert system MYCIN is efficient because it also assumes independence. That is, for each set of facts f = e, there is only one rule r: e --> h which bears on a hypothesis h. This assumption might complicate the design of MYCIN's rule base, except MYCIN incorporates a rule-checking procedure to verify rule independence. Given this verification, the user (i.e, the expert or knowledge engineer) can ignore the restrictions on rule design: he can add or amend MYCIN rules using a general format.

In MYCIN, a diagnosis is represented by a rule such as

```
PREMISE:($AND (SAME CNTXT GRAM GRAMPOS)
              (SAME CNTXT MORPH COCCUS)
              (SAME CNTXT CONFORM CLUMPS))

ACTION: (CONCLUDE CNTXT IDENT STAPH TALLY .7)
```

In other words,

if 1) the stain of the organism is gram-positive, and
   2) the morphology of the organism is coccus, and
   3) the growth conformation of the organism is clumps,

then there is suggestive evidence (0.7) that
the identity of the organism is staphylococcus.

(3.14)

In this example, the diagnosis (hypothesis) is 70% certain if laboratory data confirm all three facts. Since conditional independence is assumed, no combination of facts 1-3 in rule (3.14) may appear as the premise of another rule. MYCIN specifically avoids this possibility. During design, MYCIN (EMYCIN) checks each rule to ensure that evidence is not duplicated. It tests for rules of the form

e1 --> h, e1 & e2 --> h,

where one premise "subsumes" or is a superset of another. For each set of evidence, it attempts to generate a single rule. This format is not unreasonable in medical diagnosis where decisions are reserved until all the evidence is available. A decision based on partial or incomplete laboratory data has little credibility.

For MYCIN's problem, Bayes' theorem introduces other restrictions which are necessarily limiting:

1. It assumes that any application of the rules produces a single hypothesis or conclusion; ie., all possible outcomes are disjoint (mutually independent).
2. It assumes that the rule set is complete; ie., at least one of the hypotheses must be true (mutually exclusive).
3. It limits the modifiability of the knowledge base. Probability theory requires that the sum of all P(h) = 1. If a fact is added to the fact base or the

probability of an existing fact is changed, then all priors P(e) and P(h) must be recalculated.

Medical diagnosis is an instance where the assumptions of Bayes' theorem do not apply. Diagnoses (hypotheses) are not mutually exclusive since a patient may have two or more infections at once. Diagnoses are not mutually independent because a patient may have a disease that can't be diagnosed (e.g., AIDS, Legionnaire's). Also, it is difficult to collect all the prior and joint probabilities required by Bayes' rule (3.4). Even when statistics are readily available, they often do not provide a reliable or consistent base for medical diagnosis. The frequency of diseases changes from one population to another. The statistics provided by the Mayo clinic in Boston usually don't characterize the patterns of disease in Fredericton. Also, populations develop tolerances to prevalent diseases, so more reliable statistics are more quickly out-dated.

MYCIN is a rule-based system that measures uncertainty in terms of certainty factors (CFs) defined by CF: KB --> [-1,1] where KB is MYCIN's knowledge base of rules, facts and hypotheses. Each fact and rule in MYCIN's knowledge base is recorded with a CF value. For a fact f, CF(f) = 1 if f is certainly true. CF(f) = -1 indicates total falsity or disbelief. If CF(f) = 0, then nothing is known about f. Similarly, each MYCIN rule is recorded with certainty

CF(rule) whose interpretation is similar to CF(f). These CF values are subjective estimates which measure the "importance" of a fact or rule. They measure not only the probability that facts (rules) are true, but also the "risk" incurred in case they are false (Buchanan and Duda, 1983).

Unlike probability measures, CFs distinguish between degrees of belief and disbelief, i.e., CFs also range over negative values. This extension is designed to accomodate rules of the form

```
if <evidence>
  then <conclusion1>
  else <conclusion2>
       |<action>
```

If <evidence> is "true", then MYCIN asserts <conclusion1>. If it is "false", then MYCIN asserts <conclusion2> or it performs an appropriate procedure <action>. For the remaining discussion, without loss of generality, it is assumed that a MYCIN rule follows the usual if-then format.

In MYCIN, inference (3.2) is interpreted as

```
e --> h    CF(rule)
e          CF(e)
-----------------                          (3.15)
h          CF(h,e)
```

Initially, each hypothesis h is assumed to be false with CF(h) = 0. Following inference (3.15), CF(h) is changed to CF(h,e) whenever a rule r: e --> h is applied. Rule r is applied if h is a top-level or goal hypothesis, or when a fact CF(f) is changed and f matches the rule premise e. As in PROSPECTOR, the computation of CF(h,e) is complicated when there is uncertainty in evidence e or when more than one rule applies. Each of these cases is considered next.

Uncertain evidence. The premise (evidence e) of a MYCIN rule may combine a number of simple facts using any combination of logical operators, e.g., e = f1 & f2 or (¬f3). In this case, the antecedent CF(e) is computed using fuzzy set operators (Zadeh, 1965) to combine the different CF(f_i). The combination operators are defined recursively by

$$CF(f1 \ \& \ f2) = \min[CF(f1), CF(f2)] \ ,$$
$$CF(f1 \text{ or } f2) = \max[CF(f1), CF(f2)] \ ,$$
$$CF(\neg f) = 1 - CF(f) \ .$$

These operators return the expected Boolean values when the different CFs are 0 or 1.

In MYCIN, uncertain evidence e with CF(e) reduces the certainty CF(h,e) of a hypothesis by reducing the strength CF(rule), i.e.,

$$CF(rule) = CF(rule) * CF(e) \qquad (3.16)$$

If CF(rule) > 0, then MYCIN applies the rule to prove a conclusion h. If CF(rule) is negative, then MYCIN's goal is the alternative conclusion ¬h. Actually, MYCIN avoids applying rules which are only marginally effective by using a threshold CF value, i.e., rules are used only if CF(rule) > 0.2 or CF(rule) < -0.2.

Combining rules. The final evaluation CF(h,e) in inference (3.15) must account for the effect of chaining on more than a single rule. If rules r_i: e_i --> h are used to evaluate a hypothesis, then inference (3.15) must combine the individual e_i's in way that is consistent with the assumptions of the individual rules. In MYCIN, inference (3.15) is defined by the following algorithm:

```
Posterior(h, rule);
  begin
    if [CF(h) > 0) and (CF(rule) > 0] then
      CF(h,e) = CF(h) + CF(rule) - CF(h) * CF(rule)
    else
      if [CF(h) < 0) and (CF(rule) < 0] then
        CF(h,e) = CF(h) + CF(rule) + CF(h) * CF(rule)
      else
                                  C(h) + CF(rule)
        CF(h,e) = ------------------------------------------
                  1 - min[|CF(h)|, |CF(rule)|]
  end ;                                                    (3.17)
```

When a hypothesis h is initialized as a goal hypothesis, then CF(h) = 0. Following the firing of a first rule, algorithm (3.17) updates CF(h) to CF(rule) where CF(rule) incorporates CF(e) by equation (3.16). When a number of rules r_i: e_i --> h bear on the same goal hypothesis, the current certainty CF(h) is updated by (3.17) whenever facts f = e_i are added to the fact base.

Algorithm (3.17) is analogous to the one in PROSPECTOR where evidence e_i is combined by multiplying the weights (i.e., likelihood ratios) of the different rules r_i. MYCIN's algorithm is justified, in part, by the same independence assumption, so the analogy is expected. Algorithm (3.17) is also justified because it defines a function with the following desirable properties:

1. CF(h,e) is easy to compute.
2. The computation preserves closure, so CF(h,e) always returns a value in [-1,1]. Further, CF(h,e) = 1 if CF(h) or CF(rule) = 1. CF(h,e) = -1 if CF(h) or CF(rule) = -1.
3. If contradictory conclusions are combined, i.e, CF(h) = CF(¬rule), then the function of (3.17) returns CF(h,e) = 0.
4. Except at the singular points (-1,1) and (1,-1), CF(h,e) is a continuous function of CF(h) and CF(rule), increasing monotonically in each of these variables.

5. The function is symmetric in CF(h) and CF(rule). When more than two sets of evidence are combined, it returns a result that is independent of the order in which the evidence is combined.

Implementation. MYCINs inference engine performs a backward chaining procedure to "prove" a single top-level hypothesis. The top-level is a general goal that initiates a diagnosis based on user data. The top-level invariably introduces other goal hypotheses whose "proof" is required to complete the diagnosis. The MYCIN program is summarized by the following algorithm:

```
Evaluate(top);
  begin
    if Fact(top) then
      Assert(top, CF(top))
    else if Match(top, e_i --> top) then
          begin
            top = e_i ;
            Evaluate(top)
          end ;
    if CF(top) > 0.2 then
      Assert(top, CF(top))
    else if CF(top) < -0.2 then
          Assert(¬top, CF(top))
    else Ask_user(top)
  end ;
```

For each goal hypothesis, if the hypothesis is not a fact in the fact base, then MYCIN searches the rules for one whose conclusion matches the goal. If the set of rule premises matches facts in the fact base, then the goal is "proved" with certainty CF from algorithm (3.17). If the rule premise doesn't match existing facts, then MYCIN applies Evaluate() recursively. If a fact in the rule premise cannot be "proved", then, as a last resort, MYCIN asks the user.

MYCIN also applies inference (3.15) to update a hypothesis that is not a current goal. The certainty CF(h) of a hypothesis is changed whenever fact f is asserted and f bears on h. This procedure not only ensures a result which is more nearly correct, but it contributes to the efficiency of the MYCIN program. When a goal hypothesis is evaluated, because the fact base represents the most recent data, then the evidence on which the goal depends is more likely to be available. MYCIN, therefore, performs fewer searches through the rule base.

MYCIN represents an approach to evidential reasoning which tries to minimize the disadvantages of Bayesian theory. The system is admittedly ad hoc but is designed for a particular application. It follows probability theory in many respects, but it is not well-formulated. Attempts to provide a theoretical justification were unsuccessful (Buchanan and Shortliffe, 1984). It assumes independence

among its rules but some rules are unavoidably dependent. It uses measures provided by experts on a rule by rule basis and are, presumably, inaccurate and inconsistent. Despite these limitations, the MYCIN system is justified in practice. It produces diagnoses which, most often, are more accurate than those by a human expert.

3.3 THE DEMPSTER-SHAFER THEORY OF EVIDENCE

PROSPECTOR and MYCIN represent early attempts to accomodate reasoning that is inexact. The techniques apply to particular domains but fail to provide a method which may be generalized to any rule-based system. In attempting to minimize the complexity in a probabilistic approach, both introduce assumptions which limit their application. In PROSPECTOR and MYCIN, these assumptions entail restrictions that tend to inconsistencies. Generally, a system is inconsistent when it contains operations which return varying results. Practically, it is inconsistent when it introduces contradictions. A rule-based system is inconsistent if both a hypothesis h and its negation ¬h follow from the same premise. In practice, contradictions are not easy to detect since they may appear as intermediate hypotheses, i.e., they are generated by rules other than the one used to derive a final outcome.

Other algorithms for rule-based reasoning avoid the problems in PROSPECTOR and MYCIN by extending or

generalizing the theory on which they are based. Following their experience with MYCIN, Shortliffe and other principals in the MYCIN project developed an algorithm using Dempster-Shafer's rule of evidence (Gordon and Shortliffe, 1984). The rule parallels Bayes' rule for weighting decisions but it is more general. It is based on a theory of evidence (Shafer, 1979) which subsumes probability theory, so Bayes' rule is an instance of the Dempster-Shafer rule. It also applies in situations which are too general or too restricted for a Bayesian approach. For rule-based expert systems, it offers a framework which supports dynamic evidence-gathering (Hughes and Maksym, 1985). Its only limitation, it appears, is a complexity which is prohibitive except in limiting cases (Shafer and Logan, 1987).

3.4 FUZZY LOGIC IN ELOPS

Fuzzy logic is a system for rule-based reasoning that avoids the restrictions of other techniques. Fuzzy logic incorporates and extends the procedures of a multi-valued logic, so it is consistent and complete (Dubois and Prade, 1980). It avoids contradictions and exceptions. Fuzzy logic incorporates the semantics of fuzzy set theory (Zadeh, 1965), a general format for representing both ordinary and "fuzzy" sets. Fuzzy set theory also provides the basis for extending probability theory and other analytical techniques. These generalizations are formally explained in

Appendix I. The present discussion follows the previous sections and introduces the FLOPS (Siler and Tucker, 1986) fuzzy logic approach to uncertainty.

Fuzzy logic is a generalized or multi-valued logic. It extends the notion of "truth" by replacing the usual two-valued truth function T: P --> {0,1} with a multi-valued possibility function defined by poss: P --> [0,1]. For each proposition p in P, 'p is true' or 'p is possible' with certainty poss(p). Since p is more or less true or false, the set P is a set of "fuzzy" propositions.

Fuzzy logic provides a semantics for propositions with terms that are vague or ambiguous. For example,

'temperature is high' ,

'diagnosis is clamp is very likely' .

use "fuzzy" terms 'high' and 'very likely'. They are interpreted in fuzzy logic by

'x is HIGH'

'most DIAGNOSES are CLAMP'.

where HIGH, DIAGNOSES and CLAMP are fuzzy sets defined by membership functions m(), e.g., m(HIGH): HIGH --> [0,1]. The qualifier 'most' is a fuzzy measure defined by

$$\text{most}(X) = \frac{\text{sum}[\max(m(x \text{ in DIAGNOSES}), m(x \text{ in CLAMP})]}{\text{sum}[m(x \text{ in DIAGNOSES})]}$$

where sum() is the summation over all x in X, the universal set containing the union of sets DIAGNOSES and CLAMP. Both the membership function m() and the function most() define a possibility distribution poss(X) that specifies the "truth" 0 to 1 of fuzzy propositions. For each subset X_i in X, poss(x) = m(x in X_i) and poss(X_i) = most(X_i).

In a fuzzy logic rule-based system, inference is represented by

```
e --> h   poss(rule)
e         poss(e)
-----------------------------------------   .   (3.18)
h         poss(h)
```

The computation of poss(h) is greatly simplified because the function poss() represents a truth function. It is a logical function that is easily combined using logical operators and(), or(), i.e.,

$$poss(h) = and[poss(rule), poss(e)] \quad . \qquad (3.19)$$

Actually, equation (3.19) assumes that poss() is a valid "truth" function, i.e., it is composable (Zadeh, 1988). The poss() function of a fuzzy logic has this property. On the other hand, the PROSPECTOR and MYCIN algorithms are complicated precisely because, without the proper restrictions, probabilities and CFs are not composable logical values.

FLOPS (Siler and Tucker, 1986) is a rule-based language that incorporates the semantics of fuzzy logic. FLOPS implements inference (3.18) using confidence factors (CFs) defined by CF: KB --> [0,1] . KB is FLOPs knowledge base of facts and rules. Each fact and rule is recorded with a CF value between 0 and 1, represented as a value between 0 and 1000. FLOPS applies a threshold CF = 500 to distinguish between certainty and uncertainty. A FLOPS' fact (rule) is "true" if CF(fact) > 500. It is "false" if CF(fact) < 500. CF(fact) = 500 indicates indifference, i.e., a fact is neither true or false; or, it is equally true and false. This last possibility, a contradiction, is avoided in FLOPS by interpreting the function CF() as a necessity function. This function is the dual of poss() and is defined by

$$necc(p) = 1 - poss(\neg p) \quad ,$$

where p is a fuzzy proposition. This interpretation is implemented in FLOPS by using inference (3.18) for facts and rules that are true with CF > 500. This guarantees that the results of inference (3.18) are consistent (Dubois and Prade, 1987).

Uncertain evidence. FLOPS facts are combined using fuzzy logic operators for and(), or() and ¬(). These operators are defined by

$$CF(f1 \text{ and } f2) = \min[CF(f1), CF(f2)] \quad ,$$
$$CF(f1 \text{ or } f2) = \max[CF(f1), CF(f2)] \quad ,$$
$$CF(\neg f) = 1 - CF(f) \quad . \qquad (3.20)$$

The logical operators min(), max() correspond to fuzzy set operators for union and intersection. This is consistent with the logical interpretation of the operators and(), or(). More importantly, min(), max() are the only logical operators that guarantee all the theorems of ordinary logic; i.e., if T is a theorem in a Boolean logic then T is a theorem in fuzzy logic (Dubois and Prade, 1980).

A FLOPS rule r: e --> h is "fired" if there are facts f which match the rule premise e. If e matches a simple fact, i.e., e = f, then CF(e) = CF(f) in inference (18). If e represents a combination, then operators (3.20) are applied. Actually, FLOPS uses a restricted rule format, so the premise e can only represent a conjunction of facts in the fact base, i.e., e = f1 & f2; then CF(e) is computed by

$$CF(e) = CF(f1 \ \& \ f2) = \min(CF(f1), CF(f2)) \quad .$$

Following equation (3.19), the result of inference (3.18) is also computed using the min() operator: i.e.,

$$CF(h) = \min(CF(rule), CF(e)) \quad .$$

In other words, inference (3.18) establishes an equivalence between the proposition h and the conjunction and(rule,e), which is the logical interpretation of modus ponens.

FLOPS' implementation of inference (3.18) may now be summarized by

```
e --> h   CF(rule)
e         CF(e) = min(CF(f_i))
-------------------------------------          (3.21)
h         CF(h) = min(CF(rule),CF(e))
```

The value CF(rule) is never evaluated in FLOPS. This value is provided by the user when a rule is defined. A typical FLOPS rule is

```
rule 800 (trend ^bias > (1.5,1.0,0))
         (diagnosis)
    -->
      modify 2 ^bias.high ^dx.connector ;
```

In other words,

```
if bias is high
  then diagnosis is connector with CF = 0.8
```

If the premise 'bias is high' is certain, i.e., CF(e) = 1, then the conclusion 'diagnosis is connector' is asserted in the fact base with certainty CF(rule) = 0.8. If the premise is less than certain, then the conclusion is asserted with CF(h) equal to the least of CF(e) or CF(rule). Intuitively, a conclusion cannot be more certain than the certainty CF(rule). Also, a conclusion cannot be stronger than the evidence on which it is based.

Incidentally, CF(f) for f:'bias is high' is evaluated over a possibility distribution poss(bias in HIGH). The triple (1.5,1.0,0) defines a fuzzy set HIGH whose characteristic function is a normalized Gaussian curve G centered on 1.5 with deviation 1.0 and relative error 0. The curve G is defined over all reals, ie., G: R --> [0,1], so each real bias b is associated with a characteristic CF(f(b)) = G(b). The function f(b) indicates that b is a bias value assigned to the proposition f:'b is HIGH'.

Combining rules. An algorithm for inference (3.21) must also account for the effects of chaining on two or more rules. In FLOPS, if the rule base contains several rules r_i: e_i --> h which bear on the same hypothesis h, the effects of the different e_i are combined using the max() operator, i.e.,

$$CF(h) = \max[CF(r_i), CF(e_i)] \quad . \qquad (3.22)$$

In other words, the several rules r_i are interpreted as a disjunction or() over the conjunctions and(h, e_i). This interpretation places an additional restriction on FLOPS' rules. These restrictions, however, are not so limiting as they may appear. The FLOPS format is the standard "conjunctive normal form" (CNF) of logical calculus, i.e., the knowledge base has the form or(and()). Logical calculus provides a method for translating any logical statement to FLOPS CNF format. Computationally, CNF is the most

efficient knowledge base representation (Robinson, 1965). The result is a FLOPS format that is both powerfully expressive and computationally simple.

Implementation. FLOPS is a rule-based system using a forward chaining algorithm, i.e., FLOPS inference engine is data-driven. In each cycle, a FLOPS rule is fireable if the rule premise matches facts in the fact base. If more than one rule is fireable, FLOPS selects the rule which returns the highest CF(h). If two rules return the same CF(h), then FLOPS resolves the conflict using a standard OPS algorithm. In each cycle, the "best" rule is fired. Other rules are saved on a rule stack or agenda and they are executed after rules of greater certainty are applied.

Inference (3.21) is used in FLOPS to evaluate the CF(h) of each hypothesis and to select among competing hypotheses. It provides the control procedure for each cycle through the FLOPS rules. Chaining on several rules r_i: e_i --> h is implemented implicitly, i.e., the max() operation of equation (3.22) is not performed directly in FLOPS; rather, for each hypothesis, CF(h) is only changed by rule r_i if the new CF is greater than the current CF. This is consistent with the definition of equation (3.22). FLOPS also recognizes the redundancy of computing CF(h) for every rule r_i.

In general, FLOPS fuzzy logic provides a format for rule-based reasoning that applies to "common-sense" or "approximate" reasoning, but it also applies to exact reasoning. Also, the algorithm is computationally simple since, for each application of rules, it operates on a small number of rules (i.e., rules r_i where the premise e_i matches facts f_i). At the same time, FLOPS provides a representation that accomodates a variety of data structures. FLOPS operators apply to ordinary or fuzzy sets, fuzzy probabilities and ordinary or fuzzy logic.

Compared with the other systems in this section, FLOPS fuzzy logic has at least two advantages. First, it is consistent (Dubois and Prade, 1987), so it avoids conclusions which are contradictory. Also, it is more general than probabilistic or other reasoning because it extends to fuzzy or ambiguous statements such as 'bias is high'. If HIGH is defined by a probability distribution, then this distribution may be used to define a fuzzy set over which the fuzzy operators may be applied (Civanlar and Trussel, 1986). The reverse is not true. The definition of a fuzzy set is too general to guarantee the restrictions which are implied by probability theory (Zadeh, 1988).

FLOPS fuzzy logic offers other advantages which are implied in the preceding discussion but not elaborated. As a further comparison, Bonissone and Brown (1987) list the

"most important requirements that the ideal representation of uncertainty should satisfy". In Table 1, the different systems in this section are compared against the following criteria:

1. Combination rules should not assume independence among sets of evidence (e.g., probabilities in PROSPECTOR and CFs in MYCIN assume independence).
2. Combination rules should not assume that hypotheses are exhaustive or exclusive (e.g., probabilities in PROSPECTOR entail both assumptions).
3. For each hypothesis, the representation should make explicit the amount of supporting evidence and the amount of non-supporting evidence. (CFs in MYCIN and FLOPS provide this representation.)
4. The user should be able to describe the uncertainty of information at the available level of detail, i.e., heterogeneous information granularity. (FLOPS fuzzy sets may include any number of elements.)
5. There should be some measure of consistency to detect potential conflicts and to identify the sources of conflicts. (FLOPS guarantees consistency by interpreting CFs as necessity measures.)
6. There should be an explicit representation of ignorance to allow the user to make non-committing statements, or to express his lack of conviction about some choices. (In FLOPS, ignorance is represented by CF = 500; in MYCIN, by CF = 0.)

7. It should distinguish between a conflict in information, i.e, an inconsistency, and ignorance about the information. (FLOPS fuzzy logic avoids inconsistencies. On the other hand, if MYCIN derives hypotheses h and ¬h, it assumes ignorance by setting CF(h) = 0.)
8. It should measure not only uncertainty in the information, but also uncertainty in the measure itself, i.e., second-order uncertainty. (FLOPS fuzzy set membership values are recorded with CFs.)
9. The representation should appear natural to the user. (FLOPS fuzzy logic is "approximate" reasoning.)
10. The syntax and semantics should be closed under the rules of combination. (FLOPS fuzzy logic is complete under the logical operators min and max and under FLOPS inference.)
11. It should be feasible to make pairwise comparisons of uncertainty, because these comparisons (rankings) are needed for performing any kind of decision analysis. (FLOPS operators min and max define a group structure which is partially ordered.)

Among the different approaches for representing uncertainty, FLOPS fuzzy logic satisfies all of the criteria (Table 1). PROSPECTOR's modified Bayesian approach satisfies only three. MYCIN is slightly less restrictive. Only Dempster-Shafer's method offers the same flexibility as

TABLE 1

Comparison of Uncertainty Representations

Criteria:	Representation: PROSP'R	MYCIN	D-SHAFER	FLOPS
1. Dependencies	0	0	1	1
2. Exclusivity, exhaustivity	0	1	1	1
3. Disbelief	0	1	1	1
4. Heterogeneity	0	0	1	1
5. Consistency	0	0	1	1
6. Ignorance	0	1	1	1
7. Consistency vs ignorance	0	0	1	1
8. Second-order uncertainty	0	0	1	1
9. Natural	1	0	1	1
10. Closure	1	0	1	1
11. Rankings	1	1	1	1

Zadeh's, but it is limited by its complexity, i.e., it is computationally more expensive (Shafer and Logan, 1987).

Chapter 4

RESISTANCE TEMPERATURE DETECTORS

4.1 DESCRIPTION

Resistance temperature detectors (RTDs) are electrical sensing devices. RTDs are used in the nuclear power industry to monitor temperatures in the heat transport system (HTS) which services the reactor core (Figure 5). RTDs provide temperature data used by the control monitor, a Digital DCC-X mainframe computer, not only for control information, but to detect HTS flow obstructions or abnormal reactor activity.

The heat transport system (HTS) is a collection of 380 channels that circulate heavy water (i.e., deuterium oxide) between the reactor and boilers. Each channel of the HTS is confluent with the fuel cells containing the fuel bundles which power the nuclear reactor (Figure 6). At full power (FP), "cold" water at 260 degrees Celcius enters the core where it is heated to 310 degrees. It exits to the boilers where it provides the steam to power the plant's turbines and returns to the core at 260 degrees. Control procedures guarantee the "steady state" operation of the reactor and respond to temperature and pressure changes in the HTS. RTDs

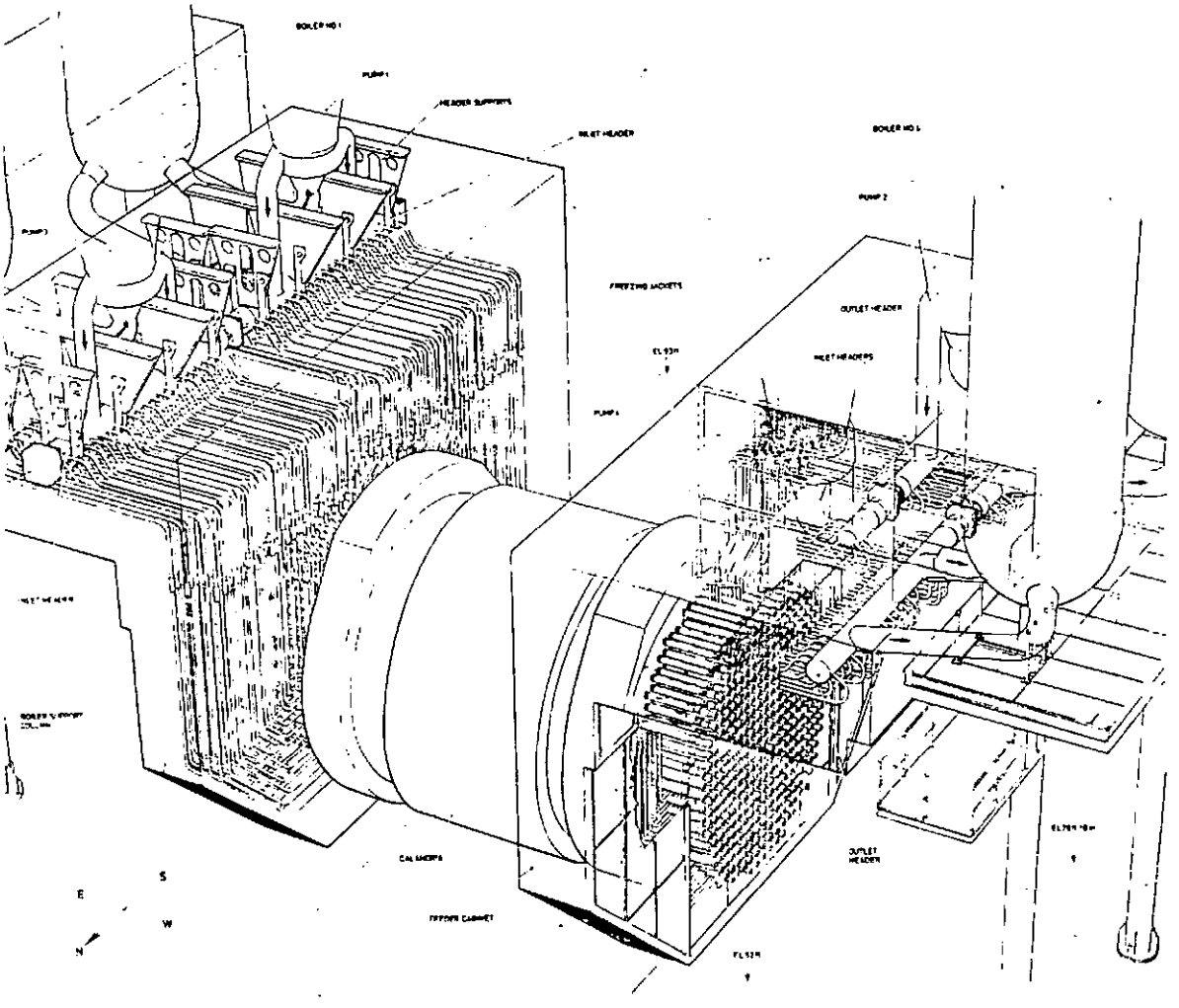


Figure 5: Heating transport system (HTS) of Candu reactor.

monitor the temperature of the HTS where it exits from the reactor core. At FP, a "normal" RTD reading is 310 degrees.

An RTD is an electrical device whose resistance responds to changes in temperature. As the channel temperature changes, the RTD's resistance also changes to produce a voltage value that may be translated directly into an actual temperature. For each degree of temperature between 0 and 320 degrees C, an RTD produces a corresponding voltage between 100 and 900 mv (Table 2). The voltage is fed to an analog-digital converter connected to the control monitor. The monitor scans the 380 channels at eight-second intervals (Figure 7) and tests for temperature readings that are "irrational". It sounds a warning or an alarm if a channel temperature is outside a specified range, ie., 4 or 7 degrees higher or lower than its neighbors.

