# A comparison of various methods for determining confidence factors within an expert system

by

Lance W. Christiansen

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Mechanical Engineering
Major: Nuclear Engineering



Iowa State University
Ames, Iowa
1990

# TABLE OF CONTENTS

# LIST OF TABLES

## LIST OF FIGURES

## ABSTRACT

An expert system has been developed at Iowa State University to diagnose common problems and prescribe maintenance planning procedures for motor-operated valves at the Duane Arnold Energy Center in Palo, Iowa. The system is capable of two methods of diagnosis. The first uses the confidence factors inherent in the system used to develop the program (known as Level5). Level5 has a one to one correspondence between a symptom and its diagnosis. The second method uses confidence factors similar to those used in the Mycin medical diagnosis expert system to determine the relative likelihood of several different possible valve problems based on a combination of symptoms. This allows the user to choose which diagnosis is the most likely.

## ACKNOWLEDGEMENTS

I would like to thank the Power Affiliates research program and Iowa Electric Light and Power company for their generous support on this and other projects in the Nuclear Engineering department. I would also like to thank the excellent staff of the Nuclear Engineering department who, despite the hard times recently experienced, have maintained a high level of professionalism. I would especially like to thank Dr. Richard Danofsky for his support and suggestions. Lastly, I would like to thank my parents and fellow graduate students for their friendship and support. They have made my stay here at Iowa State an enjoyable one.

# 1. INTRODUCTION

## 1.1 Problem Statement

Maintenance at a nuclear power plant is a time-consuming and expensive task. It has been estimated that over $100,000,000 is lost each year merely in plant down time due to maintenance [1]. One area of maintenance that consumes over 30% of the industries' annual maintenance budget is valve maintenance. Clearly, any measures that can be taken to reduce the cost of this endeavor and increase public safety by improving the quality and thoroughness of maintenance procedures would be beneficial. For this reason, a valve diagnostic and maintenance planning program was developed at Iowa State University for implementation at the Duane Arnold Energy Center in Palo, Iowa. This program is capable of diagnosing many problems common to motor-operated valves (MOVs) at the plant.

In the original expert system, each valve symptom has a corresponding diagnosis. This is a useful feature, but in the real world there is usually not a 100% correspondence between a symptom and a diagnosis. Often there is more than one symptom present, and this combination of symptoms can indicate a different problem than any one of the component symptoms alone would. That is why it would be useful to have an expert system capable of taking multiple symptoms into account. The goal of this research was to implement an expert system capable of doing this via the use of

*confidence factors.* Confidence factors will be explained fully in later chapters, but basically they are numbers reflecting an expert's degree of confidence in a conclusion. By suitably combining the confidence factors for each individual symptom, a ranking of possible causes for valve problems from most likely to least likely can be obtained.

## 1.2 Scope of Thesis

Chapter 1 provides an explanation of the field of artificial intelligence and describes how expert systems fit into this field. It begins by explaining the goal of artificial intelligence in general, then moves on to expert systems in particular in Section 1.4. Some concepts common to most expert systems are discussed here.

Chapter 2 reviews some famous successful expert systems. It also covers the status of expert systems in the nuclear industry to date, and introduces some specific applications of expert systems in the field.

Chapter 3 introduces the idea of the *confidence factor* or CF as a method of ranking diagnoses. The Mycin confidence factor system is discussed in Chapter 3. This was the first successful diagnostic program as well as the first to use the confidence factor system. The various constraints that govern CFs are described here, as well as methods for combining the confidence factors, since most expert systems that incorporate confidence factors use a modified form of the Mycin confidence factors.

Chapter 4 describes the expert system development tool (Level5) that was used to implement the valve problem diagnosis and maintenance planning program that is being developed in the nuclear engineering department at Iowa State University. The structure of its rules is discussed as well as the method of determining confidence factors used by Level5.

Chapter 5 introduces some alternatives to confidence factors that may be suitable for future development. Section 5.1 covers the topic of fuzzy logic pioneered by L. Zadeh in the early 1970s. Fuzzy logic methods may be an alternative to confidence factors in that users could indicate their confidence in a conclusion through linguistic statements rather than numbers. The conventions and mathematical notation used by the fuzzy logic system are covered in Section 5.1.1. Section 5.1.2 describes some successful system controllers that use fuzzy logic. Section 5.2 describes an alternate method of ranking possibilities called Dempster-Shafer calculus that is different from the confidence factor concept and may be suitable for development in a future expert system.

Chapter 6 describes the valve maintenance and diagnostics program developed using Level5 and goes through an example session with the expert system in order to illustrate some of its capabilities. This chapter includes an illustration of how confidence factors can be used to help improve the diagnosis of valve problems.

## 1.3 Introduction to Artificial Intelligence

Artificial intelligence (AI) has been defined as:

> "The study of mental facilities through the use of computational models" [2].

AI includes the following general areas:

1. Robotics- This area is primarily concerned with developing visual and tactile processing programs that will allow robots to observe and interpret their environments.

2. Natural language processing- Concerned with developing programs that can read, speak, and understand language as people use it in everyday conversation.

3. Expert systems- Concerned with developing programs that use symbolic knowledge to simulate the behavior of a human expert, i.e., to "reason" like a human being.

More simply put, the ultimate goal of AI as stated in [2] is to "build a person" or at least something that acts as efficiently and intelligently as a person. At present, researchers are far from achieving this goal.

In order to create a machine capable of duplicating human reasoning processes, it is necessary to have some understanding of just how people do reason. Previous tests of intelligence have concentrated on how well people think when compared to one another, rather than how people think in a qualitative sense which is what AI researchers need to establish.

The basic steps of human thinking are divided much the same way that computer "reasoning" is thought of by computer scientists. The steps involved are encoding, storing, and recalling information [3]. Although the steps are the same, the processes utilized by computers versus humans could be (and almost certainly are) very different. Human problem solving is the model for most expert systems, since even though this may not be the most reasonable way of doing things, at least AI researchers know that it is *possible* to reason in this way.

## 1.4 Introduction to Expert Systems

Expert systems can be broadly defined as:

> "computerized processes or programs that attempt to emulate human thought processes associated with the application of expertise to problem-solving" [4].

Research in the area of expert systems includes investigation into the methods and techniques for constructing man-machine systems with specialized problem-solving expertise. Expert systems are often developed with the aid of a tool or *shell* which is an interface between the user and the computer language used to implement the system. The system in use to develop the system discussed in this thesis is known as Level5 (formerly Insight2+)[1]. Its shell language is known as PRL (for **P**roduction **R**ule **L**anguage). Level5 uses Pascal as its primary processing language, although the user does not need to know Pascal to use Level5.

Knowledge in any field is of two sorts: public and private. The public facts can usually be found in textbooks and other references, but expertise generally also involves a good amount of private knowledge. This private knowledge is usually in the form of "rules of thumb," or heuristics. Heuristics enable a human expert to make educated guesses, recognize promising approaches to problems, and deal effectively with erroneous or incomplete data. It is in the area of private knowledge that expert systems can play an important role.

In the past, knowledge in one of the more heuristic areas could only be gotten by hiring an expert (at often considerable expense), who had learned the rules of

[1] Level5 is published by Information Builders, Inc. 1250 Broadway, New York, N.Y. 10001. (212) 736- 4433.

thumb through years of experience. Expert systems have the potential to change this to a large extent by capturing the knowledge of the expert in a computer program. The knowledge in an expert system is confined to a narrow area of expertise, mainly because of the sheer magnitude of trying to cover a very broad area of knowledge. Expert systems are most effective in situations where human expertise is in great demand and short supply. In these cases, expert systems offer the following advantages:

- Greater reliability and consistency.
- Greater speed.
- Increased accessibility.
- Reproducibility of results.
- The knowledge in them remains after the expert leaves.

The nuclear industry, because of the complexity of much of its technology and because of its high requirements for safety, has need of experts in many fields. These experts must be highly trained and they do not "grow on trees" so the criteria for effective expert systems is met. Expert systems can play an important role in diagnostics, operator training, safety analysis, and other areas (see Chapter 2).

Expert systems generally use a limited English vocabulary of the form **IF** (condition A) **AND** (condition B) **THEN** (conclusion) to express relationships among objects. Statements of this kind are referred to as production rules. Because of the rule-like nature of these statements, expert systems are sometimes referred to as *rule-based* systems.

Two different methods of reasoning are generally associated with these basic rules, namely backward and forward chaining. In a forward chaining (or data driven) system, an initial set of facts is used to infer new facts by executing the appropriate rules. This technique is useful when there are many possible solutions to a problem, because many different possibilities can be pursued at once. Backward chaining (or goal driven) systems begin by assuming a final or root goal and then attempting to satisfy this goal by determining the truth or falsity of the rules and facts that conclude the desired goal. This technique works well on systems that have a relatively small number of known, well-defined solutions. Level5 is a primarily backward chaining tool although it is capable of forward chaining.

One of the most useful features of expert systems is their ability to come to conclusions in the face of incomplete or even erroneous data. Expert systems accomplish this either through the use of baysian statistics (covered in Chapter 3) or through the use of *confidence factors* or CFs, which reflect the experts confidence that a given fact or condition is true. This feature will be covered extensively in the following chapters.

An expert system typically consists of two main components: the knowledge base and the inference engine. The knowledge base and working memory constitute one part of the system, and the inference engine and all the subsystems and interfaces constitute the second part. Figure 1.1 shows a block diagram of a generic expert system.

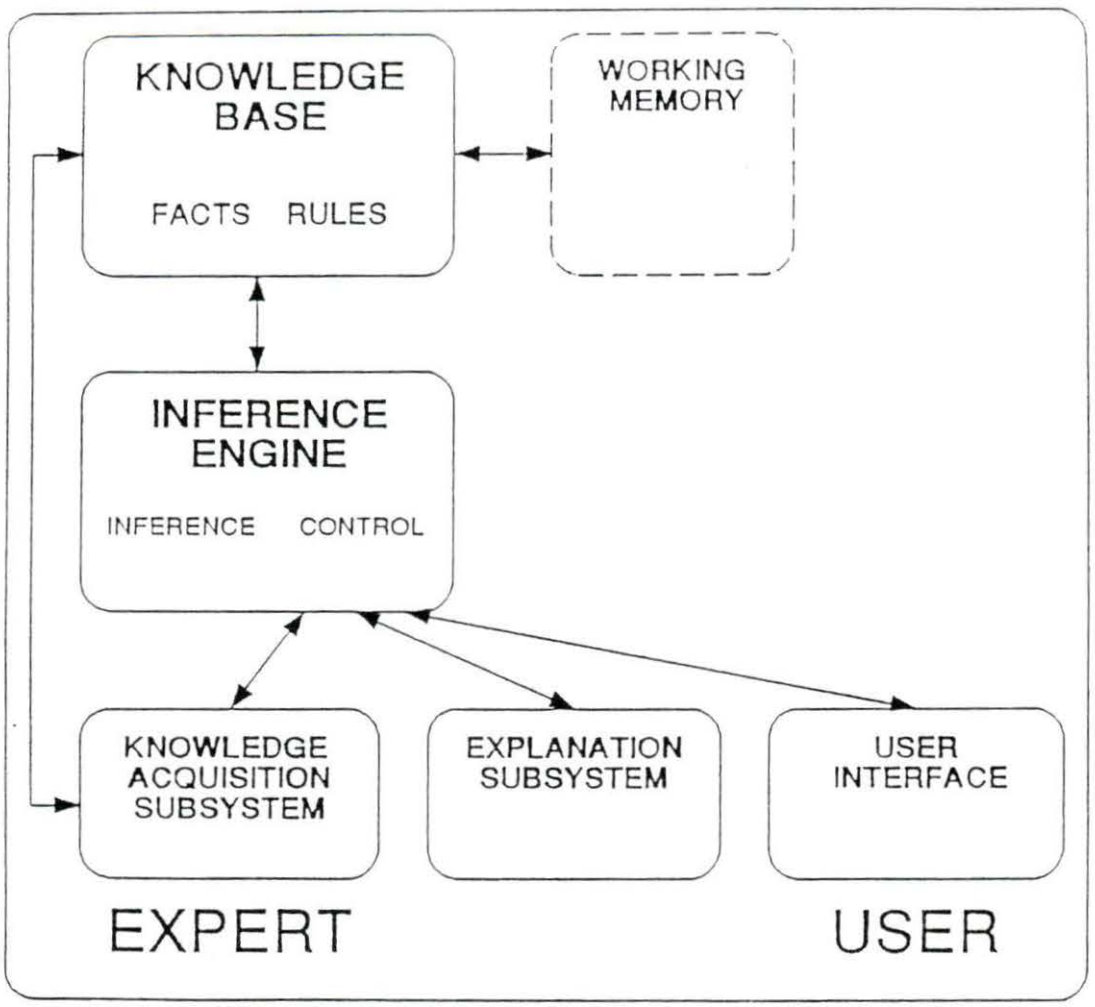


Figure 1.1: A Generic Expert System

### 1.4.1 The Knowledge Base

The knowledge base contains the facts and rules that embody the expert's knowledge. On the surface, much knowledge looks like it consists of deep or complex ideas, but it is often found that this is not the case; knowledge can often be broken down into simple relationships between objects [5].

There are five different methods which expert systems use to encode the facts and relationships that represent knowledge. A brief description of each type is presented below. The interested reader is referred to [5] for more details.

1. Semantic networks- The semantic network or semantic net is one of the oldest representational schemes in AI. A semantic net consists of a series of *nodes* connected together by *links*. The nodes usually represent objects or attributes of objects. The links connecting them represent the relationships between the nodes. Two common links are the "*has-a*" link and the "*is-a*" link. The first simply refers a to characteristic of the object such as "the valve *has-a* leak." The second refers to a general class the object belongs to such as "the valve *is-a* motor-operated valve."

2. Object-attribute-value triplets (OAVs)- Objects can be physical objects such as *door* or *window*, or they can be conceptual ideas. Attributes are characteristics associated with objects such as color, size or shape. Values are the value of the attribute. This scheme is just a specialized case of a semantic net with the object-attribute link being a has-a link and the attribute-value link being an is-a link. The OAV triplets can be modified by *certainty* or *confidence factors* which represent the user's confidence that a value for an attribute exists. This is the scheme used in the development of Mycin, which is covered in Chapter 3.

3. Rules- Rules can be used with OAV representations. They consist of a premise (also called an antecedent) which can be composed of several expressions or if-clauses. This is followed by a single conclusion or then-clause. Examples of Level5 rules are covered in Chapter 4.

4. Frames- A frame is a description of an object that contains slots for all the information associated with the object. Slots are like attributes in that they

may store values. They may also contain default values, pointers to other frames, sets of rules, or procedures by which values may be obtained. Frames are more difficult to develop than OAV triplets but they allow for a richer storage of knowledge.

5. Logical expressions- Logical expressions can also be used to represent knowledge. The two most popular methods of representing logical relationships are *prepositional logic* and *predicate calculus.* Prepositions are statements that are either true or false. They are linked together by connectives such as AND, OR, NOT, etc. Prepositional logic is concerned with the truthfulness of statements. Predicate calculus is an extension of prepositional logic. The basic units of predicate calculus are called objects. Statements about objects are called predicates. For example "*is-red(ball)*" is an assertion that the ball is red. Some AI computer languages, notably Prolog, use this type of representation in their rules.

### 1.4.2 The Inference Engine

The inference engine contains the inference strategies and controls that an expert system uses when manipulating facts and rules. It examines existing facts and rules, and adds new facts when possible. The inference engine also decides in which order the inferences are made.

**1.4.2.1 Inference** The most common reasoning strategy that inference engines employ is known as *modus ponens.* This strategy simply states that if it is given that an antecedent leads to a conclusion, and it is known that the antecedent is true, then the conclusion is true. As an example of this, consider the following example where *you have cancer* is the antecedent and *you are sick* is the conclusion:

$$\frac{\begin{array}{r}\text{IF you have cancer THEN you are sick} \\ \text{you have cancer}\end{array}}{\text{you are sick}} \tag{1.1}$$

If it is known that *you have cancer* then it can be concluded that *you are sick.* Note that this is different from reasoning by abduction which will be covered in Section 3.2.

A feature that is unique to the reasoning strategy of expert systems, which conventional languages lack, is the ability to reason with uncertain information. One method of handling uncertainty deals with handling unknown information. In this case the rules are allowed to fail if their antecedents can not be evaluated. If the rules' antecedents are connected by AND statements then the failure of one condition leads to the failure of the entire rule. If, on the other hand, the antecedents are connected by OR statements, the failure of one antecedent does not necessarily mean that the entire rule must fail.

**1.4.2.2 Control** There are two problems that the control portion of the inference engine must address:

- It must have a way to decide where to start.
- It must have a way to resolve conflicts that emerge due to alternative lines of reasoning.

Because Level5 is a backward chaining system, the first problem is solved by the fact that it starts with the goal statement and looks for rules that conclude the goal. If there is more than one rule that concludes the goal, then Level5 looks for the rule

with the highest confidence value. In the case where the confidence values are the same, the first rule encountered gets fired first. To address the second problem, an identical system is employed in the case of rules with the same conclusion if they have the same confidence factors.

# 2. LITERATURE REVIEW

## 2.1 Introduction

Expert systems had their practical debut with a program called Heuristic Dendral in 1965 [2],[5]. In that year Joshua Lederberg, who was a Nobel Prize-winning chemist, developed the basic algorithm for a system designed to establish the chemical structure of unknown molecules. Edward Feigenbaum of Stanford University and Bruce Buchanan set out to try to incorporate Lederberg's rules into an heuristic expert system. Given a molecule's atomic formula and its mass spectrograph, Dendral was able to determine the structure of unknown compounds. Obtaining the heuristic rules needed to implement Dendral took over fifteen man-years, but the knowledge gained from it helped tremendously to establish the field of knowledge engineering. The Stanford group used its knowledge to go on to develop other expert systems, probably the most famous of which is the Mycin system which will be covered in Chapter 3.

Following the success of Mycin, companies began to take note of expert systems technology. Some of the more successful are presented below.

- Macsyma- Developed at MIT to assist in solving complex mathematical problems [5]. This system is still under continual improvement and is used by hundreds of researchers daily. It is the most powerful program yet developed

to solve algebraic equations using a computer. It played a major role in convincing the artificial intelligence community that expert systems are capable of high levels of performance.

- Hearsay I and II- Developed at Carnegie-Mellon University from the late 1960s to mid 1970s [5]. This system was designed to understand spoken human speech patterns. At the close of the project, the program was able to understand over 1,000 words and respond in about the same time that a human would. Hearsay clearly demonstrated the superiority of heuristic methods over statistical methods when dealing with problems involving the meanings of words in context.

- Prospector- Developed in the early 1970s at Stanford Research Institute, International (SRI) for the U.S. Geological Survey [5]. Prospector was designed to provide consultations to geologists in the early stages of searching for ore-grade mineral deposits. It allows users to stop it at any stage in the consultation process and actually volunteer information. It then adjusts its inferencing strategy according to the information provided. In 1980, Prospector became the first expert system to achieve major commercial success when it discovered a large molybdenum deposit.

- Xcon- Developed by Digital Equipment Corporation to configure their Vax computers [2]. This forward chaining expert system has all but replaced people in its area of expertise. The Xcon program now configures all orders for Vax computers.

## 2.2 Expert Systems Uses in the Nuclear Industry

An expert system can be a vigilant assistant to plant operators as well as an aid in plant management and maintenance in nuclear power plants, where safety is of great concern. Because of demands for greater safety margins, lower environmental impacts, increased performance, etc., automation of most functions in nuclear power plants is inevitable [6]. Because of the complexity of these power plants, the use of expert systems in such areas as operator training, diagnostics, and safety analysis will be invaluable.

Because of their importance to cost and safety in the nuclear industry many different organizations including nuclear equipment vendors, engineering firms, national laboratories, the utility industry, universities, and others have shown interest in developing expert systems technology. The main areas of emphasis in the industry have been [4]:

1. Fault recognition.
2. Diagnosis and recovery.
3. Task planning.
4. Intelligent operator interfaces.
5. Intelligent systems control.

The system described in this thesis falls into the second and third areas. There are many systems being developed in each area. It is the purpose of this section to give the reader an idea of some of the systems that have been implemented.

The most coherent efforts have been undertaken by *EPRI* (the Electric Power Research Institute), which is funded by a consortium of utility companies from around the world. One of the first EPRI-sponsored projects was Realm (the **R**eactor **E**mergency-**A**larm-**L**evel **M**onitor), developed by Technology Applications, Inc. for Indian Point Unit 2 in cooperation with Consolidated Edison of New York. Realm was designed to identify emergency situations at a nuclear power plant by comparing symptoms it observes with the list of events stored in its database. This is important because the NRC (Nuclear Regulatory Commission) has issued specific guidelines on how to classify a particular emergency situation, with each situation requiring a

specific set of responses. It reportedly performed well in a recent drill when operated in parallel with the plant's normal emergency operating procedures [4].

