

A knowledge-based system for finding cutsets and
performing diagnostics

by

Ramin Mikaili

An Abstract of
A Thesis Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE

Signatures have been redacted for privacy

Iowa State University
Ames, Iowa
1989

A knowledge-based system for finding cutsets and
performing diagnostics

Ramin Mikaili

Under the supervision of Dr. Richard A. Danofsky
From the department of Nuclear Engineering
Iowa State University

In performing a probabilistic risk assessment (PRA), fault trees are constructed and evaluated; this is called fault tree analysis. The end products of fault tree analysis are cutsets. Fault tree analysis is error-prone and time-consuming. An expert system called ESAS (Expert System for Analyzing Systems) is developed which implements a method that bypasses fault tree analysis for finding cutsets. This expert system then uses these cutsets for diagnostic purposes. Given an anomaly, ESAS finds the corresponding cutsets which contain the probable causes. ESAS is written in Prolog and can be run on an IBM XT, AT, or compatible. By use of virtual windows and menu systems, ESAS acquires knowledge and encodes it into facts in Prolog. Therefore, knowledge of programming in Prolog is not required for using ESAS.

A knowledge-based system for finding cutsets and
performing diagnostics

by


Ramin Mikaili

A Thesis Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE

Major: Nuclear Engineering

Approved:


In Charge of Major Work


For the Major Department


For the Graduate College

Iowa State University
Ames, Iowa
1989

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	vi
1 INTRODUCTION	1
1.1 Problem Statement	1
1.2 Scope of Thesis	4
1.3 A Short History of Artificial Intelligence	5
1.4 Introduction to Expert Systems	9
1.4.1 Knowledge Base	12
1.4.2 Inference Engine	13
2 LITERATURE REVIEW	15
2.1 AI in the Nuclear Industry	15
2.2 Review of Computer Programs Used for Fault Tree Analysis	20
2.3 Expert Systems Developed for Diagnostics	24
3 A DESCRIPTION OF FAULT TREE ANALYSIS	31
3.1 Construction of a Fault Tree	31
3.2 Evaluation of a Fault Tree	34

4	BASICS OF PROLOG	37
4.1	Knowledge Representation in Prolog	38
4.2	Iterative Techniques in Prolog	41
4.2.1	Backtracking	42
4.2.2	Recursion	43
4.3	Turbo Prolog	46
5	DEVELOPMENT OF ESAS	48
5.1	Acquisition and Representation of Information	49
5.1.1	Semantic Network Representation of a System	49
5.1.2	Other System Information	53
5.1.3	Representation of Facts in ESAS	55
5.2	Rules for Finding Paths Through a System	58
5.3	Rules for Finding Cutsets and Failure Probability of a System	61
5.4	Rules Developed for Diagnostics	68
5.5	Interacting with ESAS	69
6	ANALYSIS OF NUCLEAR PLANT SYSTEMS	72
6.1	An Emergency Core Cooling System	72
6.2	A Pressure Tank System	75
6.3	The PWR Containment Spray Injection System	78
6.4	A Power Distribution Box	80

7	CONCLUSIONS	84
7.1	Restriction on Types of Systems Analyzed by ESAS	84
7.2	Suggestions for Future Work	85
8	BIBLIOGRAPHY	88

LIST OF FIGURES

Figure 1.1:	Expert system framework	11
Figure 3.1:	Configuration of a pumping system	36
Figure 3.2:	The fault tree for the pumping system	36
Figure 5.1:	The configuration of the modified pumping system	58
Figure 5.2:	A tree representation of a graph	59
Figure 5.3:	The menu system of ESAS	70
Figure 6.1:	The configuration of ECCS	73
Figure 6.2:	The fault tree constructed for ECCS	74
Figure 6.3:	The failure menu tree for ECCS	75
Figure 6.4:	The configuration of the pressure tank system	76
Figure 6.5:	The configuration of the PWRCSIS	79
Figure 6.6:	The configuration of the power distribution box	81

ACKNOWLEDGEMENTS

The nuclear industry has been struggling to convince the public of the safety of nuclear power plants for the past two decades. There has not been any demand for nuclear plants for the last decade. This has forced nuclear engineering and other related nuclear fields to diversify. The Nuclear Engineering Department at Iowa State University, under direction of Dr. Spinrad has chosen to respond to the demand for the diversification. I would like to acknowledge the faculty members of the Nuclear Engineering Department for providing such diversity. Exploring artificial intelligence is an example of this diversification.

The person responsible for introducing artificial intelligence to the Nuclear Engineering Department at Iowa State University is Dr. Danofsky. He, with a great deal of enthusiasm for research in new areas, has always been able to interest students. I am greatly appreciative of his guidance. It has been a pleasant experience to work with Dr. Danofsky. Mostly, I would like to thank Dr. Danofsky for his friendship and trust.