An RTD consists of a platinum coil element supported by a connecting rod of varying lengths. The different lengths facilitate the RTD's positioning at different distances from the containment wall which surrounds the reactor core. Outside the containment, the RTD is connected to other electrical devices by a male-female-type "cannon" connector. Inside the containment, the RTD is held by a "guide tube" that is continuous with a thermowell or "clamp" attached to the channel pipe (Figure 8 a). The clamp supports the RTD element (i.e., the platinum coil) and provides an effective

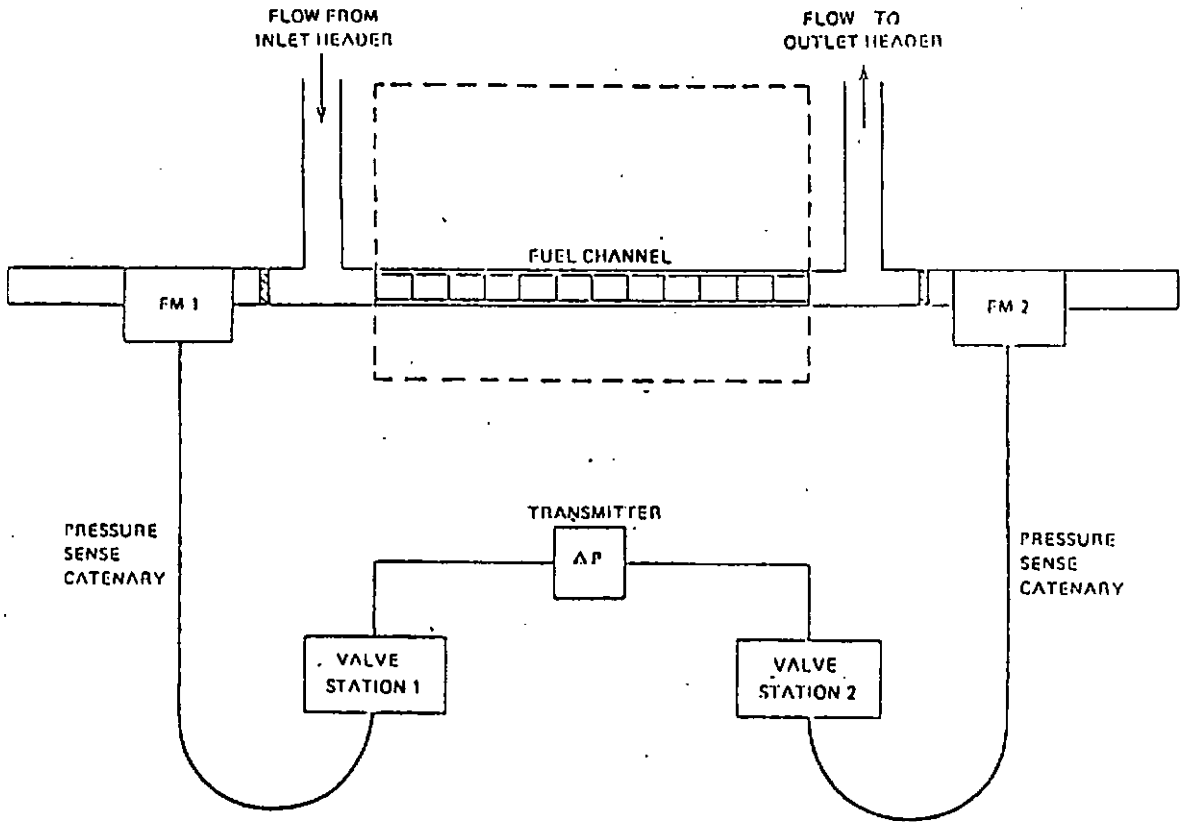


Figure 6: HTS single channel showing fuel cells.

TABLE 2

RTD temperature-voltage conversion.

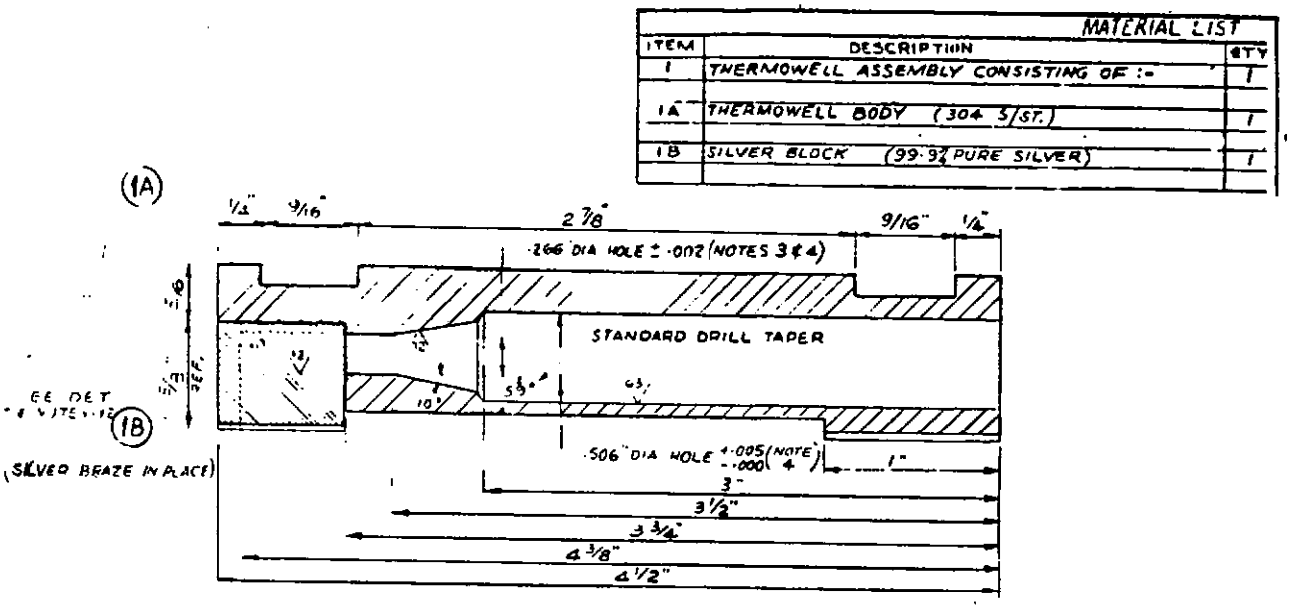
T, deg C	E, mV	T, deg C	E, mV
0	100.00	170	553.22
10	128.66	180	577.78
20	157.06	190	602.12
30	185.19	200	626.25
40	213.06	210	650.17
50	240.68	220	673.88
60	268.04	230	697.38
70	295.16	240	720.68
80	322.03	250	743.78
90	348.65	260	766.68
100	375.03	270	789.38
110	401.18	280	811.88
120	427.09	290	834.20
130	452.77	300	856.32
140	478.22	310	878.25
150	503.44	320	900.00
160	528.44		

contact with the channel through a highly conductive silver bridge. There are two guide tubes associated with each clamp to guarantee an optimum contact. The channel pipe is nearly circular and, because the clamp is mechanically attached, there is no certainty that its contact with the channel is everywhere complete. The "spare" guide tube allows that, if one side of the clamp misses the channel wall, contact is still maintained on the other.

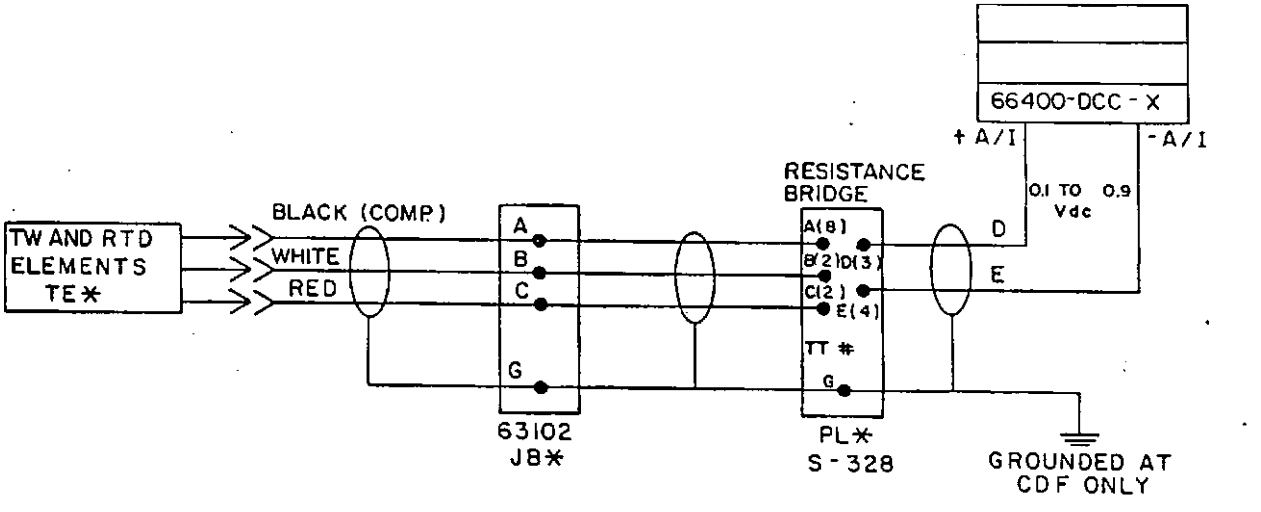
87 NOV 19 DCC X 09:33:58 CHANNEL OUTLET TEMPERATURES PRINTOUT

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
A									309	308	309	310	309	309									A
B						309	308	308	309	309	308	310	309	309	308	308	308						B
C					309	309	309	308	309	308	309	308	308	308	308	309	309	310					C
D				310	309	311	309	309	310	310	309	308	309	309	309	313	309	309	309				D
E			310	310	310	310	310	309	309	309	309	309	310	309	309	307	309	310	310	309			E
F			310	310	310	310	309	310	310	310	309	308	309	309	309	309	310	308	308	308			F
G		309	310	310	309	309	309	309	309	309	309	310	309	310	309	310	311	309	313	309	309		G
H		306	308	309	310	307	309	310	309	311	309	309	309	309	310	309	309	309	310	310	310		H
J	308	309	309	307	308	310	309	309	309	309	308	309	308	309	307	309	308	310	309	308	313	308	J
K	310	309	309	308	309	309	309	308	309	309	309	309	309	308	307	309	310	308	309	305	310	311	K
L	310	309	309	309	310	309	308	309	306	309	309	310	309	308	310	314	310	311	310	308	309	309	L
M	309	309	309	311	308	310	310	309	309	308	310	309	309	307	309	310	309	309	308	308	309	311	M
N	310	305	309	312	309	294	308	309	308	310	308	308	309	310	309	309	309	310	308	309	308	309	N
O	310	310	309	310	310	309	310	310	310	308	309	308	309	308	310	310	310	309	308	309	308	308	O
P		309	309	308	311	310	309	309	309	309	308	310	308	309	308	310	309	309	310	308	309		P
Q		310	309	308	309	310	309	310	310	310	310	307	308	309	309	307	309	308	308	309	309		Q
R			309	309	310	308	310	309	309	311	308	309	307	309	308	310	309	308	308	308			R
S			309	310	310	309	309	309	310	305	309	307	309	308	310	309	309	309	308	312			S
T				308	308	310	309	310	311	309	308	309	307	309	308	309	308	309	309				T
U					310	308	309	308	309	308	308	308	308	308	309	309	309	308					U
V						307	309	308	308	309	309	309	309	308	308	308	308						V
W									307	308	308	307	309	307									W
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	

Figure 7: Control monitor display of RTD data.



(a)



(b)

Figure 8: Resistance Temperature Detector (RTD).

Thermowell or "clamp" (a) attaches to HTS channel and holds a platinum coil element in one of two drill tapers or bores. Each bore is continuous with guide tube that supports the connecting rod between the coil and the RTD electrical circuit (b).

4.2 RTD MAINTENANCE

RTD failures during normal operations create critical alarm conditions which may result in a plant shutdown. During normal operations, RTDs provide temperature readings for the 380 channels of the HTS (Figure 7). The control monitor polls the RTD temperature data and tests for readings that are "irrational". The control monitor signals a warning if an RTD shows an error of more than 4 degrees. It sounds an alarm if the error is more than 7 degrees. An alarm halts normal operating procedures until the problem is identified and cleared. A failed RTD can generate an alarm that is costly and unnecessary. RTD maintainance, therefore, is designed to eliminate alarms which are caused by RTD faults.

RTD faults are either electrical or mechanical. Electrical faults occur in electrical devices (junction boxes, resistance bridges, the analog input device) which are associated with RTDs (Figure 8 b). Electrical faults are well-defined by the electrical characteristics of the RTD circuit. They are detected and diagnosed using simulation software and hardware. Mechanical faults are not so well-defined. There is no cause-effect model to explain RTD mechanical failures because the same symptoms identify more than one fault. Only direct inspection or testing can provide a certain diagnosis, but RTDs are located close to

the reactor core and inspection exposes staff to higher than normal levels of radiation.

A mechanical fault occurs when a clamp or other RTD component loses contact with the RTD circuit. An RTD clamp establishes a critical contact between the RTD element and the channel surface. The clamp is attached to the channel pipe by two U-type bolts and may be misplaced if the bolts are not sufficiently tight or not properly positioned. In either case, the RTD loses direct contact with the channel and produces a reading that is incorrect. Contact may also be lost through a poor connection provided by the guide tube. If only one side of the clamp touches the channel surface, then only one of the guide tubes completes the circuit between the channel and the RTD. If a guide tube is damaged, then contact is lost between the RTD and the clamp.

The clamp and guide tube present problems that are obviously mechanical. They are caused when the clamp or guide tube are physically separated from the channel. The cannon connector, on the other hand, is an electrical device that could be metered (e.g., by a voltmeter), especially since the connector is more accessible than other RTD components. It is positioned outside the containment wall. Testing is not done, however, because of the connector's sensitivity. A well-functioning connector can be damaged by the test procedure or by simply disconnecting the device. A

connector problem is identified from the same data which identifies a clamp or guide tube fault and, for this reason, it is classed as a mechanical problem.

There are six types of RTD mechanical faults depending on the RTD component and on the action that is recommended for its repair (Table 3).

TABLE 3

RTD Mechanical Faults

Component:	Repair:
1. Clamp is loose or poorly placed	Rework clamp
2. Cannon connector contact is deteriorated	Replace connector
3. Guide tube is suspect	Place RTD in spare guide tube.
4. Clamp or guide tube is suspect. RTD is already completely reworked. Performance is acceptable.	Monitor RTD for deterioration. RTD shows best possible reponse.
5. RTD performance is poor but acceptable.	Monitor RTD for deterioration.
6. RTD is failed	Replace RTD.

The expert system of this thesis identifies and diagnoses the faults listed in Table 3 . The list is short and the problem is apparently simple. In fact, the problem is

complicated by a lack of definition. The different classes of faults display the same symptoms. RTD performance is characterized by only two measures. The same measures both identify RTD malfunctions and provide the data on which RTD faults are diagnosed. Two measures are insufficient to distinguish among the different types of RTD faults.

4.3 RTD PERFORMANCE DATA

RTD faults are identified during three major nuclear plant routines. During normal operations, a poorly functioning RTD is identified by a reading that is too high or too low. The cause of a failure is diagnosed based on other measures which characterize an RTD's performance. These measures quantify an RTD's response to temperature change and the error or bias of the RTD temperature reading. Response and bias data are collected during three routines which are regular, mandatory procedures at nuclear power plants in Canada. The three routines are

1. an annual startup,
2. daily refuellings, and
3. monthly deratings.

4.3.1 Startup Bias

RTD performance is characterized by the absolute error or bias in RTD temperature data. Bias measures the difference between the actual channel temperature and the temperature

indicated by the channel RTD. Bias data is collected following an annual outage when the reactor is shut down for yearly maintainance and repairs. Following an outage, the reactor is "restarted" by stepping the power from 0 to 100% FP. This operation is called startup. RTD readings at different power levels are measured and compared with expected readings from steady-state calculations.

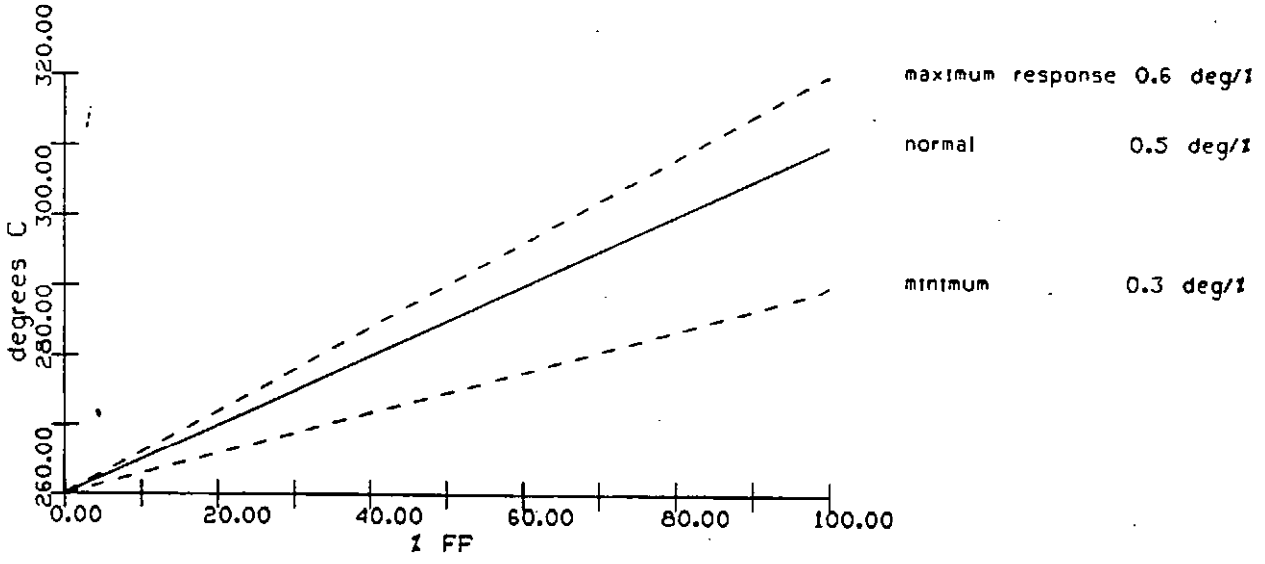


Figure 9: RTD startup response.

Startup provides two bias measures based on two methods of calculation. The accuracy of the calculation depends on an RTD's "saturation", a characteristic defined by the method which is used. Generally, an RTD is "seeing saturation" when it fails to properly respond to an increase

in the channel temperature. A normally functioning RTD records a 0.5 degree temperature increase for each one per cent increase in power (Figure 9). At power levels close to FP, an RTD's performance may deteriorate because of the extremely high temperatures at these levels.

Method 1. Bias is calculated from RTD temperatures recorded for the final two power steps. This method is used if the RTD is "saturated" as defined by the following relation:

$$\text{Temp(Last, RTD)} - \text{Temp(Last-1, RTD)} < 0.3 \text{ degrees} \quad .$$

Then,

$$\text{average(Sat,RTD)} = \text{average(Temp(Last,RTD),Temp(Last-1,RTD))} \quad ,$$

$$\text{Bias(RTD)} = \text{Temp(Sat,ROH)} - \text{average(Sat,RTD)} \quad , \qquad (4.1)$$

where Temp(Sat,ROH) in (4.1) is the expected temperature at the outlet header (OH) of the HTS at full power. Temp(Sat,ROH) is calculated based on OH pressure and flow data.

Method 2. Bias is calculated using a linear regression on RTD readings taken below 94% FP. Extrapolation provides an expected temperature reading Temp(Exp,RTD) that is compared with the actual RTD reading Temp(FP,RTD) at 100%

FP. Figure 10 demonstrates this comparison. In Method 2, an RTD is "seeing saturation" if

$$Temp(Exp,RTD) - Temp(FP,RTD) > 0.5 \text{ degrees} \quad . \tag{4.2}$$

If relation (4.2) applies, then the bias at FP is calculated as the difference between the two temperature values, i.e.,

$$Bias(FP,RTD) = Temp(Exp,RTD) - Temp(FP,RTD) \quad .$$

Bias data from Method 2 is further processed by averaging the bias at FP with a bias calculated at 0% FP. At 0% FP, an RTD temperature is compared with the temperature expected at the HTS inlet header (IH); ie.,

$$Deviation(RTD) = Temp(0,RTD) - Temp(0,RIH) \quad .$$

Deviations from the IH temperature are then averaged over all RTDs to provide a standard bias deviation:

$$Std_bias = average(Deviation(rtd)) \quad .$$

Then

$$Bias(0,RTD) = Deviation(RTD) - Std_bias \quad .$$

Finally, for each saturated RTD, a Method 2 bias is computed as the average error between the readings at 0% and 100% FP, i.e.,

$$Bias\ (RTD) = average(Bias(FP,RTD),\ Bias(0,RTD)) \quad .$$

For an unsaturated RTD, a bias is computed from the standard error:

$$Bias(RTD) = Bias(0,RTD) + Std_bias \quad .$$

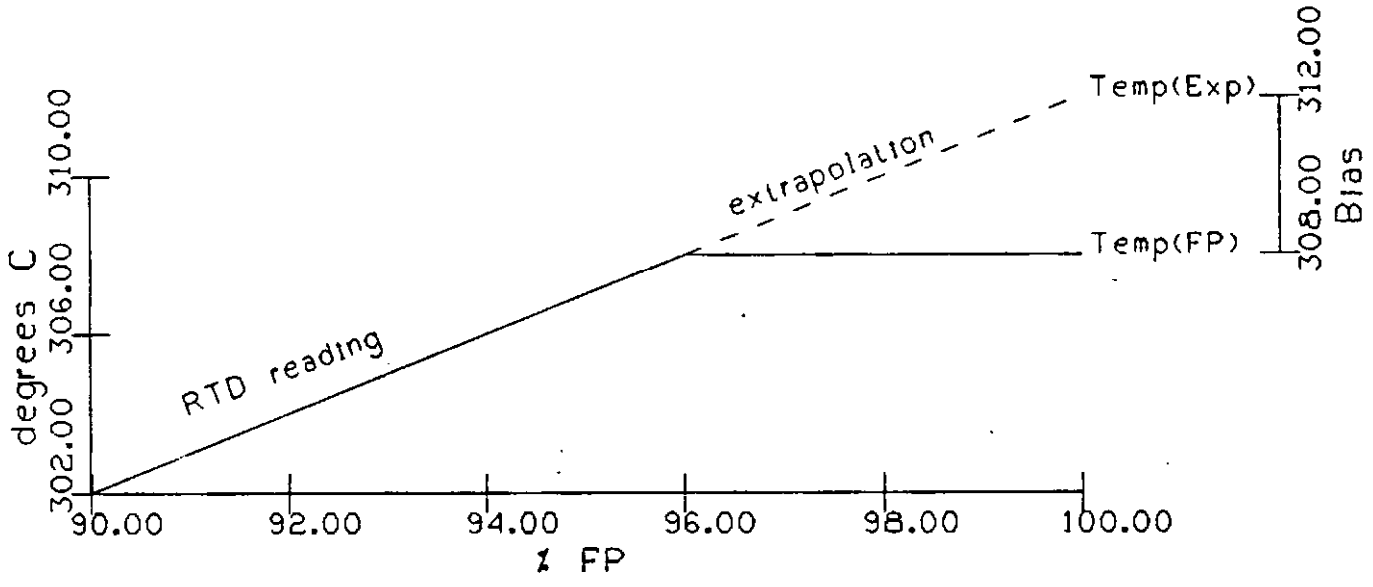


Figure 10: RTD startup bias. Method 2.

4.3.2 Refuelling Response

RTD performance is also characterized by its response. Response measures the rate at which an RTD responds to channel temperature changes. Response data is collected when the fuel cells associated with an HTS channel are "refuelled" (Figure 6). The sudden temperature changes caused by the removal and insertion of fuel bundles provides

an opportunity to test the response of channel RTD. Removing active fuel bundles results in a sudden decrease (i.e., approximately 20 degrees in 2 minutes) in channel temperature. Channel data is plotted for a specified period (i.e., 15 minutes) and, because operating conditions are strictly controlled, refuelling produces a characteristic curve or "trend" (Figure 11). A trend provides a reliable, visual indication of RTD performance. An RTD shows "good" response if the maximum temperature change (approximately 30 degrees) is registered within 10 minutes and if 2/3 of this change (20 degrees) is registered within 2 minutes.

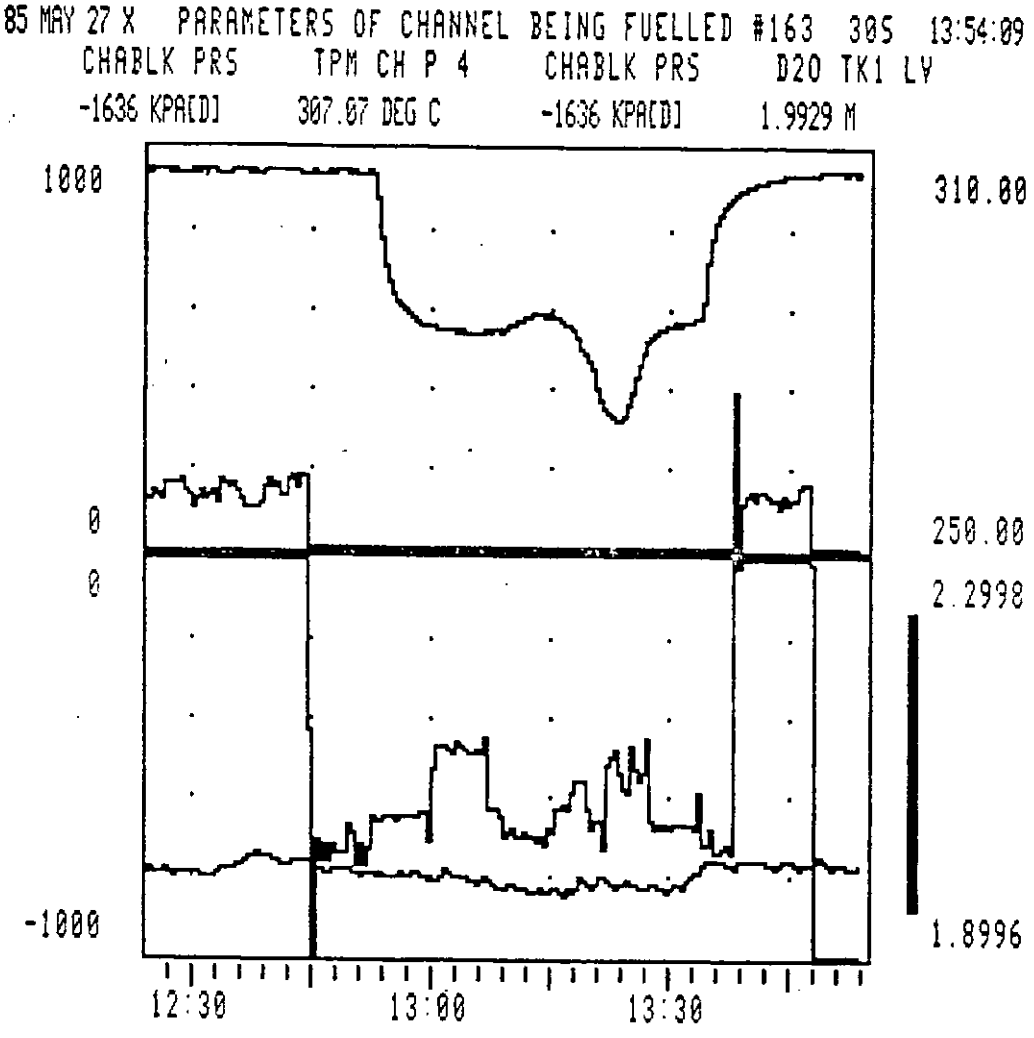


Figure 11: RTD refuelling response (trend).

4.3.3 Derating Response

Faulty or suspect RTDs are also identified during the monthly "derating" when reactor power is reduced from FP. Derating is a mandatory procedure designed to test RTD response at power levels where these devices are more responsive. At FP, RTDs are exposed to extremely high temperatures. They operate in an environment where their reliability is no longer guaranteed. Derating is designed to provide a rough check of RTD performance and to identify problems that can't be detected during normal operations.

During derating, reactor power is reduced to 80% FP. The reactor is "ramped" or stepped back to FP by incrementally increasing power levels. The performance of an RTD during the power ramp provides a coarse check of the RTD's response. RTD readings at different power levels provide the data for calculating RTD bias.

4.4 RTD DIAGNOSIS

Bias and response data are used to diagnose RTD faults and to plan RTD maintainance and repairs. Serious RTD faults may be identified directly if RTD bias is significant or RTD response is slow. Serious faults are diagnosed and repaired at the time they are detected (e.g., during derating). Otherwise, maintainance is a planned procedure. It is

performed once a year during a mandatory plant "outage" when power generation is shut down. Maintainance of RTDs and other devices close to the reactor core requires that the reactor is "cold". Outage is carefully planned to limit the time lost from power generation and to reduce the amount of radiation to which personnel are exposed.

An RTD work plan is prepared for each annual outage based on bias and response data. Reponse curves or "trends" from channel refuellings (Figure 11) are visually inspected and RTDs that show a "poor" response are flagged as suspect. Suspect RTDs are analyzed and diagnosed with reference to RTD biases and work histories (Callister, 1987).

RTD diagnosis is a typically "expert" skill. It requires decisions based on common sense and experience. RTD data are inexact and unreliable and are subject to interpretation. The diagnostic model (i.e., causes and effects) is not well-formulated so the expert must choose among competing hypotheses. In general, the problem presents ambiguities which complicate a computer solution.

RTD diagnosis is not well-defined. The diagnosis is based on subjective characterizations of RTD data and not on exact measures. During refuelling, RTD response is plotted as "trends" but, because of the volume of data, the numerical values on which trends are based are not recorded. Three to five channels are fuelled each day, so between

750-1250 trends are inspected and compared prior to the annual outage. The graphical data could be more closely analyzed to provide a quantitative measure of response but, because of the number of channels, this is not done. It is also unnecessary because, based on experience, the RTD expert can quickly identify trends which are abnormal.

Bias data is more or less reliable depending on the accuracy of bias calculations. Startup bias is more exact but it doesn't indicate an RTD's current performance. Startup bias data is almost 12 months old when outage planning is done. Bias is not a reliable measure of RTD performance because it is inconsistent. Bias may drift in either direction, so an RTD that reads high in one instance can read low in the next.

Derating bias is less exact because derating is less strictly controlled. The ramping process is performed by a human operator who is directed to provide only a coarse check on RTD data. Derating bias is based on only 20 readings (ie., 80-100% FP) but, depending on the operator, the data may be polled taking more or less time.

RTD diagnosis is not well-formulated. The unreliability of RTD data forces the RTD expert to make decisions that are highly subjective. Based on a visual inspection of response data, the expert identifies RTDs which are most likely faulty. Based on response data and bias, he then rates the

certainty that a suspected RTD is actually at fault. A final diagnosis, therefore, is more or less certain, depending on the expert's weighting of the data on which the diagnosis is based.

RTD diagnosis is not exact. It identifies only the most likely cause for an RTD fault. Bias and response together indicate a possible cause (or causes), but the relations between symptoms and faults are not explicit. A high bias identifies an electrical problem, possibily a cannon connector, but maybe another electrical component. A low bias may be caused by either a faulty connector or a loose clamp. A final diagnosis is tentative because it forces the expert to make a judgement based on possibilities. The judgement is, at best, a conjecture that identifies a possible cause based on data that is more or less certain.

Conflicting diagnoses are resolved based on an RTD's work history. If the expert must choose between a clamp or guide tube, he may reason that, if a clamp has been reworked, then a clamp problem may be minimized. He may also make a decision based on his familiarity with a particular device. For example, a top row RTD is more likely to show a guide tube problem because it was damaged during installation. Again, the reasoning is tentative and relies on the expert's intuition or common sense.

4.5 AN RTD EXPERT SYSTEM

This thesis introduces a prototype system for RTD diagnosis. It duplicates the skills of an expert RTD technician using an expert system approach. It provides a computer solution to a problem that is complex because of the number of decisions it involves. The diagnosis procedure rates and analyzes the performance of 380 RTDs.

An expert system model is appropriate because RTD diagnosis is based on decisions which are "heuristic" and not well-defined. The heuristics are the subjective, sometimes tentative, decisions learned by experience. They are rules of thumb which are applied to a highly specialized domain. They apply not to the wider field of nuclear instrumentation, but to the narrow field of temperature detector mechanical failures.

An expert system approach is the only basis for computer-assisted diagnosis of RTDs. The problem domain is not proscribed by formal rules, so standard algorithms do not apply. There is no complete set of rules that relates RTD behavior and RTD faults. Bias and response provides partial information which explains RTD mechanical faults but there are no other characteristics to complete a model of cause and effects.

Alternatively, RTD diagnosis might be approximated by statistical or other data that would provide additional

information for the decisions on which a diagnosis is based. Unfortunately, the data isn't available. The monitoring computer polls a massive amount of such data but, because of the volume, it is not stored but is lost immediately after processing.

RTD diagnosis is "modelled" by a rule-based expert system. The decisions made by the RTD expert are formulated as simple rules. A rule-based format is modular so that rules can be added, deleted or amended without changing other system components. This format simplifies the definition of an expert problem. It assists in the development of an "algorithm" where an algorithm doesn't exist.