An expert system called PIPES is a pc-based software system developed by Combustion Engineering which automates the lengthy process by which steam generator tubes are to be eddy current tested for structural flaws. PIPES also assists in implementing the inspection test pattern necessary to verify repairs [7].

Westinghouse offers a complete motor-operated valve maintenance program called VITALS (**V**alve **I**ntelligent **T**est and **A**na**L**ysis **S**ystem) and an on-line Valve Monitoring System (VMS). They have developed the system in response to tougher guidelines on motor-operated valves (MOVs) mandated by the NRC in June, 1989. The data collected by VMS and VITALS is sent to an expert system developed by Westinghouse for analysis, or printed out for human inspection. Typical time for set-up and testing of valves is less than two hours as opposed to the up to twenty hours normally required [8].

Ohio State University is developing an Operator Advisor System (OAS) to aid nuclear plant operators in identifying abnormal operating conditions [9]. The system is being developed for the Perry BWR (Boiling Water Reactor) power plant. It has three components: monitoring, procedure management, and diagnosis. The monitoring portion is designed to detect known malfunction states and threats to plant safety. The procedure management component is designed to control plant malfunction states and initiate safety maintenance procedures when it anticipates threats to safety. The diagnostic portion is designed to diagnose both known and unknown malfunction states before the traditional plant alarms are triggered. This system uses the "symptom-oriented" approach towards safety that has been in favor

since the Three Mile Island accident rather than the "event-oriented" approach. The event-oriented approach has been abandoned because too much emphasis was placed on the operator's ability to identify the exact problem quickly even in the case of multiple events occurring simultaneously, and because it is not possible to define *all* possible events and situations beforehand.

At Iowa State, a system known as ESAS (**E**xpert **S**ystem for **A**nalyzing **S**ystems) has been developed [10]. ESAS finds cut-sets for a PRA (Probabilistic Risk Assessment) without having to first generate fault-trees. A cut-set is a set of components whose failure causes the failure of the system. Constructing fault-trees is time-consuming and requires some guesswork. Also, most computer codes for finding fault-trees require a mainframe to execute. Therefore, by eliminating fault-trees, much time and effort can be saved.

As mentioned in Section 1.1, a valve problem diagnosis and maintenance planning expert system has been developed by Michael J. Winter and Dr. Richard Danofsky at Iowa State University [1]. The system is capable of diagnosing many problems commonly found in MOVs, and specifying the post-maintenance tests that need to be performed on these valves once maintenance has been performed. This system will be covered in more detail in Chapter 6.

# 3. MYCIN

## 3.1 Background of Mycin

One of the first expert systems ever created is known as Mycin [5]. Mycin was developed in the middle seventies at Stanford University, with the goal of assisting physicians in the diagnosis and treatment of meningitis (inflammation of the membranes around the brain and spinal cord) and bacteremia infections (infections caused by bacteria in the blood). Prior to the development of Mycin, AI had been criticized for solving only "toy" problems with little or no practical significance. The Stanford researchers set out to implement an expert system that had practical value.

The problem domain of meningitis was chosen because this disease requires quick treatment. However, because laboratory results may take twenty-four to forty-eight hours to complete, the attending physician was often forced to call in expert help to aid in his diagnosis. This often took precious time: time that quite often neither the patient nor the doctor could afford to waste. Because of the advantages of expert systems (see Section 1.4) this area seemed to be one that could greatly benefit from expert systems technology.

To make a diagnosis, Mycin asks the user various questions about the patient, such as his or her disease history and whether certain bacteria are present in the patient's body. Based on this information Mycin draws conclusions about which

diseases may be present and ranks them according to their likelihood.

The results of the first trials were very encouraging. Mycin gave correct diagnoses in 65% of the tested cases, while the human physicians scored 63% on the average [5]. This was somewhat surprising both because the expert system did better than expected and because the experts did worse.

Because of the early success of Mycin, many expert systems software developers looked at Mycin's reasoning strategy when designing their systems. Because of this most expert systems, including Level5, use a modified form of Mycin's confidence factor system. Mycin's developers have developed a tool known as Emycin which employs Mycin's inferencing strategy, leaving it to the individual developers to input their own rules. Because so many expert systems developers have used a modified form of Mycin confidence factors, they will be covered in detail in this chapter.

## 3.2 Baysian Statistics

Mycin uses a form of reasoning known as abduction. Reasoning by abduction has the following format; if it is known that $A$ follows from $B$, and also that $B$ exists or is true, it can inferred that $A$ exists or is true. Written in predicate calculus format this is:

$$\begin{array}{r} (\text{IF A B}) \\ B \\ \hline A \end{array} \tag{3.1}$$

This form of inference is not legal inference as deduction is, because it can lead to incorrect conclusions. For example, a use of abduction that would lead to an

incorrect conclusion might be:

$$\begin{array}{r} \text{IF you have cancer THEN you are sick} \\ \text{you are sick} \\ \hline \text{you have cancer} \end{array} \tag{3.2}$$

The patient may indeed have cancer but this is hardly certain on the basis of just one symptom. However, since this is the type of reasoning that physicians actually use in making diagnoses (although usually with more than one piece of evidence), it is still a valuable reasoning tool in areas that involve the diagnosis of a problem or disease from its "symptoms". Such areas include medical diagnosis, mineral exploration and equipment maintenance.

Another potential disadvantage of the abduction method is that it is possible to get more than one correct answer to a question. For example, if your dog comes into the house all wet you might decide that it was caught in a rain shower, but it is also possible that the dog walked by a lawn sprinkler. It is not known which of these possibilities is true with absolute certainty. The best one can do is try to determine which of several possibilities is the most probable.

In order to weigh the facts for or against each conclusion the expert system developer must somehow make two decisions. The first is to decide how strongly a fact weighs for or against a conclusion, and the second is to determine how to combine the various pieces of evidence into a final conclusion [2]. This is accomplished in many expert systems via the implementation of confidence factors or CFs which will be covered in detail in Section 3.3. Confidence factors are generally numbers between zero and one which reflect the expert system user's level of belief or disbelief in a fact

or conclusion.

The Mycin program uses a modified form of Baysian statistics, which are based on conditional probabilities. In the case of the Mycin program this is the probability that a patient has a disease $D$ in light of some evidence $E$ or $P(D \mid E)$. Each $E$ term consists of one or more $S_k's$ which correspond to the symptoms of the problem. This is in contrast to the more familiar unconditional probability $P(D)$ which is the probability that a certain disease is present before any evidence is gathered. Unconditional probabilities are sometimes called prior probabilities and conditional probabilities are called posterior probabilities.

Prior probabilities are defined mathematically by the expression:

$$P(D) = \frac{D}{Pop} \tag{3.3}$$

where $Pop$ is the total population and $D$ is the number of people in the population with condition $D$. The posterior or conditional probability is defined as:

$$P(D \mid E) = \frac{D \cap E}{E} \tag{3.4}$$

Where $P(D \mid E)$ means the probability of $D$ given the evidence $E$. It is equal to the intersection of sets $D$ and $E$ divided by the set of individuals exhibiting $E$. An expression like $P(D \mid E)$ often cannot be evaluated directly because it is never known for certain how many people with a group of symptoms $E$ actually have disease $D$ in a given population. One thing that can be much more easily determined however is the probability that a patient will exhibit a set of symptoms given that the disease is present, or $P(E \mid D)$. Physicians can also determine the probability of the disease occurring in the population $P(D)$ and the instances of a certain symptom occurring

in the population $P(E)$. From these factors, $P(D \mid E)$ can be determined using Bayes's theorem which states:

$$P(D \mid E) = \frac{P(D) \times P(E \mid D)}{P(E)} \tag{3.5}$$

Since the numbers on the right side of the equation are easier to obtain than the ones on the left, Bayes's theorem simplifies the task of calculating the probabilities involved.

But there is an even greater problem involved with calculating conditional probabilities that has not been addressed. For $m$ diseases and $n$ symptoms, there are approximately $m \times n$ numbers to calculate (actually $m \times n$ conditional probabilities $+ m$ disease probabilities $+ n$ symptom probabilities). This is not too formidable a task, especially if it is recognized that some symptoms and diseases can be ruled out as being unrelated to one another. A difficulty occurs when one must consider the possibility of multiple symptoms in a disease, however.

For example, suppose $P(D \mid S1\&S2)$ needs to be calculated, where $S1\&S2$ stands for the presence of two symptoms simultaneously. Then for each pair of symptoms $i$ and $j$ $P(S1\&S2 \mid D)$ and $P(S1\&S2)$ need to be determined. The number of these pairs is $n * (n-1) \approx n^2$ so now approximately $m * n^2$ combinations must be considered. This is a great increase over the number of possibilities that had to be considered previously, and if three symptoms must be considered the problem becomes even worse. As an example, if there are 500 diseases with 3,000 symptoms ($m = 500, n = 3,000$) the number of single symptom cases that must be considered is approximately 1,500,000, which is a large but manageable number for today's computers. If, on the other hand, we consider the two symptom case the number

of values rises to $500 * 3000^2 = 4,500,000,000$, and adding more interdependent symptoms makes the situation even worse [2]. In addition to this, in the real world the conditional probabilities of multiple symptom diseases or problems will probably not be known.

It is therefore necessary to implement a system where single symptom conditional probabilities can be combined with other single symptom probabilities to give a good approximation to the multiple symptom case without having to calculate so many values. Mycin solved this problem via the implementation of the so-called confidence factor, or CF.

### 3.3 Mycin Confidence Factors and Measures of Belief and Disbelief

In the Mycin system the concept of confirmation is used rather than that of strict probability. Confirmation does not indicate that a hypothesis is proven, but rather that an observation lends credence to it [11]. This level of support in an observation is denoted as C[h,e], or the degree of confirmation in $h$ based on the observation $e$. Confirmations cannot be manipulated as though they were probabilities, as this leads to inconsistencies or paradoxes. An example of this is the famous Paradox of the Ravens [11]. Suppose $h1$ is the statement "All ravens are black" and $h2$ is the statement "All non-black things are non-ravens." Even though these two statements are logically equivalent, it is not valid to assert that $C[h1, e] = C[h2, e]$ for all $e$, as would be the case in probability theory. For example if $e$ is the evidence that a vase is green, it seems silly to suggest that the green vase supports the fact that all ravens are black even though it does support the fact that all non-black things are non-ravens.

Another important point to consider is that $C[h, e]$ does not equal $1 - C[\bar{h}, e]$ [1]. A disconfirmation function of some sort is then needed because a supposition and its inverse are not related as they are in conventional probability theory. Another way of saying this is that the CFs are not symmetrical [12].

For example, suppose there are two rules with associated CFs (assume they are Mycin rules even though they are presented in a format similar to Level5):

```
RULE A
IF stalled motor
THEN valve stem binding CF 0.7

RULE B
IF excessive handwheel effort
THEN valve stem binding CF 0.6
```

Mycin rules use a technique known as *uncertainty addition* to combine CFs. Using Equation 3.29 (to be shown later) the combined CF of *packing too tight* is:

$$X + Y(1 - X) = 0.7 + 0.6(1 - 0.7) = 0.88 \tag{3.6}$$

In other words, it is 88 percent certain that the valve stem is binding if both symptoms are present. If, as in conventional probability, we assumed that $P(\bar{A}) = 1 - P(A)$ then the following rules would result:

```
RULE A
IF NOT stalled motor
THEN NOT valve stem is binding CF 0.3
```

[1] A bar over a variable means its negation or opposite.

```
RULE B
IF NOT excessive handwheel effort
THEN NOT valve stem is binding CF 0.4
```

This would give the chance of valve stem is not binding given the two symptoms as:

$$0.3 + 0.4(1 - 0.3) = 0.58 \tag{3.7}$$

in which case *valve stem binding* would be $1 - 0.58 = 0.42$. This is quite different from the previous answer, so obviously conventional probability theory is not relevant.

In order to determine some way to combine these confidence measures some new terms need to be introduced. These terms are called measures of belief ($MB$) and measures of disbelief ($MD$) and are defined as follows:

$$MB[h, e] = X : \text{The measure of increased belief in h based on e is X.} \tag{3.8}$$

$$MD[h, e] = X : \text{The measure of increased disbelief in h based on e is X.} \tag{3.9}$$

The evidence $e$ may not be an event but could be another hypothesis that is itself subject to confirmation.

Under Mycin's system if a rule increases belief in a hypothesis, it also proportionately decreases the disbelief in the hypothesis according to the formula:

$$\frac{P(h \mid e) - P(h)}{1 - P(h)} \text{ , if } P(h \mid E) > P(h). \tag{3.10}$$

This is called the measure of increased belief in $h$ resulting from $e$. This is equivalent to $MB[h, e]$ discussed above and, as noted, occurs when the conditional

probability is greater than the prior probability. Conversely, if the conditional probability is less than the prior probability, the systems disbelief would increase while its belief would decrease. This proportionate decrease in belief is given by:

$$\frac{P(h) - P(h \mid e)}{P(h)} \quad \text{if } P(h \mid e) < P(h) \tag{3.11}$$

This is the measure of increased disbelief in $h$ resulting from $e$ and is denoted by $MD[h, e]$.

In short, the measure of increased belief is proportionate to the decrease in disbelief, and the measure in increased disbelief is proportionate to the decrease in belief. Belief is estimated by $P(h)$ at any given time and disbelief is estimated by $1 - P(h)$. Some important points to note here are that a single piece of evidence $e$ cannot both favor and disfavor a hypothesis, so when $MB[h, e] > 0$, $MD[h, e] = 0$. Similarly, when $MD[h, e] > 0, MB[h, e] = 0$. This leads to the somewhat more formal definitions of MB and MD given below:

$$MB[h, e] = \begin{cases} 1 & \text{if } P(h) = 1 \\ \frac{max[P(h|e), P(h)] - P(h)}{max[1,0] - P(h)} & \text{otherwise} \end{cases} \tag{3.12}$$

$$MD[h, e] = \begin{cases} 1 & \text{if } P(h) = 0 \\ \frac{min[P(h|e), P(h)] - P(h)}{min[1,0] - P(h)} & \text{otherwise} \end{cases} \tag{3.13}$$

Finally, we define a certainty factor (CF) as the difference between the measures of belief and disbelief divided by a weighting factor which will be explained in Section 3.4:

$$CF = \frac{MB[h,e] - MD[h,e]}{1 - min(MB[h,e], MD[h,e])} \tag{3.14}$$

This certainty factor is necessary when two competing hypothesis must be compared. The certainty factor can range in value from $-1$ to 1 as opposed to MB and MD which both have values between 0 and 1. If a piece of evidence $e$ neither confirms nor disconfirms a hypothesis it is assigned a CF of zero.

### 3.4 Combining Measures of Belief and Disbelief

The certainty factor was originally defined as simply the difference between the MB and MD factors but this was changed to the form noted in Equation 3.14 in more recent versions of Mycin for two reasons. First, under the old rules there was a tendency for a piece of evidence with a large MD to overwhelm several pieces of positive evidence, or vice versa. The second deals with the commutivity of the CFs and will be explained below.

The tendency of one large piece of negative (positive) evidence to overwhelm several positive (negative) pieces was taken into account by a term in the denominator which acts as a weighting factor to decrease the effects of either a MB much larger than its corresponding MD or vice versa [13]. For example, under the original system if there was a large body of evidence which supported a conclusion with an MB of 0.99 and a single piece of evidence with a MD of 0.8 the CF would be:

$$CF = MB - MD = 0.99 - 0.8 = 0.19 \tag{3.15}$$

This gives a disproportionate weight to the single piece of negative evidence. Under the new system however, the CF is:

$$CF = \frac{MB - MD}{1 - min(MB, MD)} = \frac{0.99 - 0.8}{1 - min(0.99, 0.8)} = \frac{0.19}{0.20} = 0.95 \tag{3.16}$$

which is a more realistic result.

The following characteristics of MB and MD should also be noted:

$$MB[h, e] = 0 \text{ if e and h are independent or if e disconfirms h.} \tag{3.17}$$

$$MD[h, e] = 0 \text{ if e and h are independent or if e confirms h.} \tag{3.18}$$

One of the paradoxes encountered by the developers of Mycin was the fact that although an expert may agree with a hypothesis to a degree X, he does not necessarily agree with the hypothesis negation with a degree $1 - X$. In terms of the previously defined notation this is represented as:

$$CF[h, e] + CF[\bar{h}, e] \neq 1 \tag{3.19}$$

Using the rules developed so far, it can be shown that in fact the sum of these terms does not equal one, but zero, which agrees more closely with what one would expect on an intuitive level because it implies that if a piece of evidence supports a hypothesis it disfavors the negation of the hypothesis equally. This is true because $MB[h, e] = MD[\bar{h}, e]$ and, recalling the mathematical definitions of the measures of belief and disbelief:

$$CF[h, e] + CF[\bar{h}, e] = \frac{P(h \mid e) - P(h)}{1 - P(h)} + \frac{P(h) - P(h \mid e)}{1 - P(h)} = 0 \tag{3.20}$$

It is important that the developer of an expert system using certainty factors either weights the rules so that the sum of the mutually exclusive hypothesis does not exceed one, or otherwise normalizes them in some way.

It was mentioned earlier that some sort of approximation technique is needed in order to handle the case of multiple symptoms for a certain hypothesis. Remember that while it is possible to evaluate these using Bayes's theorem, large amounts of data storage are required. The confidence factor alleviates this problem because, since the confidence factors do not represent probability per se, approximations can be made using them that would not be otherwise valid.

There are several defining criteria that should be met by the confidence factors involving two or more symptoms if they are to be logically consistent and make sense intuitively. These criteria are broken down into four categories and presented below.

### 3.5 Defining Criteria of Mycin Confidence Factors

1. Limits - $MB[h, E]$ increases towards 1 as confirming evidence is found. Similarly, $MD[h, E]$ increases towards 1 as disconfirming evidence is found. In both cases, they reach 1 only if a piece of evidence either confirms or disconfirms the hypothesis completely. The combined evidence CF, $CF[h, E_+ \& E_-]$, should fall somewhere between the CF with the disconfirming evidence and the CF with the confirming evidence. In other words, $CF[h, E_-] < CF[h, E_+ \& E_-] < CF[h, E_+]$, where $E_+$ stands for confirming evidence and $E_-$ stands for disconfirming evidence.

2. Absolute confirmation/disconfirmation - If $MB[h, E_+] = 1$ then $MD[h, E_-] = 0$ and $CF[h, E] = 1$. Similarly, if the MD term equals 1, then $MB = 0$ and the $CF = -1$. It is also impossible for the MB and MD in a given rule to both equal 1.

3. Commutivity - The order that the evidence is discovered does not affect the final confidence factor.

$$MB[h, S1\&S2] = MB[h, S2\&S1] \quad (3.21)$$
$$MD[h, S1\&S2] = MD[h, S2\&S1] \quad (3.22)$$
$$CF[h, S1\&S2] = CF[h, S2\&S1] \quad (3.23)$$

4. Missing information - If the truth or falsity of a symptom in a hypothesis cannot be determined, then the various factors are determined by disregarding that symptoms' effect. If S2 is unknown:

$$MB[h, S1\&S2] = MB[h, S1] \quad (3.24)$$
$$MD[h, S1\&S2] = MD[h, S1] \quad (3.25)$$
$$CF[h, S1\&S2] = CF[h, S1] \quad (3.26)$$

These rules have a few important implications. Some of these are:

1. The MB of a hypothesis never decreases unless the associated MD goes to 1.
2. The MD of a hypothesis never decreases unless the associated MB goes to 1.
3. A CF of zero indicates either that the disconfirming and confirming evidence are equal, or that there is an absence of both confirming and disconfirming evidence.
4. If $E = E_+\&E_-$ then $CF[h, E]$ represents the CF of a rule covering only those cases wherein all the conditions of $E_+$ and $E_-$ are satisfied. Since it is impractical to write such rules, especially where more than two symptoms are involved, some method of combining the component symptoms from separate rules is needed.

## 3.6 Combining Mycin Confidence Factors

It was mentioned earlier that CFs cannot exceed 1 or go below $-1$. Because of this requirement, the designers of Mycin have implemented several rules to make

sure this does not occur. In addition, procedures dealing with disjunctions and conjunctions of hypotheses have been developed to cover those cases where a body of evidence could point to more than one hypothesis (logical AND), or where the evidence does not clearly indicate which of two functions may be true (logical OR). These combining functions are presented below.