Dr. Spinrad took part in this project by giving invaluable suggestions. His efforts are greatly appreciated not only for this project but also for motivating me to become more research oriented. Also, I express my gratitude to Dr. Wright for being a member of my graduate committee.

Also, I am indebted to the Nuclear Engineering Department for the financial support during my graduate studies. In this regard, the interest and financial support of the Power Affiliates and Iowa Electric Company is appreciated.

Special thanks to Dr. Williams (Bob) and Ruth Anderson for their assistance and advices but mostly for their friendship. Their constant help in bureaucratic and personal affairs made this project much easier. I specially wish Ruth strength and health.

Dr. Mostafa Mikaili and Hajar Mahmoudian Mikaili (my parents) played a great role in my education. They not only financially supported my undergraduate studies but also inspired me to pursue higher education. My parents' goal have always been to facilitate the education of their children by means of financial and moral support. They both had to make great sacrifices in their lives to achieve this goal. I express my deepest gratitude to them. Very special thanks to my brothers Arman and Afshin who have always been with me. Additional thanks to Afshin for teaching me the tricks of using AutoCad.

I want to extend my appreciation to my fellow graduate students Üner Çolak, Niyazi Sökmen, Mark Nutt, Dave Roth, Lance Christianson, Zekeriya Altaç, and Russ Bywater, working with whom made late night and/or early morning hours pleasant. I specially want to thank Niyazi and Üner for their help with LATEX, and their suggestions. Also, I thank my former fellow graduate students, Dr. Jordi Roglans Ribas, Mike Winters, Dr. Okan Zabunoğlu, and Dr. John Sankoorikal for the bull sessions and valuable suggestions. Special thanks to Jordi for leaving the cassette tape of the violin concert of J. S. Bach, performed by Paganini; I could not

have stayed awake without this screeching music during long nights at Nuc. E Lab. I have been very fortunate to work with this group.

For my little three and one year old friends, Deniz and Arda, I have special thanks. With their smiles and innocence, they cleared and refreshed my mind when I was frustrated and tired. Also, I thank my other little friends, my cats, Sasha, Bach, Ike, and Peeshi, who stayed up with me and kept me awake by demanding pets.

Very special thanks to my friend Linda Lamb for her advice, help, and tolerance. Linda, who has been with me in every step of the way for the past five years, made the time spent on this project more meaningful. I deeply appreciate her partnership by dedicating this work to her.

1 INTRODUCTION

1.1 Problem Statement

The task of assessing the risks to the public from potential accident scenarios involving commercial nuclear power plants achieved prominence during the early 1970s with the U.S. Reactor Safety Study [1]. Initially sponsored by the U.S. Nuclear Regulatory Commission (NRC), this study, which was based on probabilistic methods introduced the use of fault trees and event trees which are needed to perform such comprehensive risk assessments[1].

A probabilistic risk assessment (PRA) is used to determine the failure probability of important systems of a power plant. So far, all PRA studies have been carried out by developing fault trees. A fault tree is a graphic model of the various parallel and sequential combinations of faults that will result in the occurrence of a pre-defined undesired event. A fault tree depicts the logical interrelationships of basic events that lead to an undesired event, which is called the top event of the fault tree [1]. The end products of fault tree analysis are lists of components whose failures cause the top event. These lists are called cutsets.

The reactor safety study, which was conducted under the direction of Professor Norman Rassmussen, is called WASH-1400 [2]. The main task of this study was to estimate the risk to the public using PRA methods, mainly fault tree analysis,

for two types of then-modern nuclear reactors: the Surry 1 788-MW(e) Pressurized Water Reactor (PWR) and the Peach Bottom 2 1065-MW(e) Boiling Water Reactor (BWR) [3]. Although the scope of WASH-1400 was only the analysis of these two reactors, the report has been used extensively as a benchmark for the safety analysis of nuclear power plants ever since. Seventeen contractors and national laboratories were required [3] for completing WASH-1400; therefore, it is apparent that a PRA analysis requires a great deal of effort.

In an attempt to automate fault tree analysis, computer codes have been developed. These codes are restricted by the memory size and speed of the computer that is used. Modern main frame computers have large memory and high execution speed capabilities, so the use of these codes is possible but very expensive; they are also error-prone for analysis of large systems such as are encountered in nuclear power plants [1].

For this thesis, we developed an expert system called, Expert System for Analyzing Systems (ESAS), to find the cutsets without performing fault tree analysis. This is accomplished by simply tracing through the system. The method first forms lists of components which consist of all possible combinations of components in the system. Then, assuming the failure of components in each list, ESAS tries to find paths from the input to the output components. Depending on the type of accident, which can be an existence of a path between the input and the output component, ESAS decides whether the list of components (assumed failed) is a cutset.