The RTD expert system is implemented using fuzzy logic (FLOPS, Siler and Tucker, 1987). The fuzzy set representation quantifies the subjective nature of many of the diagnostic rules. It also provides a well-defined certainty measure for rating a diagnosis based on these decisions.

In the absence of more precise data, a fuzzy logic expert system is the only approach to a computer solution for RTD diagnosis. The approach is justified because it works. It produces an application that duplicates the skills of a human expert applied to a complex problem. The solution may appear ad hoc because it uses measures that are ad hoc and

arbitrary. These measures, however, are a reliable quantification of RTD behavior because they summarize RTD performance based on more than five years' observation by an expert technician. In this sense, they provide an approximation of the statistical behavior of RTDs (Duda et al., 1985) that may or may not be improved using techniques that are more exact.

Chapter 5

KNOWLEDGE ENGINEERING OF AN RTD EXPERT SYSTEM

The main effort of this thesis involves building the knowledge base for an expert system (XpeRTD) to diagnose RTD faults. This task is identified as the most difficult component of expert system design (Hayes-Roth et al., 1983). The process is typically called "knowledge engineering" and includes the identification of an expert problem and the design of a computer solution. Specifically, the task characterizes the problem of RTD diagnosis in terms of a knowledge base of rules and facts. This requires identifying what characteristics of RTDs and other devices are relevant to the diagnosis and, based on these characteristics, specifying how a diagnosis is made.

The main task of engineering the knowledge for XpeRTD is complicated by the ambiguities inherent in some of the facts. While certain lines of reasoning lead to specific diagnoses, other conclusions aren't so clear. For example, if a device has a low bias and its response is poor, then, if the RTD's performance history is poor, then a decision can be made based on its work history.

More graphically,

```
IF (bias OF rtd) IS low
  AND
   (response OF rtd) IS poor
  AND
    NOT (performance_history OF rtd) IS acceptable
THEN
    check(work_history)
```

In this example, if the performance history is acceptable, then the decision isn't clear. There is no clear decision because low bias may be due to either a mechanical or an electrical fault. Without the additional information provided by a performance history, there is no basis for a decision. Given such a case, a system designer may choose to ignore it or he can work with the expert to try and develop a decision. This development is assisted by using a logical format to define the problem and by using additional evidence to refine the problem data. The logical format is simply a rule-based expert system. Such a system is easily modified by additions or deletions to the rule set. This modification can be made without altering the control structure or the other components of the system. Probabilities or other certainty measures can assist a decision that is based on ambiguous data. Techniques for reasoning with uncertainty are also a standard part of expert systems.

A rule-based expert system provides a well-established model for knowledge representation and knowledge processing. This suggests that a prototype can be designed based on existing models, but this is not the case. The model provides a convenient format to represent the problem data, but there are no standard techniques for translating a body of knowledge into an actual set of rules and facts. Knowledge representation and knowledge engineering remain the most active and most difficult areas of research related to expert systems. Given these problems, the present system is important because it details a successful attempt to engineer an expert domain.

Knowledge engineering is characterized by a number of distinct steps which apply to any problem in software engineering. These include goal identification, concept generation, implementation, testing and debugging. These tasks are easily explained in general terms, but they are never easy in practice. Especially in engineering expert systems, the different steps merge into a single process for which there are no simple methods or techniques.

5.1 PROBLEM IDENTIFICATION

A rule-based system provides a convenient format for representing data and procedures. Unfortunately, it also restricts the solution of a problem by imposing a method which is invariable. The algorithm employed by the inference engine specifies the general framework for every problem. The fact base and rule set, therefore, must be carefully designed to account for this implementation. These difficulties may be explained by comparing Prolog's rule-based implementation of a standard algorithm (e.g., quicksort) with Pascal's, for example. Prolog's solution bears little relation with Pascal's, since it necessarily incorporates the pattern-matching facility of Prolog's inference engine. Any Pascal algorithm may be programmed using Prolog, but the translation usually involves a difficult transformation because Prolog's "main procedure" is invariant.

The development and design of a diagnostic computer system is somewhat simplified because the diagnosis problem is well-defined. It is defined by a number of general algorithms, including rule-based ones, which assists the organization and representation of similar problems.

5.1.1 The Diagnostic Model

Current research describes four practical approaches to diagnosis:

1. Statistical analysis using discriminant functions or other classification tools to identify classes of faults (de Kleer et al., 1987);
2. Probabilistic Bayesian models (Ben-Bassat et al., 1980; Reggia and Peng, 1987);
3. Simulation models using functional rules to relate cause and effect (Davis, 1983); and
4. Rule-based models where rules explicitly relate or chain a set of symptoms to their implied diagnosis (Reiter, 1987).

XpeRTD uses a rule-based approach to diagnosis. Other models were not considered based on the initial definition of the problem. Specifically, there were insufficient data to provide a statistical analysis or to generate a simulation model. From the beginning, it was obvious that such a system for RTD diagnosis would depend on the expert's diagnostic experience, on knowledge that is not well-defined but which might be duplicated in terms of a heuristic set of rules.

The XpeRTD system is typical of a rule-based diagnostic system. In general terms, it selects a class of faults depending on a small set of symptoms. It incorporates a set of rules which relate each set of symptoms to one class of faults. Because the same symptoms apply to all fault

classes, the problem incorporates heuristics, the judgemental knowledge which the expert uses to distinguish similar cases.

5.1.2 The Diagnostic Process

In a more general sense, a rule-based diagnostic system performs a classification or pattern recognition procedure. This characterization provides a framework for organizing both the XpeRTD problem and the problem data. That is, it provides a focus or perspective for engineering a fuzzy logic, rule-based solution for RTD diagnosis. Specifically, it simplifies and assists the actual design and coding of the XpeRTD rule set by identifying the essential characteristics of rule conditions (FLOPS facts) and rule conclusions (FLOPS actions).

In FLOPS, a pattern-matching procedure is the basis of the control or inference engine. The patterns are the facts in FLOPS fact base. Pattern classes are specified by FLOPS rules. For each pattern (i.e., for each rule premise), FLOPS fuzzy logic evaluates its class membership (i.e., the strength of the rule conclusion). At the same time, FLOPS' conflict resolution algorithm "discriminates" among the possible classifications and selects the "best".

In each program cycle, FLOPS' matching algorithm "recognizes" a pattern in the fact base. The features of

each pattern are specified in the antecedent clauses of a FLOPS rule. In XpeRTD, these features characterize "classes" of RTD faults. FLOPS' fuzzy logic evaluates each set of features using the min() operator. It also provides a weighting which relates patterns and classes, again using the min() operator to classify each pattern. Finally, by chaining through all possible classes, FLOPS selects the "best" or most likely class for the current data base pattern, this time using the max() operator.

Figure 12 (adapted from Klir and Folger, 1988) demonstrates FLOPS' inference algorithm as it applies to a classification problem and, specifically, as it applies to RTD diagnosis. This example also explains, graphically, the problems of knowledge engineering and representation. In the following summary, the XpeRTD problem is represented by sets

S = {s | s is a symptom for some fault f in F}
F = {f | f is a possible fault for some s in S}
R = {r | r is an RTD}
A = {a | a is a symptom indicated by a single RTD r_1}
B = {b | b is a final diagnosis for RTD r_1}

where A and B are fuzzy sets with weighted elements that indicate the degree of certainty of, for example, the RTD data in A.

The final diagnosis and XpeRTD solution is the fuzzy set B and is given by the composition, denoted 'o', over sets A and K, i.e.,

$$B = A \circ K \quad . \qquad (5.1)$$

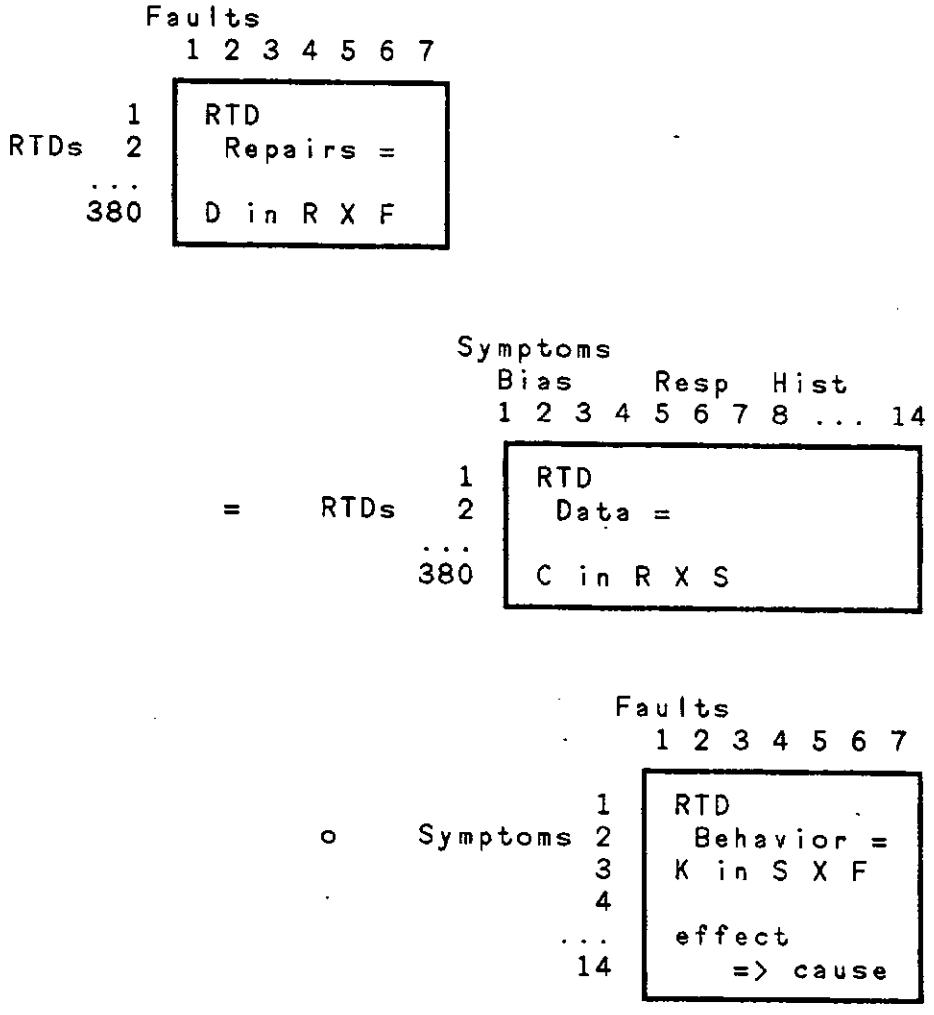


Figure 12: Knowledge representation for RTD diagnosis

In other words, faults f_i in B are indicated by symptoms a_i in A to the extent that the symptoms a_i are confirmed by some set K. K represents the technical knowledge that relates symptoms and faults, i.e., K is a subset of the relation S X F. The relation S X F represents the information that a symptom s in S indicates some fault f in F. The useful information K in S X F satisfies

$$D = C \circ K \qquad (5.2)$$

where C = R X S and D = R X F. In other words, the set D is the expert knowledge which evaluates the certainty that an RTD r in R exhibits a fault f in F.

Figure 12 is a knowledge representation for the problem of RTD diagnosis. A solution depends on engineering or specifying the knowledge represented by the relations (5.2). This is knowledge which combines a "textbook" understanding of RTD behavior with the practical experience of a diagnostician. If the representation is complete, then a solution may be "computed" by equation (5.1), which is defined for fuzzy sets by

$$m(B,f) = \max[\min[m(A,s), m(K,(s,f))]] \quad , \qquad (5.3)$$

where the function m() is a membership function which specifies the weights of elements s in A and (s,f) in K. Equation (5.3) defines a fuzzy subset B of F, so each f_i in B is weighted by m(B,f_i). For each f_i, the max() operation is performed over symptoms s in S, i.e., it selects an optimum value among different m(K,(s,f_i)). Equation (5.3) is defined as the compositional rule of inference (Zadeh, 1983). It generalizes the inference modus ponens of ordinary logic for fuzzy sets using composition. It is also the inference procedure provided by FLOPS' inference engine.

In FLOPS' fuzzy logic, equation (5.1) is interpreted by the rule

if A then B by K

or,

if symptoms are A then
diagnosis is B because
symptoms A indicate faults B ,

which is represented in FLOPS, in part, by

```
s in S --> f in F   CF(rule) = m(K,(s,f))
s in A              CF(e)    = m(A,s)
-----------------------------------------
f in B              CF(h)    = m(B,f)
                             = min[m(K,(s,f)),m(A,s)]
```

This inference is the min() operation in equation (5.3). It returns a single hypothesis h: 'f in B' with certainty CF(h) for some f in F. FLOPS performs the max() operation when it chains over all rules S --> F, i.e., whenever it asserts other hypotheses h_i: 'f_i in B'.

Following Figure 12, a solution for RTD diagnosis is specified as a fact base C and a rule base K. Fact base C is readily defined based on RTD performance data (i.e., bias and response) and RTD work histories. A rule base K is developed from descriptions of RTD behavior and from heuristics provided by an RTD expert. With this knowledge base, a solution D is "computed" using FLOPS' inference engine which performs the composition 'o' of C over K.

5.2 PROBLEM DEFINITION

XpeRTD's solution depends on a set of rules that relates RTD performance characteristics and RTD faults. Typically, these rules are not readily available. Since RTD diagnosis is a specialized or "expert" domain, many of the rules are rules of thumb. They provide no ready algorithm or mathematical model which may be applied to generate a solution. In fact, the relation between RTD data and faults is summarized by only two rules:

```
bias is low
  -->
  fault is clamp OR connector OR spare    ;

response is poor
  -->
  fault is clamp OR spare    .
```

Based on these rules, an XpeRTD solution is complicated because the same data (e.g., low bias) indicate different conclusions (e.g., clamp or connector or spare). At the same time, RTD characteristics or performance measures do not provide sufficient information to distinguish among the possible faults.

The XpeRTD solution requires a unique set of rules. RTD diagnosis is a specialized domain where the physical characteristics of an RTD distinguish it from other

electrical devices. That is, RTD faults are defined by rules which are unique. The theory which explains the behavior of other electrical devices is too general to explain the nature of RTD faults. Instead, RTD behavior is summarized by other knowledge about RTDs. This knowledge is the experience which is the basis of the expert's problem-solving skills. It is this practical knowledge on which the present system is based.

As a first step in capturing this knowledge, it was necessary to become familiar with the expert's terminology and with what exactly the expert does. The staff at Point Lepreau were especially helpful and efficient in preparing material which introduced the problem and provided the necessary background. Most importantly, the material provided the following definitions:

1. What is the function of an RTD in terms of the overall operation of a nuclear power plant? (Young, 1987)
2. What are the physical and functional characteristics of an RTD? (Young, 1987)
3. What RTD data is available and what is the format in which it is collected and stored? (Callister, 1987; Patterson, 1987)
4. Which of these data are important in diagnosing RTD faults? (Callister, 1987)

5. What are the procedures and decisions used to diagnose a faulty RTD? (Callister, 1987; Patterson, 1987)

These questions are presented and explained in Chapter 4. In terms of program design, the most important is question 3. RTD bias and response data are recorded only as computer printouts. The data aren't stored in a form which may be used for input to a computer program. Only RTD work histories are maintained in a useable format. These are recorded manually and kept in data base files using a DOS-based program called REFLEX (Figure 13).

The information from Point Lepreau also included an initial set of rules which were used to build a first XpeRTD prototype. The rules were prepared by Patterson (1987) in consultation with Callister (1987), the RTD expert. Figure 14 demonstrates one of three procedures represented by these rules. The general format of these procedures is summarized in Figure 15 . This rule set saved weeks of preparation and interviewing by providing a working definition of the problem in terms that a programmer could easily understand.

CHANNEL NO: P11

TempElement#: 1-63102-TE036	RBRoom#: R1-108
JunctionBox#: 1-63102-JB1325	Header#: 3312-HD-5
RTDLength: 9' 6"	FeederRow: F (TOP)
RTD_SCN: 611L3854	FeederSize: 4"
Thermowell_U_bolt_SCN: 392N3381	FlowDirection: 0
ElemDrawing#: 87-63102-10-07-ED-D	SDSI_Trip: no
TempTransmitter#: 1-63102-TT036	A_I#: 3525
RangeCardType: 401Y8	BridgeCardRack#: F01
RangeCardSCN: 611L3756	SecondChannel: P09

CTM_RTD_Group: M08 008 P09 P11 P13 P15 N15
CER_Panel#: 1-63102-PL101

RTD HISTORY

RTD SUSP	SUSP CABLE	CLAMP INSPECTD	SPARE USED	RTD REPLACED
1983	1986	1983	1983	1986
1984		1984		

Figure 13: RTD REFLEX data base entry, Channel P11

```
Diagnose(bias, response) ;

 { Top-level diagnosis procedure.          }
 { Calls procedure Bias_is_ok, Bias_is_high }
 { or Bias_is_low, depending on bias value. }

  begin
    if ok(bias) then
      Bias_is_ok(response)
    else
      if (bias > 1.5) then
        Bias_is_high(response)
      else
        Bias_is_low(response)
   end ;

Bias_is_ok(response) ;

 { Procedure to diagnose six of seven RTD faults,  }
 { depending on response. For each response class, }
 { evaluates RTD work histories.                   }

  begin
   if Good(response) then
     Diagnosis('RTD is OK') ;
   if Poor(response) then
     begin
      if (history.response.ok) then
        Diagnosis('ReplaceConnector') ;
      if (history.response.deteriorated) then
        if (history.clamp.adjusted) then
          if (history.spare) then
            Diagnosis('ReworkClamp & ReplaceConnector')
          else
            Diagnosis('UseSpareGT & ReplaceConnector')
        else if (history.spare) then
               Diagnosis('ReworkClamp')
             else Diagnosis('UseSpareGT')
     end ;
   if Bad(response) then
     if (history.spare) then
       Diagnosis('ReworkClamp')
     else Diagnosis('UseSpareGT')
  end ;
```

Figure 14: Patterson-Callister rules for RTD diagnosis.

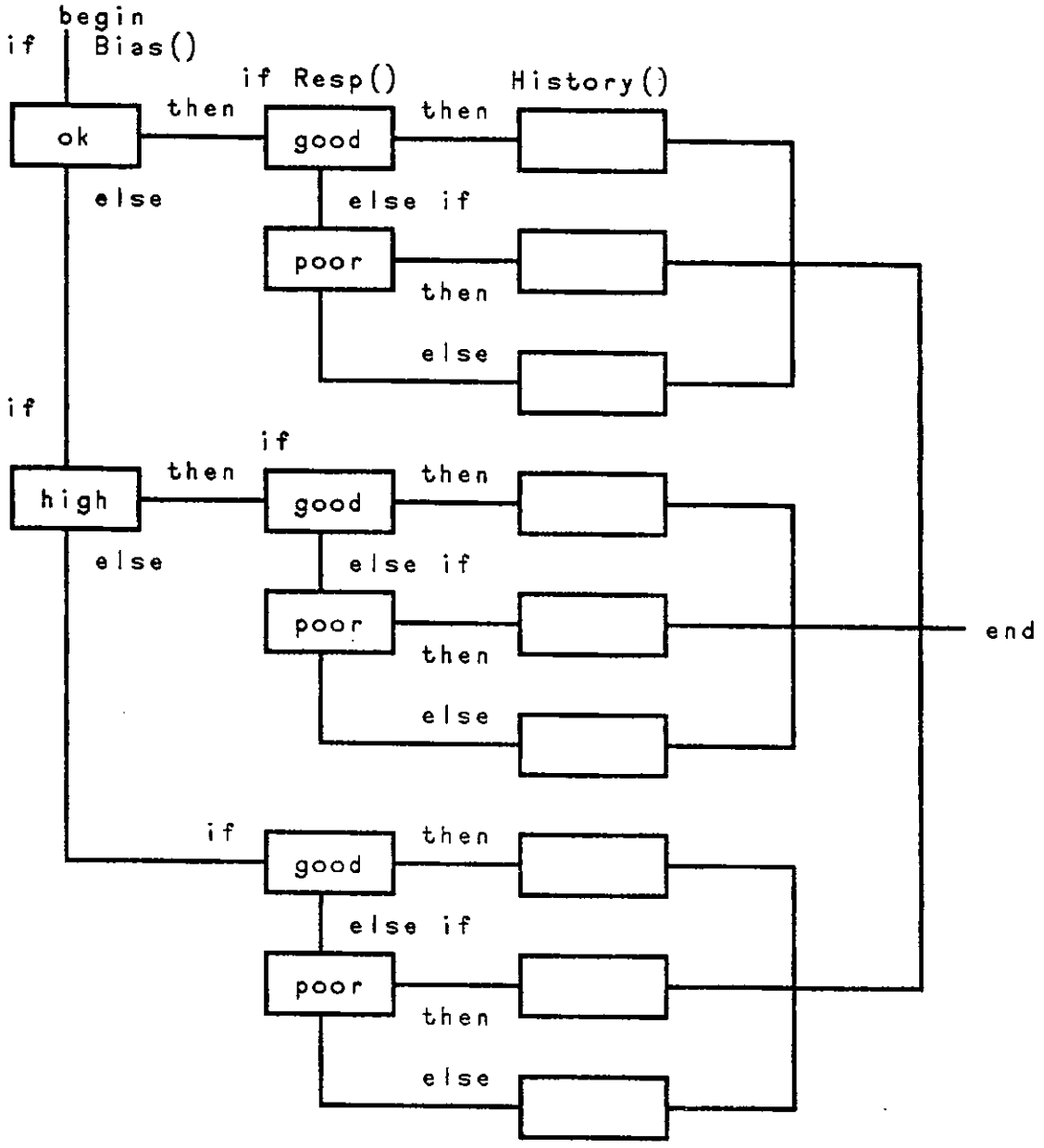


Figure 15: PRO1 forward-chaining rules: decision tree

5.2.1 Prototype PRO1: Forward Chaining

The rules supplied by Patterson (1987) were used to develop an initial prototype PRO1. PRO1 served to define the general characteristics of an RTD expert system and to provide a

working program which could be used during interviews with the expert.

PRO1 defines the general format of an RTD data base and suggests an implementation using a forward-chaining control. This definition follows translating the initial set of rules into a FLOPS (Siler and Tucker, 1986) pseudocode and testing different rules in order to achieve the desired results.

FLOPS' pattern-based format suggested an object-oriented data base with facts or objects defining RTDs, refuelling trends and work histories (Figure 16). The object (RTD) lists the physical characteristics of an RTD. Refuelling response data is represented by (trend), a list of response values and a fuzzy set which classifies an RTD's response as GOOD, POOR or BAD. RTD work histories and bias values are recorded as (hist), a list that includes a bias value and a fuzzy set which classifies the bias as HIGH, LOW, BETWEEN or OK. The REFLEX data file is used to define an (RTD) object for each RTD. Trend and bias data are not on file and must be provided by the RTD expert.

Since the REFLEX data base is not complete, it was necessary that PRO1 prompt for input values. This restriction dictated a prototype for single RTD diagnosis, i.e., a program to read trend and bias values for a single RTD, perform a diagnosis based on data base values, and record the diagnosis in the fact base.

```
(RTD
  (id "P11")
  (clamp "")
  (connector "")
  (spare "")
 )

(trend
  (id "P11")
  (dtmax 10)
  (dt2 2)
  (response (good 1000)(bad 0)(poor 0))
 )

(hist
  (id "P11")
  (abias 2.0)
  (bias (high 1000)(low 0)(between 0)(ok 0))
  (diag)
 )
```

Figure 16: PRO1 data base definitions

PRO1's rules are translated directly from Patterson's. The rule set is designed for single diagnosis, so there was no need to restrict the algorithm. It covers all possible combinations of data and provides a diagnosis for each set of symptoms. The rules operate by inferring a diagnosis from the RTD data, using forward reasoning from facts to conclusions, and were easily translated into FLOPS forward-chaining format (Figure 15).

Prototype PRO1 is complete in the sense that, given any set of symptoms, it provides a diagnosis. The diagnosis, however, is not necessarily correct. The expert, Callister, had difficulty confirming some of the rules during the first interview. For example,

```
if bias is high &
   bias is drifted &
   response is poor &
   spare is used
then rework clamp &
     replace cannon connector .
```

Callister noted the unlikelihood of both high bias and poor response appearing at the same time. In general, the rule set forces the expert to make decisions which he can't make with any degree of certainty. That is, the rules represent decisions that have no basis in theory, or that have no counterpart in practice.

PR01 is also deficient because it requires information that isn't readily available. For example, it prompts for actual response data, then, using fuzzy arithmetic, it assigns fuzzy membership values for each of the response classes (i.e., GOOD, POOR and BAD). The expert explained that trend characteristics are usually not calculated; rather, RTD diagnosis is based on a visual inspection. Response is rated 'good' or 'poor' as a matter of judgement. In other words, uncertainty is measured as a subjective estimate rather than an exact value.

5.2.2 Prototype PRO2: Backward Chaining

In another attempt to define the problem, the initial rules were translated into a hypothesis set. This is the usual format for a diagnosis problem, especially when the number of possible faults is small. Using this strategy, a diagnostician tests for each fault (hypothesis) in turn by checking for related symptoms. Since most faults exhibit some distinguishing feature, an expert diagnostician can quickly eliminate hypotheses because they are not substantiated by the data.

The Patterson rules were translated into a set of hypotheses using Prolog pseudocode (Figures 17 and 18). In this new format, the initial set is reduced from more than 20 to only five rules. The rule set for a backward chainer underlies the redundancy in the Patterson rules. Redundancy is not necessarily undesirable. In fact, it is often the only solution for a problem which is not proscribed by general rules. In this case, however, the larger rule set represents a lack of definition, not unexpected since the rules were designed as a summary of the RTD problem.

The new rules reduce the previous set to the few general rules on which RTD diagnosis is based. The rules provide no new information in the form of additional characteristics or relations. They do confirm the lack of theory or general principles which might provide an algorithmic solution.

```
% pseudoProlog:

clamp() :-
  spare,
  (response(bad);
   (response(poor),
    deter(response))
  ) .

spare() :-
  not(spare),
  (response(bad);
   (response(poor),
    deter(response))
  ) .

connector() :-
  bias(high),
  (response(good);
   (response(poor),
    drift(bias))
  ) .

connector() :-
  bias(low),
  ((response(poor),
    drift(bias));
   response(good);
   response(bad)
  ) .

connector() :-
  bias(ok),
  response(poor),
  not(deter(response))) .
```

```
% English translation:

diagnosis is clamp
 IF spare is used &
   IF response is bad
   ELSE IF response is poor &
           deteriorated

diagnosis is spare
 IF spare is not used &
   IF response is bad
   ELSE IF response is poor &
           deteriorated

diagnosis is connector
 IF bias is high &
   IF response is good
   ELSE IF response is poor &
           bias is drifted

diagnosis is connector
 IF bias is low &
   IF response is poor &
      bias is drifted
   ELSE IF response is good
   ELSE IF response is bad

diagnosis is connector
 IF bias is ok &
    response is poor &
   NOT
    response is deteriorated
```

Figure 17: PRO2 backward-chaining rules: Prolog pseudocode

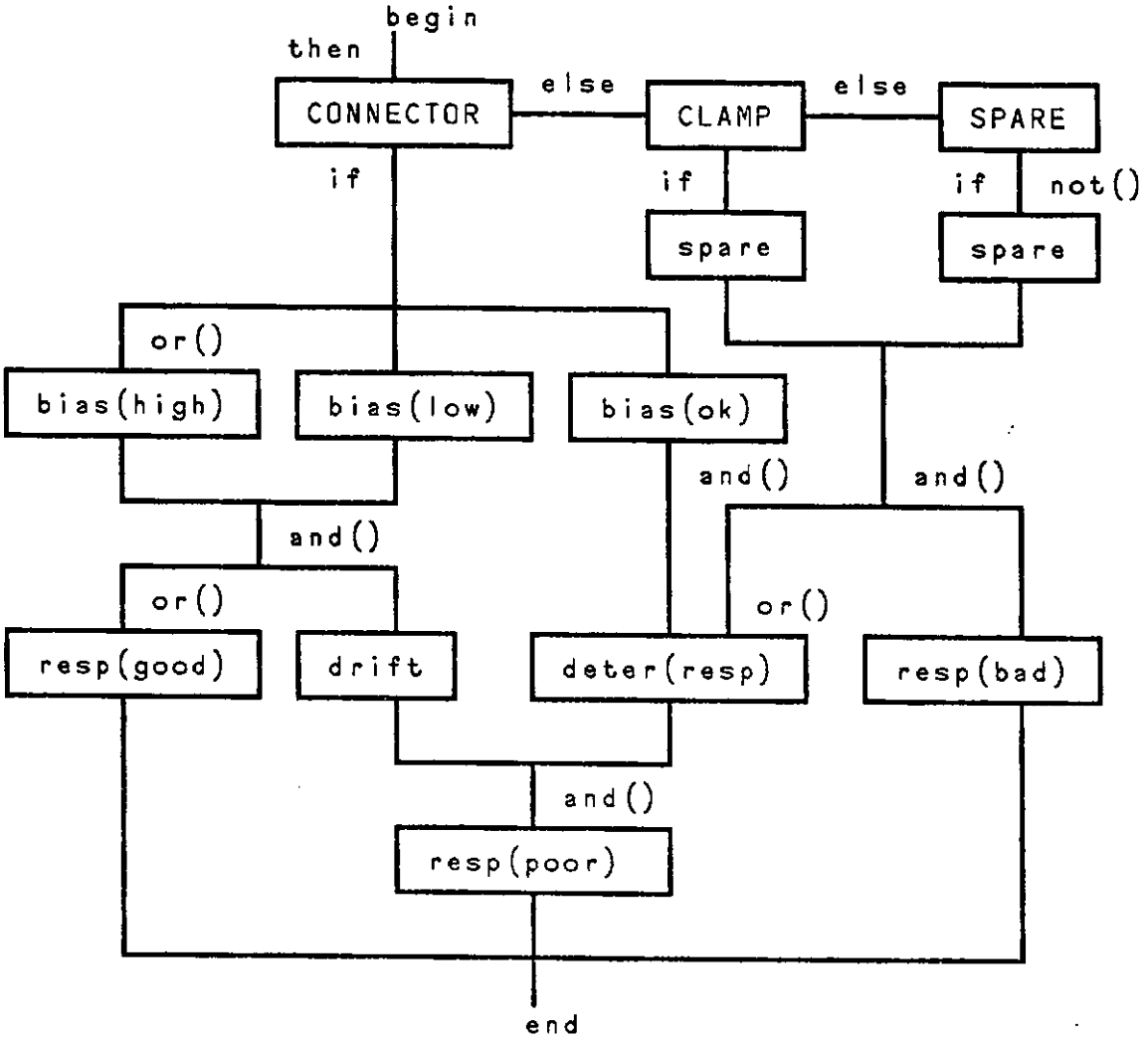


Figure 18: PRO2 backward-chaining rules: decision tree

The new set of rules was useful in suggesting a hierarchical approach. For example, if a clamp is the suspected fault, then check the response first; if a connector is suspect, then check the bias first. The exercise suggested a restructuring of the RTD program to incorporate both backward and forward chaining. Backward chaining could be applied to first isolate the most likely

fault(s). Forward chaining could then be used to confirm the fault by testing the facts in the RTD fact base.

The new strategy implements a standard algorithm for multiple fault diagnosis, i.e., diagnosis where more than one fault is indicated by the same set of symptoms. MYCIN's problem is of this kind. MYCIN's solution also incorporates a hierarchical approach. MYCIN's rules are divided into "contexts", different rule sets which apply to different cases. MYCIN uses the knowledge about the taxonomy of bacteria to eliminate sets of rules and to select among others depending on the identification of the organism.