1. Incrementally acquired evidence - This rule gives a method of modifying the existing MB or MD as new evidence comes to light, keeping in mind that the limits on the resulting CF must lie between -1 and 1. Under the old Mycin rules, the combining functions were defined as:

$$MB[h, S1\&S2] = \begin{cases} 0 & \text{if } MD[h, S1\&S2] = 1 \\ MB[h, S1] + MB[h, S2](1 - MB[h, S1]) & \text{otherwise} \end{cases} \tag{3.27}$$

$$MD[h, S1\&S2] = \begin{cases} 0 & \text{if } MB[h, S1\&S2] = 1 \\ MD[h, S1] + MD[h, S2](1 - MD[h, S1]) & \text{otherwise} \end{cases} \tag{3.28}$$

These rules were revised somewhat because under this system it was necessary to partition the evidence into positive and negative weights in order to preserve commutivity of evidence when the MBs and MDs were combined into CFs later. Under the new system Mycin simply stores the current CF value and combines it with new evidence as this becomes available. The new Mycin combining functions are [13]:

$$CF_{COMBINE}(X, Y) = \begin{cases} X + Y(1 - X) & X, Y > 0 \\ \frac{X+Y}{1-min(|x|,|y|)} & \text{one of } X, Y > 0 \\ -CF_{COMBINE}(-X, -Y) & X, Y < 0 \end{cases} \tag{3.29}$$

The results are different from the old system only in the case where the CFs to be combined are of opposite sign.

2. Conjunctions of hypothesis - The measure of belief (disbelief) in a conjunction of hypothesis is only as good as the belief (disbelief) in the hypothesis believed less (more) strongly.

$$MB[h1\&h2, E] = min(MB[h1, E], MB[h2, E]) \tag{3.30}$$

$$MD[h1\&h2, E] = max(MD[h1, E], MD[h2, E]) \tag{3.31}$$

3. Disjunctions of hypothesis - The measure of belief (disbelief) in a disjunction of a hypothesis is as good as the belief (disbelief) in the hypothesis believed more (less) strongly.

$$MB[h1 \vee h2, E] = max(MB[h1, E], MB[h2, E]) \tag{3.32}$$

$$MD[h1 \vee h2, E] = min(MD[h1, E], MD[h2, E]) \tag{3.33}$$

4. Strength of evidence - This rule applies to the case where it is not known for certain whether S is true but a CF is known reflecting the degree of belief in S. $MB'$ and $MD'$ refer to the degrees of belief in the hypothesis when the symptoms are known for certain.

$$MB[h, S1] = MB'[h, S1] \times max(0, CF[S1, E]) \tag{3.34}$$

$$MD[h, S1] = MD'[h, S1] \times max(0, CF[S1, E]) \tag{3.35}$$

As an example of the combining rules, consider the following example:

```
Symptom: Stalled motor

Cause: Valve stem binding (40%)
       Packing too tight  (60%)

Symptom: Leaking packing

Cause: Packing too loose (not too tight) (90%)
```

In this hypothetical situation, the expert has a valve symptom of "stalled motor". Sixty percent of the time this condition is caused by the valve stem binding, while forty percent of the time it is caused by the packing being too tight. On the basis of just this symptom therefore, the expert would conclude that "packing too tight" is the cause of the problem. Suppose however, that the expert also observes another symptom, namely that of "leaking packing" indicating that the packing is too loose. This changes the situation. Packing too loose can be interpreted as being a measure of disbelief in "packing too tight". The new confidence factor for "packing too tight" is then:

$$CF = \frac{60 - 90}{1 - min(60, 90)} = -75 \tag{3.36}$$

On the basis of the second symptom "valve stem binding" becomes the most likely candidate.

### 3.7 Problems with Mycin's Approximations

The four combining functions satisfy the defining criteria mentioned earlier, but there are some problems. First of all, it has been assumed throughout that the symptoms $S1$ and $S2$ are independent; this may not be true in actual practice. Secondly, the combining criteria always cause the MB or MD to increase regardless of the relationship between the new and prior evidence. However, the developers of Mycin point out in [11] that confirmation theory has little to do with probability theory in the numerical sense, and that the usefulness of the theory depends on its accuracy in a given context. In other words: if it works, use it .

In their original analysis, Shortliffe and Buchanan compared the CF's obtained

from simulated data using combining functions 3.27 and 3.28 and compared them with the exact CFs calculated from Equations 3.12 and 3.13. They found that the functions give good approximations to the numbers obtained using exact data. The greatest discrepancies occurred when the functions were applied many times to reach the final result [2] and when the pieces of evidence were strongly related (i.e., not independent) for the hypothesis under consideration. This is to be expected because the rules developed by Shortliffe and Buchanan tacitly assumed independence of symptoms.

For convenience, some important Mycin relations are summarized in Table 3.1.

[2] Reflecting Zadeh's postulate that the more steps involved in reaching a result, the "fuzzier" that result is (see Section 5.1).

Table 3.1: Some important Mycin relations

| Purpose | Equation |
|---|---|
| Conditional probability of $D$ given $E$. | $P(D \mid E) = \frac{P(D) \times P(E \mid D)}{P(E)}$ |
| Measure of belief. | $MB[h,e] = \frac{P(h) - P(h \mid e)}{P(h)}$ if $P(h \mid e) < P(h)$ |
| Measure of belief (alternate). | $MB[h,e] = \begin{cases} 1 & \text{if } P(h) = 1 \\ \frac{max[P(h \mid e), P(h)] - P(h)}{max[1,0] - P(h)} & \text{otherwise} \end{cases}$ |
| Measure of disbelief. | $MD[h,e] = \frac{P(h \mid e) - P(h)}{1 - P(h)}$ if $P(h \mid E) > P(h)$ |
| Measure of disbelief (alternate). | $MD[h,e] = \begin{cases} 1 & \text{if } P(h) = 0 \\ \frac{min[P(h \mid e), P(h)] - P(h)}{min[1,0] - P(h)} & \text{otherwise} \end{cases}$ |
| Combining confidence factors. | $CF_{COMBINE}(X,Y) = \begin{cases} X + Y(1-X) & X, Y > 0 \\ \frac{X+Y}{1 - min(\lvert x \rvert, \lvert y \rvert)} & \text{one of } X, Y > 0 \\ -CF_{COMBINE}(-X,-Y) & X, Y < 0 \end{cases}$ |

# 4. LEVEL5

An expert system will often be developed with the help of a tool or shell, which is a system designed to facilitate the rapid development of knowledge-based systems that address a specific class of problems. The shell is basically an "overlay" on an existing language and is designed to make program development easier. As mentioned in Section 1.4, Level5's platform language is Pascal. Other tools have been developed using Lisp and Prolog. The Level5 tool was developed by Information Builders, Inc. [14]. The language used by Level5 is known as PRL for Production Rule Language and is constructed in a straightforward IF-THEN structure; this makes program development and modification easy to perform.

## 4.1 Basics of Level5 Rules

The basic structure of a Level5 program consists of one or more goals, each of which may have sub-goals. The inference engine looks through the knowledge base to find a rule that has a goal as its conclusion. It then looks at the conditional statements that must be satisfied to meet this conclusion. Each of these conditional statements may be the conclusion of other rules, and these rules will also have conditions to satisfy. The knowledge base tries to determine the values of these statements by using the rules encoded in it or by querying the database. If it cannot determine the

values in this way, the knowledge base asks the user for the information it needs.

**4.1.0.3 PRL fact types.** There are four basic fact types in Level5 version 1.0. These are [14]:

1. simple fact-These are variables that have a true or false value. For example, the user might be asked:

   `"Will welding be performed that violates the pressure boundary?"`

   Where the entire clause

   `"Will welding...boundary?"`

   would be the simple fact. The user would respond by pressing the TRUE function key or the FALSE function key. When queried by the knowledge base for the value of a simple fact, the user can also be asked to provide his confidence in truth or falsity of the fact on a scale of 0 to 100. In this case, the simple fact will be assigned a value of true or false depending on if the user's answer was above or below a certain threshold assigned by the program developer.

2. attribute value-This type of variable designation is used whenever there is an antecedent part of a clause that can have one of several values. For example:

   ```
   valve operator maintenance IS operator replaced
   valve operator maintenance IS spring pack reworked
   valve operator maintenance IS limit switch maintained
   valve operator maintenance IS motor replaced
   ```

   Each one of the statements with the antecedent

   `valve operator maintenance`

   would most likely come from separate rules within the knowledge base. When the knowledge base needs to know the value of

```
valve operator maintenance
```

the user will be presented with a screen similar to:

```
What is: valve operator maintenance

operator replaced
spring pack reworked
limit switch maintained
motor replaced
```

The user would then select the desired answer, which would then become the value of the attribute value variable. Attribute value variables can take on multiple values simultaneously if they are declared using the reserved MULTI command.

3. string-This variable type is analogous to the string variables in other languages and can be any ASCII character. They must be declared at the beginning of the knowledge base by preceding the variable name with the reserved word STRING, otherwise the PRL compiler cannot distinguish them from numeric fact types. When the knowledge base queries the user for an answer (or checks the data base), the case of the characters in the answer is not important.

4. numeric-The numeric data type is used when the user needs to respond with a number that will be used in a subsequent mathematical of logical operation. If the number need not be used in one of these operations, it could just as easily be assigned as a string variable.

Subsequent versions of Level5 include the fact types of *time* and *interval* but since this system was developed using version 1.0 these fact types will not be covered.

## 4.2 CF and CONF Statements

Level5 has two different methods of handling uncertainty. These are the CONFIDENCE (CONF) statement and the CF statement. The way that Level5 handles

these two types of confidence is somewhat different. The CF statement will be examined first, then the CONF statement.

Before this it will be necessary to explain the term *OR-class* as it is referred to in the PRL language. The OR-class of a fact consists of all the rules that can conclude that fact via a THEN, AND, or ELSE PRL statement [15]. Rules in the same OR-class are evaluated following the order in which they appear in the knowledge base unless they have confidence factors in the conclusion of each rule. In this case, the rules are pursued in order of highest confidence value (CF). If all members of a fact's OR-class are considered and a conclusion cannot be reached, the conclusion is set to false if the fact is a simple fact. If the fact type is attribute value any values not assigned are set to false. CFs can only be used in the conclusion portion of a rule or in conjunction with an INIT or REINIT statement, which are two statements used to initialize a fact's value after the knowledge base has chained to another program.

The CONF statement is a confidence assignment operator that can be used from within the antecedent portion of a rule. Two examples of the use of the CONF statement would be:

SIMPLEFACT fact type:

```
RULE A
IF facta
THEN factb
AND CONF (factc) := 80
!
```

This would set the confidence value of *factc* to 80.

ATTRIBUTE-VALUE fact type:

```
AND CONF (valve operator maintenance IS motor
replaced) := 75
```

This would set the confidence of *motor replaced* to 75.

The CONF statement is treated differently from the CF statement. For example, in the following rule using the CF statement if the user answers false when queried about facta, then factb is set to false. this is true because in this case of a single rule the rule itself is a member of factb's OR-class of rules.

```
RULE A
IF facta
THEN conclusion CF 100
AND factb
```

Now consider two miniature knowledge bases, or *KBs*. One without the CONF statement, and one including it.

```
1. conclusion example 1
!
RULE A
IF    facta
THEN  conclusion example 1
AND   factb
!
RULE B
IF    factb
THEN  conclusion example 1
!
END
```

In this case, if the user answers false to facta, no further conclusions can be reached and factb's value is false. Now consider a second KB that is nearly identical, except that it uses the CONF statement:

```
1. conclusion example 2
!
RULE A
IF facta
THEN conclusion example 2
AND CONF (factb) := 100
!
RULE B
IF factb
THEN conclusion example 2
!
END
```

In this case if the user responds with false when queried for facta's value factb remains uninitialized (see Section 4.5 following) because the CONF assignment does not include the rule in the assigned fact's OR-class of rules. This is because factb is not concluded in that rule by a THEN, AND, or ELSE statement alone but rather by an AND CONF(factb) statement. The user will then also be queried for factb's value when RULE B is fired.

### 4.3 Calling External Programs

Level5 also has the useful ability to call external programs from within the knowledge base itself. This feature allows Level5 to call programs in the Fortran or C languages, for example. Through Fortran, Level5 can perform complex arithmetical functions or use dBase III data files to recover valve information. The first feature was used when incorporating the Mycin-like CFs into the expert system.

## 4.4 Parameter Passing Routine

In order to pass parameters between a Fortran program and Level5, an intermediate parameter-passing routine named ASCIIPRM must be used (see Appendix A). ASCIIPRM has routines for passing character, numeric, and logical data between Fortran and Level5. ASCIIPRM must be linked with any Fortran program that passes parameters to the Level5 program or receives parameters from Level5.

Level5 has the ability to call on any external programs with a .BAT, .COM or .EXE file designation in order to perform mathematical or data base functions or any other tasks the developer believes would be more conveniently handled by a program external to Level5. These programs are called using the ACTIVATE command followed by the name and the full path name of the desired program if the external program is not in the same directory as the PRL program. For example, if the program to be called is called PARMTEST, a sample rule calling this program would look something like the following:

```
RULE For testing parameter passing
IF Outputs displayed
AND  ACTIVATE c:\temp\PARMTEST.EXE
DISK  c:\temp\PARAM.DAT
SEND Real out
SEND Integer out
SEND String out
SEND Character out
SEND Boolean out
RETURN Real in
RETURN Integer in
RETURN String in
RETURN Character in
RETURN Boolean in
```

```
THEN Parameter test
AND DISPLAY Input parameters
```

The DISK command identifies the disk file name where the knowledge base and the external programs exchange ASCII information. The SEND command designates the variables the knowledge base sends to the external program. The RETURN command requests the values to be returned to the knowledge base from the external program. The ACTIVATE command also allows a program to be started by the external program activated by the knowledge base. This is done using the COMMAND function. For example the line:

```
AND  ACTIVATE c:\temp\PARMTEST.EXE COMMAND example
```

instructs the external program to activate the program *example*. Any .BAT, .COM or .EXE file can be called in this way.

The knowledge base will write the out-going data into the disk file designated by the DISK command using the standard ASCII format which is as follows:

Line 1: number of parameters to be sent or received in the file.

- "C" in column one for each line of character data with a space between the "C" and the data
- "N" in column one for each item of numeric data with a space between the "N" and the data.
- "L" in column one for each item of logical (true or false) data with a space between the "L" and the data.

The correspondence between PRL fact types and external fact types is given below:

Table 4.1: Correspondence between ASCII and PRL data types [14].

| External Fact Type | PRL Fact Type |
|---|---|
| boolean | SIMPLEFACT or ATTRIBUTE |
| real | NUMERIC |
| string | STRING |
| character | STRING |

It may seem strange that Level5 transfers the value of an ATTRIBUTE value fact as true or false. It does this because when an attribute value is sent to an external program, Level5 looks up the attribute's confidence value and compares it with the value's associated THRESHOLD statement. If the value has a CF greater than or equal to the threshold, then a value of true is sent to the external program; otherwise a false value is sent. Conversely, when an external program returns a boolean fact type, it is assigned a CF of 100 if the boolean is true, or a CF of 0 if the boolean is false.

The Level5 knowledge base must receive any data from external programs in the same format, and in the same order in which they are requested in the RETURN statement by the knowledge base. This is because values are sent and returned in the order they are declared in the knowledge base. As a result the names of the variables in the external programs do not have to match the names in the knowledge base, but the corresponding data types must be the same between Level5 and the external program.

## 4.5 Level5 and Mycin Confidence Factors Compared

Level5 uses a different approach to confidence factors than Mycin. In the Level5 system, confidence factors range from 0 to 100 with the exceptions noted below. If the CF of a fact is not known at a given point in time it is assigned a value of -1. This can happen as a result of three conditions. These are:

1. If the CF is part of a rule that has not yet been active.
2. It was not initialized.
3. Its value has not been received from an external program.

Finally, if the state of the variable is unknown the CF is assigned a value of $-2$. This occurs if the user responds by pressing the UNKN key when queried for the CF [15].

This is in contrast to the Mycin method, where values range between $-1$ and 1, with confirming evidence ranging between 0 and 1 while disconfirming evidence ranges between 0 and $-1$. Actually, the measure of disbelief (MD) has a value between 0 and 1, but acts as a negative number because of the formula for confidence factors in Mycin (see Equation 3.14).

The major difference however, is that under the Level5 system decreases in confidence are not possible. As pointed out in Section 4.2 confidence factors in Level5 are assigned on the basis of whichever rule concluding a given fact has the higher CF.

# 5. OTHER METHODS FOR DETERMINING CONFIDENCE

## 5.1 Fuzzy Logic

Fuzzy logic was pioneered by Lotfi A. Zadeh (among others) and was introduced to the world at large in his famous 1973 paper "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes" [16]. Zadeh's intention was to introduce a system of logic that would allow the modeling of systems that were generally considered to be too complicated for accurate modeling using conventional mathematical techniques. This is stated succinctly by the **principle of incompatibility** which says that as the complexity of a system increases, our ability to make precise and significant statements about its behavior diminishes [16]. Zadeh's original concept was to apply the techniques of fuzzy logic to some of the "softer" sciences such as sociology, politics, and economics, where the rigorous mathematical techniques that serve physics and engineering so well have largely failed due to the much greater complexity and, at the same time, vagueness inherent in these sciences. Because fuzzy logic employs linguistic variables in the form of IF-THEN rules, it is classed under the heading of artificial intelligence.

Fuzzy logic differs from conventional logic theory mainly because an element of a fuzzy logic set can have partial membership in a set, i.e., it need not be completely "in" a set. A degree of membership is generally characterized by a number between

zero and one, depending on how confident the developer is that a given value fits a certain description. Systems using fuzzy logic do not need to explicitly specify the relationships between every single input and output variable via differential equations or other mathematical relationships.

Fuzzy logic presents an interesting use relating to confidence factors in expert systems. Instead of asking the users of the system to input their confidence in a fact as a precise number, they could instead choose from a list of terms such as "positive", "fairly certain", etc. These linguistic variables could then be mapped onto an array of values similar to Equation 5.21 in Section 5.1.2. This would avoid having the users input their confidence in a fact, leaving it the job of the expert system developer to assign the various confidence values. This would make the expert system more transparent to the end user.

### 5.1.1 Fuzzy Logic Rules

The rules of fuzzy logic have a certain resemblance to Boolean algebra, upon which they are based. However, there is an important difference; namely, that the values of the variables are not strictly limited to on/off or $+/-$ values. This is in keeping with the fact mentioned earlier which stated that a variable can be partially in a set without being completely in it. The set of all the possible values a fuzzy variable can take on is referred to as a "universe of discourse" denoted by the symbol "U". Each member is given a value, usually between zero and one, indicating its degree of membership in the set. Thus the expression:

$$U = 1/1 + 0.8/2 + 0.5/3 \tag{5.1}$$

states that the set member **1** is completely in the set, the member **2** is 80% in the set, and so on. A typical rule in a fuzzy logic controller might look like: "IF $x$ is **small** THEN $y$ is **very large**."where small is defined as:

$$small = 0.5/0.4 + 0.7/0.5 + 1/0.6 + 0.7/0.7 + 0.5/0.8 \tag{5.2}$$

Thus if $x$ were 0.6 it would have a 100% membership in **small**, and correspondingly $y$ would be 100% "**very large**". If $x$ was not 100% **small** then $y$ would not be 100% **very large** but would have partial membership in that particular set. Generally, there would be no change in the value of **very large** until a certain threshold value was reached [16].

A fuzzy relation, denoted by **R**, is defined by the expression:

$$R = \int_{X \times Y} m_R(x, y)/(x, y) \tag{5.3}$$

where the degree of membership is given by the numerator and the denominator characterizes the combinations between the members of the sets $X$ and $Y$. For example, if

$$X = \{a, b\} \tag{5.4}$$

$$Y = \{c, d\} \tag{5.5}$$

then if a fuzzy relation between the members of $X$ and $Y$ called "relationship" was defined it might look like:

$$relationship = 0.9/(a, c) + 0.7/(a, d) + 0.3/(b, c) + 0.2/(b, d) \tag{5.6}$$

Alternatively, if $R$ is a relation from $X$ to $Y$, and $S$ is a relation from $Y$ to $Z$, then the **composition** of $R$ and $S$ is defined to be:

$$R \circ S = \int_{X \times Z} \vee(\mu_R(x,y) \wedge \mu_S(y,z))/(x,z) \tag{5.7}$$

where $\vee$ is the max function and $\wedge$ is the min function defined such that:

$$a \vee b = \begin{cases} a & \text{if } a \geq b \\ b & \text{if } b > a \end{cases} \tag{5.8}$$

$$a \wedge b = \begin{cases} a & \text{if } a \leq b \\ b & \text{if } b < a \end{cases} \tag{5.9}$$

As an example of the max function, consider the matrices defined below:

$$\begin{bmatrix} 0.3 & 0.8 \\ 0.6 & 0.9 \end{bmatrix} \begin{bmatrix} 0.5 & 0.9 \\ 0.4 & 1.0 \end{bmatrix} = \begin{bmatrix} 0.32 & 0.8 \\ 0.36 & 0.9 \end{bmatrix} \tag{5.10}$$

The matrix products would be taken in the normal way by multiplying the corresponding row entry by the appropriate column entry but instead of adding the products to get the resulting final matrix entry, the maximum of the two products formed for each entry is used, for example:

$$entry(1,1) = (0.3)(0.5) + (0.8)(0.4) = 0.15 + 0.32 \rightarrow 0.32 \tag{5.11}$$

$$entry(2,1) = (0.6)(0.5) + (0.9)(0.4) = 0.3 + 0.36 \rightarrow 0.36 \tag{5.12}$$

and so on. Some other important relations are defined in Table 5.1.