PRO2 uses FLOPS forward reasoning to simulate backward chaining. It shows the feasibility of a contextual approach and, also, the flexibility of the FLOPS interpreter. Subsequent development of the RTD problem focused on developing the rules which could test the different hypotheses. While PRO2 suggested the best solution to RTD diagnosis, the solution depended on decisions or rules which had still to be defined. Specifically, the rules of Figure 18 must be extended to include other classes of RTD faults and additional RTD data.

5.2.3 Rule Definition

The eventual design of the XpeRTD system depends on a minimum set of rules. This rule set summarizes the expert's approach to RTD diagnosis. Initially, it was expected that the expert would provide these rules directly. It was expected that the rules could be easily summarized and that a final set would follow by elimination. It was discovered, however, that knowledge engineering does not proceed so easily. This section discusses two basic approaches used to define rules for XpeRTD.

5.2.3.1 Interviews with an Expert.

Sessions with the expert served to clarify both program data and the diagnostic process. Following the first interview, it was decided that the initial rule set was unworkable. Attempts to salvage the information contained in the rules were mostly unproductive. Typically, the expert didn't have a ready list of rules and showed difficulty articulating the decisions used in diagnosis. Efforts to develop another rule set were frustrated by this lack of definition.

An obvious approach to rule definition uses scenarios. The expert is presented with possible rules in the form of a question:

Scenario:

if bias is high
  AND response is poor
then
  (what diagnosis?)
  OR (what other RTD
    characteristics are needed for a decision?)

In response to such a question, the expert gave an answer based on his knowledge of instrumentation. The response was an explanation of RTD behavior which invariably lead to other scenarios or other questions not related to RTD mechanical faults. In most cases, the answer provided nothing conclusive. Often, the expert's response appeared evasive. Sometimes, the interviewer suspected the expert's interests. At times, he wondered whether the problem was too complex, or too simple. It was later learned that this situation is typical of expert interviews.

Hayes-Roth et al. (1983) note that the vocabulary used by most experts is often inadequate for problem-solving. Expertise entails skills which, because they are practised, are automatic. They no longer involve deliberate decisions or procedures that are easily summarized. Typically, expert system problems are imprecisely defined. The essential task of knowledge engineering doesn't summarize but refines and extends the expert's knowledge. It provides a structure

where none exists; it identifies and formalizes domain concepts in terms which are undefined.

This problem is well known in psychology, a field which specializes in formalizing and understanding human intelligence and behavior. Psychology recognizes the difficulty in soliciting explanations for why people do what they do. It has developed a battery of techniques which, ignoring what people say, characterize behavior that is unconscious or automatic. These techniques assume that what people say is not what they mean (Gaines and Shaw, 1986).

5.2.3.2 Defining Rules by Induction.

Shaw and Gaines (1984) describe an engineering technique for expert systems based on methods in psychology. The technique provides a convenient algorithm for defining an expert problem. It was especially useful for suggesting a general approach for interviews with the RTD expert. It was this approach which eventually provided the additional rules to complete the XpeRTD rule base.

The Shaw and Gaines algorithm (SG) reduces an expert problem to a set of rules which are used to prime an expert system shell. Their algorithm is the basis of at least two interactive computer programs that assist the expert in defining the expert domain (Shaw and Gaines, 1984; Boose, 1984). An SG program produces a table or repertory grid that

summarizes the problem domain. The grid is then used to provide a weighted set of rules, based on standard clustering algorithms. Figure 19 shows a sample grid for the RTD problem. In Figure 19, columns 1-7 represent typical RTDs. They are typical because each RTD represents one class of faults. The SG algorithm was not implemented here, so the grid in Figure 19 and the program dialogue in Figure 20 are both simulations.

```
                RTD r1 r2 r3 r4 r5 r6 r7
                 ******************************
response is good 1 *                            *  1 bad
 bias reads high 2 *                            *  2 low
     drift is up 3 *                            *  3 down
  sym_op_ch high 4 *                            *  4 low
 position is top 5 *                            *  5 bottom
   resp is deter 6 *                            *  6 same
suspected > once 7 *                            *  7 never
 rtd is replaced 8 *                            *  8 not repl
clamp is reworkd 9 *                            *  9 no
  spare is used 10 *                            * 10 not used
 conn is replcd 11 *                            * 11 not repl
                   *  *  *  *  *  *  *  *  *  *
                      electrical fault
                      *  canon connector
                      *  *  clamp
                      *  *  *  spare guide tube
                      *  *  *  *  RTD failure
                      *  *  *  *  *  ok but monitor
                      *  *  *  *  *  *  best response
```

Figure 19: Repertory grid for RTD diagnosis

The algorithm SG first prompts the expert to list the objects or elements which are relevant to the problem. For RTD diagnosis, these are individual RTDs. SG then asks for the classes or categories which distinguish the elements of

the problem, i.e., the different types of faults. SG continues by prompting for class features, ie., what characteristics identify the different faults. The questions posed to the expert are designed to force comparisons which he would not ordinarily make. SG provides a format for knowledge engineering which is simple and, at the same time, accounts for the subtleties which underlie the engineering process. Incidentally, Shaw and Gaines (1984) discuss the representational theory on which their algorithm is based.

From an initial list of elements (RTDs) and categories (faults), SG refines and completes the problem description by prompting for the similarities and dissimilarities among the elements. Figure 20 simulates a dialogue between an SG program and the RTD expert. In this example, SG compares a clamp fault and a connector fault. It first characterizes these faults by their similarity. It then tries to find different features. It continues to develop a feature set by looking for faults with features that haven't been defined.

The SG program also defines the rule set for a rule-based expert system. When a new feature is introduced, the expert is asked to rate, for each class of fault, the importance of this feature for identifying the fault. Figure 20 demonstrates this procedure. In the sample dialogue, the program prompts the expert for an interval estimate. The

```
SG: What is similar about
      an RTD whose fault is * clamp *
      an RTD whose fault is * connector *
    ?
_: BIAS

SG: What is different about
      an RTD whose fault is * clamp *
      an RTD whose fault is * connector *
    ?
_: RESPONSE
...

SG: Please specify
      the interval over which * response * is defined
    ?

      low value is
    ?
_: GOOD
      high value is
    ?
_: BAD

SG: If 1 is good and 5 is bad then
      an RTD whose fault is * clamp * has response
    ?
_: 5

SG:   an RTD whose fault is * connector * has response
    ?
_: 3

SG: List another type of fault that is not indicated by
      * response * is good or bad
    ?

_: SPARE
SG: What is similar about
      an RTD whose fault is * clamp *
      an RTD whose fault is * spare *
    ?
_: RESPONSE
...
```

Figure 20: Sample dialogue for program SG (adapted from Shaw and Gaines, 1984)

estimate may be the interval of a probability distribution or another characteristic function. This function provides weights for the "rules" or conditional relations between elements and categories, which are defined by the repertory grid.

The SG algorithm interprets the estimates as fuzzy set membership values. In Figure 20, response values range from good = 1 to bad = 5. In this case, 1 represents a minimum membership and 5, a maximum, in a fuzzy set BAD. Opposite membership values are recorded for the fuzzy set GOOD. A scale may be defined using a finer gradient but, Shaw and Gaines suggest, a gradient of between 3 to 10 is usually sufficient to completely characterize the problem domain.

(Ben-Bassat et al., 1980), employs a similar technique in MEDAS, an expert system for medical diagnosis. MEDAS uses an 8-point scale to solicit estimates for the conditional probabilities that a feature or symptom exists given the fact that a disease or disorder is present. The estimates form the basis of a non-rule-based expert system using a multimembership (Bayesian) classification algorithm.

(Gaglio et al., 1985) details a number of analytical techniques for summarizing subjective estimates provided by two or more experts.

The SG algorithm was not implemented here but the general approach was useful in interviews with the RTD expert. It suggested a case by case analysis of RTD failures rather than a theoretical analysis of cause and effects. Instead of asking questions of the form

if bias is high
then (what diagnosis?) ,

it was more productive to use questions such as

if rtd P11 was diagnosed with a clamp problem
then (what were the symptoms?) .

In other words, it proved more instructive to discuss the history of particular RTDs, rather than the general nature of RTD behavior. This line of questioning introduced new terms to characterize RTDs and RTD diagnosis. For example, it provoked a discussion of 'symmetrically opposite channels'. The discussion emphasized, for the first time, the importance of location in evaluating RTD performance. It also clarified the nature of the uncertainty attached to bias data. Bias is, generally, an unreliable indicator of RTD behavior. If bias is the only indication of an RTD fault, then the expert verifies the biased reading by comparing the suspected RTD with the RTD on the channel which is symmetrically opposite. The two RTDs occupy the same position relative to the reactor core and should,

therefore, show the same response to changes in reactor activity or in HTS parameters (ie., channel pressure and flow rate).

The SG approach was especially useful in suggesting the final solution to the thesis problem. It clarified the source of uncertainty in the data and provided a means to quantify the uncertainty. The expert found the SG weighting procedure awkward and tedious, so the repertory grid in Figure 19 wasn't completed. This option obviated the detailed computation and analysis of the SG rule-producing procedure. Similar results were generated more easily as prior CFs attached to FLOPS rules. This approach at least approximates the more rigorous mathematical treatment defined by SG since they both depend on estimates provided by the expert. The less rigorous approach was deemed sufficient for prototyping a system that will eventually be verified and adjusted based on outage inspection and repairs.

5.2.4 Prototype PR03: Parallelism and Contexts

Prototype PR03 provides a diagnosis which is more nearly correct in the sense that it better duplicates the expert's diagnosis. Generally, it redefines RTD diagnosis using the terms and data which the expert applies. PR03 operates on the RTD data following the patterns which the expert employs. Specifically, it extends the hypothesis set of

Figure 18 to include all fault classes. It incorporates new rules which, for each fault, defines the sets of RTD data which bear on this fault. The PRO3 rule base is summarized in Figure 21 .

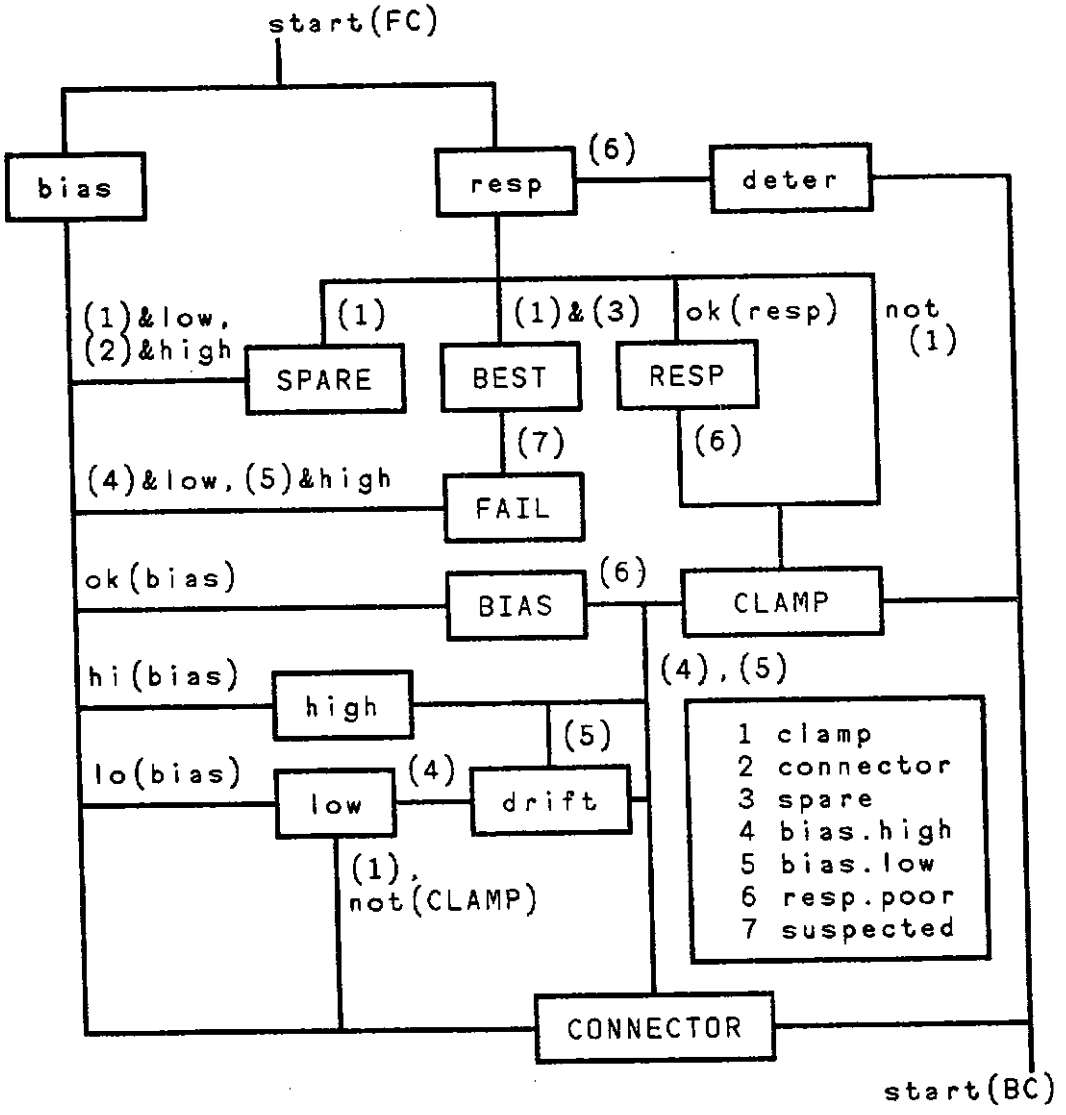


Figure 21: PRO3 forward- (FC) and backward-chaining (BC) rules: semantic net

PR03 is data-driven so, for each RTD, rules are fired only if their LHSs match the RTD data. That is, for each rule, PR03 chains forward from facts to conclusions. PR03 also simulates backward chaining by selecting each hypothesis in turn. In each cycle, PR03 selects a different class of fault (e.g., connector) and activates only rules which bear on this class. This control is effected using FLOPS' rule definitions to identify different rule "blocks". Backward chaining is necessary because some faults are diagnosed depending on other faults, e.g., a rule to diagnose a connector fault is conditioned on whether a clamp fault is eliminated.

PR03 is a more general program for diagnosis. It accepts whatever data is available. It produces one set of results for all 380 RTD's. In these respects, PR03 follows the procedure used by the expert for outage planning. It first requests a list of RTDs which are suspected to be faulty. This list is normally prepared by the expert in the planning process and is based on a visual inspection of refuelling trends. An RTD is suspect if the trend shows a "poor" response. Other data such as bias or drift may not be available. PR03 requests this data if it is available, but the data is not necessary. PR03 is designed to provide a diagnosis regardless of what RTD data is available.

PR03 uses "contexts" to diagnose 380 RTDs at once. It makes decisions depending on the facts in the fact base and each decision considers all RTDs. PR03 is implemented using a parallel version of FLOPS. This version recognizes different blocks of rules and it fires each block in sequence.

In parallel FLOPS, a block is a set of rules which is fired at the same time. In FLOPS' data-driven format, rules are selected for firing if they match facts in the data base. Parallel FLOPS performs the matching over one block of rules for each cycle; sequential FLOPS performs matching over all the rules. If one rule matches more than one set of facts (i.e., an "instantiation"), then sequential FLOPS selects the "best" match over all the facts in the fact base. It fires the rule once but remembers other instantiations, in case it must backtrack. By comparison, parallel FLOPS, for each block of rules, finds the "best" rule for each fact, then it fires all the best rules. Parallel FLOPS does not backtrack because it activates all instantiations at the same time.

For example, PR03 hypothesizes a connector fault using one block of rules. This block contains four general rules which bear on the connector problem. When this block is activated, parallel FLOPS applies all four rules and only these rules. It then evaluates each RTD. If an RTD exhibits

characteristics which satisfy the conditions of any one of four rules, then FLOPS records a connector problem. If an RTD satisfies more than one rule, FLOPS diagnoses a connector problem, but it also records a CF value which accounts for the different evidence. In other words, for each RTD with a suspected connector fault, parallel FLOPS records a fault with a CF value that depends on the RTD's characteristics. By comparison, sequential FLOPS diagnoses all faults at the same time. For each fault, it records the RTD which most likely exhibits this fault.

PRO3 uses the algorithm developed with PRO2. It first isolates the most likely faults. It then tests the certainty of each fault using work histories and other RTD data. PRO3 returns a diagnosis which, for each RTD, lists the possible faults as well as the likelihood that a fault exists. A diagnosis is recorded as a FLOPS object defined by

```
(diag
  (id "s10" 1000)
  (dx (elect 0)
      (connector 800)
      (clamp 0)
      (spare 0)
      (repl 750)
      (monitor 700)
      (best 700)
      (needs_work 750)
))                                                    (5.4)
```

The element (dx()) is a FLOPS fuzzy set where each fuzzy variable has a characteristic certainty or CF value that is either user-defined or modified by a FLOPS' rule. The variable (connector) above, for example, is recorded with CF = 800. A CF may be modified by a FLOPS' rule of the form

```
rule 900 (trend ^id <id> ^bias "high")
         (diag ^id <id>)
      -->
      modify 2 ^dx.connector ;                        (5.5)
```

which changes the CF value in (connector CF) to a "posterior" CF which depends on the rule CF = 900 and on the CF value of bias = "high". In example (5.4), rule (5.5) is triggered by a fact (trend (id "s10")(bias "high" 800)), so (diag (id "s10")) is modified to include (dx (connector 800)).

The PR03 rule set is more flexible than PR01's. For each type of RTD fault, PR03 considers all possibilities. Indirectly, it incorporates the certainty values or fuzzy measures provided by the Gaines and Shaw (1986) technique. These values are important in distinguishing among the many similar cases among RTD faults. In fact, these values define the decisions which are essential for solving the RTD problem. In other words, the values are simply a quantification of the common-sense, expert knowledge that is critical in completing the diagnosis.

With this prototype, the RTD problem is well-defined. The new rule set using certainty measures defines what RTD characteristics are important in fault identification and how these characteristics are used to distinguish among the different RTD faults. Further development from this point involves testing the system for completeness using actual RTD data and for consistency using "exceptions to the rules".

5.3 TESTING AND DEBUGGING

The PR03 algorithm was tested on 1987 outage data from the N. B. Power Point Lepreau nuclear generating station. The test data included the REFLEX file of RTD histories and the expert's list of suspected RTDs. The REFLEX file is the one which existed at the time of the 1987 outage. It does not include the 1987 results, so it contains the actual data

which existed for the 1987 planning. The expert's list was compiled from the report that the expert prepared following the outage (Callister, 1987). This report lists a final diagnosis for each suspect RTD. It provides not only the input required for a PRO3 diagnosis, but also a basis to compare PRO3's diagnosis with the expert's.

Initially, the test version of PRO3 produced results that were inconsistent with the expert's. The results contained decisions that didn't agree with the expert's diagnosis. The rules did not distinguish among the different levels of bias, for example. PRO3 diagnosed a connector problem when it should have recognized a bias error. The initial results were also incomplete. PRO3 missed some diagnoses which it should have made based on the data. To correct these errors, the PRO3 rule set was revised. Additional rules were added to produce results which were complete. Rules were eliminated or additional conditions were added to rule premises to avoid inconsistencies.

The revised version of PRO3 uses 28 rules. This version is listed in Appendix III.3. The rule set is minimized, so it contains only those rules which are required for the test data. All the rules bear on RTD diagnosis, since other procedures (i.e., XpeRTD's control and interface) are provided by the FLOPS inference engine. The PRO3 rule set performs a hypothesize-and-test procedure. It contains all

the rules needed to generate a hypothesis set. It also contains the rules which tests hypotheses for the few exceptions among the test data.

PR03's rule set is correct because it duplicates exactly the expert's 1987 diagnosis (Callister, 1987). PR03's diagnosis is listed in Appendix IV. PR03's performance is highly dependent on the test data, but it also depends on initial hypotheses which are most likely correct for any data. In other words, while PR03 is revised to suit a particular data set, it depends on a number of rules which are invariable. These last rules summarize the decisions which the expert employs in an actual diagnosis. The expert's rules are probably correct. This means that his judgements apply in most cases with a probability which depends on the expert's familiarity with these cases.

PR03's algorithm is correct in another sense. A prerequisite for a "good" working expert system is modularity. Rules should be easily added and changed. PR03 exhibits a flexibility that is surprising because it was not specifically planned. PR03 is modular, in part, because it employs a parallel version of FLOPS. Parallel FLOPS provides modularity by forcing a blocked or structured format of rules. It also contributes to a program's modifiability by providing a procedure which may be variously applied.

Parallel FLOPS facilitates diagnosing all RTDs at once, but it also follows sequential FLOPS and allows diagnosing one RTD at a time. PRO3, using parallel FLOPS, performs single or multiple diagnosis, depending on the data. In either case, PRO3 uses the same rules. The rule base, then, may be modified by viewing the PRO3 algorithm as a solution for either single RTD diagnosis, or for the more general outage planning problem. Rules will be different for each problem, but PRO3 accomodates any inconsistencies using a hypothesize-and-test cycle. It distinguishes between rules which are general (i.e., which apply to the hypothesize cycle), and the rules which are the exceptions (i.e., which test individual cases).

PRO3 is also modular because it uses rule blocks. PRO3's block structure simplifies debugging and modification. Program errors are isolated and easily identified as rule decisions or rule actions which are the result of rules in particular rule blocks. Each block bears on a single result, so each program decision or effect is easily modified by adding or deleting only rules in the appropriate block. For example, if PRO3 makes an error in diagnosing a connector problem, then the problem is easily corrected by amending one or more rules in the rule block that bears on a connector fault.

5.4 VERIFICATION

The thesis was completed just prior to the 1988 outage at the Point Lepreau facility, so verification of the final rule set is not yet available. It is expected, however, that verification will be provided at a later date. In this case, the PRO3 algorithm will be applied to the data prepared for the 1988 outage to produce a work plan for RTD outage repairs. The PRO3 work plan may then be compared with that developed by the RTD expert. This comparison will validate the success of PRO3 in capturing the expert's diagnostic skills.

Following the current outage, the PRO3 work plan may be compared with the reports which follow outage work. These reports summarize the observations of RTD technicians during the inspection and repair of suspected RTDs. A comparison with these results will validate the success of PRO3's reliability in summarizing and predicting RTD behavior.

Chapter 6

SYSTEM DESCRIPTION

6.1 OPERATING PHILOSOPHY

XpeRTD is a prototype expert system for RTD diagnosis which is based on the analysis in Chapter 5. XpeRTD is designed to assist an expert RTD technician in performing a routine task which is both time-consuming and complex. The complexity of the task depends, in part, on the number of RTDs and the amount of data which must be considered. XpeRTD duplicates the skills which the expert applies to outage planning, ie., to preparing an annual work plan for RTD maintainance. XpeRTD reviews the histories and current data for 380 RTDs. It summarizes data sets as large as 1500 items. XpeRTD saves the human expert weeks or months of preparation by recording and processing this volume of data.

XpeRTD formalizes and defines a procedure that is currently tentative and *ad hoc*. XpeRTD's rule set structures the diagnosis problem and approximates an exact model of RTD behavior. This definition of RTD diagnosis has at least three advantages:

1. It provides a basis for improving and refining RTD diagnosis.

2. It documents the theory (rules) on which the diagnosis is based and records the results of each diagnosis.
3. It provides a teaching tool for training other technicians in RTD diagnosis.

XpeRTD is a prototype whose implementation would contribute to the safety and economy of nuclear power plant operations. The greater efficiency from computer-assisted diagnosis would

1. reduce incorrect RTD diagnoses by making explicit the rules on which the diagnoses are based;
2. reduce the time that personnel are exposed to higher than normal radiation levels by reducing the time required for RTD inspection and repair; and,
3. reduce the time lost from power generation by limiting the time required for outage work and by limiting the number of alarms caused by failed RTDs.

XpeRTD's greater efficiency would depend not only on faster, more consistent diagnosis but also on a format which may be verified and improved.

6.2 HARDWARE AND SOFTWARE

XpeRTD is a prototype application developed for use in the Canadian nuclear industry. It is compatible with hardware which is installed at nuclear power plants in Canada. It is implemented in FLOPS (Siler and Tucker, 1986), a rule-based programming language that is an inexpensive DOS software package which can be applied as a general programming tool.

FLOPS is an IBM-PC-compatible software package which provides the control procedures required to implement a rule-based system. It permits the definition of fuzzy sets, fuzzy numbers and the other elements of fuzzy logic which provide for handling uncertainty in the data and in the rules themselves. FLOPS also supports other features (e.g., external calls in C, rule blocking, firing control) that allow the programmer to design a system tailored to a particular application.

6.3 DATA ORGANIZATION

XpeRTD is a rule-based expert system which incorporates a data base of rules and facts (Figure 22). XpeRTD's rules are recorded and maintained as a FLOPS program. The fact base exists as a separate data file which is read into working memory during run time.

The XpeRTD fact base is generated from an existing file of RTD work histories. RTD data is normally recorded and

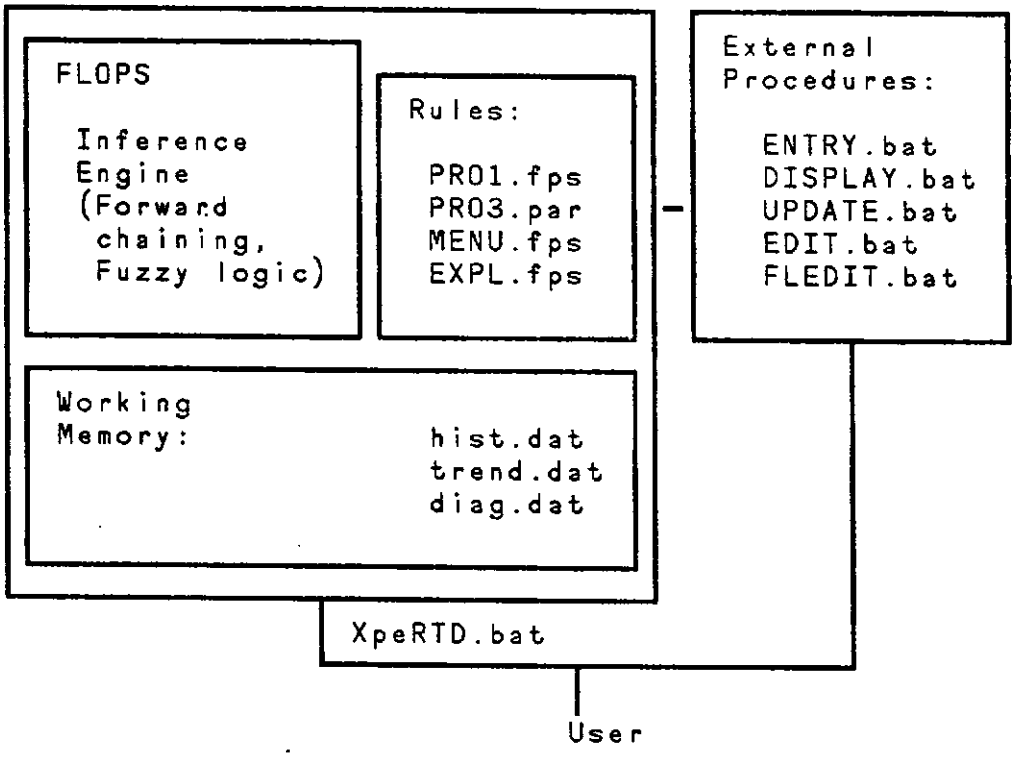


Figure 22: XpeRTD Expert System

maintained using a data base software package called REFLEX. REFLEX includes a file utility for translating between its own data base and other file formats, including FLOPS ASCII "stream" files. The translation facility wasn't used because the REFLEX file contains a data base not designed for rule-based programming. The REFLEX file contains information which is redundant for RTD diagnosis. Each record in the REFLEX data base contains 46 fields of which only six are relevant. A FLOPS program designed around such a record would be unnecessarily complicated and inefficient.

The XpeRTD fact base uses only a portion of the REFLEX file. This facilitates defining facts as objects. Facts are structured following FLOPS "patterns" and the structure is dictated by the characteristics of the RTD problem and the logic of RTD diagnosis, ie., facts are patterned after FLOPS rules, and vice versa.

XpeRTD's fact base is a standard ASCII file. Its contents may be displayed and edited using a standard text editor. XpeRTD includes file-handling procedures which are external to FLOPS and which provide the following facilities:

1. Display and entry of RTD work histories (file hist.dat).
2. Entry of RTD diagnosis data (trend.dat).
3. Recording of the results of a FLOPS diagnosis (diag.dat).
4. Updating of RTD work histories (hist.dat) based on data manipulated and returned by XpeRTD (diag.dat, trend.dat).
5. Updating of RTD work histories (hist.dat) following outage inspection and repair based on user-supplied data.

XpeRTD files and file-handling procedures are summarized in Figure 23 .

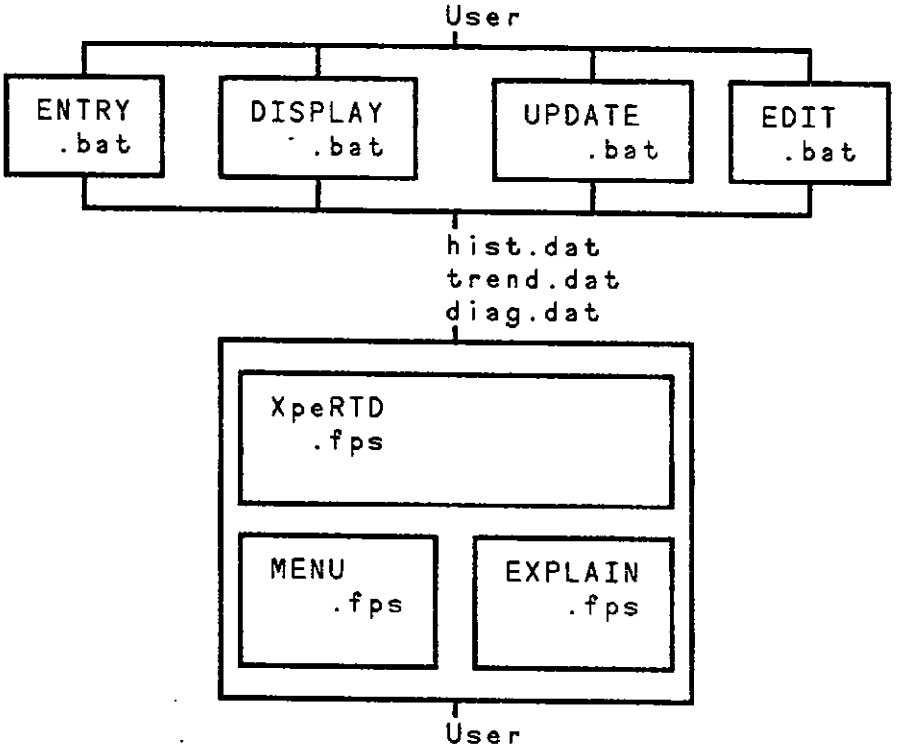


Figure 23: XpeRTD data and data communication

6.4 SYSTEM OPTIONS

XpeRTD is a menu-driven, interactive expert system. Its main menu presents the user with the following options:

1) Display/enter RTD Data
2) Diagnose Single RTD
3) Prepare RTD Outage Plan
4) Update RTD REFLEX file
5) Edit RTD REFLEX file
6) Explain RTD Diagnosis

XpeRTD's top level is FLOPS forward-chaining inference engine. FLOPS first fires the rules MENU.fps which prompts

the user for a selection from the main menu. Depending on the user's response, FLOPS calls the appropriate program. Diagnosis for a single RTD is initialized by FLOPS using a single trend record. Outage planning is initialized using an external program to create the trend.dat file.

The user has easy access to XpeRTD's data. He may either choose XpeRTD's menu-driven format or use a standard text editor to display and alter XpeRTD files. The 'Display' option calls a menu-driven external procedure that presents RTD data in a formatted display (Figure 24) and that allows the user to display and compare different RTD features. The 'Enter' option calls FLEDIT, a FLOPS menu-driven editor designed to control data entry for FLOPS (XpeRTD) files. 'Edit' calls a standard text editor and provides an alternative program for assessing and editing the XpeRTD data base. 'Update' executes an external routine which prompts the user for work data and records this information in the RTD history file.

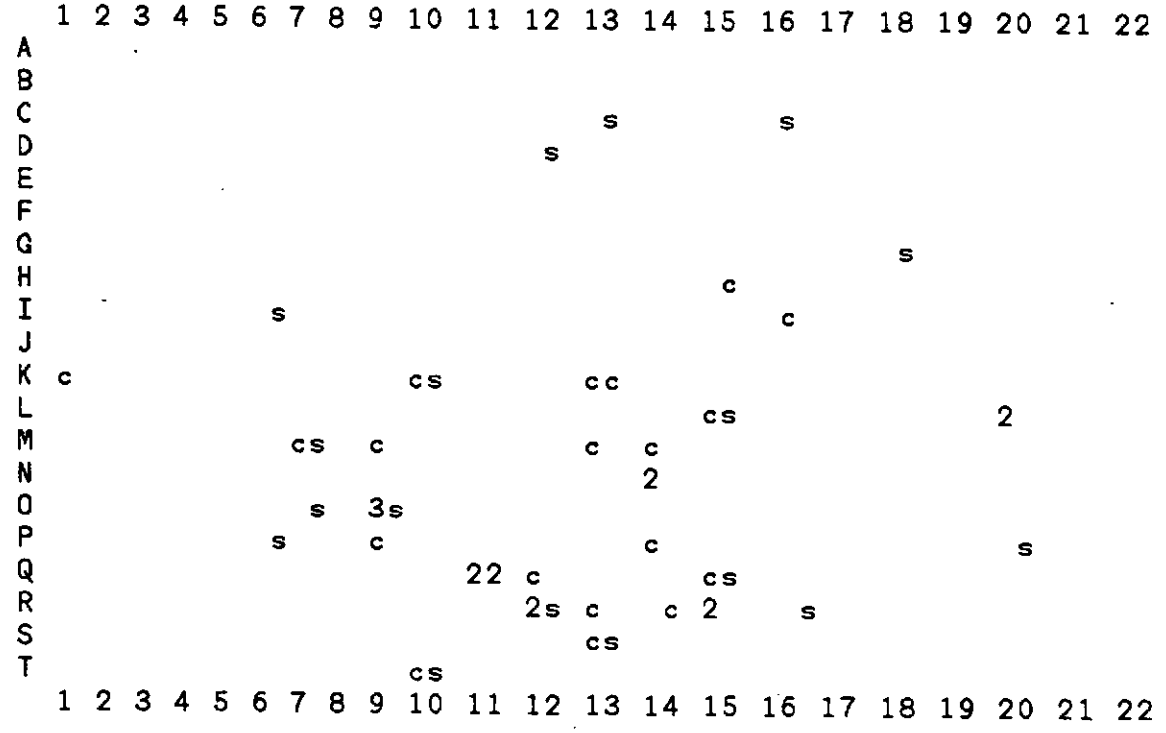


Figure 24: XpeRTD screen display.

This display summarizes history data for 380 RTDs, identified by channel numbers A10-W13: 'c' = clamp is reworked, 's' = spare is used. Number entry indicates that a clamp (spare) is worked 2 or 3 times.

6.4.1 Explanation

Explanation is an essential element of an expert system. It initially provides the programmer with a debugging tool for identifying which rules are used to produce the results. In a working system, it allows the user to follow the expert's reasoning so he may verify the system's decisions. For example, XpeRTD's explanation facility provides the following dialogue:

```
XpeRTD: *** Procedure Explain ***
         What RTD?
>> s10

XpeRTD: RTD s10 diagnosis is connector with CF 800
        because

        Rule 32:
            IF bias is high &
               response is ok &
               opposite is ok &
               elect is no
            THEN diagnosis is connector

        Complete diagnoses are listed in the file
        diag.dat. The final diagnosis for RTD s10
        is

        (diag
         (id "s10" 1000)
         (dx (elect 0)
             (connector 800)
             (clamp 0)
             (spare 0)
             (repl 750)
             (monitor 700)
             (best 700)
             (needs_work 750)
         ))
```

```
This diagnosis is based on current data in the
files trend.dat and hist.dat. RTD s10 trend and
history data are

(trend
 (id "s10" 1000)
 (resp "" 0)
 (bias "high" 800)
 (drift "" 0)
 (deter "" 0)
 (elect "" 0)
)

(hist
 (id "s10" 1000)
 (pos "" 0)
 (bias "" 1000)
 (connector "" 1000)
 (resp "poor" 700)
 (spare "spare" 1000)
 (repl "" 1000)
 (susp 3 1000)
 (clamp 1 1000)
)

...

*** exiting Explain...
```

XpeRTD remembers the rules it used to establish a diagnosis. It records a trace or record of its reasoning with the facts in the diag.dat output file. If the user requests an explanation, XpeRTD consults this file, finds the rule or rules which it used and displays the rules in English.

6.4.2 Training

XpeRTD's explanation facility also serves as a teaching tool. During a single RTD diagnosis, XpeRTD keeps the user informed of its decisions and indicates the basis on which a decision is made. The information recreates the decisions made by the expert and teaches the novice the basics of the diagnostic procedure. The explanation facility provides additional instruction by reproducing the background data and reasoning processes on which the expert's skills depend.

6.4.3 Documentation

XpeRTD displays another feature of a typical expert system. It provides a useful form of documentation which includes

1. a formal description of the diagnostic process,
2. a history of RTD maintainance and repairs, and
3. a statistical record of RTD performance.

The explanation facility captures the essential elements of RTD diagnosis. It summarizes the expert's skill and understanding from five years' experience. This knowledge is

difficult to encode otherwise since the expert, typically, has trouble verbalizing skills which are automatic. The XpeRTD program provides a record of this knowledge which can be maintained and amended. A record is also available when the expert leaves because of a transfer or retirement.

XpeRTD includes an external procedure for updating the RTD history file. This program is designed to record the results of outage repairs and offers a convenient format for maintaining statistical information about RTD performance.

6.5 SAMPLE DIAGNOSIS

XpeRTD produces a diagnosis that is complete. It considers each possibility and recognizes the occurrence of multiple faults. For each RTD, XpeRTD lists data similar to the following output:

```
(diag
 (id "s10" 1000)
 (dx (elect 0)
     (connector 800)
     (clamp 0)
     (spare 0)
     (repl 750)
     (monitor 700)
     (best 700)
     (needs_work 750)
 ))
```

(6.1)

XpeRTD's output gives the expert a complete analysis for each RTD. The explanation and display facilities allow him to verify the analysis. If he is not satisfied with a result, then he may alter the data and execute another diagnosis using the option for single RTD diagnosis.

For each possible fault, XpeRTD evaluates the "possibility" that it causes the suspected RTD failure. The evaluation is based on the certainty of the evidence, e.g., bias is high, and on the expert's certainty that the evidence indicates the fault. This last certainty is recorded as CF(rule) and is specified when a FLOPS rule is coded, e.g.,

```
   bias is high
     -->
     (0.8) diag is connector
is implemented by
   rule 800 (trend ^bias "high")
            (diag )
       -->
       modify 2 ^dx.connector ;
```

The certainty CF(e) of FLOPS program data (e.g., bias is high) is also provided by the user when the data are entered. Data entry during outage planning may involve a large data set, so, for the user's convenience, he is not required to specify a CF(e) value. In this case, FLOPS assumes CF(e) = 1, that is, the evidence is certainly true.

The sample diagnosis (6.1) is based on information which is volunteered by the user and recorded in the file trend.dat. This is an ASCII "stream" file which contains a record for each suspect RTD, e.g.,

```
trend,"s10",1000,"",0,"high",800,"",0,"",0,"",0
```

(6.2)

XpeRTD diagnosis is also based on information in the hist.dat file of RTD work histories. This file contains records of the form

```
hist,"s10",1000,"",0,"",1000,"",1000,"poor",700,
"spare",1000,"",1000,3,1000,1,1000
```

(6.3)

The record structures (6.2) and (6.3) are defined by declarations in a FLOPS program. The files trend.dat and hist.dat may be generated using a standard text editor or using a FLOPS data base editor called FLEDIT. FLEDIT is a menu-driven editor which displays the field data for each record entry. For example, the trend.dat record (6.2) is displayed as

```
ELEMENT DESCRIPTOR NAME: trend
 ATTRIBUTE              id   TYPE ATM    s10
 ATTRIBUTE           id.cf   TYPE CONF   1000
 ATTRIBUTE            resp   TYPE ATM
 ATTRIBUTE         resp.cf   TYPE CONF   0
 ATTRIBUTE            bias   TYPE ATM    high
 ATTRIBUTE         bias.cf   TYPE CONF   800
 ATTRIBUTE           drift   TYPE ATM
 ATTRIBUTE        drift.cf   TYPE CONF   0
 ATTRIBUTE           deter   TYPE ATM
 ATTRIBUTE        deter.cf   TYPE CONF   0
 ATTRIBUTE           elect   TYPE ATM
 ATTRIBUTE        elect.cf   TYPE CONF   0
```

During program execution, XpeRTD reads the trend.dat and hist.dat files into FLOPS' working memory. Each record is represented internally as a list similar to (6.1).

During a diagnosis, XpeRTD first hypothesizes a class of fault, then identifies all RTDs for which this fault is indicated. For example, in one cycle, FLOPS activates all the rules which bear on a connector fault. These are the rules in block 4 of the PRO3 listing in Appendix III.3. Specifically, when block 4 is activated, rule r8 is instantiated with facts (6.2):

```
:rule r8  -- if bias is low &
:              not(diagnosis is clamp)
:            then
:              diagnosis is connector
:
rule 1000 4 (trend ^id <id> ^bias "high" ^bias.cf > 700)
            (diag ^id <id> ^dx.clamp < 500)
    -->
    modify 2 ^dx.conn ;
```

Since facts (6.2) match the rule pattern (trend), the variable <id> is instantiated as "s10". The rule modifies

the working memory element (diag (id "s10")) so the value CF in (dx (conn CF)) is modified. This CF is changed to the posterior CF of rule r8. The posterior CF depends on the CF in (trend (bias "high" CF)).

Similarly, other values in the element (diag (id "s10")) shown in (6.1) are modified by other rules which match facts (6.2) or (6.3).

Appendix IV lists a sample XpeRTD diagnosis. In this example, XpeRTD reads the file trend.dat of suspected RTDs and produces an outage plan for RTD repair and maintainance. The outage plan is written to file diag.dat and includes a complete diagnosis for each suspected RTD.

Chapter 7

CONCLUSION

7.1 THESIS SUMMARY

The objectives proposed for this thesis were both practical and theoretical. Following these objectives, a prototype expert system XpeRTD was designed and tested. Also, the XpeRTD prototype was evaluated with reference to its implementation using a FLOPS "shell" and to its use of a fuzzy logic representation. Specifically, the thesis establishes the following:

1. An expert system provides the best computer solution for RTD diagnosis, a problem which is not well-defined. In fact, without a simulation or statistical model, an expert system is the only solution.
2. The prototype system XpeRTD is capable of performing the expert task of RTD diagnosis. XpeRTD's diagnosis appears to be at least as good as that of the human expert, based on initial test results.
3. Knowledge engineering is assisted by a flexible software tool which doesn't unnecessarily restrict the problem solution. XpeRTD's solution was facilitated by a FLOPS forward-chaining "shell" which also simulates backward chaining and parallel

execution. The solution combines all these features. While a hypothesis-driven backward chainer is usually applied to diagnosis, it was found that this approach is not always the best.

4. The FLOPS programming language exhibits other features which support flexibility in program design and which assist program solutions by offering alternative control and data structures. FLOPS inference engine not only responds to alternate rule formats (i.e., alternate chaining), but it may be sequenced or sectioned using "contexts", rule blocks and rule CFs. FLOPS fact base not only represents exact data, but also data that is "tentative" or "noisy" (i.e., fuzzy sets).

5. Fuzzy logic provides additional flexibility in a rule-based system for representing uncertain data and inexact reasoning. Other schemas entail a number of constraints which limit their application and restrict the design of rule-based systems. Chapter 4 of the thesis compares FLOPS fuzzy logic with the logic used in other expert systems. Specifically, a FLOPS system is compared with the expert systems PROSPECTOR (i.e., a probablistic approach) and MYCIN (i.e., an "ad hoc" representation).

The XpeRTD solution for RTD diagnosis is listed in Appendix III. Programs PR01 and PR02 incorporate rules for

interfacing and initializing the XpeRTD system. PR03 provides a working set of rules for RTD outage planning. In total, these programs represent approximately 2200 lines of FLOPS code and 114 rules. XpeRTD's rules may be categorized as follows:

35 rules:		XpeRTD interface and initilization
21	:	single RTD diagnosis (PR01)
28	:	RTD outage planning (PR03)
30	:	XpeRTD explanation facility
---		
114 rules		

Finally, the sample diagnosis in Appendix IV provided a coarse evaluation of XpeRTD performance. This diagnosis is based on test data containing 112 records of 10 fields and 68 records of 7 fields for a total 180 objects (i.e., RTD records) and 1596 attributes (i.e., RTD facts). Another 200 RTDs were not included in the test data because this number has no history of faults and were not suspect. The program PR03 returned a completed diagnosis after six minutes' run time. This is the time between the user's 'run' command and FLOPS' 'halt', and after FLOPS' compilation of the PR03 rules. FLOPS used almost two minutes' run time to organize the program data, i.e., to perform the matchings on 112 X 68 record combinations over 28 rules of two or three patterns

per rule. The remaining time (i.e., approximately four minutes) includes 248 I/Os (i.e., reads and writes) and more than 200 printlines.

7.2 FURTHER RESEARCH

The XpeRTD prototype is an initial design for a software system applied to RTD diagnosis. Further development of XpeRTD and the problems attending its design suggest other research, which is summarized as follows:

1. XpeRTD diagnoses only classes of mechanical faults. Electrical faults are better defined and they are diagnosed using simulation hardware and software (Patterson, 1987). XpeRTD could be expanded to incorporate these other faults. An expanded system would provide a single program for diagnosing RTD failures and further rationalize the diagnostic process by consolidating all the relevant data.
2. An expanded XpeRTD system could be developed as an on-line system. This system would identify RTD failures from control data and diagnose RTD faults in case of an alarm or warning during normal operations (Patterson, 1987).
3. XpeRTD might be improved using statistical data which is not now available. RTD bias and response data could be quantified and accumulated over time. This data could be correlated with RTD maintainance

records to provide a statistical model of RTD behavior. Diagnostic algorithms using statistical data are detailed in de Kleer and Williams (1987) and Ben-Bassat et al (1980). Civanlar and Trussell (1986) explain how this data might be translated into a FLOPS fuzzy set representation.

4. FLOPS applications would be assisted by a library of C-based routines for file-handling, data entry and display, and other programs for user interface and data communication. A good deal of the development time for XpeRTD was given to designing external procedures to perform these tasks. The existing routines accomodate RTD data collection and XpeRTD file-handling, but they could be generalized for use with any FLOPS application.

5. XpeRTD's interface could be improved using graphics software to provide a mouse-sensitive graphics display for input of program data and to interpret file data and program results. To access these routines, FLOPS includes a 'call' command which communicates with both external procedures and the DOS operating system.

6. A FLOPS library could also contain a program based on those of Shaw and Gaines (1984) and Boose (1984) to assist in engineering other FLOPS' expert systems. These programs automate the interviewing process.

They provide the expert with a user-friendly tool which he may use at his convenience. In practice, experts are often more responsive to this type of interview than a formal one with a knowledge engineer, especially when the information being solicited must be quantified.

7. FLOPS' rule-based implementation could possibly support a more sophisticated interface, based on natural languages, for example, an area of active research in expert system design. XpeRTD's explanation facility, for example, uses "canned text". Each rule is duplicated as a string of text and the appropriate string is written to the screen to "explain" a rule. A rule-based production system such as FLOPS has a built-in language processor, i.e., its rule format "understands" a grammar which is represented as a set of productions (Michalski et al., 1983). FLOPS fuzzy logic also "understands" English statements which are vague or ambiguous (Zadeh, 1979).

8. XpeRTD could be developed to include some of the higher level routines of other expert systems. FLOPS incorporates a facility for "metarules", i.e., rules about rules (Siler and Tucker, 1986). Metarules could possibly be used to implement learning in XpeRTD. Additional routines could give the system an

ability to learn from its experience or to coach the user for new rules when it is unable to make a decision (Michalski et al., 1983). For example, XpeRTD could use metarules to query the user about symptoms and faults. This information could then be applied to generate a new XpeRTD rule for RTD diagnosis.

Appendix I

FUZZY SETS AND FUZZY LOGIC

Fuzzy set theory (Zadeh, 1965), despite its "fuzzy" denotation, is grounded in ordinary set theory. It also extends the concept of ordinary sets to provide a framework that, for many practical applications is more workable. Fuzzy set theory has been successfully used to control industrial processes (Kickert and van Nauta Lemke, 1976) and to simulate aircraft flight (Larkin, 1985). It was originally developed for solving pattern recognition problems where classes are approximately described or where patterns are not strictly random (Zadeh, 1979). Generally, it serves to estimate concepts which humans apply to everyday problems and to practical decision-making. Such estimates often suffice where more precise information is not available because processes are too complex.

This appendix presents a formal definition of fuzzy sets and fuzzy set operators. It also introduces the basic concepts of fuzzy logic, which extends fuzzy set theory to fuzzy or "approximate" reasoning.

Fuzzy Sets (Dubois and Prade, 1980). If U is a set of objects, or universe, with elements u, then an ordinary

subset A of U may be defined by a membership or characteristic function m(A,u): U --> [0,1] such that

$$m(A,u) = 1 \text{ iff } u \text{ in } A ,$$

$$m(A,u) = 0 \text{ iff } u \text{ not in } A .$$

In other words, for each u in U, 'u in A' is either true or false. A fuzzy set generalizes the concept of membership by defining the characteristic function of a subset F over the interval [0,1].

Definition. A fuzzy set F is a set of ordered pairs

$$F = \{(m(F,u),u) \mid u \text{ in } U\} ,$$

where m(F,u) specifies the degree of membership of u in F.

For example, let F be the fuzzy set of real numbers around 10. For U = R = reals, define a membership function m so that

$$m(F,u) = \frac{1}{1 + (u - 10)^2} ;$$

then

$$F = \{\ldots,(0.01,0),\ldots,(0.25,5),\ldots,(1,10),\ldots\} .$$

In other words, 10 is certainly a 'real number around 10'; 5 is such a number with less certainty (0.25) and 0, with even less certainty (0.01).

Definition. The union and intersection of two fuzzy subsets A, B are defined as operations on the membership functions m(A,u) and m(B,u) over U, i.e,

$$m[\text{union}(A,B),u] = \max[m(A,u),m(B,u)] \quad , \qquad (I.1)$$

$$m[\text{intersection}(A,B),u] = \min[m(A,u),m(B,u)] \quad . \qquad (I.2)$$

The min(), max() operators are the only operators which guarantee the usual properties of combinations over ordinary sets and fuzzy sets, i.e., definitions (I.1) and (I.2) define functions which have the following properties:

1. The membership value of u in union(A,B) or intersection(A,B) depends on its membership value in A and B. Also, membership in the intersection set is more constrained than membership in a singleton set; union is less constrained.
2. Complete membership in A and in B implies complete membership in the intersection(A,B); and, complete lack of membership implies lack of membership in the union(A,B): i.e., min(1,1) = 1 and max(0,0) = 0.
3. Operators min(), max() are commutative, associative and mutually distributive.
4. Min(), max() are continuous and nondecreasing functions with respect to each of their arguments; i.e., the membership value of u in union(A,B) or intersect(A,B) doesn't decrease when m(A,u) or m(B,u) increases; and, it doesn't strongly increase when m(A,u) or m(B,u) increases only slightly.

5. For f() = min() or max(), f[m(A,x1),m(A,x2)] is strictly increasing: i.e., for *() = union() or intersection(), if m(A,x1) = m(B,x1) > m(A,x2) = m(B,x2) then m[*(A,B),x1] is certainly greater than m[*(A,B),x2].

Generally, these restrictions guarantee the "effectiveness" or "computability" of the min(), max() functions, even when they are extended to three or more arguments. Property 5, for example, guarantees that the results of the operations min(), max() are consistent.

Definition. The complement ¬F of a fuzzy set F is defined by the membership function

$$m(\neg F,u) = 1 - m(F,u) \qquad (I.3)$$

Fuzzy structures. The concept of fuzzy sets defined by equations (I.1) - (I.3) can be extended to a fuzzy "algebra". Specifically, a set P(U) of fuzzy subsets of U with union, intersection and complementation operators denoted by '+', '*' and '¬' defines a structure S = (P(U), +, *, ¬) with the following algebraic properties:

a. Commutivity:
$$A + B = B + A \ ;$$
$$A * B = B * A \ .$$

b. Associativity:
$$A + (B + C) = (A + B) + C \ ;$$
$$A * (B * C) = (A * B) * C \ .$$

c. Idempotency:

$$A + A = A \ ;$$
$$A * A = A \ .$$

d. Distributivity:

$$A + (B * C) = (A + B) * (A + C) \ ;$$
$$A * (B + C) = (A * B) + (A * C) \ .$$

e. Absorption by { } and U:

$$A + U = U \ .$$
$$A * \{\ \} = \{\ \} \ ;$$

f. Identity:

$$A + \{\ \} = A \ ;$$
$$A * U = A \ .$$

g. Absorption:

$$A + (A * B) = A \ ;$$
$$A * (A + B) = A \ .$$

h. De Morgan's Laws:

$$\neg(A * B) = (\neg A + \neg B) \ ;$$
$$\neg(A + B) = (\neg A * \neg B) \ .$$

i. Involution:

$$\neg\neg A = A \ .$$

j. Equivalence:

$$[(\neg A + B) * (A + \neg B)] = [(\neg A * \neg B) + (A * B)] \ .$$

k. Symmetry:

$$[(\neg A * B) + (A * \neg B)) = [(\neg A + \neg B) * (A + B)] \ .$$

Only two laws of ordinary set theory do not apply to fuzzy structures. They are

l. Contradiction:

$$A * \neg A \neq \{\ \} \ ;$$

m. Excluded middle:

$$A + \neg A \neq U \ .$$

In other words, since fuzzy sets A, ¬A have no definite boundaries, then union(A,¬A) does not exactly cover the

universe U. Since A, ¬A may overlap, intersection(A,¬A) is not necessarily empty.

The structure S = (P(U), +, *, ¬) is a lattice induced by the set [0,1]; i.e.,

1. S is partially ordered by [0,1] and the ordering, defined by '≤':p1,p2 in P --> [0,1] X [0,1], is reflexive and transitive;
2. for each pair p1,p2 in P and u in U, there exist unique elements glb and lub where
   a) glb = max[m(p1,u),m(p2,u)] ,
   b) lub = min[m(p1,u),m(p2,u)] .

In other words, max() defines a greatest lower bound and min(), a least upper bound, for sets in P. S is a complete lattice because it contains least and greatest elements 0, 1; i.e., for any element p in P and u in U,

min[m(p,u),m({},u)] = 0 ,
max[m(p,u),m(P,u)] = 1 .

Fuzzy logic. The fuzzy structure S also provides a framework for a fuzzy logic F = (V(P),and,or,¬). V(P) is a valuation function over the set of fuzzy propositions P so that V: P --> [0,1]. Fuzzy logic extends ordinary propositional and predicate calculus by replacing the usual truth function T:P --> {0,1} by V(P). For each proposition p in P, V also defines a possibility function poss(p):p -->

[0,1]. For p:'x is A', poss(A) estimates the possibility that p is true. The possibility function, an assignment function in logic, also specifies a membership function m(A) for x in A.

Fuzzy logic operators for logical combinations are defined by

$$\begin{aligned} &p \text{ and } q : \min[\text{poss}(p), \text{poss}(q)] ; \\ &p \text{ or } q : \max[\text{poss}(p), \text{poss}(q)] ; \\ &\neg p : 1 - \text{poss}(p) . \end{aligned} \qquad \text{(I.4)}$$

These operators reduce to the usual Boolean functions when poss(p) is 1 or 0.

The logic operator '-->' for conditional statements is variously defined depending on the interpretation of p:'if x is A then y is B'. Gaines (1979) suggests at least two definitions:

$$p \dashrightarrow q : \min[1, 1 - \text{poss}(p) + \text{poss}(q)] ; \qquad \text{(I.5)}$$

$$p \dashrightarrow q : \min[1, \text{poss}(q)/\text{poss}(p)] . \qquad \text{(I.6)}$$

Equation (I.5) follows the usual interpretation of a multivalued logic which identifies p --> q with or(¬p,q). Equation (I.6) assumes a probabilistic interpretation which equates the conditional p --> q with the probability prob(q|p):

$$\text{poss}(p \dashrightarrow q) = \text{prob}(q|p) = \text{prob}(\text{and}(p,q)) / \text{prob}(p) .$$

Equation (I.5) guarantees that the fuzzy logic F is complete under the operators min(), max() and -->(). Equation (I.6) is inconsistent unless sets A and B in 'A --> B' are ordinary sets (Dubois and Prade, 1980; Zadeh, 1988).

Zadeh (1978) provides another interpretation for -->() based on the definition that 'A --> B' is a relation A X B on fuzzy sets A and B:

$$m[A \times B,(a,b)] = \min[m(A,a),m(B,b)] \quad . \qquad (I.7)$$

Rules of fuzzy logic based on the conditional will be evaluated differently depending on definitions (I.5) - (I.7). For example, the rule modus ponens is

```
p --> q     (CF1)
p           (CF2)
-------
q           (CF3)   .                    (I.8)
```

In other words, if 'p --> q' is true with certainty CF1 and 'p' is true with certainty CF2, then conclude 'q' with certainty CF3. The "truth" CF3 of the proposition 'q' depends on the certainties CF1 and CF2. The "truth" assignment T(CF1,CF2) = CF3 is evaluated differently by definitions (I.5) - (I.7). In practice, the evaluation is useful if it guarantees consistency.

FLOPS fuzzy logic is consistent because it incorporates the fuzzy logic operators (I.4) - (I.5). FLOPS' "inference engine" interprets modus ponens inference by

```
p --> q    poss(rule)
p          poss(p)
-----------------------------------
q          poss(q) = min[poss(p),poss(rule)]
```

(I.9)

In other words, FLOPS interprets inference (I.9) as a conjunction and(). This is consistent with a logical interpretation which establishes an equivalence between and[(p --> q),p] and q; i.e., the conclusion q has the same truth value as and().

FLOPS implements chaining on two or more rules r_i: p_i --> q following equation (I.7). Equation (I.7) defines the compositional rule of inference, a generalization of modus ponens in terms of fuzzy set operations (Zadeh, 1983). Equation (I.7) is consistent in a rule-based system under a number of logical restrictions. FLOPS guarantees these restrictions by interpreting the values poss() in inference (I.9) as necessity measures (Dubois and Prade, 1987). This interpretation accounts for the inconsistency of fuzzy sets introduced by the contradiction of fuzzy intersection; i.e., for fuzzy set A,

$$\text{intersection}(A, \neg A) \neq \{\} .$$

FLOPS also restricts the format of rules p --> q so that its knowledge base conforms to a standard logical format called "conjunctive normal form" (CNF). FLOPS' knowledge base is interpreted as a disjunction or(f_i, r_i) of facts f_i and rules r_i. Each rule r_i is a conjunction and(f_j) of facts. Then,

$$KB = or[f_i, and(f_j)] \quad , \qquad (I.10)$$

where KB is FLOPS' knowledge base. In FLOPS, a set of propositions or(and()) is evaluated as max[min[]], which is consistent with the equation in (I.7).

Equation (I.7) justifies using inference (I.9) to evaluate fuzzy sets, i.e., to "prove" fuzzy propositions of the form p:'bias is high' where 'high' is interpreted as a fuzzy set. Dubois and Prade (1987) also justify using inference (I.9) to evaluate "fuzzy" probabilities, i.e., to prove fuzzy propositions of the form p:'bias is high is likely' where 'likely' is represented as a fuzzy measure most(p). Dubois and Prade establish that FLOPS inference (I.9) is consistent when it is applied to fuzzy necessity measures. They also demonstrate that probabilities are inconsistent in the same system. Further, probabilities have lower bounds which, in the case of chaining, quickly converge to zero. Fuzzy necessity measures have greater lower bounds, so, in practice, they are more informative.

Appendix II

FLOPS COMPARED WITH OTHER EXPERT SHELLS

FLOPS (Siler and Tucker, 1986) is a rule-based programming language. It is a high-level language that provides the "shell" or top-level procedures for a rule-based expert system. FLOPS' interpreter operates as a forward-chaining "inference engine" that searches the knowledge base and matches rules and facts. FLOPS' syntax includes specific commands to define the rules and facts of an expert system knowledge base. These characteristics are the basic elements of an ever-growing number of expert system shells, software and/or hardware dedicated to expert systems applications. The range of other features which are available are summarized in Table 4 (Cutter Information Corp., 1986). Table 4 lists the more advanced features first, so the example shells at the top of the list inherit the characteristics of the other shells.

FLOPS is a DOS-based PC software package that is recommended because of its flexibility. Based on the programming language OPS (Brownston et al., 1986), FLOPS inherits the general applicability of the OPS language. It lacks the sophisticated interface of other similarly priced PC expert system shells, but this sophistication can be

TABLE 4

Features: Expert System Shells.

User Interface	ES Shells	FLOPS
Graphics	ART, KEE	no
Online editors	PC Plus	no
Browse knowldege base	KES	yes
Trace, debug	OPS5, Nexpert, PC Easy	yes
External procedures	Exsys	yes

Inference and Control	ES Shells	FLOPS
Semantic networks, truth maintainance	ART, KEE	no
Objects, frames	PC Plus	no
Multiple instantiations, Structured rules, Backward & forward chaining, Advanced CFs	KES, OPS5, Nexpert, PC Easy	yes
Simple rules, User CFs	Exsys	yes

developed using FLOPS' facility to incorporate external procedures. The best PC-based shells are rated "cost-inefficient" precisely because they are not flexible. They dictate the format for rule and database representations. They also dictate a system design by restricting user access to the top-level control and other procedures .

FLOPS is unique among PC-based shells in handling "common sense" reasoning. It incorporates fuzzy set definitions and fuzzy logic procedures to provide a sophisticated and general-purpose format for reasoning with uncertain data. FLOPS also offers the choice of sequential or parallel firing of rules. Parallel execution facilitates a situation where a single task applies to a number of facts at once.

FLOPS flexibility offers the potential for development which is limited only by the size and speed of the hardware. As a demonstration of this potential, it is interesting to compare (Table 5) FLOPS capabilities with a state-of-the-art expert system shell called ART (Inference Corp., 1987). This comparison is suggested because both shells incorporate the syntax and implementation of an OPS production system.