Table 5.1: Fuzzy mathematical relations

| | |
|---|---|
| $\neg A = (1 - m(y))/y$ | compliment (not) |
| $A + B = (m_A(y) \vee m_B(y))/y$ | logical or/union |
| $A \cap B = (m_A(y) \wedge m_B(y))/y$ | logical and/intersect |
| $AB = m_A(y)m_B(y)/y$ | product |
| $CON(A) = A^2$ | concentration or very |
| $DIL(A) = A^{0.5}$ | dilation |
| $A^{1.25}$ | plus A |
| $A^{0.75}$ | minus A |

As an example of some of these rules, consider a universe of discourse defined as:

$$U = 1 + 2 + 3 + ... + 10 \text{ and,} \tag{5.13}$$

$$Y = 0.4/1 + 0.7/2 + 0.6/3 + 0.2/6 \tag{5.14}$$

$$X = 0.5/1 + 0.4/2 + 1/5 + 1/4 \tag{5.15}$$

then

$$X \text{ and } Y = 0.5/1 + 0.7/2 + 0.6/3 + 1/4 + 1/5 + 0.2/6, \tag{5.16}$$

while

$$X \text{ or } Y = 0.4/1 + 0.4/2 \tag{5.17}$$

As an example of evaluating a linguistic expression according to the fuzzy logic rules, consider the expression **not very small** or, as it can also be written ¬(**very small**) where we will define small as being:

$$\textbf{small} = 1/1 + 0.8/2 + 0.6/3 + 0.4/4 + 0.2/5 \tag{5.18}$$

then very small would be:

$$\textbf{very small} = 1/1 + 0.64/2 + 0.36/3 + 0.16/4 + 0.04/5 \tag{5.19}$$

where the degrees of membership have been squared as per the definition given above. Finally,

$$\neg\ \textbf{very small} = (1-1)/1 + (1-0.64)/2 + (1-0.36)/3 + (1-0.16)/4 + (1-0.04)/5 \tag{5.20}$$

Problems can sometimes arise in fuzzy control systems when the final control action must be chosen. The output of any fuzzy algorithm is a fuzzy set itself, and so various grades of membership are assigned to the members of this fuzzy set. There are two methods of determining the appropriate "defuzzified" output to be sent to the controller. If one of the members has a degree of membership in excess of the others then no problem occurs,and its value is sent to the controller, but if there are two values that have a nearly identical degree of membership (or if all values have the same degree of membership, and a "plateau" occurs), then the appropriate control action must somehow be chosen from among them, and the other method is chosen. In this method, the "center of mass" or weighted average of the individual values is taken.

### 5.1.2 Examples of Successful Fuzzy Controllers

Mamdani and Assilian [17] have implemented an interesting application of a fuzzy logic controller. Their paper describes a device designed to control a steam engine and boiler combination. The controller used two algorithms to govern the system: one to respond to the heat change of the steam, the other to respond to changes in the throttle position at the input to the engine. The steam engine and boiler combination had two inputs (heat input to the boiler and throttle opening at the input of the engine cylinder), and two outputs (steam pressure in the boiler and the speed of the engine). The fuzzy controller used six fuzzy variables to accomplish its task. These were:

1. PE = Pressure error or difference between present value of pressure and the set point.
2. SE = Speed error or difference between present value of speed and the set point.
3. CPE = Delta PE or difference between PE at time t and $t-1$.
4. CSE = Delta SE or difference between present SE and SE at $t-1$.
5. HC = Heat change.
6. TC = Throttle change.

These variables could each take on seven values corresponding to varying membership in seven fuzzy subsets. These subsets ranged from PB (Positive Big) to NB (Negative Big). The results of the fuzzy controller were compared to those obtained using a fixed digital controller (see Figure 5.1, p. 60).

The results seem to indicate better performance by the fuzzy controller as shown by the overshoot evidenced by the digital controller in the one case, and the relatively long time it took to reach the set point in comparison to the fuzzy controller in the other. The fuzzy controller worked so well that the original idea of designing a learning fuzzy controller was discarded as being unnecessary.

Feeley and Johnson [18] have demonstrated an interesting application of a fuzzy logic controller to a pressurized water reactor. The reactor was modeled using a set of nine non-linear coupled differential equations. The nine state variables included in their analysis were:

1. Neutron density.
2. One group of delayed neutrons.
3. Control rod position.
4. Fuel pin temperature.
5. Hot leg temperature.
6. Cold leg temperature.
7. Steam generator saturation temperature.
8. Steam flow control valve position.
9. Turbine speed.

The two controller output (plant input) variables were the voltages applied to the control rod and steam flow control valve actuators. The controller input (plant

output) variables were average primary coolant temperature and turbine speed. The power removed through the turbine was treated as a controller disturbance input. In this case, each input or output was converted into one of five states ranging from "positive big" to "negative big". The controller used two relational matrices (**R**'s); one for the control rod actuator signal and one for the steam flow valve signal. The output variables were defuzzified using a center of area (mass) technique. Figures 5.2 and 5.3 (pp. 61-62) show that the controller responded quite well to changes in the set point. Figure 5.2 shows the controllers response to a set point change in $T_{av}$ of plus and minus five degrees, while the turbine speed $S_t$ was held constant. Figure 5.3 shows the controller response to simultaneous $T_{av}$ and $S_t$ set point changes, and again the response was good. Figure 5.4 (p. 63) illustrates the disturbance rejection capabilities of the controller as the turbine load was varied between $+/-2.5$ MW. As shown, the controller was able to maintain $T_{av}$ to within one degree and $S_t$ to within one rev./sec.

Roglans-Ribas [19] has implemented a fuzzy controller to perform small power level changes in a nuclear reactor. The controller utilized for this purpose had two input variables:

1. DEM - Delta power, or difference between power demanded and actual power.

2. CHE - Rate of change of power level.

and one output variable, CONR - Reactivity inserted by the controller.

The fuzzy set for power difference had five members from big positive to big negative. The sets for control reactivity and change of power were divided into five and three sets, respectively. Each fuzzy set was divided into seven categories from

negative to positive three, and the degrees of membership for each of the fuzzy sets were assigned on that basis. For example, the fuzzy set for power difference, DEM, would look like:

| | **−3** | **−2** | **−1** | **0** | **1** | **2** | **3** |
|---|---|---|---|---|---|---|---|
| **BIP** | 0.0 | 0.0 | 0.0 | 0.1 | 0.4 | 0.8 | 1.0 |
| **SMP** | 0.0 | 0.0 | 0.0 | 0.3 | 1.0 | 0.5 | 0.1 |
| **ZER** | 0.0 | 0.1 | 0.3 | 1.0 | 0.3 | 0.1 | 0.0 |
| **SMN** | 0.1 | 0.5 | 1.0 | 0.3 | 0.0 | 0.0 | 0.0 |
| **BIN** | 1.0 | 0.8 | 0.4 | 0.1 | 0.0 | 0.0 | 0.0 |

(5.21)

thus if an input had a fuzzified value of 2, its membership in BIP (Big Positive) would be 0.8, its membership in SMP (Small Positive) would be 0.5 and so on.

The fuzzified values to be used in the linguistic controller were determined (for the case of power difference) by expressing the DEM change as a percentage of the original power. A unit of DEM would then be 0.1% of the power difference, for example. The overall performance of this controller seemed to be very good, with rapid approaches to the set point being common. Only very small over- or undershoots were experienced. The only problem seemed to be in the fact that once the final power was inside of a certain percentage of its final value, the controller would allow it to oscillate in a "dead band". This was because the power was so close to its final value that the controller determined that no further control action was required. The assignment of fuzzified values to the controller via the fuzzy algorithm was not fine enough to distinguish any difference in values at the output. This could easily be remedied by further "quantizing" the possible fuzzy output values, however.

### 5.1.3 Expert Systems Uses of Fuzzy Logic

Fuzzy logic presents a unique opportunity to model systems that are either too complex or too tedious to model using conventional mathematical techniques. By using a series of fairly simple rules, fuzzy logic permits the construction of linguistically-based rules of an IF-THEN format. As noted in Section 5.1 this makes expert systems easier for the average person to use.

The various control systems implemented using fuzzy algorithms have proven both its usefulness and effectiveness as a systems controller. In fact, fuzzy controllers seem to be quite robust, in many cases out-performing their digital counterparts. In the future, we will no doubt begin to see fuzzy logic applied to the more esoteric sciences such as economics and psychology as well.

## 5.2 Dempster-Shafer Belief Calculus

The Dempster-Shafer belief calculus is another method of finding confidence factors. It was not used within this expert system; however because it offers an interesting alternative to CFs, it will be covered briefly here. A suggestion for future work would be to compare the Dempster-Shafer method for determining confidence factors with the Mycin method to see if there are any significant differences.

### 5.2.1 Dempster-Shafer Calculus Compared to Baysian Statistics

Dempster-Shafer calculus is similar to Baysian statistics in that both assign a number between zero and one to reflect the degree of belief in a fact or conclusion. However, it is different from conventional probability theory in that the belief in an event occurring $P(A)$ plus the belief in an event not occurring $P(\bar{A})$ need not sum

to one. In fact, the belief in both may be zero. For example, an expert may assign a belief of zero to "valve packing is too tight" while simultaneously assigning a value of zero to his belief that it is not too tight because the evidence to date is inconclusive. Section 5.2.2 will show how Dempster-Shafer calculus handles a state of ignorance in a situation.

### 5.2.2 The Belief Function

In Dempster-Shafer calculus, the developer begins with a set of possible events here denoted by $E$. Suppose our set of possible events includes three elements as in [20]. The set is denoted by:

$$E = \{B, J, S\} \tag{5.22}$$

The belief function is not defined over this set. It is defined over the power set of $E$, which is defined as the set consisting of all the subsets of $E$ and is denoted by $2^E$. This set contains the following elements:

$$2^E = \{\emptyset, \{B\}, \{J\}, \{S\}, \{B, J\}, \{B, S\}, \{J, S\}, \{B, J, S\}\} \tag{5.23}$$

where $\emptyset$ is the empty set.

The belief function maps the power set of events into the range $[0, 1]$. The notation for the belief function is:

$$Bel : 2^E \rightarrow [0, 1] \tag{5.24}$$

The belief function must also satisfy the following three conditions:

$$Bel(\emptyset) = 0 \tag{5.25}$$

$$Bel(E) = 1 \tag{5.26}$$

$$Bel(A_1 \cup ... \cup A_n) \geq \sum_i Bel(A_i) - \sum_{i<j} Bel(A_i \cap A_j) + ... + (-1)^{n+1} Bel(A_1 \cap ... \cap A_i) \tag{5.27}$$

The third condition provides a constraint over building up a belief for a set of events. As an example, suppose that belief values were assigned to $B$ and $S$. The third condition simply states:

$$Bel(\{B\}) + Bel(\{S\}) \leq Bel(\{B, S\}) \tag{5.28}$$

or, in other words, the belief that the symptom is in the set $\{B, S\}$ can be no less than the sums of the individual beliefs.

A belief function that represents a state of ignorance is given by:

$$Bel(A) = \begin{cases} 0 & \text{if } A \neq \\ 1 & \text{if } A = E \end{cases} \tag{5.29}$$

This is referred to as a *vacuous belief function.* It indicates that while it is known that the correct answer is somewhere in the total set of possibilities, there is no similar belief regarding any of the subsets of the total set. The vacuous belief function allows a distinction to be made between lack of belief (or ignorance) and disbelief.

### 5.2.3 Probability Masses

Dempster-Shafer belief calculus does not ask the user to assign values to belief functions over a set of alternatives, because this would require the user to constantly keep in mind the constraints of Equations 5.25 through 5.27. Instead, these values are built from basic probability assignments that can be determined by the user [20].

A basic probability assignment, $m$, is like a belief function in that it maps the power set of the set of alternatives into the range $[0, 1]$. It is different, however, because the only requirements for the basic probability assignment are:

$$m(\emptyset) = 0 \tag{5.30}$$

$$\sum_{A \subset S} m(A) = 1 \tag{5.31}$$

The first condition states that no probability assignment should be made to the empty set while the second states that the sum of all the assignments over all the subsets equals one. $m$ is sometimes referred to as the *probability mass* of subset $A$. The relationship between the probability masses assigned to the subsets of the set of possible events and the belief value assigned to each subset is given by:

$$Bel(A) = \sum_{B \subset A} m(B) \tag{5.32}$$

In other words, the belief in a subset $A$ is the sum of the probability mass $m(B)$ assigned to all proper subsets $B$ of $A$.

As an example of the uses of probability masses in the determination of belief values, suppose the probability masses listed in Table 5.2 have been assigned by the user.

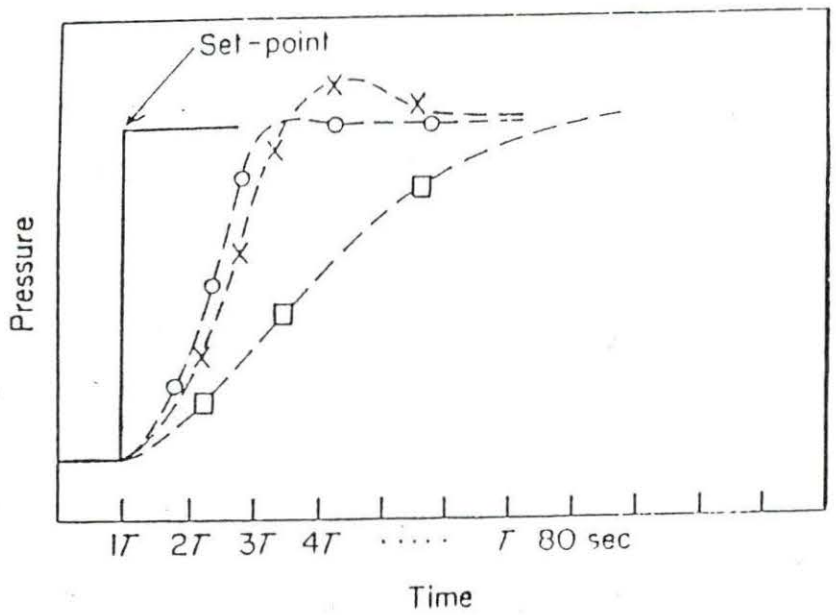


Figure 5.1: Comparison Between a Fuzzy and Digital Controller for a Steam Engine. (Digital controller: × ,□ . Fuzzy controller: ⊙)

Table 5.2: Example of assignment of probability masses

| Event | Mass |
|---|---|
| {B} | 0.1 |
| {J} | 0.2 |
| {S} | 0.1 |
| {B,J} | 0.1 |
| {B,S} | 0.1 |
| {J,S} | 0.3 |
| {B,J,S} | 0.1 |

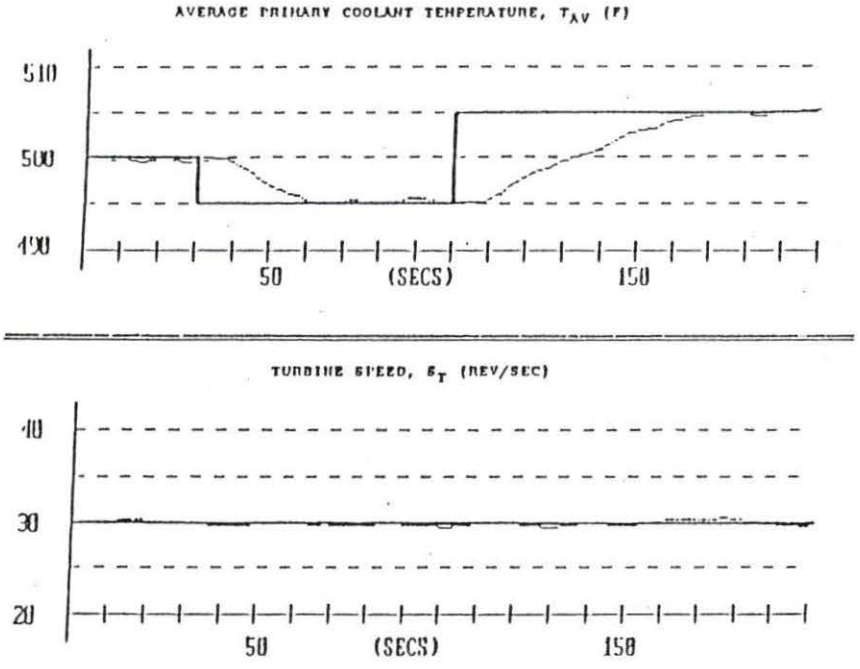


Figure 5.2: Response of a Fuzzy Nuclear Power Plant Controller to a Change in Setpoint $T_{av}$

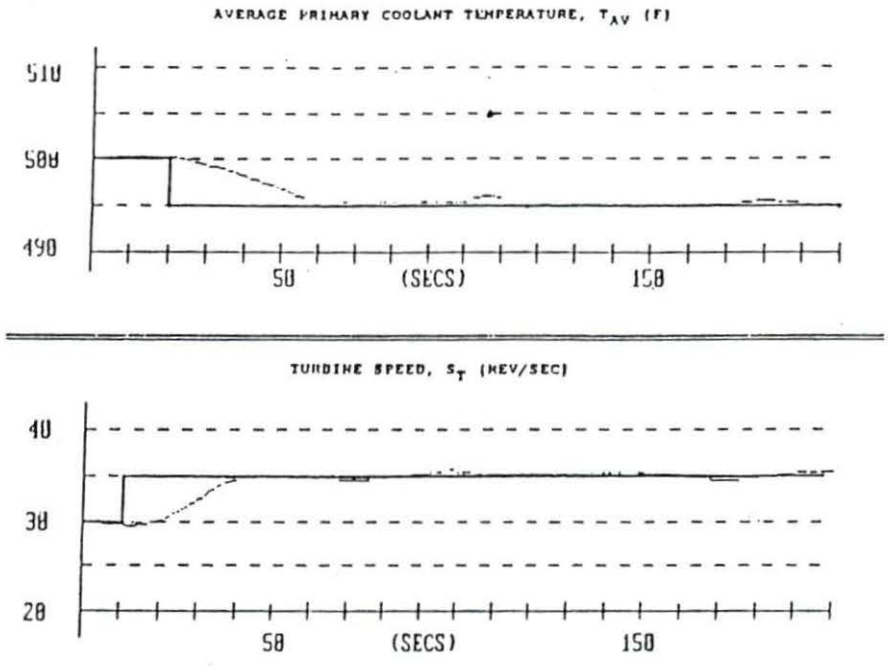


Figure 5.3: Response of a Fuzzy Nuclear Power Plant Controller to Simultaneous Changes in $T_{av}$ and $S_t$

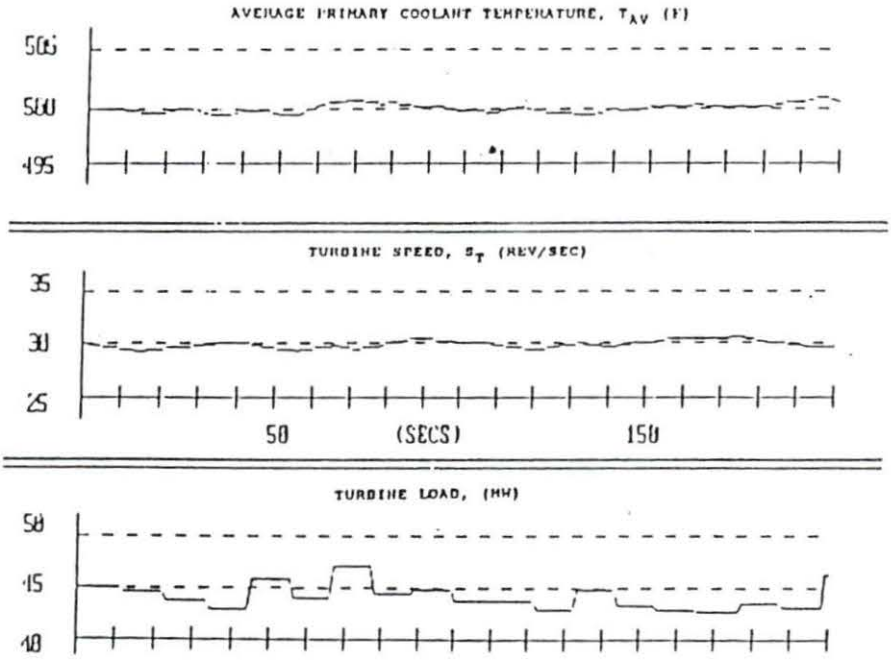


Figure 5.4: Rejection of Turbine Load Disturbance by a Fuzzy Nuclear Power Plant Controller

The probability mass is not the same as probability because it is known, for example, that the probability that one of the alternatives $\{B, J, S\}$ is true is one but the probability mass assigned to $\{B, J, S\}$ is only 0.1. Also note that the sum of the probability masses is one as required by Equation 5.31. The probability masses can be used directly to determine that the degree of belief in $B$ is 0.1, in $J$ is 0.2 and in $S$ is 0.1 because these three sets have no subsets. But if the degree of belief in $\{B, J\}$ is required then Equation 5.32 must be used so that:

$$Bel(\{B, J\}) = m(\{B\}) + m(\{J\}) + m(\{B, J\}) = 0.1 + 0.2 + 0.1 = 0.4 \qquad (5.33)$$

Using Equation 5.32 the set of belief values is given in Table 5.3.