The major differences between ART and FLOPS are due to their different applications. ART is a sophisticated, user-friendly design tool for expert systems. It operates in a powerful, Lisp-based workstation environment. ART provides menu-driven graphics and extensive editing and debugging tools. It accomodates advanced data structures (i.e., semantic networks) which are defined as sets of frames or "schemas" and which are managed by built-in routines to assist the user in defining inheritance and transitivity among schema. These routines also guarantee "truth maintainance" so, when a fact is deleted from the fact base,

TABLE 5

FLOPS compared with ART.

ART	FLOPS
1. Relational data and object-oriented data	Relational data
2. Totally data-driven, opportunistic	Data-driven, opportunistic or optimal
3. Logical dependencies provide logical consistency	Logical consistency based on fuzzy logic
4. Forward and backward reasoning in the same application	Same
5. Alternative hypotheses and temporal states	Alternative hypotheses and temporal data
6. Extensive development and debugging aids	Limited development and debugging

ART erases other facts whose "proof" depends on the one deleted.

Expert systems are designed in ART as rule-based systems in a Lisp environment. Rules are developed using an extended Lisp. They are defined as an ART program which may be compiled and maintained as object file. ART object files may then be edited to accomodate rule modifications. ART rules may contain conditions (LHSs) with any combination of logical operators. Rule actions (RHSs) and conditions (LHSs) may call Lisp functions and procedures.

FLOPS is a general-purpose programming language for DOS-based hardware. Its rule-based format is easily applied to rule-based expert systems but it lacks the other features which are built into ART. FLOPS operates in an interactive environment and responds to FLOPS commands. FLOPS provides a record data structure which is represented as an object list of attribute-value pairs that must be declared in a FLOPS' program. FLOPS communicates with external procedures using a calling convention that is compatible with the C programming language. The call command also communicates directly with the DOS operating system.

An expert system is designed in FLOPS as a FLOPS' program of rules. Rules are defined using a rule command and may be maintained and edited as a source file. They are compiled at run time but FLOPS saves no object code, i.e., the compilation is lost when the user terminates an interactive session. Rule conditions (LHSs) are interpreted as a conjunction and() of FLOPS' "patterns" which must match exactly with FLOPS' facts. External procedures and FLOPS functions may be used only as rule actions (RHSs).

FLOPS provides other features which, allowing for the different environments, compares favorably with ART. FLOPS includes a data base editor and debugging facility which allow the user to browse FLOPS' knowledge base of facts and rules, and to access FLOPS' inference engine (i.e., the

agenda or rule stack). FLOPS' record structure defines an object-attribute-value relation and provides the basis for building composite structures including frames and networks. Networks cannot be represented directly but they may be built by rule definitions using "contexts" and rule blocks. Also, FLOPS' records are also defined by "time tags", the basis for ART's temporal states and truth maintainance procedures.

Appendix III

FLOPS PROGRAM LISTINGS

III.1 PROTOTYPE PRO1

```
:
:
: program pro1.fps
:
: FLOPS sequential program to diagnose single RTD.
:
: ...reads RTD history data from file pro1.dat. Generates
: a single diagnosis based on file data and on information
: which is volunteered by the user.
:
: PRO1 is menu-driven and allows the user to select among
:
:      1) RTD diagnosis ,
:      2) display/entry of RTD data , or
:      3) explanation of PRO1 diagnosis .
:
: During diagnosis, PRO1 rules match bias and response
: "classes". Classifications are either given by the
: user, or determined from bias and response data in the
: PRO1 fact base. Delta_T temperature values are used to
: classify response as GOOD, POOR or BAD. An abias value
: classifies device bias as LOW, HIGH, BETWEEN and NONE.
: These computations may use fuzzy sets and are based on
: mean values which are defined in the database.
:
: Generally, PRO1 rules are fired using FLOPS forward
: chaining with data-driven and event-driven control.
: That is, rules are fired when LHS conditions match with
: facts in the fact base. Also, rules are sectioned as
: "blocks" whose firing can be ordered using block commands.
:
: A FLOPS fact base is defined by 'literalize' and 'make'
: commands. PRO1's fact base records RTD device characteristics
: and performance data. Specifically, these structures represent
:
:  RTD device --            (RTD (id)(clamp "yes") (spare "no"))
:
:  performance history --   (hist (rtd)(abias degrees)(bias class)
:                              (drift degrees))
:
:  performance data --      (trend (rtd)(max_delta_T minutes)
```

```
:                                   (2/3_delta_T minutes)
:                                   (response class)(deter "yes"))
:
:
: This version of PRO1 uses strict but not fuzzy rules. It
: provides an explanation facility using a "history tree" to
: record the rules employed during a diagnosis. This facility
: doesn't explain "how" PRO1 performs, but it explains the
: "why" of PRO1's diagnosis. This is an efficient explanation
: facility for a program which, using fuzzy logic, may return
: more than one conclusion.
:
:
write '\nPROGRAM RTD DIAGNOSIS';
write ' n...is menu-driven and includes the following options:';
write '\n';
write ' n      1) Diagnose RTD';
write ' n      2) Display/Enter RTD Data';
write ' n      3) Explain RTD Diagnosis';
write '\n';
write ' nThe Display/Enter option lets you see or change the
write ' nprogram's knowledge base. This data includes...';
write '\n';
write ' n      a) RTD Response is...GOOD POOR or BAD';
write ' n      b) RTD Response has deteriorated...YES or NO';
write ' n      c) RTD 2/3, Max Delta T is...(minutes)';
write ' n      d) RTD Clamp is reworked...YES or NO';
write ' n      e) RTD Spare Guide Tube is used...YES or NO';
write ' n      f) RTD Bias has drifted...YES or NO';
write ' n      g) RTD Bias is...(degrees C)';
write ' n      h) Channel Mean Bias/Delta T is...';
write ' n                                   (degrees/minutes)';
write ' nRTD DIAGNOSIS Main Menu to follow...';
write '\n';


:declarations
:
write ' n   compiling pro1.fps data...';
:
literalize rtd
  id       atm
  clamp    atm
  spare    atm
  connector  atm
  work     atm
;

literalize trend
  rtd      atm
  dt2      flt
  dtmax    flt
  resp     fzset  (  good
                     poor
```

```
                        bad   )
  deter   atm
;

literalize hist
  rtd      atm
  abias    flt
  bias     fzset   (  high
                      low
                      between
                      none   )
  drift    flt
;

literalize mean
  bias    flt
  nbias   flt
  dt2     flt
  dtmax   flt
  delta   flt
;

literalize work
  rtd    atm
  rule   int
  diag   atm
  conf   int
  trace  int
;

literalize explan
  rule int
;

literalize menu
  action   int
;

literalize data
  entry   int
  rtd     atm
  val     int
  temp    flt
;

literalize enable
  block   int
;

write 'ok n';
write '\n   compiling rules...';
:
: Rules
:
```

```
: PRO1 rules define at least three different procedures...
: RTD diagnosis, explanation and knowledge base operations.
: Rule r0 is the driver which presents these options.
: Subsequent rules are fired depending on the user's choice...
:
write 'block prelim...';


:-------------------------------------------------------------
: Block 1. Provides user menu and initializes RTD data.
:
: Rule 0 --
:   Displays MAIN MENU and requests user selection...
:
rule menu 1000 1 (menu ^action.cf 0)
    -->
    fire all off,
    fire block 1 on,
    write '\n',
    write '\n*****************************************',
    write '\n**  RTD Diagnosis Expert...Main Menu  **',
    write '\n*****************************************\n',
    write '\n      1) Diagnose RTD',
    write '\n      2) Display/Enter RTD Data',
    write '\n      3) Explain RTD Diagnosis',
    write '\n      4) Exit to DOS\n',
    write '\n   Please type a NUMBER 1-4...\n',
    read 1 ^action;

: Rule 1.1
:
: If user selects main/DIAGNOSIS...
:   prompts user for RTD data, initializes RTD work and resets
:   blocking...
rule start 1000 1 (menu ^action 1)
                  (work ^rtd.cf 0)
                  (enable ^block.cf 0)
    -->
    write '\n***procedure Diagnosis***\n',
    write ' n With test database, RTDs are identified by codes
    write ' n that represent corresponding data values, ie.,',
    write ' n Bias/Response/Clamp/Spare/Deterioration... n',
    write ' n For example, "HPCSD" names the device with HIGH',
    write ' n bias, POOR response, CLAMP is reworked, SPARE',
    write ' n is used and response shows DETERIORATION.',
    write '\n',
    write ' n  Using this code, what RTD will be diagnosed? n',
    read 2 ^rtd,
    write '\n   Diagnosing RTD...\n',
    fire all on,
    fire block 1 off,
    fire block 2 off,
    fire block 3 off;
```

```
:
: Rule 1.1.0
:
: If user selects MAIN/DIAGNOSIS and response is reset,
:   sets blocking to turn off class rules...
:r0
rule 1000 1 (menu ^action 1)
            (work ^rtd.cf 0)
            (enable ^block.cf > 0)
    -->
    write '\n***procedure Diagnosis***\n',
    write ' n With test database, RTDs are identified by codes',
    write ' n that represent corresponding data values, ie.,',
    write ' n Bias/Response/Clamp/Spare/Deterioration... n',
    write ' n For example, "HPCSD" names the device with HIGH',
    write ' n bias, POOR response, CLAMP is reworked, SPARE',
    write ' n is used and response shows DETERIORATION. n',
    write ' n  Using this code, what RTD will be diagnosed? n',
    read 2 ^rtd,
    write '\n   Diagnosing RTD...\n',
    fire all on,
    fire block 1 off,
    fire block 2 off,
    fire block 3 off,
    fire block 5 off;

:
: Rule 1.2
:
:  If user selects MAIN/DATA,
:    sets blocking for data rules...
:r1
rule 1000 1 (menu ^action 2)
    -->
    fire block 1 off,
    fire block 2 on;


:-------------------------------------------------------------
: Block 2. Includes data entry and display routines.
:
: Rule 1.2.0.
:   If user selects DATA...
:   prints DATA menu and prompts for user option...
:r2
rule data 1000 2 (menu ^action 2)
                 (data ^entry.cf 0  ^rtd.cf 0)
    -->
    write '\n*****************************************',
    write '\n**  RTD Diagnosis Expert...Data Menu  **',
    write '\n*****************************************\n',
    write ' n   1) RTD Response is...GOOD POOR or BAD',
    write ' n   2) RTD Response has deteriorated...YES or NO',
    write ' n   3) RTD 2/3, Max Delta T is...(minutes)',
```

```
    write ' n   4) RTD Clamp is reworked...YES or NO',
    write ' n   5) RTD Spare Guide Tube is used...YES or NO',
    write ' n   6) RTD Bias drift is...(degrees C)',
    write ' n   7) RTD Bias is...(degrees C)',
    write ' n   8) Channel Mean is...(Bias/Delta T)',
    write ' n   9) Display RTD Data.',
    write ' n  10) Exit to MAIN MENU. n',
    write ' n What NUMBER? n', read 2 ^entry,
    write ' n What RTD do you want to see or change? n',
    read 2 ^rtd ;

: Rule 1.2.1
:
: Depending on user selection from DATA menu,
:   reads response class...
:r3
rule 1000 2 (menu ^action 2)
            (data ^entry  1  ^rtd <id>  ^val.cf 0)
    -->
    write '\n',
    write '\n***procedure Change_Bias***\n',
    write ' n   Based on current refuelling trend for',
    write ' n      channel <id>, RTD response is... n',
    write '\n                          1) GOOD',
    write '\n                          2) POOR',
    write '\n                          3) BAD\n',
    write ' n      Enter a NUMBER 1-3...n',
    read 2 ^val;

: Rule 1.2.1.1
:
: Depending user DATA entry,
:    classifies response as GOOD and sets blocking flag...
:r4
rule 1000 2 (menu ^action 2)
            (data ^entry  1  ^rtd <id>  ^val 1)
            (trend ^rtd <id>)
            (enable)
    -->
    write ' nRecording response...',
    modify 3 ^resp.good  1000  ^deter "no",
    write ' nAssuming RTD response shows no deterioration... n',
    modify 4 ^block 5,
    delete 2, make data;

: Rule 1.2.1.2
:
: Depending user DATA entry,
:    classifies response as POOR and sets blocking flag...
:r5
rule 1000 2 (menu ^action 2)
            (data ^entry  1  ^rtd <id>  ^val 2)
            (trend ^rtd <id>)
            (enable)
```

```
    -->
    write '\n   Recording response...';
    modify 3 ^resp.poor  1000  ^deter "yes",
    write '\n   Assuming RTD response has deteriorated...\n',
    modify 4 ^block 5,
    delete 2, make data;

: Rule 1.2.1.3
:
: Depending user DATA entry,
:    classifies response as BAD and sets blocking flag...
: r6
rule 1000 2 (menu ^action 2)
            (data ^entry  1  ^rtd <id>  ^val 3)
            (trend ^rtd <id>)
            (enable)
    -->
    write '\n   Recording response...',
    modify 3 ^resp.bad  1000  ^deter "yes",
    write '\n   Assuming RTD response has deteriorated...\n',
    modify 4 ^block 5,
    delete 2, make data;

: Rule 1.2.2
:
: Depending on DATA entry,
:   records response deterioration...
: r7
rule 1000 2 (menu ^action 2)
            (data ^entry 2  ^rtd <id>)
            (trend ^rtd <id>)
    -->
    write '\n',
    write '\n***procedure Change_Response***\n',
    write ' n Based on channel refuelling trends, n',
    write ' n RTD <id> response has deteriorated...YES or NO? n',
    read 3 ^deter,
    delete 2, make data,
    write '\n';

: Rule 1.2.3
:
: If user  selects DATA,
:   reads and records delta t...
: r8
rule 1000 2 (menu ^action 2)
            (data ^entry 3  ^rtd <id>)
            (trend ^rtd <id>)
    -->
    write '\n',
    write '\n***procedure Change_Delta_T***\n',
    write ' n With reference to channel <id> refuelling trend...',
    write ' n    what is Max Delta T in minutes? n',
    read 3 ^dtmax,
```

```
    write '\n       what is 2/3 Delta T?\n',
    read 3 ^dt2,
    delete 2, make data;

: Rule 1.2.4
:
: If user selects DATA,
:   reads and records new clamp value...
: r9
rule 1000 2 (menu ^action 2)
            (data ^entry 4  ^rtd <id>)
            (rtd ^id <id>)
    -->
    write '\n',
    write '\n***procedure Change_Clamp_is_Reworked***\n',
    write '\n       RTD <id> clamp was reworked...YES or NO?\n',
    read 3 ^clamp,
    delete 2, make data;

: Rule 1.2.5
:
: Depending on user selection from DATA menu,
:   reads and resets spare...
: r10
rule 1000 2 (menu ^action 2)
            (data ^entry 5  ^rtd <id>)
            (rtd ^id <id>)
    -->
    write '\n',
    write '\n***procedure Change_Spare_is_Used***\n',
    write '\n       RTD <id> in spare guide tube...YES or NO?\n',
    read 3 ^spare,
    delete 2, make data;

: Rule 1.2.6
:
: Depending on user selection from DATA menu,
:   reads and resets drift...
: r11
rule 1000 2 (menu ^action 2)
            (data ^entry 6  ^rtd <id>)
            (hist ^rtd <id>)
    -->
    write '\n',
    write '\n***procedure Change_Drift***\n',
    write '\n   Based on channel bias data,\n',
    write '\n       RTD <id> bias has drifted...YES or NO?\n',
    read 3 ^drift,
    delete 2, make data;
: Rule 1.2.7
:
: Depending on user selection from DATA menu,
:   saves current bias as temp value...
: r12
```

```
rule 1000 2 (menu ^action 2)
            (data ^entry 7  ^rtd <id>  ^temp.cf 0)
            (hist ^rtd <id>  ^abias <b>)
    -->
    modify 2 ^temp <b>;

: Rule 1.2.7.1
:
: Depending on user choice, records new bias and,
:   if bias has drifted, records bias drift...
: r13
rule 1000 2 (menu ^action 2)
            (data ^entry 7  ^rtd <id>  ^temp.cf > 0  ^temp <t>)
            (hist ^rtd <id>  ^abias <b>  ^drift <d>)
    -->
    write '\n',
    write '\n***procedure Change_Bias***\n',
    write ' n Based on channel bias data, n',
    write '\n       what is RTD <id> bias in degrees C?\n',
    read 3 ^abias,
    modify 3 ^drift ( <t> - <b> ),
    write ' n   Recording RTD drift  n',
    write ' n       = current bias - previous bias... n',
    delete 2, make data;

: Rule 1.2.8
:
: Depending on user selection from DATA menu,
:   records bias and delta T mean values...
: r14
rule 1000 2 (menu ^action 2)
            (data ^entry 8  ^val.cf 0)
    -->
    write '\n',
    write '\n***procedure Change_Means***\n',
    write ' n Record new mean value for...',
    write '\n',
    write '\n                1) Bias  (degrees C)',
    write '\n                2) Max Delta T  (minutes)',
    write '\n                3) 2/3 Delta T  (minutes)',
    write '\n',
    write ' n Enter a NUMBER 1-3. n',
    read 2 ^val;

: Rule 1.2.8.1
:
: If user selects to update mean value,
:   records mean BIAS...
: r15
rule 1000 2 (menu ^action 2)
            (data ^entry 8  ^val 1)
            (mean ^bias <b>  ^nbias <nb>)
    -->
    write '\n',
```

```
    write '\n***subroutine Change_Mean_Bias***\n',
    write ' n Change mean BIAS to...? n',
    read 3 ^bias,
    modify 3 ^nbias ( <b> * -1 ),
    write ' n Based on your entry, an RTD reading is... n',
    write '\n',
    write ' n  1) HIGH if Bias is less than -<b> degrees C n',
    write ' n  2) LOW    ..is greater than +<b>',
    write ' n  3) BETWEEN    ...is between [1.0, <b>]',
    write ' n  4) OK         ...is between [-<b>,1.0] n',
    delete 2, make data;

: Rule 1.2.8.2
:
: If user selects update mean value,
:       records mean MAX DELTA T...
: r16
rule 1000 2 (menu ^action 2)
            (data ^entry 8  ^val 2)
            (mean ^ dt2 <d2>  ^dtmax <dm>  ^delta <d>)
    -->
    write '\n',
    write '\n***subroutine Change_Mean_Delta***\n',
    write ' n Change MEAN max Delta T to...? n',
    read 3 ^dtmax,
    modify 3 ^delta ( (<dm> - <d2>) + <dm>),
    write ' n Based on your entry, a channel response is ... n',
    write ' n  1) GOOD  if RTD response is less than <dm>',
    write ' n  2) POOR                ...is between [<dm>, <d>]',
    write ' n  3) BAD                 ...is greater than <d> n',
    delete 2, make data;

: Rule 1.2.8.3
:
: If user selects update mean value,
:       records mean 2/3 DELTA T...
: r17
rule 1000 2 (menu ^action 2)
            (data ^entry 8  ^val 1)
            (mean ^dt2 <d2>)
    -->
    write '\n',
    write '\n***subroutine Change_Mean_Delta***\n',
    write ' n Change MEAN 2/3 Delta T to...? n',
    read 3 ^dt2,
    write ' n Based on your entry, a channel response is... n',
    write ' n   1) GOOD if RTD response is less than <d2>',
    write ' n   2) POOR ',
    write ' nor 3) BAD                ...is greater than <d2> n',
    delete 2, make data;

: Rule 1.2.9
:
: If user selects DATA,
```

```
:   displays RTD knowledge base data...
: r18
rule 1000 2 (menu ^action 2)
            (data ^entry 9  ^rtd <id>)
            (rtd ^id <id>)
            (hist ^rtd <id>)
            (trend ^rtd <id>)
    -->
    write '\n',
    write '\n***procedure Display_Knowledge_Base***\n',
    write '\n       RTD <id> knowledge base data...\n',
    prmem 3, prmem 4, prmem 5,
    delete 2, make data;

: Rule 1.2.10
:
: If user selects data/EXIT,
:   set blocking and return Rule 0, main menu...
: r19
rule 1000 2 (menu ^action 2)
            (data ^entry 10)
    -->
    fire block 2 off,
    fire block 1 on,
    write '\n',
    write '\nExiting Data Menu...\n',
    delete 1, make menu,
    delete 2, make data;

: Rule 1.2.11
:
: If user selects data/DIAGNOSIS,
:   display work information...
: r20
rule 1000 2 (menu ^action 2)
            (data ^entry 11)
    -->
    write '\n',
    write ' n***Procedure incomplete. Refer Rule 1.2.11*** n',
    delete 2, make data;

: Rule 1.2.12
:
: If user selects data/UPDATE,
:   record work information and delete work items...
: r21
rule 1000 2 (menu ^action 2)
            (data ^entry 12)
            (work ^rtd <id>)
    -->
    write '\n',
    write ' n***Procedure incomplete. Refer Rule 1.2.12*** n',
    delete 2, make data;
```

```
:-------------------------------------------------------------
: Block 1, con'd.
:
: Rule 1.3.0.
:   If user selects main/EXPLAIN,
:   sets blocking to trace RTD diagnosis...
: r22
rule 1000 1 (menu ^action 3)
    -->
    fire block 1 off,
    fire block 3 on;


:-------------------------------------------------------------
: Block 3. Explains RTD diagnosis.
:
: Rule 1.3.1.
:   If user selects main/EXPLAIN...
:   displays trace of current diagnoses...
:
rule trace 1000 3 (menu ^action 3)
                  (explan ^rule.cf 0)
                  (work ^diag.cf > 0  ^trace.cf 0)
    -->
    prmem 3,
    modify 3 ^trace 1;

:
: Rule 1.3.2
:
: If user selects main/EXPLAIN...
:   prompts for rule id...
: r23
rule 800 3 (menu ^action 3)
           (explan ^rule.cf 0)
    -->
    reset,
    write '\n***procedure Explain***\n',
    write ' n Above is a list of the diagnoses along with ',
    write ' n the rules used to make these diagnoses... n',
    write ' n     which RULE would you like to see? n',
    read 2 ^rule;

:
: Rule 1.4
:
: If user selects main/EXIT or makes and entry error,
:  re-initializes menu and verifies user choice...
:
rule exit 1000 1 (menu ^action > 3)
    -->
    delete 1, make menu,
    write '\n',
```

```
    write '\nType RUN to continue, STOP to exit...\n',
    halt;

write 'ok';
write '\n                          block diagnose...';


:-------------------------------------------------------------
: Block 5. Evaluates RTD response data.
:
: Rule 2.
:   Depending on delta values, classifies response as GOOD...
: r24
rule 1000 5 (work ^rtd <id>)
            (mean ^dt2 <m1>  ^dtmax <m2>)
            (trend ^rtd <id> ^resp.good 0  ^dt2 <= <m1>
                   ^dtmax <= <m2>)
    -->
    modify 3 ^resp.good 900,
    write '\n   Response RTD <id> rated GOOD...\n';

: Rule 3
:
:   Classifies response as POOR...
: r25
rule 1000 5 (work ^rtd <id>)
            (mean ^dtmax <m2>  ^delta <d>)
            (trend ^rtd <id> ^resp.poor 0  ^dtmax > <m2>
                   ^dtmax <= <d>)
    -->
    modify 3 ^resp.poor 900,
    write '\n   Response RTD <id> rated POOR...\n';

: Rule 4
:
:   Classifies response as BAD...
: r26
rule 1000 5 (work ^rtd <id>)
            (mean ^dtmax <m2> ^delta <d>)
            (trend ^rtd <id> ^resp.bad 0  ^dtmax > <d>)
    -->
    modify 3 ^resp.bad 900,
    write '\n   Response RTD <id> rated BAD...\n';


:-------------------------------------------------------------
: Block 6. Evaluates RTD bias data.
:
: Rule 5.
:   Depending on database bias value, classifies bias as HIGH...
: r27
rule 1000 6 (work ^rtd <id>)
            (mean ^nbias <nb>)
            (hist ^rtd <id> ^bias.high 0  ^abias <= <nb>)
```

```
    -->
    modify 3 ^bias.high 900,
    write '\n   Bias RTD <id> rated HIGH...\n';

: Rule 6
:
:   Classifies device bias as LOW...
: r28
rule 1000 6 (work ^rtd <id>)
            (mean ^bias <b>)
            (hist ^rtd <id> ^bias.low 0  ^abias >= <b>)
    -->
    modify 3 ^bias.low 900,
    write '\n   Bias RTD <id> rated LOW...\n';

: Rule 7
:
:   Classifies device bias as BETWEEN...
: r29
rule 1000 6 (work ^rtd <id>)
            (mean ^bias <b>)
            (hist ^rtd <id> ^bias.between 0  ^abias > 1.0
                  ^abias < <b>)
    -->
    modify 3 ^bias.between 900,
    write '\n   Bias RTD <id> rated BETWEEN high and low ...\n';

: Rule 8
:
:   Classifies device bias as NONE...
: r30
rule 1000 6 (work ^rtd <id>)
            (mean ^nbias <nb>)
            (hist ^rtd <id> ^bias.none 0  ^abias > <nb>
                  ^abias <= 1.0)
    -->
    modify 3 ^bias.none 900,
    write '\n   Bias RTD <id> rated OK ...\n';


:-------------------------------------------------------------
: Block 10. Diagnoses bias is NONE.
:
: Rule 9.
:  Diagnoses RTD ok...
: r31
rule 1000 10 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.good)
             (hist ^rtd <id>  ^bias.none)
    -->
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> OK with conf ', pconf,
    write '\n',
    make work ^rtd "<id>"  ^rule 9  ^diag "none",
```

```
    delete 1;

: Rule 10
:
:  Diagnoses RTD spare and connector...
: r32
rule 1000 10 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.poor  ^deter "yes")
             (hist ^rtd <id>  ^bias.none)
             (rtd  ^id <id>  ^clamp "yes"  ^spare "yes")
    -->
    make work ^rtd "<id>"  ^rule 10  ^diag "Spare & connector",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n  ***Use spare',
    write '\n  ***& Rework connector\n';

: Rule 11
:
:  Diagnoses RTD clamp and connector...
: r33
rule 1000 10 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.poor  ^deter "yes")
             (hist ^rtd <id>  ^bias.none)
             (rtd ^id <id>  ^clamp "yes"  ^spare "no")
    -->
    make work ^rtd "<id>" ^rule 11 ^diag "Rework clamp & conn",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework clamp and connector.';

: Rule 12
:
:  Diagnoses RTD clamp...
: r34
rule 1000 10 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.poor  ^deter "yes")
             (hist ^rtd <id>  ^bias.none)
             (rtd ^id <id>  ^clamp "no"  ^spare "yes")
    -->
    make work ^rtd "<id>"  ^rule 12  ^diag "Rework clamp",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework clamp.';

: Rule 13
:
:  Diagnoses RTD spare...
: r35
rule 1000 10 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.poor  ^deter "yes")
```

```
                (hist ^rtd <id>  ^bias.none)
                (rtd ^id <id>  ^clamp "no"  ^spare "no")
    -->
    make work ^rtd "<id>"  ^rule 13  ^diag "Use spare",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n     RTD <id> needs work with conf ', pconf,
    write '\n  ***Use spare.';

: Rule 14
:
:  Diagnoses RTD connector...
:
rule 1000 10 (work ^rtd <id> ^diag.cf 0)
                (trend ^rtd <id>  ^resp.poor  ^deter "no")
                (hist ^rtd <id>  ^bias.none)
                (rtd ^id <id>)
    -->
    make work ^rtd "<id>"  ^rule 14  ^diag "Rework connector",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n     RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework cannon connector.';

: Rule 15
:
: Diagnoses RTD clamp...
:
rule 1000 10 (work ^rtd <id> ^diag.cf 0)
                (trend ^rtd <id>  ^resp.bad)
                (hist ^rtd <id>  ^bias.none)
                (rtd ^id <id>  ^spare "yes")
    -->
    make work ^rtd "<id>"  ^rule 15  ^diag "Rework clamp",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n     RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework clamp.';

: Rule 16
:
:  Diagnoses RTD spare ...
:
rule 1000 10 (work ^rtd <id> ^diag.cf 0)
                (trend ^rtd <id>  ^resp.bad)
                (hist ^rtd <id>  ^bias.none)
                (rtd ^id <id>  ^spare "no")
    -->
    make work ^rtd "<id>"  ^rule 16  ^diag "Use spare",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n     RTD <id> needs work with conf ', pconf,
    write '\n  ***Use spare guide tube.';
```

```
:-------------------------------------------------------
: Block 11. Diagnoses bias is BETWEEN.
:
: Rule 17.
:  Diagnoses cannon connector ...
:
rule 1000 11 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>)
             (hist ^rtd <id>  ^bias.between  ^drift > 0)
             (rtd ^id <id>)
   -->
   make work ^rtd "<id>" ^rule 17 ^diag "Replace conn & cable",
   delete 1,
   write '\n***DIAGNOSIS:\n',
   write '\n     RTD <id> needs work with conf ', pconf,
   write '\n  ***Replace canon connector and cable to first JB.';

: Rule 18
:
:  Diagnoses RTD ok ...
:
rule 1000 11 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>)
             (hist ^rtd <id>  ^bias.between  ^drift > 0)
             (rtd ^id <id>)
    -->
    make work ^rtd "<id>"  ^rule 18  ^diag "RTD ok and monitor",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n     RTD <id> OK with conf ', pconf,
    write ' but...',
    write '\n  ***Monitor bias drift.';


:-------------------------------------------------------
: Block 12. Diagnoses bias is LOW.
:
: Rule 19.
:  Diagnoses electrical connection ...
:
rule 1000 12 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.good)
             (hist ^rtd <id>  ^bias.low)
             (rtd ^id <id>)
   -->
   make work ^rtd "<id>" ^rule 19 ^diag "Suspect elect conn",
   delete 1,
   write '\n***DIAGNOSIS:\n',
   write '\n     RTD <id> needs work with conf ', pconf,
   write '\n  ***Suspect electrical connection.';

: Rule 20
:
```

```
:  Diagnoses clamp and connector ...
:
rule 1000 12 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.poor  ^deter "yes")
             (hist ^rtd <id>  ^bias.low)
             (rtd ^id <id>  ^spare "yes")
    -->
    make work ^rtd "<id>" ^rule 20 ^diag "Rework clamp & conn",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n     RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework clamp and connector.';

: Rule 21
:
:  Diagnoses RTD spare and connector...
:
rule 1000 12 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.poor  ^deter "yes")
             (hist ^rtd <id>  ^bias.low)
             (rtd ^id <id>  ^spare "no")
    -->
    make work ^rtd "<id>"  ^rule 21  ^diag "Use spare & conn",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n     RTD <id> needs work with conf ', pconf,
    write '\n  ***Use spare guide tube.',
    write '\n  ***Rework canon connector.';

: Rule 22
:
:  Diagnoses unavailable ...
:
rule 1000 12 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.poor  ^deter "no")
             (hist ^rtd <id>  ^bias.low)
             (rtd ^id <id>)
    -->
    make work ^rtd "<id>"  ^rule 22  ^diag "No data",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n  ***RTD <id> work can't be identified...',
    write '\n     ***Rule 22 insufficient data***';

: Rule 23
:
:  Diagnoses RTD clamp ...
:
rule 1000 12 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.bad)
             (hist ^rtd <id>  ^bias.low)
             (rtd ^id <id>  ^spare "yes")
    -->
    make work ^rtd "<id>"  ^rule 23  ^diag "Rework clamp",
```

```
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework clamp.';

: Rule 24
:
:  Diagnoses RTD spare ...
:
rule 1000 12 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.bad)
             (hist ^rtd <id>  ^bias.low)
             (rtd ^id <id>  ^spare "no")
    -->
    make work ^rtd "<id>"  ^rule 24  ^diag "Use spare",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n  ***Use spare guide tube.';


:-------------------------------------------------------------
: Block 13. Diagnoses bias is HIGH.
:
: Rule 25.
:  Diagnoses RTD connector ...
:
rule 1000 13 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.good)
             (hist ^rtd <id>  ^bias.high)
             (rtd ^id <id>)
    -->
    make work ^rtd "<id>"  ^rule 25  ^diag "Rework connector",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework canon connector.';

: Rule 26
:
:  Diagnoses RTD clamp and connector...
:
rule 1000 13 (work ^rtd <id> ^diag.cf 0)
             (trend ^rtd <id>  ^resp.poor  ^deter "yes")
             (hist ^rtd <id>  ^bias.high)
             (rtd ^id <id>  ^spare "yes")
    -->
    make work ^rtd "<id>" ^rule 26 ^diag "Clamp and connector",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework RTD clamp and canon connector.';

: Rule 27
```

```
:
:  Diagnoses RTD spare and connector...
:
rule 1000 13 (work ^rtd <id> ^diag.cf 0)
              (trend ^rtd <id>  ^resp.poor  ^deter "yes")
              (hist ^rtd <id>  ^bias.high)
              (rtd ^id <id>  ^spare "no")
    -->
    make work ^rtd "<id>" ^rule 27 ^diag "Spare and Connector",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework RTD spare and canon connector.';

: Rule 28
:
:  Diagnoses unavailable...
:
rule 1000 13 (work ^rtd <id> ^diag.cf 0)
              (trend ^rtd <id>  ^resp.poor  ^deter "no")
              (hist ^rtd <id>  ^bias.high)
    -->
    make work ^rtd "<id>"  ^rule 28  ^diag "No Data",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n  ***RTD <id> work cannot be decided...',
    write '\n     ***Rule 28 insufficient data***\n';

: Rule 29
:
:  Diagnoses RTD clamp and connector...
:
rule 1000 13 (work ^rtd <id> ^diag.cf 0)
              (trend ^rtd <id>  ^resp.bad)
              (hist ^rtd <id>  ^bias.high)
              (rtd ^id <id>  ^spare "yes")
    -->
    make work ^rtd "<id>" ^rule 29 ^diag "Clamp and Connector",
    delete 1,
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n  ***Rework RTD clamp and canon connector.';

: Rule 30
:
:  Diagnoses RTD spare and connector...
:
rule 1000 13 (work ^rtd <id> ^diag.cf 0)
              (trend ^rtd <id>  ^resp.bad)
              (hist ^rtd <id>  ^bias.high)
              (rtd ^id <id> ^spare "no")
    -->
    make work ^rtd "<id>" ^rule 30 ^diag "Spare and Connector",
    delete 1,
```

```
    write '\n***DIAGNOSIS:\n',
    write '\n      RTD <id> needs work with conf ', pconf,
    write '\n   ***Use RTD spare. ',
    write '\n      Rework canon connector.';


:-------------------------------------------------------
: Block 0. Re-initializes program to start.
:
: Rule Last.
:    Following MAIN/DIAGNOSIS,
:    resets blocking and returns to Rule 0, main menu...
:
rule last 600 (menu ^action 1)
              (work ^rtd <id> ^diag.cf > 0)
    -->
    fire all on,
    write '\n',
    write '\n***Diagnosis complete***\n',
    delete 1, make menu,
    make work;

write 'ok';
write '\n                    block explain...';


:-------------------------------------------------------
: Block 3, con'd. Explains diagnoses.
:
: Rule 3.9.
:  For RTD, if ^rule = 9 in (work ^rtd = RTD),
:  provides English translation of rule 9, block 10...
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 9)
    -->
    write '\n***Rule 9',
    write '\n',
    write '\n    IF  1) bias is ok',
    write '\n     &  2) response is good',
    write '\n',
    write '\n    THEN',
    write '\n       RTD is ok',
    write '\n',
    modify 2 ^rule 40;

:
: Rule 3.10
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 10)
    -->
    write '\n',
    write '\n***Rule 10',
```

```
    write '\n',
    write '\n    IF  1) bias is ok',
    write '\n     &  2) response is poor',
    write '\n     &  3) response has deteriorated',
    write '\n     &  4) clamp has been reworked',
    write '\n     &  5) RTD has been placed in spare guide tube',
    write '\n',
    write '\n    THEN',
    write '\n       Use spare guide tube',
    write '\n     & Rework canon connector',
    write '\n',
    modify 2 ^rule 40;

:
: Rule 3.11
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 11)
    -->
    write '\n',
    write '\n***Rule 11',
    write '\n',
    write '\n    IF  1) bias is ok',
    write '\n     &  2) response is poor',
    write '\n     &  3) response has deteriorated',
    write '\n     &  4) clamp has been reworked',
    write '\n     &  5) spare guide tube has NOT been used\n',
    write '\n    THEN',
    write '\n       Rework clamp',
    write '\n     & Rework canon connector',
    modify 2 ^rule 40;

:
: Rule 3.12
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 12)
    -->
    write '\n',
    write '\n***Rule 12',
    write '\n',
    write '\n    IF  1) bias is ok',
    write '\n     &  2) response is poor ',
    write '\n     &  3) response has deteriorated',
    write '\n     &  4) RTD is in spare guide tube',
    write '\n     &  5) clamp has NOT been reworked',
    write '\n',
    write '\n    THEN',
    write '\n       Rework clamp',
    modify 2 ^rule 40;

:
: Rule 3.13
:
```

```
rule 1000 3 (menu ^action 3)
            (explan ^rule 13)
    -->
    write '\n',
    write '\n***Rule 13',
    write '\n',
    write '\n    IF  1) bias is ok',
    write '\n     &  2) response is poor',
    write '\n     &  3) response has deteriorated',
    write '\n     &  4) RTD is NOT in spare guide tube',
    write '\n     &  5) clamp has NOT been reworked',
    write '\n',
    write '\n    THEN',
    write '\n       Place RTD in spare',
    modify 2 ^rule 40;

:
: Rule 3.14
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 14)
    -->
    write '\n',
    write '\n***Rule 14',
    write '\n',
    write '\n    IF  1) bias is ok',
    write '\n     &  2) response is poor',
    write '\n     &  3) response has NOT deteriorated',
    write '\n ',
    write '\n    THEN',
    write '\n       Rework canon connector',
    write '\n',
    modify 2 ^rule 40;

:
: Rule 3.15
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 15)
    -->
    write '\n',
    write '\n***Rule 15',
    write '\n',
    write '\n    IF  1) bias is ok',
    write '\n     &  2) response is bad',
    write '\n     &  3) RTD is in spare guide tube',
    write '\n',
    write '\n    THEN',
    write '\n       Rework clamp',
    write '\n',
    write '\n',
    modify 2 ^rule 40;

:
```

```
: Rule 3.16
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 16)
    -->
    write '\n',
    write '\n***Rule 16',
    write '\n',
    write '\n    IF  1) Bias is Ok',
    write '\n     &  2) Response is Bad',
    write '\n     &  3) RTD is NOT in Spare Guide Tube',
    write '\n',
    write '\n    THEN',
    write '\n       Place RTD in Spare ',
    write '\n',
    write '\n',
    modify 2 ^rule 40;

:
: Rule 3.17
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 17)
    -->
    write '\n',
    write '\n***Rule 17',
    write '\n',
    write '\n    IF',
    write '\n       1) Bias is between High and Low',
    write '\n     & 2) Bias has drifted ',
    write '\n',
    write '\n    THEN',
    write '\n       Rework Canon Connector',
    write '\n     & Rework Cable to first JB',
    write '\n',
    modify 2 ^rule 40;

:
: Rule 3.18
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 18)
    -->
    write '\n',
    write '\n***Rule 18',
    write '\n',
    write '\n    IF',
    write '\n       1) Bias is between High and Low',
    write '\n     &  2) Bias shows NO drift',
    write '\n',
    write '\n    THEN',
    write '\n       RTD is Ok',
    write '\n     & Monitor RTD Bias',
    modify 2 ^rule 40;
```

```
:
: Rule 3.19
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 19)
    -->
    write '\n',
    write '\n***Rule 19',
    write '\n',
    write '\n    IF',
    write '\n       1) Bias is Low',
    write '\n     & 2) Response is Good',
    write '\n',
    write '\n    THEN',
    write '\n       Suspect electrical connector',
    write '\n',
    modify 2 ^rule 40;

:
: Rule 3.20
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 20)
    -->
    write '\n',
    write '\n***Rule 20',
    write '\n',
    write '\n    IF',
    write '\n       1) Bias is Low ',
    write '\n     & 2) Response is Poor',
    write '\n     & 3) Response has deteriorated',
    write '\n     & 4) RTD is in Spare Guide Tube',
    write '\n',
    write '\n     THEN',
    write '\n        Rework Clamp ',
    write '\n      & Rework Canon Connector\n',
    modify 2 ^rule 40;

:
: Rule 3.21
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 21)
    -->
    write '\n',
    write '\n***Rule 21',
    write '\n',
    write '\n    IF',
    write '\n       1) Bias is Low',
    write '\n     & 2) Response is Poor',
    write '\n     & 3) Response has deteriorated',
    write '\n     & 4) RTD is NOT in Spare Guide Tube',
    write '\n',
```

```
    write '\n   THEN',
    write '\n      Place RTD in Spare Guide Tube',
    write '\n    & Rework Canon Connector\n',
    modify 2 ^rule 40;

:
: Rule 3.22
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 22)
    -->
    write '\n',
    write '\n***Rule 22',
    write '\n',
    write '\n   IF',
    write '\n      1) Bias is Low',
    write '\n    & 2) Response is Poor',
    write '\n    & 3) Response shows NO deterioration',
    write '\n',
    write '\n   THEN',
    write '\n      Insufficient data',
    write '\n    & No decision is available\n',
    modify 2 ^rule 40;

:
: Rule 3.23
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 23)
    -->
    write '\n',
    write '\n***Rule 23',
    write '\n',
    write '\n   IF',
    write '\n      1) Bias is Low',
    write '\n    & 2) Response is Bad',
    write '\n    & 3) RTD is in Spare Guide Tube',
    write '\n',
    write '\n   THEN',
    write '\n      Rework Clamp',
    write '\n',
    modify 2 ^rule 40;

:
: Rule 3.24
:
rule 1000 3 (menu ^action 24)
            (explan ^rule 9)
    -->
    write '\n',
    write '\n***Rule 24',
    write '\n',
    write '\n   IF',
    write '\n      1) Bias is Low',
```

```
    write '\n    & 2) Response is Poor',
    write '\n    & 3) RTD is NOT in Spare Guide Tube',
    write '\n',
    write '\n   THEN',
    write '\n      Place RTD in Spare Guide Tube',
    write '\n',
    modify 2 ^rule 40;

:
: Rule 3.25
:
rule 1000 3 (menu ^action 25)
            (explan ^rule 9)
    -->
    write '\n',
    write '\n***Rule 25',
    write '\n',
    write '\n   IF',
    write '\n      1) Bias is High',
    write '\n    & 2) Response is Good ',
    write '\n',
    write '\n   THEN',
    write '\n      Rework Canon Connector',
    write '\n',
    modify 2 ^rule 40;

:
: Rule 3.26
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 26)
    -->
    write '\n',
    write '\n***Rule 26',
    write '\n',
    write '\n   IF',
    write '\n      1) Bias is High',
    write '\n    & 2) Response is Poor ',
    write '\n    & 3) Response has deteriorated',
    write '\n    & 4) RTD is in Spare Guide Tube',
    write '\n',
    write '\n   THEN',
    write '\n      Rework Clamp',
    write '\n    & Rework Canon Connector',
    modify 2 ^rule 40;

:
: Rule 3.27
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 27)
    -->
    write '\n',
    write '\n***Rule 27',
```

```
    write '\n',
    write '\n   IF',
    write '\n      1) Bias is High ',
    write '\n    & 2) Response is Poor ',
    write '\n    & 3) Response has deteriorated ',
    write '\n    & 4) RTD is NOT in Spare Guide Tube',
    write '\n',
    write '\n   THEN',
    write '\n      Place RTD is Spare Guide Tube',
    write '\n    & Rework Canon Connector',
    modify 2 ^rule 40;

:
: Rule 3.28
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 28)
    -->
    write '\n',
    write '\n***Rule 28',
    write '\n',
    write '\n   IF',
    write '\n      1) Bias is High',
    write '\n    & 2) Response is Poor',
    write '\n    & 3) Response shows NO deterioration',
    write '\n',
    write '\n   THEN',
    write '\n      Insufficient data',
    write '\n    & No decision is available',
    modify 2 ^rule 40;

:
: Rule 3.29
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 29)
    -->
    write '\n',
    write '\n***Rule 29',
    write '\n',
    write '\n   IF',
    write '\n      1) Bias is High',
    write '\n    & 2) Response is Bad',
    write '\n    & 3) RTD is in Spare Guide Tube',
    write '\n',
    write '\n   THEN',
    write '\n      Rework Clamp',
    write '\n    & Rework Canon Connector',
    modify 2 ^rule 40;

:
: Rule 3.30
:
rule 1000 3 (menu ^action 3)
```

```
            (explan ^rule 30)
    -->
    write '\nRule 30',
    write '\n',
    write '\n   IF',
    write '\n      1) Bias is High ',
    write '\n    & 2) Response is Bad ',
    write '\n    & 3) RTD is NOT in Spare Guide Tube',
    write '\n',
    write '\n   THEN',
    write '\n      Place RTD in Spare',
    write '\n    & Rework Canon Connector',
    modify 2 ^rule 40;

:
: Rule 3.40
:
: Resets work records by reinitializing trace value...
:
rule 1000 3 (menu ^action 3)
            (explan ^rule 40)
            (work ^trace 1)
   -->
    modify 3 ^trace 0;

:
: Rule 3.41
:
: If trace is complete,
:   returns main menu...
:
rule 800 3 (menu ^action 3)
           (explan ^rule 40)
           (work ^trace 0)
    -->
    reset,
    write '\n',
    write '\n   Exit main/EXPLAIN...',
    write '\n',
    delete 1, make menu,
    delete 2, make explan,
    fire all on;

write 'ok\n';
:
: Knowledge database
:
write ' n   reading database pro1.dat...';
open pro1.dat ;
write 'ok\n';
:
: Working memory
:
write '\n   making w. memory...';
```

```
:
make work;
make menu;
make data;
make explan;
make enable;
:
:
write 'ok\n';
write '\n';
write '\nPROGRAM RTD ready to run...';
write '\n     Type RUN to continue, STOP to exit...\n';
:
: eof pro1.fps
```

III.2

PRO2

```
:
:
: program pro2.fps
:
: FLOPS sequential program for single RTD diagnosis.
:
: ...reads RTD performance and history data in file pro2.dat.
: Generates initial hypotheses based on file data, then tests
: each hypothesis with queries to the user.
:
:
write '\nPROGRAM RTD DIAGNOSIS';
write '\n  ...is menu-driven and includes the following options:';
write '\n';
write '\n    *** 1) RTD Diagnosis          ***';
write '\n    *** 2) Display/enter RTD Data ***';
write '\n    *** 3) Explain RTD Diagnosis  ***';
write '\n';
write '\n';
write ' nRTD DIAGNOSIS Main Menu to follow...';
write '\n';
write '\n   compiling data types...';


:declarations
:
literalize rtd
  id       atm
  clamp    atm
  spare    atm
  connector  atm
  work     atm
;
```

```
literalize trend
  rtd      atm
  resp     atm
  deter    atm
;

literalize hist
  rtd      atm
  bias     atm
  drift    atm
;

literalize work
  rtd    atm
  rule   int
  diag   atm
  trace  int
;

literalize explan
  rule int
  mode atm
;

literalize menu
  action  int
  proc  atm
;


write 'ok n';
write ' n   compiling pro2.fps rules...';
:
: Rules
:
: The following rules define three procedures...
: RTD diagnosis, explanation and knowledge base operations.
: Rule r0 is the driver which presents a menu of options.
: Subsequent rules are fired depending on the user's choice...
:
write 'block prelim...';


:-------------------------------------------------------------
: Block 1. Provides user menu.
:
: Rule 0.
:   Displays MAIN MENU and requests user selection...
:
rule menu 700 1 (menu ^action.cf 0)
    -->
    reset,
    write '\n',
```

```
    write '\n*****************************************',
    write '\n**  RTD Diagnosis Expert...Main Menu  **',
    write '\n*****************************************\n',
    write ' n This program includes the following options... n',
    write '\n       1) Diagnose RTD',
    write '\n       2) Enter RTD Data',
    write '\n       3) Display RTD Data',
    write '\n       4) Explain RTD Diagnosis',
    write '\n       5) Exit to DOS\n',
    write '\n   What NUMBER 1-5?...\n',
    read 1 ^action;

: Rule 1.1
:
: If user selects main/DIAGNOSIS...
:   sets procedure HYPOTHESIS and identifies rtd in kbase...
:
rule start 1000 1 (menu ^action 1  ^proc.cf 0)
                  (rtd ^id <id>)
    -->
    modify 1 ^proc "H",
    write '\n   Diagnosing RTD <id>...\n';

:
: Rule 1.2
:
:  If user selects main/ENTER,
:    sets procedure flag and calls editor...
:
rule 1000 1 (menu ^action 2)
    -->
    modify 1 ^proc "call",
    call edit;

:
: Rule 1.2.1
:
:  If user selects main/ENTER,
:    close data file and call editor...
:
rule 1000 1 (menu ^action 2  ^proc "call")
    -->
    delete 1, make menu;

:
: Rule 1.3.0
:
: If user selects main/DISPLAY,
:   call REFLEX data...
:
rule 1000 1 (menu ^action 3)
    -->
    write ' n Load program SCREEN.BAT in default drive a:...',
    write '\n    Type any key to continue...',
```

```
    read 1 ^proc,
    modify 1 ^proc "call";

:
:
rule 1000 1 (menu ^action 3  ^proc "call")
    -->
    delete 1, make menu,
    call screen
;


:-------------------------------------------------------------
: Block 4. Explains RTD diagnosis.
:
: Rule 1.4.
:   If user selects main/EXPLAIN,
:   prompts for rule id...
:
rule 1000 4 (menu ^action 4)
            (work ^trace 0)
    -->
    prmem 2,
    modify 2 ^trace 1;


:
: Rule 1.5
:
: If user selects main/EXIT or makes and entry error,
:  re-initializes menu and verifies user choice...
:
rule exit 1000 1 (menu ^action > 4)
    -->
    delete 1, make menu,
    write '\n',
    write '\nType RUN to continue, STOP to exit...\n',
    halt;

write 'ok';
write '\n                        block diagnose...';


:-------------------------------------------------------------
: Block 5. Diagnoses single RTD.
:
: Rule 2.
:   If main/DIAGNOSE, defines hypothesis set...
:
rule 1000 5 (menu ^action 1  ^proc.cf 0)
    -->
    modify 1 ^proc "H";

: Rule 3
```

```
:
:
:
rule 1000 5 (menu ^action 1  ^proc "H")
            (hist ^rtd <id>  ^bias "high")
    -->
    make work ^rtd "<id>" ^diag "connector",
    write '\n   Hypothesis...    Work is connector',
    make work ^rtd "<id>" ^diag "clamp",
    write '\n                    Work is clamp\n',
    modify 1 ^proc "T";

: Rule 4
:
:
:
rule 1000 5 (menu ^action 1  ^proc "T")
            (work ^diag "connector")
            (rtd  ^connector.cf 0)
            (explan)
    -->
    write '\nTesting hypothesis Work is Connector...',
    write ' nHas canon_connector been reworked?...YES or NO n',
    read 3 ^connector,
    modify 4 ^rule 4,
    modify 1 ^proc "E";

: Rule 5
:
:
:
rule 1000 5 (menu ^action 1  ^proc "E")
            (work ^diag "connector")
            (rtd ^connector "yes")
    -->
    write '\nEvaluating hypothesis Work is Connector...\n',
    modify 1 ^proc "X",
    modify 2 ^diag.cf 700;

rule 1000 5 (rtd ^id <id>  ^connector "why")
            (explan)
    -->
    delete 1,
    make rtd ^id "<id>",
    modify 2 ^mode "why";

:
:
rule reason 1000 5 (explan ^rule <R>  ^mode "why")
    -->
    write '\nBecause I\'m testing rule <R>...\n';


:-------------------------------------------------------
```

```
: Block 0. Re-initializes program to start.
:
: Rule Last.
:    Following main/DIAGNOSIS,
:     resets blocking and returns to Rule 0, main menu...
:
rule last 800 (menu ^action 1  ^proc "X")
              (work)
    -->
    reset,
    delete 1, make menu,
    write '\nDiagnosis:\n',
    prmem 2;

write 'ok';
write '\n                          block explain...';

:-------------------------------------------------------------
: Block 3, con'd. Explains RTD diagnosis.
:
: Rule 3.9.
:  Depending on user's response,
:  explains program request for data...
:
rule 1000 3 (explan ^rule 4  ^mode "why")
    -->
    write '\n***Rule 4',
    write '\n',
    write '\n    IF  1) bias is high',
    write '\n     &  2) connector is reworked',
    write '\n',
    write '\n     THEN',
    write '\n        work is connector with conf 700',
    write '\n',
    delete 1, make explan;


write 'ok\n';
:
: Knowledge database
:
write ' n   reading data base pro2.dat...';
open pro2.dat;
write 'ok\n';
:
: Working memory
:
write '\n   making w. memory...';
:
make menu;
make explan;
:
:
```

```
write 'ok\n';
write '\n';
write '\nPROGRAM RTD ready to run...';
write '\n     Type RUN to continue, STOP to exit...\n';
:
: eof pro2.fps
```

III.3

PRO3

```
:
: pro3.par
:
:   PFLOPS parallel program for RTD diagnosis
:
:   ...reads RTD history data in file hist.dat and user
:   information in trend.dat. Generates RTD work plan for
:   for annual outage.
:
:   Results are output to files rtd.out and diag.dat.
:
write '\ncompiling pro3.par...\n' ;


: declarations
:
literalize trend        : for each suspect RTD
  id      atm           :    id is RTD channel number
  resp    atm           :    resp is POOR
  bias    atm           :    bias is HIGH, LOW
  drift   atm           :    bias is drifted UP, DOWN
  deter   atm           :    response is deteriorated YES, NO
  elect   atm ;         :    electrical problem is YES, NO

literalize hist         :
  id   atm              :    id is RTD channel number
  pos   atm             :    position is TOP, BOTTOM
  bias   atm            :    bias is HIGH, LOW
  conn   atm            :    connector is reworked YES, NO
  resp   atm            :    response is POOR
  spare   atm           :    spare is used YES, NO
  repl   atm            :    RTD is replaced YES, NO
  susp   int            :    RTD is suspected 0 times
  clamp   int ;         :    clamp is reworked 0 times

literalize diag         :
  id   atm              :    id is RTD channel number
  dx fzset (elect       :    fault is electrical
            conn        :      ... is replace connector
            spare       :          ...use spare guide tube
            clamp       :          ...rework clamp
            not_clamp   :          ...clamp is ok
```

```
            repl       :            ...replace RTD
            monitor    :            ...monitor RTD
            bias       :            ...bias error
            resp       :            ...slow response
            best       :            ...response is best possible
            needs_work) : RTD needs work
  data atm ;           :    program data is INCONSISTENT

literalize explain     :
  id     atm           :    id is RTD channel number
  elect  int           :    fault is electrical because rule 0
  conn   int           :       ...is replace connector because...
  spare  int           :           ...use spare guide tube...
  clamp  int           :           ...rework clamp...
  not_clamp int        :           ...clamp is ok...
  repl       int       :           ...replace RTD...
  monitor    int       :           ...monitor RTD...
  best   int ;         :           ...response is best possible...

literalize enable      :    program parameter ENABLE enables
  block int ;          :      /disables rule block 0


: Rules
:
:  ...are identified as rule BLOCKS.
:
:  PFLOPS simulates parallel execution by firing one block
:  at a time and, for each "fireable" rule, it applies the
:  rule to every fact in the data base.
:
:  In PRO3, the firing of blocks is ordered by a single rule
:  which enables or disables rule blocks in sequence.
:
:  PRO3 rules are blocked by their function and, generally,
:  perform the following actions...
:
:     1. hypothesize fault from RTD data,
:     2. modify hypotheses based on testing of other data, and
:     3. assess certainty of each hypothesis.
:
:  Rule blocking facilitates a generate-and-test procedure for
:  RTD diagnosis. It also modularizes the diagnostic process and
:  faciliates the addition and deletion of individual rules.
:


:-------------------------------------
: Block 16
:
:  Enables and disables rule blocks in sequence.
:

:rule r0 -- Controls firing of rule blocks.
```

```
:
rule 1000 16 (enable ^block <n>  ^block < 16)
    -->
    write '\n',
    write '\n ********** firing block b<n>+1 **********\n',
    modify 1 ^block (<n> + 1),
    fire block <n> del,
    fire block (<n> + 1) on;


:-------------------------------------
: Block 0.
:
:  Initializes w. memory.
:
:  For each suspect rtd, creates "diag" element
:  in PFLOPS working memory.
:

:rule r1 -- For each trend, make (diag ())...
rule 1000 (trend ^id <id>)
    -->
    make diag ^id "<id>";


:-----------------------------------------
: Block 1.
:
:  Checks RTD file data and user information.
:
:  Verifies user information is consistent and
:  RTD history data is complete.
:

:rule r2 -- if response is poor &
:                bias is high
:           then
:                data(trend) is inconsistent
:
rule 1000 1 (trend ^id <id> ^resp "poor"  ^bias "high")
            (diag ^id <id>)
    -->
    modify 2 ^data "trend is inconsistent",
    write '\n',
    write ' n<id>  Data entry is inconsistent.',
    write ' n      Bias is HIGH, response is POOR.' ;


:-----------------------------------------
: Block 2.
:
:   Updates RTD bias, response history data.
:
:   Records bias drift, response deterioration based on
```

```
:   RTD history and user information.

:rule r3 -- if bias is low &
:              history(bias) is lower
:           then
:              bias is drifted(down)
:
rule 1000 2 (trend ^id <id> ^bias "low" ^bias.cf <cf>)
            (hist ^id <id> ^bias "low" ^bias.cf < <cf>)
    -->
    modify 1 ^drift "down",
    write '\n',
    write '\n<id>  Bias is drifted.';


:rule r4 -- if bias is high &
:              history(bias) is lower
:           then
:              bias is drifted(up)
:
rule 1000 2 (trend ^id <id> ^bias "high" ^bias.cf <cf>)
            (hist ^id <id> ^bias "high" ^bias.cf < <cf>)
    -->
    modify 1 ^drift "up",
    write '\n',
    write '\n<id>  Bias is drifted.';


:rule r5 -- if response is poor &
:              history(response) is better
:           then
:              response is deteriorated
:
rule 1000 2 (trend ^id <id> ^resp "poor" ^resp.cf <cf>)
            (hist ^id <id> ^resp "poor" ^resp.cf > <cf>)
    -->
    modify 1 ^deter "yes",
    write '\n',
    write '\n<id>  Response is deteriorated.';


:-------------------------------------------------
: Block 3 -- Diagnoses clamp.
:


:rule r6 -- if response is poor &
:              not history(clamp is reworked)
:           then
:             diagnosis(clamp)
:
:  e.g., h06, n07, o06
:
```

```
rule 1000 3 (trend ^id <id> ^resp "poor" ^resp.cf > 700)
            (hist ^id <id> ^clamp 0)
            (diag ^id <id>)
    -->
    modify 3 ^dx.clamp,
    write '\n',
    write '\n<id>  Rework RTD clamp.';



:-------------------------------------------------
: Block 4 -- Diagnoses cannon connector.
:


:rule r7 -- if bias is high
:           then
:              diagnosis(connector)
:
:  e.g., e03, f08, g17, g19, h10, j21,
:        k14, m04, m13, m14, 019, p19
:
rule 1000 4 (trend ^id <id> ^bias "high" ^bias.cf > 700)
            (diag ^id <id>)
    -->
    modify 2  ^dx.conn  ^dx.not_clamp,
    write '\n',
    write ' n<id>  Replace RTD cannon connector ',
    write ' n              and cable to first J.B.';

:rule r8 -- if bias is low &
:              not diagnosis(clamp),
:           then
:              diagnosis(connector)
:
:  e.g., s10
:
rule 1000 4 (trend ^id <id> ^bias "low" ^bias.cf > 700)
            (diag ^id <id> ^dx.clamp < 500)
    -->
    modify 2  ^dx.conn ,
    write '\n',
    write ' n<id>  Replace RTD cannon connector ',
    write ' n              and cable to first J.B.';

:rule r9  -- if bias is low &
:               history(clamp)
:            then
:               diagnosis(connector)
:
:  e.g., s10?
:
rule 1000 4 (trend ^id <id> ^bias "low")
            (hist ^id <id> ^clamp > 0)
```

```
            (diag ^id <id>)
    -->
    modify 3  ^dx.conn ,
    write '\n',
    write ' n<id>  Replace RTD cannon connector',
    write ' n            and cable to first J.B.';

:rule r10 -- if rtd is suspect &
:                bias is drifted
:             then
:                diagnosis(connector)
:
:  e.g., j15, l16, o09, p20, r18, w12
:
rule 1000 4 (trend ^id <id> ^drift <> "")
            (diag ^id <id>)
    -->
    modify 2  ^dx.conn ,
    write '\n',
    write ' n<id>  Replace RTD cannon connector',
    write ' n            and cable to first J.B.';


:-----------------------------------------------
: Block 5 -- Diagnoses spare.
:

:rule r11 -- if response is poor &
:                history(clamp) &
:                not history(spare)
:             then
:                diagnosis(spare)
:
:  e.g., k20, l13, m14
:
rule 1000 5 (trend ^id <id> ^resp "poor")
            (hist ^id <id> ^clamp > 0 ^spare "")
            (diag ^id <id>)
    -->
    modify 3 ^dx.spare,
    write '\n',
    write '\n<id>  Place RTD in spare guide tube.';


:rule r12 -- if bias is high &
:                history(connector) &
:                not history(spare)
:             then
:                diagnosis(spare)
:
:  e.g., k6, k18, m8, n19, p4, q15
:
rule 1000 5 (trend ^id <id> ^bias "high")
            (hist ^id <id> ^conn <> "" ^spare "")
```

```
            (diag ^id <id>)
    -->
    modify 3 ^dx.spare,
    write '\n',
    write '\n<id>  Place RTD in spare guide tube.';

:rule r13 -- if bias is low &
:                history(connector) &
:                history(clamp)
:             then
:                diagnosis(spare)
:
:  e.g., l09
:
rule 1000 5 (trend ^id <id> ^bias "low")
            (hist ^id <id> ^conn <> "" ^clamp > 0)
            (diag ^id <id>)
    -->
    modify 3 ^dx.spare,
    write '\n',
    write '\n<id>  Place RTD in spare guide tube.';


:-----------------------------------------------
: Block 6 -- Diagnoses bias error is ok.
:

:rule r14 -- if bias is high or
:                bias is low (<= 0.7)
:              then diagnosis(bias_error) &
:                   diagnosis(monitor)
:
rule 1000 6 (trend ^id <id> ^bias <> "" ^bias.cf <= 700)
            (diag ^id <id>)
 -->
 modify 2 ^dx.bias ^dx.monitor,
 write ' n',
 write ' n<id>  Bias error. To be tracked over next 12 months.';


:-----------------------------------------------
: Block 7 -- Diagnoses best response.
:

:rule r15 -- if response is ok &
:                history(clamp) &
:                history(spare)
:             then diagnosis(best response) &
:                  diagnosis(monitor)
:
:  e.g., h06, j10, l7, n07, o06, p11?, p15, q12, q14, r13, s10, t10
:
rule 1000 7 (trend ^id <id> ^resp "poor")
            (hist ^id <id> ^spare <> "" ^clamp > 0)
```

```
            (diag ^id <id>)
 -->
 modify 3 ^dx.best ^dx.monitor,
 write ' n',
 write ' n<id> Channel has history of poor response to fuelling.',
 write ' nRTD already located in spare guide tube. To be watched.'


:-----------------------------------------------
: Block 8 -- Diagnoses response is acceptable.
:

:rule r16 -- if response is poor (<= 700) &
:                not diagnosis(best)
:              then diagnosis(response_is_slow) &
:                   diagnosis(monitor)
:
rule 1000 8 (trend ^id <id> ^resp "poor" ^resp.cf <= 700)
            (diag ^id <id> ^dx.best < 500)
 -->
 modify 2 ^dx.resp ^dx.monitor,
 write ' n',
 write ' n<id>  Fuelling trend indicates slow response.
                To be watched.' ;


:-----------------------------------------------
: Block 9. Updates diagnosis is clamp, connector.
:
:  Modifies diagnosis
:     (clamp) = (bias) * (clamp), and
:     (connector) = (bias) * (connector) ;
:  i.e., clamp or connector is suspected when
:  bias error is ok but RTD history is poor.
:

:rule r17 -- if diagnosis(bias) &
:                history(response)
:             then diagnosis(clamp)
:
:  e.g., a6, s10
:
rule 1000 9 (diag ^id <id> ^dx.bias)
            (hist ^id <id> ^resp <> "")
    -->
    modify 1 ^dx.clamp ;


:rule r18 -- if diagnosis(bias) &
:                history(bias)
:             then diagnosis(connector)
:
```

```
:  e.g., f10, t9
:
rule 1000 9 (diag ^id <id> ^dx.bias)
            (hist ^id <id> ^bias <> "")
    -->
    modify 1 ^dx.conn ;



:-------------------------------------------------
: Block 10. Updates diagnosis is clamp.
:
:  Modifies diagnosis (clamp) = (resp) * (clamp); i.e.,
:  clamp is suspect when response is acceptable but
:  history is poor.
:

:rule r19 -- if diagnosis(resp) &
:                history(response)
:             then diagnosis(clamp)
:
:  e.g., j11, u8
:
rule 1000 10 (diag ^id <id> ^dx.resp)
             (hist ^id <id> ^resp <> "")
    -->
    modify 1 ^dx.clamp ;



:-------------------------------------------------
: Block 11 -- Diagnoses RTD failure.
:

:rule r20 -- if diagnosis(best) &
:                history(suspected) = s
:             then diagnosis(RTD failure, cf = s * 250)
:
:  e.g., r13, q12, s10
:
rule 1000 11 (diag ^id <id> ^dx.best)
             (hist ^id <id> ^susp > 0 ^susp <s>)
    -->
    modify 1 ^dx.repl (<s> * 250);


:rule r21 -- if bias is high &
:                history(bias) = low
:             then RTD is irratic &
:                  diagnosis(RTD failure)
:
rule 1000 11 (trend ^id <id> ^bias "high")
```

```
                (hist ^id <id> ^bias "low")
                (diag ^id <id>)
    -->
    modify 3 ^dx.repl;

:rule r22 -- if bias is low &
:               history(bias) = high
:            then RTD is irratic &
:               diagnosis(RTD failure)
:
rule 1000 11 (trend ^id <id> ^bias "low")
                (hist ^id <id> ^bias "high")
                (diag ^id <id>)
    -->
    modify 3 ^dx.repl;



:-----------------------------------------------
: Block 12. Evaluates needs_work.
:
:  Identifies suspect RTDs. Records poor performance
:  based on repeated faults.
:

:rule r23 if RTD is suspected &
:               history(suspected) = s
:         then RTD needs work with cf = (s * 0.25)
:
:  e.g., m14
:
rule 1000 12 (trend ^id <id>)
                (hist ^id <id> ^susp > 0  ^susp <s>)
                (diag ^id <id>)
    -->
    modify 3 ^dx.needs_work (<s> * 250);



:-----------------------------------------------
: Block 13. Records electrical fault.
:
:  Rates electrical fault based on user information that
:  fault is ruled out by electrical testing.
:

:rule r24 -- if not (fault is electrical) (cf)
:               then diagnosis(electrical, cf = 1 - cf)
:
rule 1000 13 (trend ^id <id> ^elect "yes" ^elect.cf <cf>)
                (diag ^id <id>)
    -->
```

```
    modify 2 ^dx.elect (1 - <cf>);


:rule r25 -- if fault is electrical
:             then diagnosis(electrical)
:
rule 1000 13 (trend ^id <id> ^elect "yes" ^elect.cf <cf>)
             (diag ^id <id>)
    -->
    modify 2 ^dx.elect;



:-------------------------------------------------
: Block 14. Updates diagnosis is connector.
:
:  Modifies diagnosis (conn) = (elect) * (conn), i.e.,
:  connector is suspect when other electrical faults
:  are ruled out based on user information.
:

:rule r26 -- if not (fault is electrical, 1 - cf)
:            then
:               diagnosis(electrical) &
:               diagnosis(connector) with cf
:
rule 1000 14 (diag ^id <id> ^dx.elect)
    -->
    modify 1 ^dx.conn;



:-------------------------------------------------
: Block 15 -- Prints results of completed diagnosis.

:rule r27
rule 1000 15 (enable ^block 15)
    -->
    write '\n',
    write '\nDiagnosis complete...',
    write '\n  transferring results to file DIAG.DAT.\n',
    transfer diag to diag.dat;

:rule r28
rule 1000 15 (diag)
    -->
    prmem 1 ;

:-------------------------------------------------
: rtd fact base

write '\n transferring history data...\n';
transfer hist from out87.dat;
```

```
write '\n transferring trend data...\n';
transfer trend from trend.dat;

make enable ^block 0 ;
:
fire all off;
fire block 0 on;
fire block 16 on;
:
write '\nready to run...\n';
:
:
:eof pro3.par
```