Table 5.3: Probability masses and corresponding belief functions

| A | {B} | {J} | {S} | {BJ} | {BS} | {JS} | {BJS} |
|---|---|---|---|---|---|---|---|
| m(A) | 0.1 | 0.2 | 0.1 | 0.1 | 0.1 | 0.3 | 0.1 |
| Bel(A) | 0.1 | 0.2 | 0.1 | 0.4 | 0.3 | 0.6 | 1.0 |

### 5.2.4 Dempster's Rule

Dempster's rule allows several combined belief functions to form a new single belief function. In order to use Dempster's rule, the user must have two or more belief functions defined over the same set of possible events but based on different evidence. These differing belief functions could represent the set of valve problems based on different sets of symptoms, for example.

If we refer to one set of symptoms as $A$ and another set of symptoms as $B$, then Dempster's rule is written as [20]:

$$m(A) = \frac{\sum_{A_i \cap B_j = A} m_1(A_i) m_2(B_j)}{1 - \sum_{A_i \cap B_j = \emptyset} m_1(A_i) m_2(B_j)} \text{ Note: } \sum_{A_i \cap B_j = \emptyset} m_1(A_i) m_2(B_j) < 1 \tag{5.34}$$

Dempster's rule adds up the intersections of the probability masses for the symptoms and normalizes the result to produce a new probability mass. The numerator represents the summation of the intersection of the probability masses for sets $A$ and $B$. The "$B_j = A$" condition indicates that sets $A$ and $B$ are taken over the same set of symptoms. Note the condition on Equation 5.34. This term represents the amount of probability mass distributed among subsets that have empty intersections. If this term equals one, it means that the two subsets have nothing in common. This term is independent of the subset $A$ and only needs to be determined once for a given $A$ and $B$.

# 6. VALVE DIAGNOSIS AND MAINTENANCE PLANNING EXPERT SYSTEM

## 6.1 Introduction

This chapter describes an expert system being developed by the Nuclear Engineering Department of Iowa State University in cooperation with the engineering staff of Duane Arnold Energy Center in Palo, Iowa. The purpose of the expert system is to assist the engineers in valve problem diagnosis and maintenance. The purpose of this research is to demonstrate the applicability of expert system technology to nuclear power plant operation [21],[22]. As previously mentioned, the expert system tool Level5 is being used to develop the program.

This chapter provides a status report on the project and describes the features of the program. Following this is a description of the module added to allow incorporation of multiple-symptom cases using a Mycin-like CF system.

## 6.2 Valve Maintenance Planning

Valve maintenance planning at a complex installation like a nuclear power plant is a time-consuming task because of the large number of valves involved and the high level of quality required. Valve maintenance planning typically involves the following steps:

1. Diagnosing the problem.

2. Prescribing maintenance. and,

3. Determining the factors which affect the maintenance task.

The maintenance personnel must consult many sources to collect the information necessary for analyzing of the problem. These sources include the operations staff who reported the problem, a database that contains valve information, procedures and guides for determining maintenance requirements, parts inventory lists, and so on. The engineer must also coordinate the required maintenance with previously scheduled maintenance plans, testing requirements, and plant operation schedules. The development of a sufficiently sophisticated knowledge-based system to assist in maintenance planning would result in a substantial time savings, chiefly by automating information gathering and routine decision making [1].

## 6.3 Features of the Expert System

The knowledge base currently has three main sub-systems. These are diagnosis, prescription of suggested maintenance, and maintenance planning. These features are shown in Figure 6.1.

There are four main modules in the valve maintenance program. IEVALVE.PRL is the main calling program. It asks the user whether valve maintenance planning, valve diagnosis, diagnosis and maintenace planning, batch testing, or database editing functions are required. It also retrieves information about the valve from the database. Depending on the user's answer IEVALVE.PRL chains to one of the other three programs.

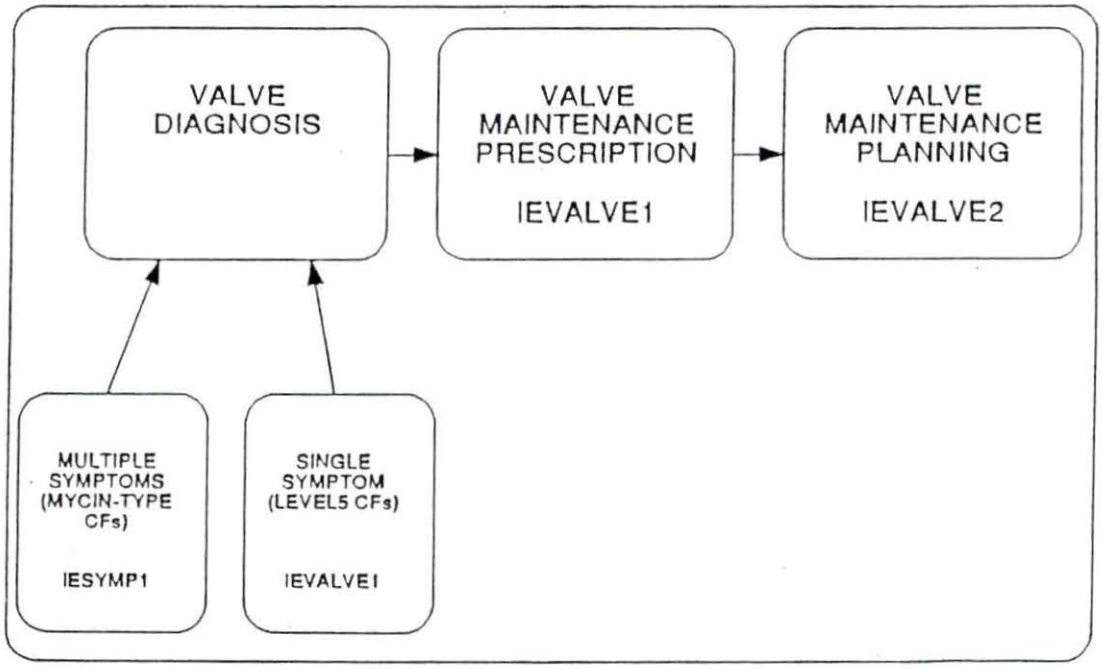


Figure 6.1: Block Diagram of Valve Maintenance Knowledge Base

IEVALVE1.PRL is the diagnosis portion of the knowledge base. It contains twelve common valve symptoms for the valve operator and valve body. Based on the symptoms selected by the user, the system returns with a diagnosis for each symptom.

IEVALVE2.PRL is the maintenance planning portion of the knowledge base. It can be entered in either of two ways. If maintenance planning alone is selected from the main menu, the user is presented with three separate lists dealing with the valve body, valve operator, and motor control center. The user then selects which maintenance actions were performed. See Figures 6.2 through 6.4. The expert system returns with a CMAR (Corrective Maintenance Action Request) form detailing which post-maintenance tests need to be performed (see Figure 6.11 in Section 6.4).

If diagnosis and maintenance planning is selected then the expert system generates the CMAR form based on the maintenance required to repair the problem based on its diagnosis. Currently, there are no diagnoses present for motor control center problems, so only maintenance planning alone can be done for it, not diagnosis.

IESYMP1.PRL is the portion of the program that takes multiple symptom diagnoses into account. After the maintenance task and the valve ID have been entered, the user is asked whether single or multiple symptom diagnosis is required. If single symptom diagnosis is selected, the knowledge base chains to IEVALVE1.PRL. If multiple symptom diagnosis is chosen the knowledge base chains to IESYMP1.PRL.

### 6.3.1 Diagnosis

At the beginning of the session the user is asked which function is desired, and the ID number of the valve in question. If diagnosis is chosen, the user is asked which

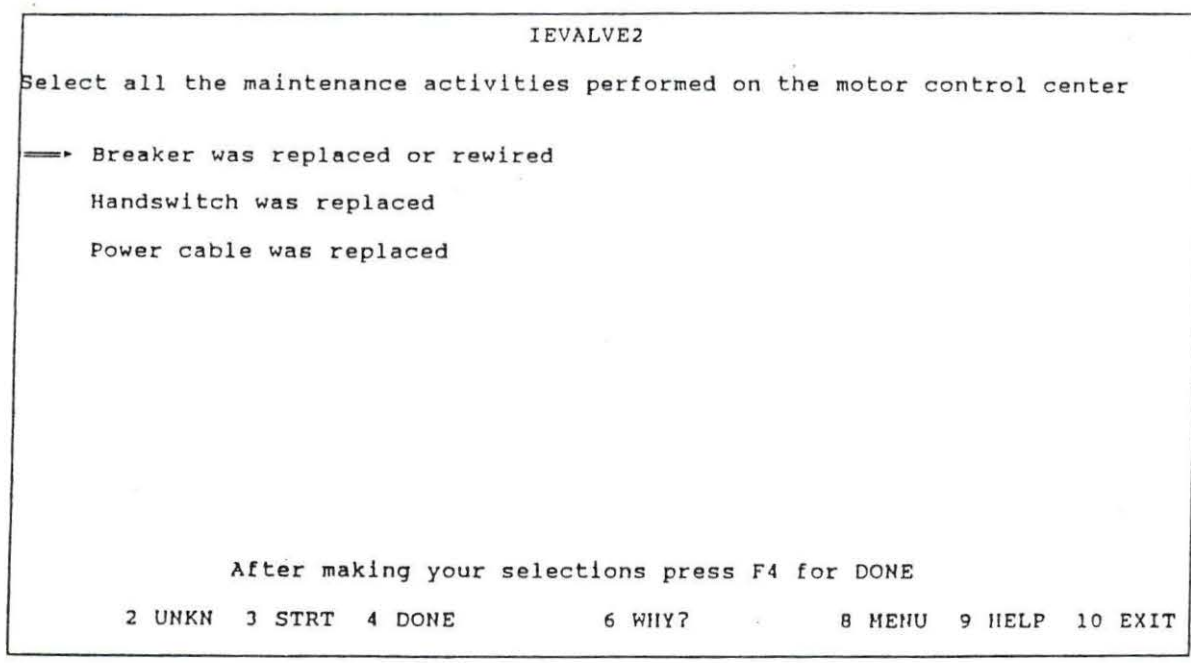


Figure 6.2: Selection of Valve Body Maintenance Actions

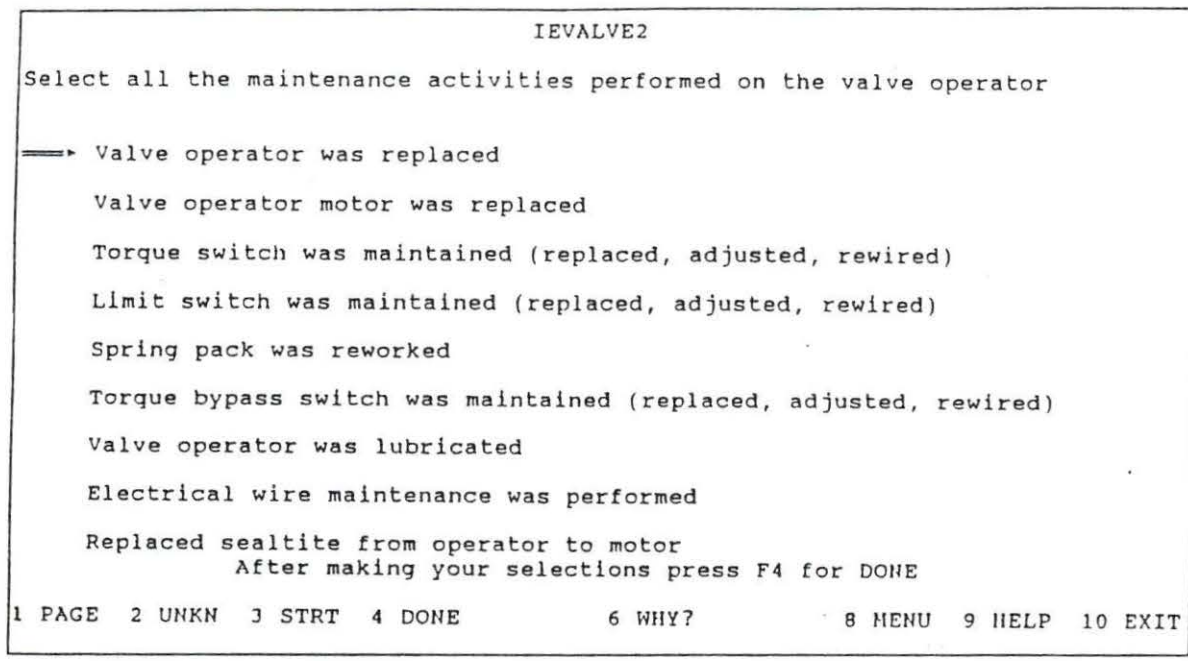


Figure 6.3: Selection of Valve Operator Maintenance Actions

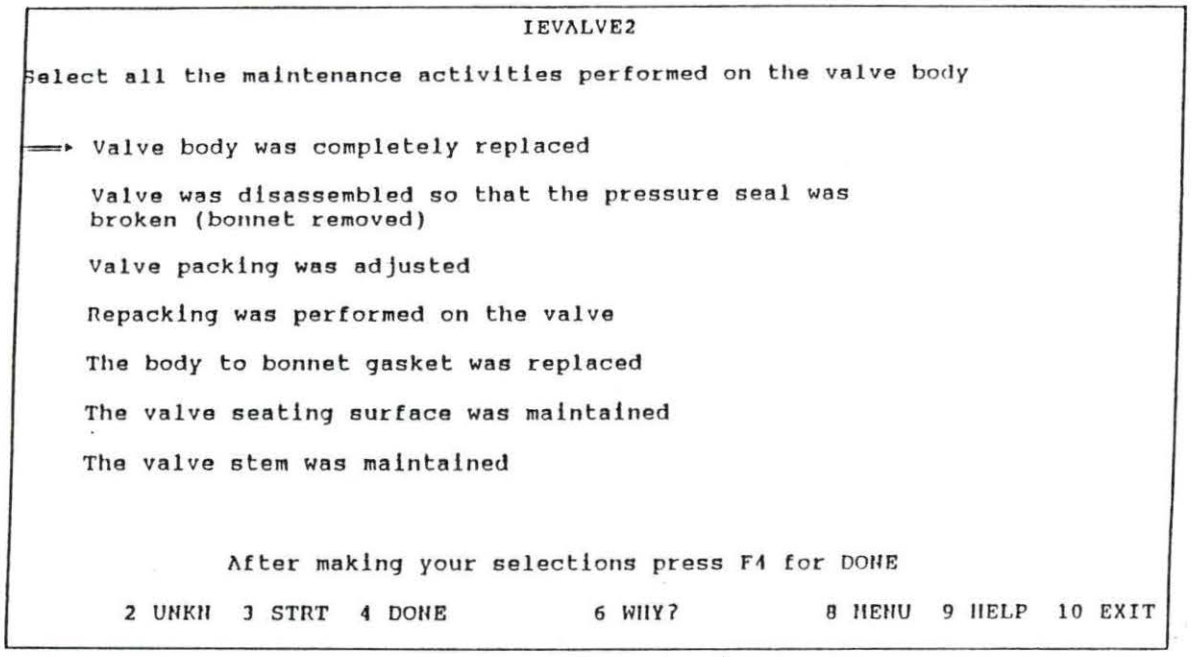


Figure 6.4: Selection of Motor Control Center Maintenance Actions

part of the valve is malfunctioning and the symptoms exhibited by that part. The knowledge base will then tell the user the probable cause of the malfunction and the steps necessary to correct the problem. More than one area of the valve and more than one symptom can be selected per session. The reader is referred to Section 6.4 for an example session with the expert system under single-symptom conditions.

Note that under single-symptom conditions, there is a 100% confidence that a certain problem is being caused by a given symptom. Under the multiple-symptom case discussed in Section 6.5 a given symptom can point to more than one possible diagnosis. Each diagnosis has varying degrees of confidence based on the symptoms present.

### 6.3.2 Maintenance Planning

After diagnosis, the post-maintenance testing requirements can be determined. The user can select the valve body, the valve operator, the motor control center, or any combination of the three as possible areas to be maintained. The user then selects the maintenance to be performed and the knowledge base recommends which tests on the CMAR form must be performed to verify that the valve operates correctly after the maintenance is completed. The fields describing radiation work permit (RWP), primary containment (pri cont), cleanliness control procedure (CCP) and NPRDS have recently been added.

The user could also request combined diagnosis, prescription of maintenance, and determination of post-maintenance information. In this case, the expert system automatically determines the areas to be repaired and the maintenance to be performed on the basis of the diagnosis it makes.

Finally, the batch testing mode allows test cases with known answers to be compared with a subsequent batch of results. This allows the user to check the effects of changes on the knowledge base.

**6.3.2.1 Verification Testing** The area of verification testing has recently been added to the program. Rules in the knowledge base now enable the user to determine whether one of four so-called "VT" tests need to be performed. The appropriate tests are determined on the basis of valve size, whether welding has been performed, whether the valve has pressure-retaining bolting, etc.

The program file IEVTTEST.PRL, which is incorporated in IEVALVE2.PRL, contains rules dealing with verification testing of motor-operated valves. These are additional tests that must be performed if maintenance is done on certain valves that have a diameter greater than one inch. There are four VT tests that may have to be performed. These are:

1. VT1 test- This is a visual inspection of the valve looking mainly for corrosion and damaged threads on the valve body. It is performed if the valve has been disassembled and if it has pressure-retaining bolting. This includes cases of valve replacement on either the valve body or operator. It also includes cases in which valve body maintenance includes repairing or replacing the seating surface. It is also required if the valve operator is replaced.
2. VT2 hydro test- This test is done by pressurizing the valve at a high pressure. It is done whenever welding has been performed that violates the valve's pressure boundary.
3. VT2 pressure test- This test is done by pressurizing the valve at lower pressure than the VT1 test. It is done whenever the pressure boundary has been violated by something other than welding, such as valve disassembly.
4. VT3 test- This is a visual inspection looking for internal signs of cracks and corrosion as well as evidence of steam cuts, which occur when steam causes

erosion of the valve threads. It is required if the valve diameter is greater than four inches and the valve has been disassembled.

The reader is referred to Appendix B for a listing of the rules used to prescribe verification testing.

## 6.4 Example Session

The features of the expert system can best be illustrated by an example. Upon entering the program, the user is presented with a screen similar to Figure 6.5 that asks which of the functions shown there the user wishes to perform.

Suppose the user chooses to diagnose, prescribe, and plan maintenance tasks and selects both the valve body and operator as the locations of the diagnosis to be made. See Figure 6.6.

He would then select both by positioning the cursor next to "valve operator," then next to "valve body." The user could then select which problems were present in each area. For example, the user might indicate that there is a problem with both the valve operator and body and that the valve operator exhibits excessive handwheel effort while the valve body may have both a binding valve stem and leakage between the valve disk and seating area. See Figures 6.7 and 6.8. The knowledge base would then recommend actions to correct these problems. See Figures 6.9 and 6.10 for the system's diagnosis.

The system would then generate a form similar to that shown in Figure 6.11 to aid in the completion of the CMAR form. The form of the output was chosen to mimic that of the CMAR report so the staff could more easily fill in the CMAR from the print out.