Appendix IV

SAMPLE RTD DIAGNOSIS

```
>> open pro3;
compiling pro3.par...

 transferring history data...
112 RECORDS READ

 transferring trend data...
68 RECORDS READ

ready to run...
>> run;

 ********** firing block 1 **********

 ********** firing block 2 **********

d06  Bias is drifted.

j15  Bias is drifted.

l16  Bias is drifted.

o09  Bias is drifted.

r18  Bias is drifted.

w12  Bias is drifted.

 ********** firing block 3 **********

 ********** firing block 4 **********

e03  Replace RTD cannon connector and cable to first J.B.

f08  Replace RTD cannon connector and cable to first J.B.

g17  Replace RTD cannon connector and cable to first J.B.

g19  Replace RTD cannon connector and cable to first J.B.
```

```
h10  Replace RTD cannon connector and cable to first J.B.
j21  Replace RTD cannon connector and cable to first J.B.
k14  Replace RTD cannon connector and cable to first J.B.
m04  Replace RTD cannon connector and cable to first J.B.
m14  Replace RTD cannon connector and cable to first J.B.
o19  Replace RTD cannon connector and cable to first J.B.
p19  Replace RTD cannon connector and cable to first J.B.
p21  Replace RTD cannon connector and cable to first J.B.
s10  Replace RTD cannon connector and cable to first J.B.
m14  Replace RTD cannon connector and cable to first J.B.
s10  Replace RTD cannon connector and cable to first J.B.
d06  Replace RTD cannon connector and cable to first J.B.
j15  Replace RTD cannon connector and cable to first J.B.
l16  Replace RTD cannon connector and cable to first J.B.
o09  Replace RTD cannon connector and cable to first J.B.
r18  Replace RTD cannon connector and cable to first J.B.
w12  Replace RTD cannon connector and cable to first J.B.

FSETCPY -- .conn not updated
        -- current confidence > new confidence

 ********** firing block 5 **********

h15  Place RTD in spare guide tube.
k06  Place RTD in spare guide tube.
k18  Place RTD in spare guide tube.
m08  Place RTD in spare guide tube.
n19  Place RTD in spare guide tube.
p04  Place RTD in spare guide tube.
q15  Place RTD in spare guide tube.
```

k20 Place RTD in spare guide tube.

l13 Place RTD in spare guide tube.

l09 Place RTD in spare guide tube.

m14 Place RTD in spare guide tube.

********** firing block 6 **********

d04 Bias error. To be tracked over next 12 months.

d16 Bias error. To be tracked over next 12 months.

e07 Bias error. To be tracked over next 12 months.

f10 Bias error. To be tracked over next 12 months.

g12 Bias error. To be tracked over next 12 months.

h04 Bias error. To be tracked over next 12 months.

h21 Bias error. To be tracked over next 12 months.

l17 Bias error. To be tracked over next 12 months.

l19 Bias error. To be tracked over next 12 months.

m06 Bias error. To be tracked over next 12 months.

m07 Bias error. To be tracked over next 12 months.

m22 Bias error. To be tracked over next 12 months.

n01 Bias error. To be tracked over next 12 months.

n04 Bias error. To be tracked over next 12 months.

n06 Bias error. To be tracked over next 12 months.

o04 Bias error. To be tracked over next 12 months.

q06 Bias error. To be tracked over next 12 months.

s20 Bias error. To be tracked over next 12 months.

t09 Bias error. To be tracked over next 12 months.

d06 Bias error. To be tracked over next 12 months.

j15 Bias error. To be tracked over next 12 months.

l16 Bias error. To be tracked over next 12 months.

o09 Bias error. To be tracked over next 12 months.

r18 Bias error. To be tracked over next 12 months.

w12 Bias error. To be tracked over next 12 months.

h15 Bias error. To be tracked over next 12 months.

********** firing block 7 **********

h06 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

j10 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

l07 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

n07 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

o06 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

p11 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

p15 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

q12 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

q14 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

r13 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

t10 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

s10 Channel has history of poor response to fuelling.
RTD already located in spare guide tube. To be watched.

********** firing block 8 **********

f20 Fuelling trend indicates slow response. To be watched.

```
g09  Fuelling trend indicates slow response. To be watched.

j11  Fuelling trend indicates slow response. To be watched.

o13  Fuelling trend indicates slow response. To be watched.

p08  Fuelling trend indicates slow response. To be watched.

r09  Fuelling trend indicates slow response. To be watched.

u08  Fuelling trend indicates slow response. To be watched.

 ********** firing block 9 **********

FSETCPY -- .conn not updated
        -- current confidence > new confidence

 ********** firing block 10 **********

 ********** firing block 11 **********

 ********** firing block 12 **********

 ********** firing block 13 **********

 ********** firing block 14 **********

 ********** firing block 15 **********

(diag (
 ('tt',260)
 ('id','e03',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',261)
 ('id','f08',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',262)
 ('id','g17',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',263)
 ('id','g19',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',264)
```

```
 ('id','h10',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',265)
 ('id','j21',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',266)
 ('id','k14',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',267)
 ('id','m04',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',269)
 ('id','o19',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',270)
 ('id','p19',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',271)
 ('id','p21',1000)
 ('dx',(('conn',900)('not_clamp',900)))))
(diag (
 ('tt',296)
 ('id','d04',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',297)
 ('id','d16',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',298)
 ('id','e07',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',300)
 ('id','g12',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',301)
 ('id','h04',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',302)
 ('id','h21',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',303)
 ('id','l17',1000)
 ('dx',(('monitor',600)('bias',600)))))
```

```
(diag (
 ('tt',304)
 ('id','l19',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',305)
 ('id','m06',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',306)
 ('id','m07',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',307)
 ('id','m22',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',308)
 ('id','n01',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',309)
 ('id','n04',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',310)
 ('id','n06',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',311)
 ('id','o04',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',312)
 ('id','q06',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',313)
 ('id','s20',1000)
 ('dx',(('monitor',600)('bias',600)))))
(diag (
 ('tt',337)
 ('id','f20',1000)
 ('dx',(('monitor',600)('resp',600)))))
(diag (
 ('tt',338)
 ('id','g09',1000)
 ('dx',(('monitor',600)('resp',600)))))
(diag (
 ('tt',340)
 ('id','o13',1000)
 ('dx',(('monitor',600)('resp',600)))))
(diag (
 ('tt',341)
```

```
 ('id','p08',1000)
 ('dx',(('monitor',600)('resp',600)))))
(diag (
 ('tt',342)
 ('id','r09',1000)
 ('dx',(('monitor',600)('resp',600)))))
(diag (
 ('tt',367)
 ('id','k06',1000)
 ('dx',(('spare',900)('needs_work',500)))))
(diag (
 ('tt',368)
 ('id','k18',1000)
 ('dx',(('spare',900)('needs_work',500)))))
(diag (
 ('tt',369)
 ('id','m08',1000)
 ('dx',(('spare',900)('needs_work',500)))))
(diag (
 ('tt',370)
 ('id','n19',1000)
 ('dx',(('spare',900)('needs_work',250)))))
(diag (
 ('tt',371)
 ('id','p04',1000)
 ('dx',(('spare',900)('needs_work',250)))))
(diag (
 ('tt',372)
 ('id','q15',1000)
 ('dx',(('spare',900)('needs_work',750)))))
(diag (
 ('tt',373)
 ('id','k20',1000)
 ('dx',(('spare',800)('needs_work',500)))))
(diag (
 ('tt',374)
 ('id','l13',1000)
 ('dx',(('spare',800)('needs_work',500)))))
(diag (
 ('tt',375)
 ('id','l09',1000)
 ('dx',(('spare',900)('needs_work',500)))))
(diag (
 ('tt',379)
 ('id','m14',1000)
 ('dx',(('conn',900)('spare',800)('not_clamp',900)
        ('needs_work',500)))))
(diag (
 ('tt',380)
 ('id','f10',1000)
 ('dx',(('monitor',600)('bias',600)('needs_work',250)))))
(diag (
 ('tt',381)
 ('id','t09',1000)
```

```
 ('dx',(('monitor',600)('bias',600)('needs_work',250)))))
(diag (
 ('tt',382)
 ('id','h15',1000)
 ('dx',(('spare',600)('monitor',600)('bias',600)
        ('needs_work',250)))))
(diag (
 ('tt',383)
 ('id','j11',1000)
 ('dx',(('monitor',600)('resp',600)('needs_work',250)))))
(diag (
 ('tt',384)
 ('id','u08',1000)
 ('dx',(('monitor',600)('resp',600)('needs_work',250)))))
(diag (
 ('tt',385)
 ('id','d06',1000)
 ('dx',(('conn',600)('monitor',700)('bias',700)
        ('needs_work',250)))))
(diag (
 ('tt',386)
 ('id','j15',1000)
 ('dx',(('conn',600)('monitor',700)('bias',700)
        ('needs_work',250)))))
(diag (
 ('tt',387)
 ('id','l16',1000)
 ('dx',(('conn',600)('monitor',700)('bias',700)
        ('needs_work',250)))))
(diag (
 ('tt',388)
 ('id','o09',1000)
 ('dx',(('conn',600)('monitor',700)('bias',700)
        ('needs_work',250)))))
(diag (
 ('tt',389)
 ('id','r18',1000)
 ('dx',(('conn',600)('monitor',700)('bias',700)
        ('needs_work',250)))))
(diag (
 ('tt',390)
 ('id','w12',1000)
 ('dx',(('conn',700)('monitor',700)('bias',700)
        ('needs_work',250)))))
(diag (
 ('tt',391)
 ('id','h06',1000)
 ('dx',(('repl',500)('monitor',700)('best',700)
        ('needs_work',500)))))
(diag (
 ('tt',392)
 ('id','j10',1000)
 ('dx',(('repl',500)('monitor',700)('best',700)
        ('needs_work',500)))))
```

```
(diag (
 ('tt',393)
 ('id','107',1000)
 ('dx',(('repl',500)('monitor',700)('best',700)
        ('needs_work',500)))))
(diag (
 ('tt',394)
 ('id','n07',1000)
 ('dx',(('repl',500)('monitor',700)('best',700)
        ('needs_work',500)))))
(diag (
 ('tt',395)
 ('id','o06',1000)
 ('dx',(('repl',750)('monitor',700)('best',700)
        ('needs_work',750)))))
(diag (
 ('tt',396)
 ('id','p11',1000)
 ('dx',(('repl',1000)('monitor',700)('best',700)
        ('needs_work',1000)))))
(diag (
 ('tt',397)
 ('id','p15',1000)
 ('dx',(('repl',500)('monitor',700)('best',700)
        ('needs_work',500)))))
(diag (
 ('tt',398)
 ('id','q12',1000)
 ('dx',(('repl',750)('monitor',700)('best',700)
        ('needs_work',750)))))
(diag (
 ('tt',399)
 ('id','q14',1000)
 ('dx',(('repl',500)('monitor',700)('best',700)
        ('needs_work',500)))))
(diag (
 ('tt',400)
 ('id','r13',1000)
 ('dx',(('repl',1000)('monitor',700)('best',700)
        ('needs_work',1000)))))
(diag (
 ('tt',401)
 ('id','t10',1000)
 ('dx',(('repl',500)('monitor',700)('best',700)
        ('needs_work',500)))))
(diag (
 ('tt',405)
 ('id','s10',1000)
 ('dx',(('conn',800)('not_clamp',800)('repl',750)
        ('monitor',700)('best',700)('needs_work',750)))))

********** firing block 16 **********
```

```
Diagnosis complete...
  transferring results to file DIAG.DAT.
68 RECORDS WRITTEN
>> stop;
 ****** system exiting ******
```

REFERENCES

Abelson, H., G. Sussman. Structure and Interpretation of Computer Programs. New York: McGraw-Hill, 1985.

Ben-Bassat, M., R. W. Carlson, V. K. Puri, M. D. Davenport, J. A. Schriver, M. Latif, R. Smith, L. D. Portigal, E. H. Lipnick, M. H. Weil. "Pattern-based interactive diagnosis of multiple disorders: The MEDAS system". IEEE Trans. Pattern Analysis and Machine Intelligence 2(2), 1980, pp. 148-160.

Bonissone, P. P., A. L. Brown, Jr. "Expanding the horizons of expert systems", Technical Report, General Electric Corporate Research and Development, Schenectady, NY, 1987.

Boose, J. H. "Personal construct theory and the transfer of human expertise" in Proc. AAAI-84 (American Association for Artificial Intelligence, Palo Alto, CA), 1984, pp. 27-33.

Buchanan, B. G., R. O. Duda. "Principles of rule-based expert systems" in M. C. Yovits, ed., Advances in Computers 22. New York: Academic Press, 1983, pp. 163-216.

Buchanan, B. G., E. H. Shortliffe. Rule-based Expert Systems. The MYCIN Experiment. Reading, MA: Addison-Wesley, 1984.

Callister, M. P. "Channel temperature monitoring review 1987". New Brunswick Power Commission Technical Report BSI-63102, Point Lepreau, N. B., April 1987.

Civanlar, M. R., H. J. Trussell. "Constructing membership functions using statistical data". J. Fuzzy Sets and Syst. 18, 1986, pp. 1-13.

Clocksin, W. F., C. S. Mellish. Programming in Prolog. New York: Springer-Verlag, 1984.

Cohen, J. "Describing Prolog by its interpretation and compilation". Comm. ACM 28 (12), 1985, pp. 1311-1324.

Cutter Information Corp., "Expert systems strategies", Arlington, MA, 1986.

Davis, R. "Diagnosis via causal reasoning: paths of interaction and the locality principle" in Proc. Natl. Conf. on Artif. Intell. (AAAI-83), 1983, pp. 88-94.

Dubois, D., H. Prade. Fuzzy Sets and Systems: Theory and Applications. New York: Academic Press, 1980.

Dubois, D., H. Prade. "Necessity measures and the resolution principle". IEEE Trans. Syst. Man and Cyber. 17 (3), 1987, pp. 474-478.

Duda, R. O., P. E. Hart, P. Barrett, J. G. Gaschnig, K. Konolige, R. Reboh, J. Slocum. "Development of the PROSPECTOR consultation system for mineral exploration", Final Report, SRI Projects 5821 and 6415, SRI International, Menlo Park, CA, 1978.

Duda, R. O., J. G. Gaschnig. "Knowledge-based expert systems come of age" in S. J. Andriole, Applications in Artificial Intelligence, Princeton, NJ: Petrocelli Books Inc., 1985, pp. 45-86.

Friedman, E. The resource model of human intelligent behavior in cognitive psychology. (Private communication.) Department of Psychology, University of Alberta, Edmonton, Alberta, 1982.

Gaglio, S., R. Minciardi, P. P. Puliafito. "Multiperson decision aspects in the construction of expert systems". IEEE Trans. Syst. Man and Cyber. 15 (4), 1985, pp. 536-539.

Gaines, B. R. "Foundations of fuzzy reasoning". Int. J. Man-machine Stud. 8, 1976, pp. 623-668.

Gaines, B. R., M. L. G. Shaw. "Induction of inference rules for expert systems". J. Fuzzy Sets and Syst. 18, 1986, pp. 315-328.

Gordon, J., E. H. Shortliffe. "The Dempster-Shafer theory of evidence" in B. G. Buchanan, E. H. Shortliffe, Rule-based Expert Systems. The MYCIN Experiment. Reading, MA: Addison-Wesley, 1984, pp. 272-292.

Hayes-Roth, F., D. A. Waterman, D. B. Lenat. Building Expert Systems. Reading, MA: Addison-Wesley, 1983.

Inference Corporation. ART Automated Reasoning Tool. Reference Manual. Los Angeles, CA: 1987.

Hughes, R. C., J. N. Maksym. "Acoustic signal interpretation: reasoning with non-specific and uncertain information". Pattern Recognition 18 (6), 1985, pp. 475-483.

Kickert, W. J. M., H. R. van Nauta Lemke. "Application of a fuzzy controller in a warm water plant". Automatica 12, 1976, pp. 301-308.

de Kleer, J., B. C. Williams. "Diagnosing multiple faults". Artif. Intell. 32, 1987, pp. 97-130.

Klir, G. J., T. A. Folger. Fuzzy Sets, Uncertainty, and Information. Englewood Cliffs, NJ: Prentice Hall, 1988.

Kowalski, R. Logic for Problem Solving. New York: Elsevier North-Holland, 1979.

Larkin, L. I. "A fuzzy logic controller for aircraft flight control" in M. Sugeno, ed., Industrial Applications of Fuzzy Control. New York: North-Holland, 1985, pp. 87-103.

Lindsay, R. K., B. G. Buchanan, E. A. Feigenbaum, J. Lederberg. Applications of Artificial Intelligence for Organic Chemistry. The DENDRAL Project. New York: McGraw-Hill, 1981.

Michalski, R. S., J. G. Carbonell, T. M. Mitchell, eds. Machine Learning: An Artificial Intelligence Approach. Palo Alto, CA: Tioga, 1983.

Minsky, M. "A framework for representing knowledge" in P. Winston, ed., The Psychology of Computer Vision, New York: McGraw-Hill, 1975, pp. 211-277.

Patterson, B. EXSYS expert system diagnosis of RTD electrical problems. (Private communication.) New Brunswick Power Commission, Point Lepreau, N.B., October 1987.

Peng, Y., J. A. Reggia. "A probabilistic causal model for diagnostic problem-solving". Part I in IEEE Trans. Syst. Man and Cyber. 17 (2), 1987, pp. 395-406; Part II, IEEE Trans. Syst. Man and Cyber. 17 (3), 1987, pp. 395-406.

Reboh, R. "Knowledge engineering techniques and tools in the PROSPECTOR environment", Technical Note 243, SRI Projects 5821, 6415 and 8172, SRI International, Menlo Park, CA, 1981.

Reiter, R. "A theory of diagnosis from first principles". Artif. Intell. 32, 1987, pp. 57-95.

Robinson, J. A. "A machine-oriented logic based on the resolution principle". J. ACM 12 (1), 1965, pp. 23-41.

Robinson, J. A. Logic: Form and Function. New York, NY: Elsevier North-Holland, 1979.

Shafer, G. A Mathematical Theory of Evidence. Princeton, NJ: Princeton Univ. Press, 1976.

Shafer, G., R. Logan. "Implementing Dempster's rule for hierarchical evidence". Artificial Intelligence 33, 1987, pp. 271-298.

Shaw, M. L. G., B. R. Gaines. "Deriving the constructs underlying decision" in H.-J. Zimmermann, L. A. Zadeh, B. R. Gaines, eds., Fuzzy Sets and Decision Analysis Amsterdam: North-Holland, 1984, pp. 335-355.

Siler, W., D. Tucker. FLOPS A Fuzzy Logic Production System: User's Manual. Birmingham, AL: Kemp-Carraway Heart Institute, 1986.

Wirth, N. Algorithms + Data Structures = Programs. Englewood Cliffs, NJ: Prentice-Hall, 1976.

Young, E. "Reactor physics and dynamics". New Brunswick Power Commission Operating Manual BSI-03100, Point Lepreau, N. B., November 1985. Revised August 1987.

Zadeh, L. A. "Fuzzy sets". Information and Control 8, 1965, pp. 338-353.

Zadeh, L. A. "Fuzzy sets as a basis for a theory of possibility". J. Fuzzy Sets and Syst. 1, 1978, pp. 3-28.

Zadeh, L. A. "A theory of approximate reasoning" in Machine Intelligence 9, 1979, pp. 149-194.

Zadeh, L. A. "The role of fuzzy logic in expert systems". J. Fuzzy Sets and Syst. 11, 1983, pp. 199-227.

Zadeh, L. A. "Fuzzy logic". IEEE Computer, 21 (4), 1988, pp. 83-93.

VITA

Candidate's name:	Kirby Ward
Place of birth:	Charlottetown, P E Island
Permanent address:	10 Pine Drive Charlottetown, P E I C1A 6R8
Schools attended:	Prince of Wales College, Charlottetown, P E I. Senior matriculation (Honors), 1966.
Universities attended:	University of P E Island, Charlottetown, P E I. BSc (Mathematics), 1971.

TECHNICAL REPORTS

SCHOOL OF COMPUTER SCIENCE

Number	Date	Author	Title	No.of Pages	Price
TR74-001	Feb 1974	L.F. Johnson	A Search Algorithm for the Simple Cycles of a Directed Graph	33	$1.65
TR74-002	Oct 1974	W.D. Wasson R. McIssaac	A New Spanning Tree Algorithm	15	$0.75
TR74-003	Oct 1974	U.G. Gujar	Remote Job Entry and Output Through APL	13	$0.65
TR75-004	Apr 1975	U.G. Gujar	Subroutines with Variable Numbers of Arguments	17	$0.85
TR75-005	Jul 1975	L.E. Garey	Block Methods for Nonlinear Volterra Integral Equations	16	$0.80
TR75-006	Aug 1975	D.M. Fellows	Comments on "A General Fortran Emulator for IBM 360/370 Random Number Generator 'RANDU'"	13	$0.65
TR75-007	Aug 1975	L.E. Garey M. LeBlanc	Quadrature Formulae for Functions of Two Variables and Applications	22	$1.10
TR75-008	Sep 1975	L.F. Johnson	Determining Cliques of a Graph	14	$0.70
TR75-009	Oct 1975	D.M. Miller	An Algorithm for Determining the Chromatic Number of a Graph	21	$1.05
TR76-010	Jan 1976	L.E. Garey	Step by Step Methods for the Numerical Solution of Volterra Integro-Differential Equations	27	$1.35
TR76-011	Jan 1976	U.G. Gujar	A device Independent Computer Plotting System	21	$1.05
TR76-012	Mar 1976	P.P. Emin	A Partition Monitor for Fast-Batch-Processing with Limited Execution (Fable)	54	$2.70
TR77-013	Dec 1976	U.G. Gujar J.A. Fitzgerald	A Driver for Raster-Like Plotting Devices	24	$1.20
TR77-014	Jan 1977	U.G. Gujar D.M. Fellows	Automatic Job Scheduling in HASP	30	$1.50
TR79-015	May 1979	U.G. Gujar J.M. DeDourek M.E. McIntyre	A Method for Designing a Lexical Analyzer	30	$1.50

Number	Date	Author	Title	No.of Pages	Price
TR79-016	Jun 1979	U.G. Gujar A.R. Nagesh	ϕSCUBA A Buffered Core Graphics System	33	$1.65
TR79-017	Jun 1979	T.A. Middleton	A Transformation Approach to Implementing Aggregate Operations	30	$1.50
TR79-018	Jul 1979	T.A. Middleton	On Assignment Between Data Paths	40	$2.00
TR79-019	Aug 1979	T.A. Middleton	On the Use of "Fixed" Data Paths for Assignment	25	$1.25
TR79-020	Aug 1979	T.A. Middleton	Representing Data Paths by Program Structures	35	$1.75
TR79-021	Sep 1982	J.D. Horton	Sets With No Empty Convex 7-Gons	10	$0.50
TR83-022	Jun 1983	J.D. Horton	Resolvable Path Designs	29	$1.10
TR83-023	Aug 1983	U.G. Gujar F.W.L. So	Computer Display of Characters	62	$3.72
TR84-024	Jun 1984	J.D. Horton	A Lower Bound on the Number of One-Factors in Bicubic Graphs	22	$1.10
TR84-025	Jul 1984	U.G. Gujar	3-D Graphics in APL: User Perspective	27	$1.35
TR84-026	Aug 1984	J.D. Horton	A Polynomial-Time Algorithm to Find the Shortest Cycle Basis of a Graph	16	$ .80
TR84-027	Sep 1984	L. Goldfarb	The Design of Efficient Pattern Recognition Systems	19	$0.95
TR84-028	Nov 1984	W.R. Knight	A Fortran to PL1 Dictionary	15	$0.75
TR85-029	Sep 1985	U.G. Gujar C.A. Crawford	Computer Processing of Musical Scores	40	$2.00
TR85-030	Dec 1985	V.C. Bhavsar J.R. Isaac	Design and Analysis of Parallel Monte Carlo Algorithms	49	$2.45
TR86-031	Jan 1986	N.N. Datar U.G. Gujar V.C. Bhavsar	Analysis and Synthesis of Interpolating Functions for 3-D Objects	22	$1.10
TR86-032	Jan 1986	T.A. Tassou	Adaptation of a Monte Carlo Radiation Transport Code to Supercomputers	143	$8.58
TR86-033	Feb 1986	L.A. Lambrou	Pseudo-Random Number Sequences For Parallel Computers	209	$12.54

Number	Date	Author	Title	No. of Pages	Price
TR86-034	May 1986	U.G. Gujar L. Slipp	A Computer Aided Design And Tutorial System (CADETS)	23	$1.15
TR86-035	Jun 1986	T. Tassou V.C. Bhavsar E. Hussein K.A. Gallivan	Monte Carlo Neutron Transport on the Alliant FX/8	25	$1.25
TR86-036	Oct 1986	U.G. Gujar	User Interface to a Flexible Character Generator	18	$0.90
TR87-037	Apr 1987	J.D. Horton	Orthogonal Starters in Finite Abelian Groups	31	$1.55
TR87-038	Aug 1987	U.G. Gujar V.C. Bhavsar N.N. Datar	Interpolation Techniques For 3-D Object Generation	45	$2.25
TR87-039	Oct 1987	V.C. Bhavsar	Time Complexity of Sequential and Parallel Monte Carlo Solution of Linear Equations	33	$1.65
TR87-040	Oct 1987	V.C. Bhavsar L.A. Lambrou J.D. Horton U.G. Gujar	Evaluation of the Discrepancy of the Linear Congruential Pseudo-Random Number Sequences	26	$1.30
TR88-041	Apr 1988	U.G. Gujar I.V. Nagendra	3D Solids from 2D Views	46	$2.30
TR88-042	Jun 1988	Kirby Ward	XpeRTD: An Expert System for Diagnosing Resistance Temperature Detectors	246	$14.76