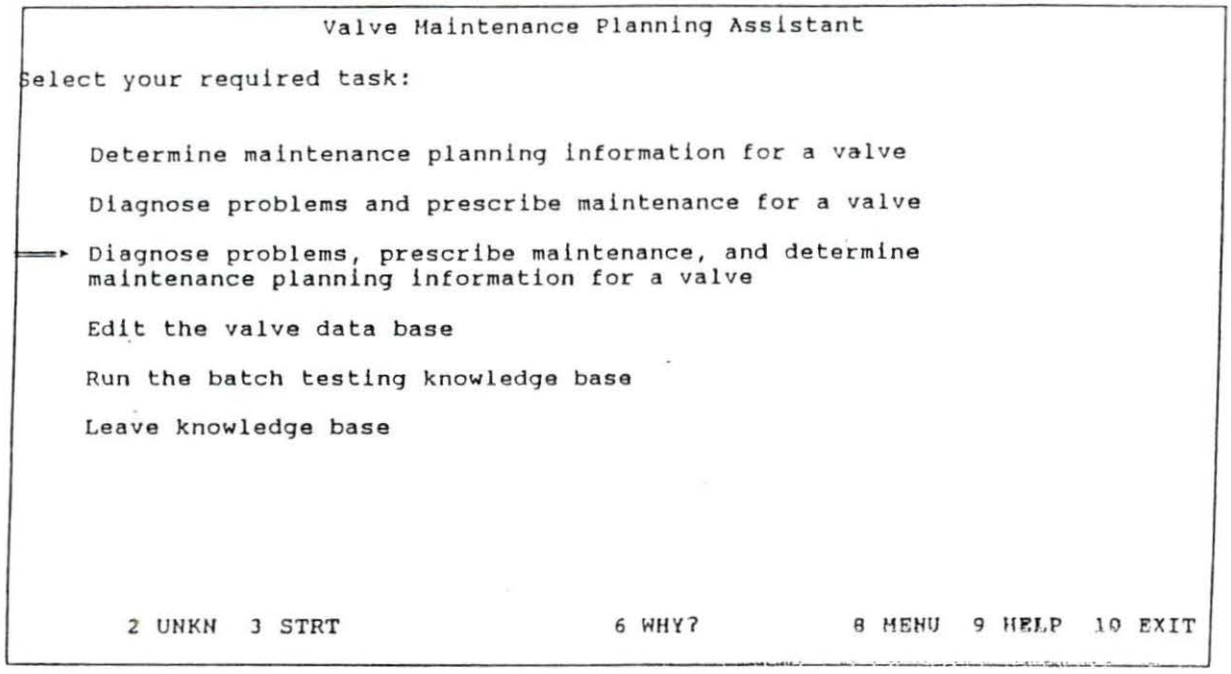


Figure 6.5: System Main Menu

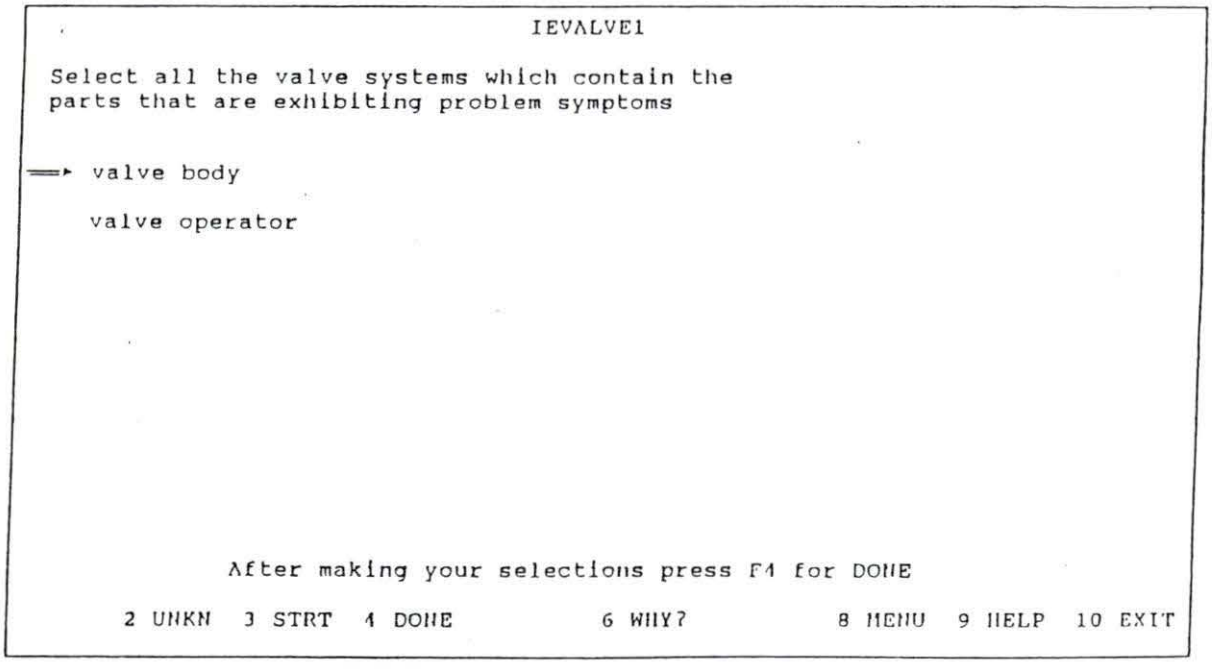


Figure 6.6: Selection of Locations for Diagnosis

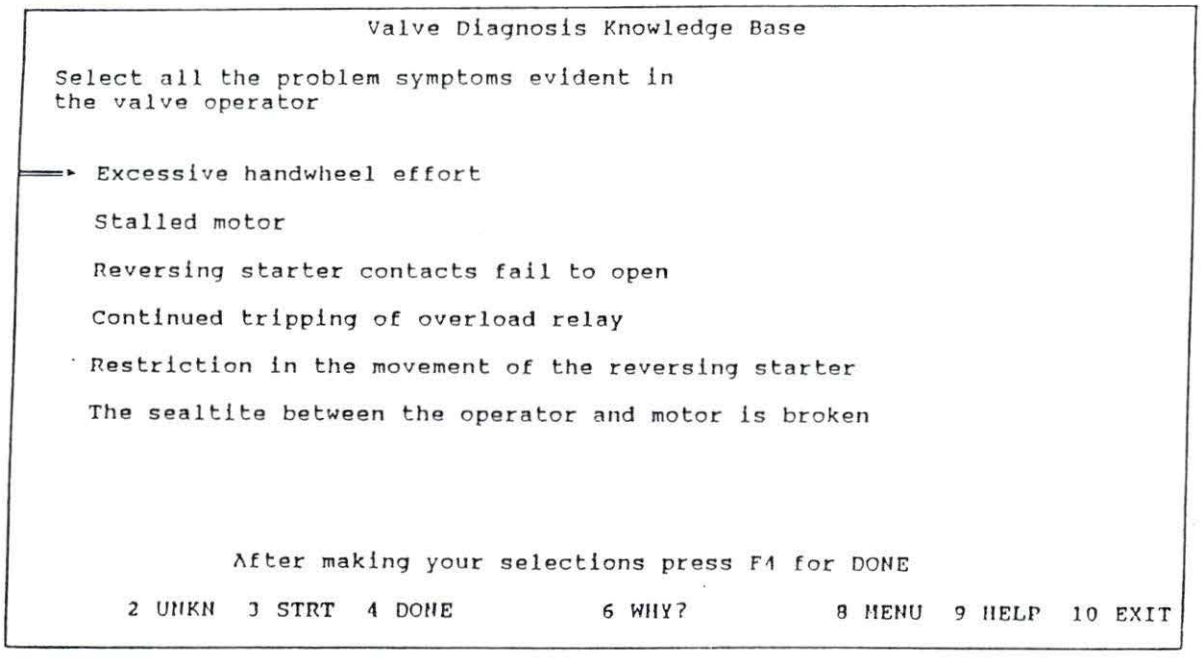


Figure 6.7: Selection of Valve Body Symptoms

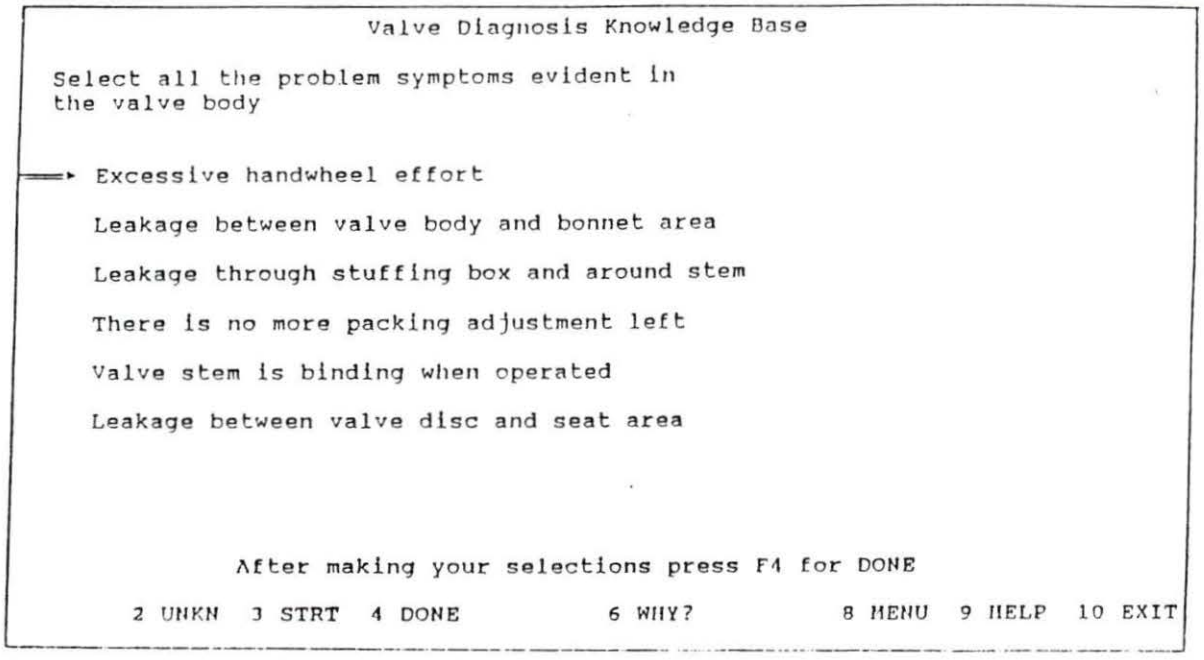


Figure 6.8: Selection of Valve Operator Symptoms

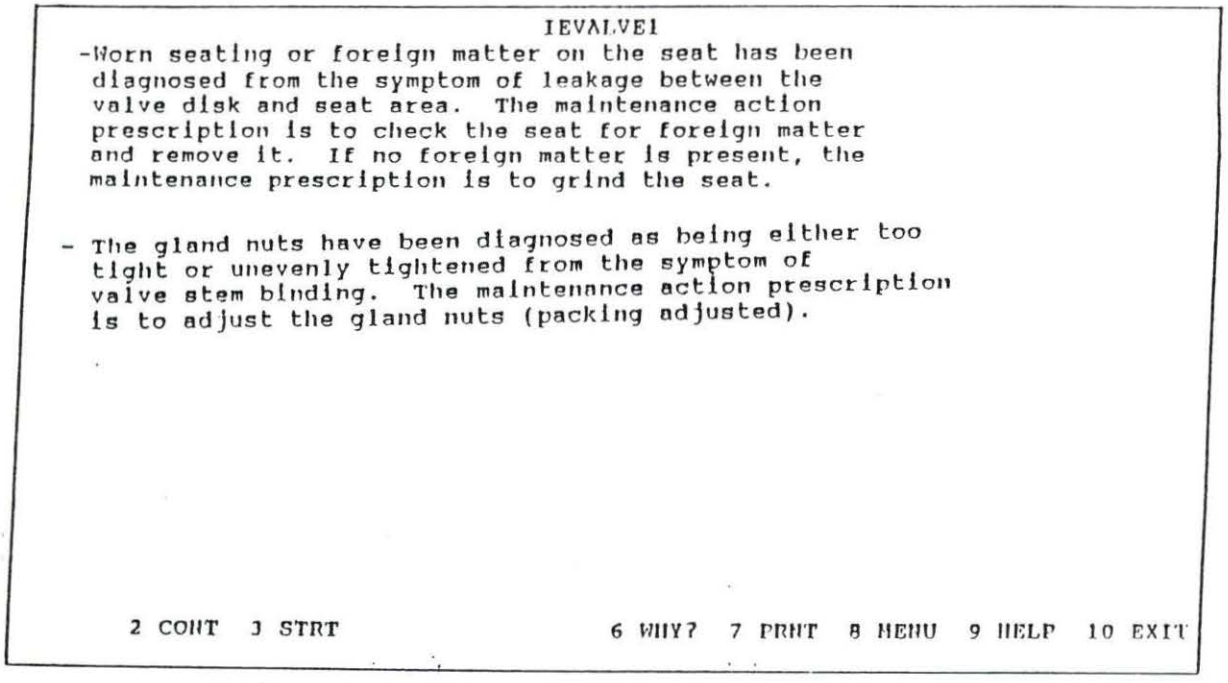


Figure 6.9: Diagnosis of Valve Body Symptoms

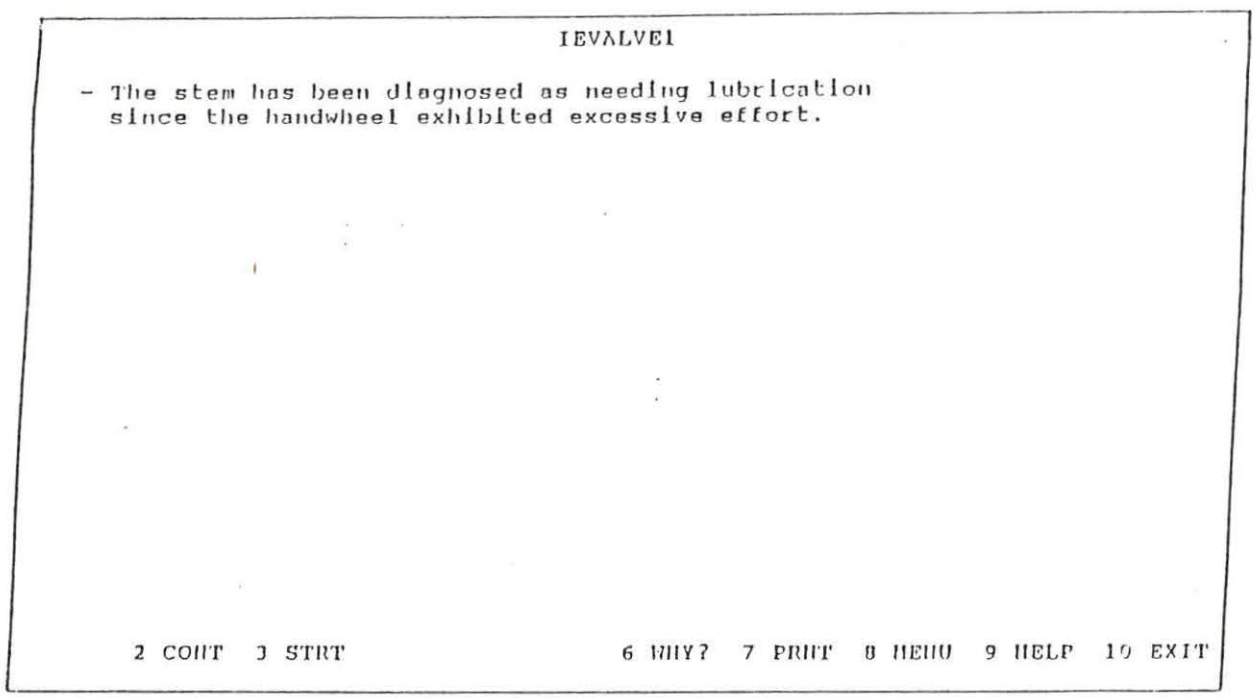


Figure 6.10: Diagnosis of Valve Operator Symptoms

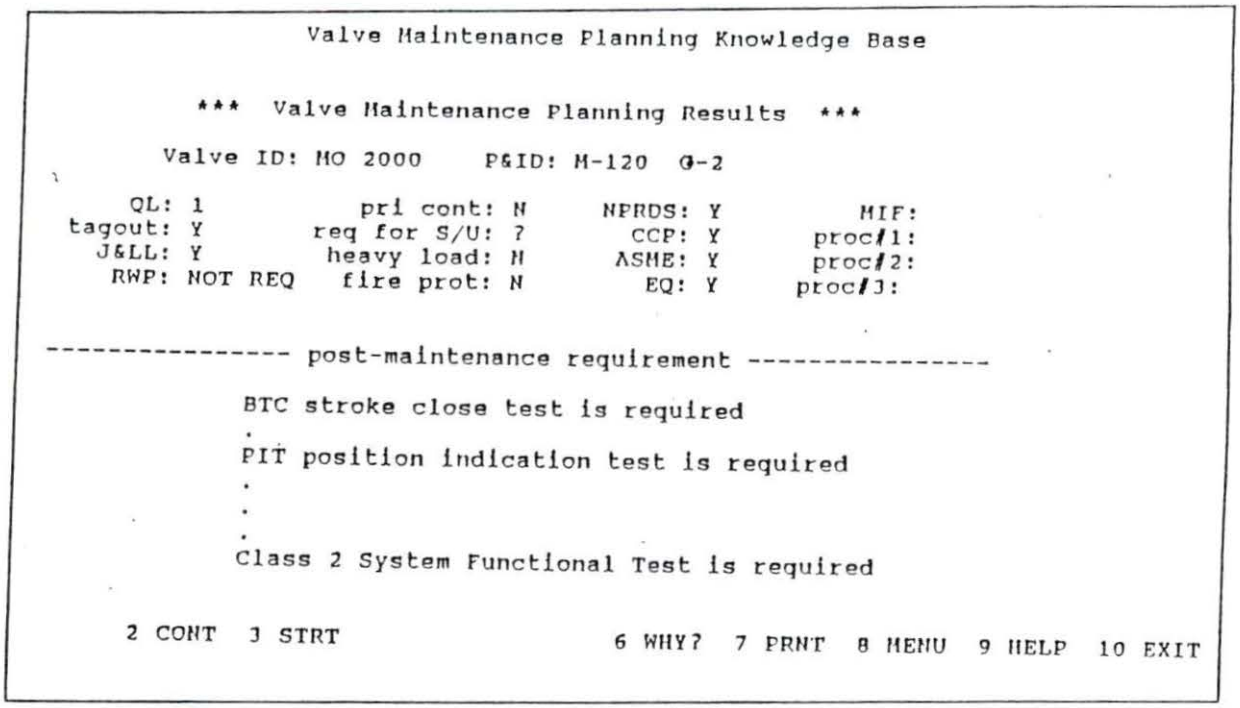


Figure 6.11: Corrective Maintenance Action Request (CMAR) Form as Presented by the System

The current database includes approximately 120 safety-related motor-operated valves (MOV's). It includes the information listed as well as other information necessary to determine the appropriate post-maintenance tests:

- The valve's P&ID coordinate which tells the location of the valve.
- The valve's diameter in inches.
- The valve's maximum stroke close time.
- Whether the valve is in primary containment.
- The valve type (although currently all valve types are MO or motor-operated).
- The valve's in-service testing (IST) class.
- The normal position of the valve (open or closed).
- Whether the valve is in a harsh environment.
- Whether a radiation work permit is needed to work on the valve.

## 6.5 Incorporation of Multiple Symptom Diagnosis

After entering the valve ID when requested by the system, the user is asked whether single symptom or multiple symptom diagnosis is required. Currently this option is available for the *diagnose and prescribe maintenance* menu selection. If multiple-symptom diagnosis is chosen the knowledge base chains to IESYMP1.PRL.

The user is presented with a series of symptoms and asked whether they apply to the case at hand. Each symptom has a series of possibilities as to what could

be causing the problem. If a symptom is totally unrelated to a given problem, then the problem is given a CF of zero for that particular symptom. In a similar fashion, a symptom could indicate to the system developer that a certain problem is not likely. In this case, the likelihood of the problem would be given a negative CF for the given symptom. As an example, some typical rules from IESYMP1.PRL (the multiple-symptom CF program) are given below.

```
RULE    for leaking packing
IF      leaking packing present
THEN    leaking packing
AND     paktite:=-0.90
AND     stembind:=0
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY leaking packing display
AND     FILE leaking packing display
ELSE    NOT leaking packing
AND     paktite:=0.90
AND     stembind:=0
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY not leaking packing display
AND     FILE not leaking packing display
!
RULE    for stalled motor
IF      stalled motor present
THEN    stalled motor
```

```
AND     paktite:=0.6
AND     stembind:=0.4
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY stalled motor display
AND     FILE stalled motor display
ELSE    NOT stalled motor
AND     paktite:=-0.6
AND     stembind:=-0.4
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY not stalled motor display
AND     FILE not stalled motor display
!
```

The first rule states that there is a 90% confidence that the packing is *not* too tight based on the symptom of leaking packing. If leaking packing were not present, then the rule states that there is a 90% confidence that the packing is too tight. Similarly, the second rule states that there is a 60% confidence that the packing is too tight and a 40% confidence that the valve stem is binding based on the symptom of stalled motor. If stalled motor is not present then the confidence value for packing too tight is -60% and the confidence value of binding valve stem is -40%.

Based on the two symptoms *leaking packing* and *stalled motor*, the combined CFs based on these two symptoms would be:

leaking packing = -0.75
valve stem binding = 0.40

open circuit = 0.00

as discussed in Section 3.6. The zero CF on *open circuit* indicates that no conclusions can be reached regarding this problem based on these symptoms.

The confidence factors presented in the above rule and in the remainder of the knowledge base are for illustrative purposes only. Because of the time limit involved, and the difficulty sometimes presented in assigning meaningful confidence factors, the CFs used in the IESYMP1 knowledge base are for demonstrative purposes only and should not be construed as being accurate in a real-life situation.

See Appendix C for a listing of the multiple-symptom knowledge base IESYMP1.PRL and the associated Fortran programs which it calls.

## 6.6 Conclusions and Suggestions for Future Work

The expert system will show its usefulness in the time it will save in maintenance planning and diagnosis. Suggestions by the Duane Arnold staff have helped to expand the scope of tests covered by the maintenance planning section and have improved the usability of the program.

The current system includes about 120 motor-operated safety-related valves. It can prescribe which post-maintenance tests need to be performed on the valve based on the maintenance done on it. The system is also capable of diagnosing common problems with the valve body and operator. If the diagnosis function is selected, the system can prescribe the necessary post-maintenance tests based on its diagnosis of the problem.

While the maintenance section is the most developed at this time, future plans

should include expanding and refining the diagnosis section of the knowledge base. Work should also be done in expanding the number of symptoms with associated problem CFs that are present in the multiple-symptom knowledge base, as well as determining accurate confidence factors for these cases. The knowledge base, or one similar to it, could also be expanded to cover other component areas.

As mentioned in Chapter 5, work could be done studying the feasability of incorporating some sort of fuzzy logic or Dempster-Shafer method into an expert system in order to compare its results to those of the present expert system.

# 7. BIBLIOGRAPHY

[1] Winter, M. J. "A Knowledge-Based Assistant for Valve Maintenance Planning." *Master of Science thesis for Department of Nuclear Engineering.* Iowa State University, 1987.

[2] Charniak, E. and D. McDermott. *Introduction to Artificial Intelligence.* Reading, Mass: Addison-Wesley, 1987.

[3] Hayes-Roth, F., D. A. Waterman and D. B. Lenat. "An Overview of Expert Systems." *Building Expert Systems.* Reading, Mass: Addison-Wesley, 1983.

[4] Uhrig, R. E. "Nuclear Utilities put Artificial Intelligence to Work." *Power* **132**, No. 6, 35-38 (1988).

[5] Harmon, P. and D. King. *Expert Systems.* New York: John Wiley & Sons, Inc., 1985.

[6] Uhrig, R. E. "Applications of Artificial Intelligence in the U.S. Nuclear Industry." *Proceedings of ANS Conference on Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry.* Held Aug. 31-Sept.2 1987, Snowbird, Utah. New York: Plenum Press, 1988.

[7] Neuschaefer, C. H. and P. Rzasa. "Pipes Expert System Speeds up Eddy Current Inspection Planning." *Nuclear Engineering International* **35**, No. 426, 37-38 (1990).

[8] "A VITAL Service for Predictive MOV Maintenance." *Nuclear Engineering International* **35**, No. 429, 40-41 (1990).

[9] Bhatnagar, R., D. W. Miller, B. K. Hajek and J. E. Stasenko. "An Integrated Operator Advisor System for Plant Monitoring, Procedure Management, and Diagnosis." *Nuclear Technology* **89**, No. 3, 281-317 (1990).

[10] Mikaili, R., R. Danofsky and B. Spinrad, "A Knowledge-Based System for Finding Cutsets and Performing Diagnostics." *Transactions of the American Nuclear Society* **59**, 121 (1989).

[11] Shortliffe, E. and B. Buchanan. "A Model of Inexact Reasoning in Medicine." *Mathematical Biosciences* **23**, 351-379 (1975).

[12] Minasi, M. "Exhibiting Uncertainty (Part 1)." *AI Expert* **5**, No. 6, 13-14 (1990).

[13] Shortliffe, E. and B. Buchanan. "Uncertainty and Evidential Support." In *Rule-Based Expert Systems* edited by E. Shortliffe and B. Buchanan. Reading, Mass: Addison-Wesley, 1984.

[14] *Level5 User's Manual.* New York: Information Builders Inc., 1987.

[15] Cumella, J. "Uncertainty In Level5." *FOCUS Systems Journal* **2**, No. 5, 57-63 (1989).

[16] Zadeh, L. A. "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes." *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-3**, No. 1, 28-44 (1973).

[17] Mamdani, E. H. and S. Assilian. "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller." *Int. Journal of Man-Machine Studies* **7**, 1-13 (1975).

[18] Feeley, J. J. and J. C. Johnson. "Linguistic Control of a Nuclear Power Plant." *Proceedings of ANS Conference on Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry.* Held Aug. 31-Sept.2 1987, Snowbird, Utah. New York: Plenum Press, 1988.

[19] Roglans-Ribas, J. "Development of a Simple Fuzzy Controller to Perform Power Level Changes in a Nuclear Reactor." *Special Project for Nuclear Engineering 544.* Iowa State University. December 1986.

[20] Spillman, R. "Managing Uncertainty with Belief Functions." *AI Expert* **5**, No. 5, 44-49 (1990).

[21] Christiansen, L. "A Knowledge-based Assistant for Valve Maintenance Planning." *Annual Report to Program Sponsers, Affiliate Research Program in Electrical Power.* ERI Project 3025, ISU-ERI-Ames 89284, May 1989.

[22] Christiansen, L. "Use of Confidence Factors in Expert Systems." *Annual Report to Program Sponsors, Affiliate Research Program in Electrical Power.* ERI Project 3025, ISU-ERI-Ames 90222, May 1990.

[23] Borse, J. G. *Fortran and Numerical Methods for Engineers.* Boston, Mass: PWS Engineering, 1985.

# 8. APPENDIX A. FORTRAN PARAMETER PASSING PROGRAM

```
C*************************************************************************
C*                                                                       *
C*   Program    :  Fortran Library for ASCII parameter passing to       *
C*                 Level5                                                *
C*                                                                       *
C*   File Name  :  ASCIIPRM.FOR                                          *
C*                                                                       *
C*   Version    :  1.0                                                   *
C*                                                                       *
C*   Compiler   :  IBM Fortran & PROFORT COMPILERS                       *
C*                                                                       *
C*   Comments   :  This is the library of Fortran subroutines that       *
C*                 provide all the utilities for passing parameter       *
C*                 data between a Fortran program and Level5.            *
C*                                                                       *
C*                 The file ASCIIPRM must be linked with the Fortran     *
C*                 program as follows:                                   *
C*                                                                       *
C*                     link prog+ASCIIPRM;                               *
C*                                                                       *
C*                 These routines provide parameter passing via disk     *
C*                 file.  The following statements must be included      *
C*                 in your main Fortran program prior to any routine     *
C*                 calls in order to access the routines:                *
C*                                                                       *
C*                       Character*20 FilNam                             *
C*                       Common /Param/ Iunit,NmParm,NmRead              *
C*                       Common /CharC/ FilNam                           *
C*                       Iunit = xx   (unit number)                      *
```

```
C*                      FilNam = 'name'  (must be same name as in     *
C*                                        the Level5 knowledge base)  *
C*                                                                    *
C*               where                                                *
C*                      Iunit = unit number of parameter file         *
C*                      FilNam = parameter file name (may include     *
C*                               full DOS path)                       *
C*                      NmParm = # of parameters to be received       *
C*                               (only of use to routines)            *
C*                      NmRead = # parameters read so far             *
C*                               (only of use to routines)            *
C*                                                                    *
C*********************************************************************
C
C
      Subroutine Messag (MessNo)
C
C*********************************************************************
C*                                                                    *
C*    This routine will display error messages related to             *
C*    the parameter passing                                           *
C*                                                                    *
C*********************************************************************
C
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      If (MessNo .EQ. 0) then
        Write (*,*) 'Parameter file empty'
      Elseif (MessNo .EQ. 1) then
        Write (*,*) 'I/O Error upon Read'
      Elseif (MessNo .EQ. 2) then
        Write (*,*) 'Attempted to read more parameters than passed'
      Elseif (MessNo .EQ. 3) then
        Write (*,*) 'Parameter type mismatch'
      Elseif (MessNo .EQ. 4) then
        Write (*,*) 'Character or Boolean not found'
      Endif
      Pause
```

```
      Return
      End
C
C
      Subroutine OpenF
C
C*********************************************************************
C*                                                                   *
C*    This routine opens access to a specified disk file expected    *
C*    to contain parameter data.  Also the following variables are   *
C*    set:                                                           *
C*                NmParm                                             *
C*                NmRead                                             *
C*                                                                   *
C*********************************************************************
C
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      NmRead = 0
      Open (Iunit,File=FilNam)
      Read (Iunit,*,End=500,Err=600) NmParm
      Return
 500  Call Messag (0)
      Return
 600  Call Messag (1)
      Return
      End
C
C
      Subroutine ResetF (NmRetr)
C
C*********************************************************************
C*                                                                   *
C*    This routine resets a disk-based parameter file to the         *
C*    beginning of the file so that return data can be sent.  The    *
C*    number of return parameters is also written at the top of      *
C*    the file (this is the routine argument).                       *
C*                                                                   *
```

```
C*********************************************************************
C
      Integer*4 form
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      Open (Iunit,File=FilNam,Status='OLD')
      Close (Iunit,Status='DELETE')
      Open (Iunit,File=FilNam,Status='NEW')
      If (NmRetr .LT. 10) then
        Assign 901 to form
      Elseif (NmRetr .LT. 100) then
        Assign 902 to form
      Elseif (NmRetr .LT. 1000) then
        Assign 903 to form
      Else
        Assign 904 to form
      Endif
      Write (Iunit,form) NmRetr
 901  Format (1X,I1)
 902  Format (1X,I2)
 903  Format (1X,I3)
 904  Format (1X,I4)
      Return
      End
C
C
      Subroutine CloseF
C
C*********************************************************************
C*                                                                   *
C*    This routine closes a disk-based parameter file.               *
C*                                                                   *
C*********************************************************************
C
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
```

```
      Close (Iunit)
      Return
      End
C
C
      Subroutine ReadI (Value)
C
C*********************************************************************
C*                                                                   *
C*    This routine reads an integer from the parameter data file.    *
C*                                                                   *
C*********************************************************************
C
      Integer*2 Value
      Character Type
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      NmRead = NmRead + 1
      If (NmRead .GT. NmParm) then
        Call Messag (2)
      Else
        Read (Iunit,901,End=500,Err=600) Type
        If (Type .NE. 'N') then
          Call Messag (3)
        Else
          BackSpace Iunit
          Read (Iunit,902,End=500,Err=600) Type,Rval
          Value = Int(Rval)
        Endif
      Endif
      Return
 500  Call Messag (0)
      Return
 600  Call Messag (1)
 901  Format (A)
 902  Format (A,F16.7)
      Return
      End
```

```
C
C
      Subroutine ReadR (Value)
C
C*********************************************************************
C*                                                                   *
C*    This routine reads a real number from the parameter data file.  *
C*                                                                   *
C*********************************************************************
C
      Character Type
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      NmRead = NmRead + 1
      If (NmRead .GT. NmParm) then
        Call Messag (2)
      Else
        Read (Iunit,901,End=500,Err=600) Type
        If (Type .NE. 'N') then
          Call Messag (3)
        Else
          BackSpace Iunit
          Read (Iunit,902,End=500,Err=600) Type,Value
        Endif
      Endif
      Return
 500  Call Messag (0)
      Return
 600  Call Messag (1)
 901  Format (A)
 902  Format (A,F16.7)
      Return
      End
C
C
      Subroutine ReadC (Value)
C
C*********************************************************************
```

```
C*                                                                     *
C*    This routine reads a character from the parameter data file.     *
C*                                                                     *
C***********************************************************************
C
      Character Value,Type
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      NmRead = NmRead + 1
      If (NmRead .GT. NmParm) then
        Call Messag (2)
      Else
        Read (Iunit,901,End=500,Err=600) Type
        If (Type .NE. 'C') then
          Call Messag (3)
        Else
          BackSpace Iunit
          Read (Iunit,902,End=500,Err=600) Value
          If (Value .EQ. ' ') then
            Call Messag (4)
            Return
          Endif
        Endif
      Endif
      Return
 500  Call Messag (0)
      Return
 600  Call Messag (1)
 901  Format (A)
 902  Format (A2)
      Return
      End
C
C
      Subroutine ReadB (Value)
C
C***********************************************************************
C*                                                                     *
```

```
C*    This routine reads a logical value from the parameter          *
C*    data file.                                                     *
C*                                                                   *
C*******************************************************************
C
      Character Cval,Type
      Logical Value
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      NmRead = NmRead + 1
      If (NmRead .GT. NmParm) then
        Call Messag (2)
      Else
        Read (Iunit,901,End=500,Err=600) Type
        If (Type .NE. 'L') then
          Call Messag (3)
        Else
          BackSpace Iunit
          Read (Iunit,902,End=500,Err=600) Cval
          If (Cval .EQ. ' ') then
            Call Messag (4)
          Else
            If (Cval .EQ. 'T') then
              Value = .TRUE.
            Else
              Value = .FALSE.
            Endif
          Endif
        Endif
      Endif
      Return
 500  Call Messag (0)
      Return
 600  Call Messag (1)
 901  Format (A)
 902  Format (A2)
      Return
      End
```

```
C
C
      Subroutine ReadS (Value)
C
C*********************************************************************
C*                                                                   *
C*    This routine reads a character string from the parameter       *
C*    data file.                                                     *
C*                                                                   *
C*********************************************************************
C
      Character Type
      Character*60 Value
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      NmRead = NmRead + 1
      If (NmRead .GT. NmParm) then
        Call Messag (2)
      Else
        Read (Iunit,901,End=500,Err=600) Type
        If (Type .NE. 'C') then
          Call Messag (3)
        Else
          BackSpace Iunit
          Read (Iunit,902,End=500,Err=600) Value
        Endif
      Endif
      Return
 500  Call Messag (0)
      Return
 600  Call Messag (1)
 901  Format (A)
 902  Format (A61)
      Return
      End
C
C
      Subroutine WriteI (Value)
```

```
C
C*********************************************************************
C*                                                                   *
C*    This routine writes an integer to the parameter data file.     *
C*                                                                   *
C*********************************************************************
C
      Real Rval
      Integer*2 Value
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      Rval = Float(Value)
      Write (Iunit,991) Rval
 991  Format ('N ',E13.6)
      Return
      End
C
C
      Subroutine WriteR (Value)
C
C*********************************************************************
C*                                                                   *
C*    This routine writes a real number to the parameter data file.  *
C*                                                                   *
C*********************************************************************
C
      Real Value
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      Write (Iunit,991) Value
 991  Format ('N ',E13.6)
      Return
      End
C
C
      Subroutine WriteB (Value)
```

```
C
C*********************************************************************
C*                                                                   *
C*    This routine writes a logical value to the parameter           *
C*    data file.                                                     *
C*                                                                   *
C*********************************************************************
C
      Logical Value
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      If (Value) then
        Write (Iunit,'(''L T'')')
      Else
        Write (Iunit,'(''L F'')')
      Endif
      Return
      End
C
C
      Subroutine WriteS (Value)
C
C*********************************************************************
C*                                                                   *
C*    This routine writes a character string to the parameter        *
C*    data file.                                                     *
C*                                                                   *
C*********************************************************************
C
      Character*60 Value
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      Write (Iunit,'(''C '',A60)') Value
      Return
      End
C
```

```
C
      Subroutine WriteC (Value)
C
C*********************************************************************
C*                                                                   *
C*    This routine writes a character to the parameter data file.    *
C*                                                                   *
C*********************************************************************
C
      Character Value
      Integer*2 Iunit,NmParm,NmRead
      Character*20 FilNam
      Common /Param/ Iunit,NmParm,NmRead
      Common /CharC/ FilNam
      Write (Iunit,'(''C '',A)') Value
      Return
      End
```

# 9. APPENDIX B. PROGRAM LISTING FOR IEVTTEST.PRL

Program IEVTTEST is called and compiled with program IEVALVE2. As mentioned in Chapter 6, this program checks the database to determine if the requirements are met to perform one or more of the following four VT tests: the VT1, VT2 hydro, VT2 pressure or VT 3 tests. The reader is referred to Section 6.3.2.1 for more details on these tests.

```
TITLE IEVTTEST
!
$ IEDECLAR.PRL

1. VT tests examined
!
RULE  For if all VT tests examined
IF    VT1 test required tested
AND   VT2 hydro test required tested
AND   VT2 pressure test required tested
AND   VT3 test required tested
AND   VT test not required tested
AND   display option decided
THEN  VT tests examined
!
RULE  For assigning format of displays
IF    VT test not required
THEN  display option decided
AND   VT not display := VT exam not required
```

```
AND   DISPLAY short
ELSE  display option decided
AND   DISPLAY long
!
RULE  For no VT tests
IF    Valve size < 1
OR    no VT tests at all are required
THEN  VT test not required
!
RULE  For no VT tests at all being required
IF    NOT VT1 test required
AND   NOT VT2 hydro test required
AND   NOT VT2 pressure test required
AND   NOT VT3 test required
THEN  no VT tests at all are required
!
RULE  For VT1 test
IF    Valve size >= 1
AND   valve has been disassembled
AND   valve has pressure retaining bolting
THEN  VT1 test required
!
RULE  For if valve has been disassembled for body selected
IF    Affected system for maintenance IS valve body
AND   valve body maintenance IS valve disassembled
OR    valve body maintenance IS maintained seating surface
OR    valve body maintenance IS valve replaced
THEN  valve has been disassembled
ELSE  NOT valve has been disassembled
!
RULE  For if valve has been disassembled for operator selected
IF    Affected system for maintenance IS valve operator
AND   valve operator maintenance IS operator replaced
THEN  valve has been disassembled
ELSE  NOT valve has been disassembled
!
RULE  For if valve has pressure retaining bolting
IF    Valve ID = "MO 4601"
OR    Valve ID = "MO 4627"
OR    Valve ID = "MO 4602"
```

```
OR    Valve ID = "MO 4628"
OR    Valve ID = "MO 4629"
OR    Valve ID = "MO 4630"
THEN  valve has pressure retaining bolting
ELSE  NOT valve has pressure retaining bolting
!
RULE  For VT2 hydrostatic test
IF    Valve size >= 1
AND   welding has been performed that violates pressure boundary
THEN  VT2 hydro test required
!
RULE  For VT2 pressure test
IF    Valve size >=1
AND   NOT VT2 hydro test required
AND   pressure boundary violated
THEN  VT2 pressure test required
!
RULE  For determining condition of pressure boundary
IF    welding has been performed that violates pressure boundary
OR    valve has been disassembled
THEN  pressure boundary violated
!
RULE  For if VT3 needed
IF    Valve ID = "MO 4601"
OR    Valve ID = "MO 4627"
OR    Valve ID = "MO 4602"
OR    Valve ID = "MO 4628"
OR    Valve ID = "MO 4441"
OR    Valve ID = "MO 4442"
OR    Valve ID = "MO 2312"
OR    Valve ID = "MO 2238"
OR    Valve ID = "MO 2239"
OR    Valve ID = "MO 2117"
OR    Valve ID = "MO 2137"
OR    Valve ID = "MO 1900"
OR    Valve ID = "MO 1901"
OR    Valve ID = "MO 1908"
OR    Valve ID = "MO 1909"
OR    Valve ID = "MO 2003"
OR    Valve ID = "MO 1905"
```

```
OR    Valve size > 4
AND   valve has been disassembled
THEN  VT3 test required
ELSE  NOT VT3 test required
!
!
RULE  For assigning VT1 display #1
IF    VT1 test required
THEN  VT1 test required tested
AND   VT1 display := VT1 exam is required
!
RULE  For assigning VT1 display #2
IF    NOT VT1 test required
THEN  VT1 test required tested
AND   VT1 display := .
!
RULE  For assigning VT2 hydro display #1
IF    VT2 hydro test required
THEN  VT2 hydro test required tested
AND   VT2 hydro display := VT2 hydrostatic exam is required
!
RULE  For assigning VT2 hydro display #2
IF    NOT VT2 hydro test required
THEN  VT2 hydro test required tested
AND   VT2 hydro display := .
!
RULE  For assigning VT2 pressure display #1
IF    VT2 pressure test required
THEN  VT2 pressure test required tested
AND   VT2 pressure display := VT2 pressure exam is required
!
RULE  For assigning VT2 pressure display #2
IF    NOT VT2 pressure test required
THEN  VT2 pressure test required tested
AND   VT2 pressure display := .
!
RULE  For assigning VT3 display #1
IF    VT3 test required
THEN  VT3 test required tested
AND   VT3 display := VT3 exam is required
```

```
!
RULE  For assigning VT3 display #2
IF    NOT VT3 test required
THEN  VT3 test required tested
AND   VT3 display := .
!
RULE  For assigning VT test not required tested
IF    VT test not required
OR    NOT VT test not required
THEN  VT test not required tested
!
!     TEXT References
!
!TEXT valve has pressure retaining bolting
!Does the valve have pressure retaining bolting?
!
TEXT welding has been performed that violates pressure boundary
Will welding been performed that violates the pressure boundary?
!
!
     DISPLAY References
!
DISPLAY long
                The following VT exams are required:

                        [VT1 display]

                        [VT2 hydro display]

                        [VT2 pressure display]

                        [VT3 display]

DISPLAY short
                 The following VT exams are required:
```

[VT not display]

END

# 10. APPENDIX C. PROGRAM LISTINGS FOR IESYMP1.PRL AND ASSOCIATED FORTRAN PROGRAMS

## 10.1 IESYMP.PRL

The following program, IESYMP1.PRL, is the main program for calculating multiple symptom confidence factors. The user is asked which symptoms are present in the valve operator and body. A CF is assigned to all possible causes of problems in the valve operator and body. If a symptom does not point to a particular problem, the problem is assigned a CF of zero. If a symptom indicates by its presence that a given problem is unlikely, then a negative CF is assigned.

The set of possible problems is sent to a data file for each symptom present. Then the combined CFs for each possible problem based on the symptoms present are calculated in the Fortran program CFCALC and sent back to the knowledge base.

```
TITLE IESYMP1
!TITLE   Valve Maintenance Planning Assistant
!
!*************************************************************************
!     Shared Parameters
!*************************************************************************
!
$ IESHARED.PRL
!
```

```
!*************************************************************************
!     Data Type Declarations
!*************************************************************************
!
$ IEDECLAR.PRL
!
!*************************************************************************
!     Parameter Initialization Statements
!*************************************************************************
!
INIT cf matrix values  = "cfdata.dat"
REINIT cf matrix values  "cfdata.dat"
!
!*************************************************************************
!     Control Element Selectors
!*************************************************************************
!
GOALSELECT OFF
$ IEMULTI.PRL
SUPPRESS ALL
FILE cfdata.dat
!
!*************************************************************************
!     Goal Outline
!*************************************************************************
!
1. Diagnose and prescribe for multiple symptoms
!2. Diagnose prescribe and plan for multiple symptoms
!
!*************************************************************************
!     Knowledge Base Rules
!*************************************************************************
!
RULE    For starting diagnosis and prescription for multisymptom
IF      Major task IS valve diagnosis
AND     Diagnosis of multiple valve problems complete
AND     final cfs displayed
AND     cf files are to be saved checked
THEN    Diagnose and prescribe for multiple symptoms
AND     ACTIVATE delete.exe
```

```
AND     FORGET Run mode
AND     FORGET Major task
AND     FORGET Affected system for maintenance
AND     FORGET valve body maintenance
AND     FORGET valve operator maintenance
AND     FORGET motor control center maintenance
AND     FORGET Print item
AND     FORGET Batch test task
!AND    CHAIN IEVALVE
!************************************************************
RULE    to check if cf file saving options looked at
IF      save the file
OR      NOT save the file
THEN    cf files are to be saved checked
!
RULE    for if user wants to save cf file
IF      save cf file option IS save file
THEN    save the file
AND     ACTIVATE cfnfile.com
DISK    param2.dat
SEND    cf matrix values
!
RULE    for if user does not want to save cf file
IF      save cf file option IS dont save file
THEN    NOT save the file
!************************************************************
RULE    to display final confidence factors
IF      diagnosis of multiple valve problems complete
THEN    final cfs displayed
AND     DISPLAY final cfs
!************************************************************
!RULE   For starting diagnosis prescription and planning for
 multisymptom
!IF     major task IS diagnosis and planning
!AND    Diagnosis of multiple valve problems complete
!THEN   Diagnose prescribe and plan for multiple symptoms
!AND    DISPLAY documentation file notice
!AND    CHAIN IEVALVE2
!
RULE    for determining if diagnosis of multiple symptom problems
```

```
complete
IF      all valve body problems analyzed
AND     all valve operator problems analyzed
THEN    diagnosis of multiple valve problems complete
AND     ACTIVATE transpose.exe
AND     ACTIVATE cfcalc.exe
DISK    PARAM3.dat
RETURN  cf1
RETURN  cf2
RETURN  cf3
!
RULE    for checking if all valve body problems considered
IF      leaking packing considered
AND     excessive handwheel effort body considered
AND     body symptomc considered
THEN    all valve body problems analyzed
!AND     DISPLAY body cons
!
RULE    for checking if all valve operator problems considered
IF      stalled motor considered
AND     holding coil failed considered
AND     valve operator symptomc considered
THEN    all valve operator problems analyzed
!AND     DISPLAY operator cons
!************************************************************
RULE    to check if leaking packing checked
IF      leaking packing
OR      NOT leaking packing
THEN    leaking packing considered
!
RULE    to check if excessive handwheel effort body checked
IF      excessive handwheel effort body
OR      NOT excessive handwheel effort body
THEN    excessive handwheel effort body considered
!
RULE    to check if body symptomc checked
IF      body symptomc
OR      NOT body symptomc
THEN    body symptomc considered
!************************************************************
```

```
RULE    for leaking packing
IF      leaking packing present
THEN    leaking packing
AND     paktite:=-0.90
AND     stembind:=0
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY leaking packing display
AND     FILE leaking packing display
ELSE    NOT leaking packing
AND     paktite:=0.90
AND     stembind:=0
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY not leaking packing display
AND     FILE not leaking packing display
!
RULE    for excessive handwheel effort body
IF      excessive handwheel effort body present
THEN    excessive handwheel effort body
AND     paktite:=0
AND     stembind:=0
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY excessive handwheel effort display
AND     FILE excessive handwheel effort display
ELSE    NOT excessive handwheel effort body
AND     paktite:=0
```

```
AND     stembind:=0
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY not excessive handwheel effort display
AND     FILE not excessive handwheel effort display
!
RULE    for body symptomc
IF      body symptomc present
THEN    body symptomc
AND     paktite:=0.45
AND     stembind:=0.56
AND     opencirc:=0.67
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY symptomc display
AND     FILE symptomc display
ELSE    NOT body symptomc
AND     paktite:=-0.45
AND     stembind:=-0.56
AND     opencirc:=-0.67
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY not symptomc display
AND     FILE not symptomc display
!
!         and so on....
!*************************************************************
RULE    to check if stalled motor checked
IF      stalled motor
OR      NOT stalled motor
```

```
THEN    stalled motor considered
!
RULE    to check if holding coil failed checked
IF      holding coil failed
OR      NOT holding coil failed
THEN    holding coil failed considered
!
RULE    to check if operator symptomc checked
IF      operator symptomc
OR      NOT operator symptomc
THEN    valve operator symptomc considered
!************************************************************
RULE    for stalled motor
IF      stalled motor present
THEN    stalled motor
AND     paktite:=0.6
AND     stembind:=0.4
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY stalled motor display
AND     FILE stalled motor display
ELSE    NOT stalled motor
AND     paktite:=-0.4
AND     stembind:=-0.6
AND     opencirc:=0
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY not stalled motor display
AND     FILE not stalled motor display
!
RULE    for holding coil failed
IF      holding coil failed present
THEN    holding coil failed
```

```
AND     paktite:=.25
AND     stembind:=0
AND     opencirc:=0.7
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY holding coil failed display
AND     FILE holding coil failed display
ELSE    NOT holding coil failed
AND     paktite:=0
AND     stembind:=0
AND     opencirc:=-0.7
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY not holding coil failed display
AND     FILE not holding coil failed display
!
RULE    for operator symptomc
IF      operator symptomc present
THEN    operator symptomc
AND     paktite:=0.10
AND     stembind:=0.20
AND     opencirc:=0.30
AND     ACTIVATE storcf.exe
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY symptomc display
AND     FILE symptomc display
ELSE    NOT operator symptomc
AND     paktite:=-0.10
AND     stembind:=-0.20
AND     opencirc:=-0.30
AND     ACTIVATE storcf.exe
```

```
DISK    cf.dat
SEND    paktite
SEND    stembind
SEND    opencirc
AND     DISPLAY not symptomc display
AND     FILE not symptomc display
!
!       and so on...
!
!*************************************************************
!               DISPLAYS
!*************************************************************
TEXT leaking packing present

Is the valve packing leaking?
!
TEXT stalled motor present

Is the valve motor stalled?
!
TEXT excessive handwheel effort body present

Does the handwheel require excessive effort to turn?
!
TEXT body symptomc present

Does the valve body exhibit symptomc?
!
TEXT stalled motor present

Is the valve motor stalled?
!
TEXT holding coil failed present

Has the holding coil failed?
!
TEXT operator symptomc present

Is operator symptomc present?
!
```

```
TEXT  save cf file option

Would you like to save the previous file of cf values?
!
TEXT save file
    Yes
!
TEXT dont save file
    No
!
DISPLAY leaking packing display
leaking packing
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
DISPLAY not leaking packing display
not leaking packing
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
DISPLAY stalled motor display
stalled motor
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
DISPLAY not stalled motor display
not stalled motor
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
DISPLAY excessive handwheel effort display
excessive handwheel effort body
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
```

```
DISPLAY not excessive handwheel effort display
not excessive handwheel effort body
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
DISPLAY holding coil failed display
holding coil failed
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
DISPLAY not holding coil failed display
not holding coil failed
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
DISPLAY symptomc display
symptomc
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
DISPLAY not symptomc display
not symptomc
packing too tight cf=[paktite]
binding valve stem cf=[stembind]
open circuit cf=[opencirc]
!
!DISPLAY body cons
!All valve body problems considered.
!
!DISPLAY operator cons
!All valve operator problems considered.
!
DISPLAY final cfs
!
confidence that the cause is packing too tight=[cf1]
confidence that the cause is binding valve stem=[cf2]
```

```
confidence that the cause is open circuit=[cf3]
!
END
```

## 10.2 STORCF.FOR

This program receives the set of problem CFs associated with each symptom from IESYMP1.PRL. It writes each group of CFs into an array with each row consisting of the CFs for each problem for a given symptom. It uses the procedures OPENF, READR, and RESETF from ASCIIPRM.FOR (see Appendix A). It also writes the dimensions of the array read from IESYMP1 to a data file for use in subsequent programs. This program and all subsequent programs were written in standard Fortran 77 [23].

```
c       this program recieves the confidence factors from IESYMP1.PRL
c       and writes them into an array called CFMATRIX.DAT. It also
c       sends the array dimensions to subsequent programs via the data
c       file ARRAYDIMS.DAT.
c
        PROGRAM STORCF
        DIMENSION CFMATRIX(0:100),CFVAL(0:100)
        INTEGER DIAGI,IUNIT,NMPARM,NMREAD
        REAL A,PAKTITECF,STEMBINDCF,OPENCIRCCF
        CHARACTER*20 FILNAM
        COMMON/PARAM/ IUNIT,NMPARM, NMREAD
        COMMON/CHARC/FILNAM

        IUNIT=10
c
c       NMPARM must be changed if more diagnoses are added
c       it is the number of parameters expected from the calling PRL
c       program.
```

```
c
      NMPARM=3

      FILNAM='C:\PRL\CF.DAT'

      OPEN(UNIT=10,FILE='C:\PRL\CF.DAT')
      OPEN(UNIT=20,FILE='C:\PRL\CFMATRIX.DAT',STATUS='UNKNOWN')
      OPEN(UNIT=30,FILE='C:\PRL\ARRAYDIMS.DAT',STATUS='UNKNOWN')
c
c     more read statements must be added if more symptoms are included
c     in IESYMP1.
c
      CALL OPENF
      CALL READR(PAKTITECF)
      CFVAL(1)=PAKTITECF
      CALL READR(STEMBINDCF)
      CFVAL(2) =STEMBINDCF
      CALL READR(OPENCIRCCF)
      CFVAL(3)=OPENCIRCCF

      CALL RESETF(NMPARM)

      DO 20 DIAGI=1,NMPARM
         CFMATRIX(DIAGI)=CFVAL(DIAGI)
 20   CONTINUE

      DO 30 K=1,1000
         READ(20,*,END=999)A
 30   CONTINUE

 111  FORMAT(1X,F5.2,1X,F5.2,1X,F5.2)

      STOP
c
c     this write statement must and associated format must be changed
c     if more diagnoses are added.
c
 999  WRITE(20,*) CFVAL(1),CFVAL(2),CFVAL(3)
      WRITE(30,*) NMPARM,K-1
      END
```

## 10.3 TRANSPOSE.FOR

This program transposes the array determined by STORCF.FOR so that the rows and columns are interchanged. This was done so that the CFs for each problem are stored in their own row, with each symptom having its own column. This made it easier for CFCALC.FOR to perform the calculations necessary to combine the component CFs into one final CF for each problem.

```
c       this program takes the array generated by STORCF.FOR and
c       transposes the rows and columns so that each row of the new
c       array consists of the CFs for one diagnosis. These CFs are
c       then combined in CFCALC.FOR.
c
        PROGRAM TRANSPOSE
        REAL ARRAY(0:100,0:100),TRANS(0:100,0:100),ROW,COLUMN
        INTEGER M,N
        OPEN(UNIT=10,FILE='C:\PRL\CFMATRIX.DAT',STATUS='UNKNOWN')
        OPEN(UNIT=20,FILE='C:\PRL\CFTRANS.DAT',STATUS='UNKNOWN')
        OPEN(UNIT=30,FILE='C:\PRL\ARRAYDIM.DAT',STATUS='UNKNOWN')
c
c       array dimensions passed from STORCF are read here.
c
        READ(30,*) N,M
c
c       this reads in the array stored in CFMATRIX.DAT.
c
        READ(10,*) ((ARRAY(ROW,COLUMN),COLUMN=1,N),ROW=1,M)
c
c       this transposes the array.
c
        DO 10 ROW=1,N
           DO 20 COLUMN=1,M
```

```
                TRANS(ROW,COLUMN)=ARRAY(COLUMN,ROW)
 20          CONTINUE
 10       CONTINUE
c
c         this writes the new array to CFTRANS.DAT.
c
          DO 30 ROW=1,N
             WRITE(20,111) (TRANS(ROW,COLUMN),COLUMN=1,M)
 30       CONTINUE
c
c         this format must be changed if more diagnoses are added.
c
 111      FORMAT (10(1x,f5.2),1x)

          STOP
          END

          SUBROUTINE PRNTARRY(X,Y,ARRAY)
          INTEGER X,Y,COL,ROW,I
          DIMENSION ARRAY(0:100,0:100)
          WRITE(*,30)(I,I=1,X)
          DO 10 ROW=1,X
             WRITE(*,20)ROW,(ARRAY(ROW,COL),COL=1,Y)
 10       CONTINUE
 20       FORMAT(2X,I2,2X,10(F6.2,1X))
 30       FORMAT(5X,10(2X,I2,3X))
          RETURN
          END
```

## 10.4 CFCALC.FOR

This program calculates the final confidence factors based on the component CFs sent to it via IESYMP1.PRL. The subroutine COMBIN calculates the CFs according to the rules of Equation 3.29. First, the first two values of row one are combined. The result is stored in the position of the second value. This is repeated with the second and third values being combined and stored in the third position, and so on

for each row. The final CFs are sent back to IESYMP1.PRL via data file PARAM3.

```
c       this program calculates the final CFs based on the component
c       CFs recieved from IESYMP1.
c
        PROGRAM CFCALC
        INTEGER I,COL,ROW,J
        REAL CF(0:100,0:100),CFCOMB(0:100)
        INTEGER*2 IUNIT,NMPARM,NMREAD
        CHARACTER*20 FILNAM
        COMMON /PARAM/ IUNIT,NMPARM,NMREAD
        COMMON /CHARC/ FILNAM


        OPEN(UNIT=10,FILE='C:\PRL\CFTRANS.DAT',STATUS='UNKNOWN')
        OPEN(UNIT=20,FILE='C:\PRL\PARAM3.DAT',STATUS='UNKNOWN')
        OPEN(UNIT=30,FILE='C:\PRL\ARRAYDIM.DAT',STATUS='UNKNOWN')
c
c       this reads the dimensions of the array of CF values.
c
        READ(30,*) ROW,COL
        NMPARM=ROW


        IUNIT = 20
        FILNAM = 'PARAM3.DAT'
c
c       this reads in the array created in TRANSPOSE.FOR
c
 9      READ(10,*)((CF(I,J),J=1,COL),I=1,ROW)
c
c       this loop calls the subroutine responsible for calculating
c       the combined CFs for each diagnosis based on the symptoms
c       observed.
c
        DO 10 I=1,ROW
           DO 20 J=1,COL-1
              CALL COMBIN(CF,CFCOMB,I,J)
```

```
 20         CONTINUE
 10      CONTINUE

         CALL RESETF (NMPARM)
c
c        this writes the combined CFs to the data file PARAM3
c
         DO 30 I=1,ROW
            CALL WRITER(CFCOMB(I))
 30      CONTINUE

         CALL CLOSEF

         STOP
         END

         SUBROUTINE COMBIN(X,CFCOMB,I,J)
c
c        this subroutine calculates the combined CFs for each diagnosis
c        based on three cases: both component CFs greater than zero, both
c        less than zero or one greater than zero, the other less than zero.

         REAL X(0:100,0:100),CFCOMB(0:100)
         INTEGER I,J

         IF (X(I,J) .GT. 0 .AND. X(I,J+1) .GT. 0) THEN
            CFCOMB(I) = X(I,J) + X(I,J+1)*(1 - X(I,J))
         ENDIF

         IF ((X(I,J) .GE. 0 .AND. X(I,J+1) .LE. 0)
     &   .OR. (X(I,J) .LE. 0 .AND. X(I,J+1) .GE. 0)) THEN
            CFCOMB(I) = (X(I,J) + X(I,J+1))/
     &      (1-MIN(ABS(X(I,J)),ABS(X(I,J+1))))
         ENDIF

         IF (X(I,J) .LT. 0 .AND. X(I,J+1) .LT. 0) THEN
            CFCOMB(I) = X(I,J) + X(I,J+1)*(1 + X(I,J))
         ENDIF
c
c        the combined CF for each row is calculated for each adjacent
```

```
c       pair of component CFs in the above procedures. this value is
c       set equal to CFCOMB and combined with the next CF in the row,
c       and so on until the end of each row. the old value is set equal
c       to zero for safety purposes.
c
        X(I,J+1)=CFCOMB(I)
        X(I,J) = 0

        RETURN
        END
```

## 10.5 DELETE.FOR

This program simply deletes the data files used for storage of values at the end of program execution. If this was not done then STORCF.FOR would keep adding rows onto the end of the existing data file.

```
        PROGRAM DELETE
c
c       this program clears the data files just before exiting IESYMP1.
c       this was done because otherwise STORCF would continue writing
c       new CF values to the end of the old cf file.
c
        INTEGER NMREAD,NMPARM
        CHARACTER*20 FILNAM
        COMMON/PARAM/ IUNIT,NMPARM, NMREAD
        COMMON/CHARC/FILNAM

        IUNIT=10
        NMPARM=3

        FILNAM='C:\PRL\CFMATRIX.DAT'

        OPEN(UNIT=10,FILE='C:\PRL\CFMATRIX.DAT')
```

```
CALL OPENF
CALL RESETF(NMPARM)

IUNIT=20
NMPARM=2

FILNAM='C:\PRL\ARRAYDIMS.DAT'

OPEN(UNIT=20,FILE='C:\PRL\ARRAYDIMS.DAT')

CALL OPENF
CALL RESETF(NMPARM)

STOP
END
```