This method is analogous to finding paths in a road map when certain roads have been closed due to bad weather, for example. If our desire is to reach the destination (e.g., going home) and we cannot reach it then we call the list of roads

closed a cutset. On the other hand, if we have no desire to reach the destination (e.g., going to the dentist) and there is a path possible then the list of roads open is a cutset. Although this idea is simple, application of it to the complex systems in a nuclear power plant is very tedious if performed manually. This implies that developing an expert system to perform this task is most appropriate; expert systems lend themselves to problem areas where the expertise is well defined, but tedious if resolved by humans.

Cutsets found for a top event in a system are stored in a database and used for two purposes: to calculate the probability of occurrence of a top event, and to diagnose a system. Given the failure probability rates for each component in the system, ESAS is capable of finding the total failure probability of the system, assuming a constant failure rate. In diagnosing a system, the consequences are known and the causes are desired. Another way of defining cutsets is that the components in the cutsets are possible causes of the top event (consequence). Given the consequence (top event) of an accident, ESAS carries out a diagnosis by listing the components in the cutset that were previously found. Another diagnostic feature of ESAS is providing "what if" scenarios. The path-finding procedure could be used in the case where the importance of success of components in the system is examined by trying to find paths through the system, assuming the failure of those components. Thus, ESAS could be applied to assist in conducting PRA studies or to advise the operator or staff who are diagnosing anomalies.

1.2 Scope of Thesis

For describing the development and implementation of ESAS, some background information in three areas, namely expert systems, fault tree analysis, and Prolog, is required.

In Chapter 1, Section 1.3, a brief history of artificial intelligence (AI) and the evolutionary stages which lead to the development of expert systems are described. A more formal definition and descriptions of expert systems in use today are presented in Section 1.4 with emphasis on two major components of an expert system, the inference engine and knowledge base.

In Chapter 2, the literature review is chosen to serve three purposes, namely to give the reader an overview of the problem areas in the nuclear industry for which the implementation of expert systems has been considered, to demonstrate the need for a method other than fault tree analysis which could be used to find cutsets, and to summarize the methods used in expert systems, which are also used in ESAS, for knowledge representation and diagnostics. A brief history of AI in the nuclear industry, a review of computer programs for fault tree analysis, and a summary of expert systems for diagnostics is also presented.

In Chapter 3, a brief description of fault tree analysis is given. The basic steps followed in fault tree construction and evaluation are described. The scope of this thesis did not allow for an elaborate description of fault tree analysis methods, but the terms commonly used in fault tree analysis and methods used to develop ESAS are defined.

In Chapter 4, to give non-Prolog-programmers proper background information for understanding the parts of ESAS presented in the later chapters, a short manual

on Prolog is presented. Knowledge representation and iterative techniques are the topics covered. Also, the Prolog software used to develop ESAS is briefly described.

In Chapter 5, the information acquired by ESAS for performing its tasks, namely finding paths and cutsets and performing diagnostics, is described. The representation of this information, encoding facts into Prolog, is then explained. To completely present the knowledge base of ESAS, which is comprised of facts and rules, the methods used for developing the rules which enables ESAS to perform its various tasks, are described. Finally, the interfacing features of ESAS with the user are described.

In Chapter 6, the cutsets found, using ESAS, for four nuclear power plant systems are reported and compared to the ones obtained by performing fault tree analysis. Also, some diagnostics features of ESAS are demonstrated by use of these system.

In Chapter 7, conclusions and the restrictions on the systems that can be analyzed by ESAS are reviewed, followed by suggestions for future work.

1.3 A Short History of Artificial Intelligence

It is difficult to pinpoint an exact starting date for the introduction of the term commonly called artificial intelligence (AI). However, the term artificial intelligence is credited to Marvin Minsky at the Massachusetts Institute of Technology (MIT), who in 1961 wrote a paper entitled "Steps Toward Artificial Intelligence" [4]. The 1960s were a period of intense optimism over the possibility of making a computer think. During that decade, the first computerized mathematical theorem-proving program, MACSYMA, and the famous program that acted like a psychoanalyst,

ELIZA, were developed. The development of these pioneer programs in AI was possible due to the creation of LISP, which is claimed to be the first AI language. It was created by John McCarthy at MIT [4].

Even before the 1960s it was well known that computers could do numerical calculations with an unbelievable speed, but MIT's MACSYMA went one step further by performing symbolic processing of mathematical formulas using knowledge required for matrix multiplication, integration, differentiation, etc. The development of MACSYMA has been continued to this date [5]. ELIZA was programmed to act like a Rogerian psychoanalyst. In this style of analysis, the psychiatrist takes a passive role, by simply echoing the patient's own remarks. Development of ELIZA raised the question, "Should computers be used in this way?" Even Weizenbaum, ELIZA's creator, in one of his books discredited his own program [4]. Schildt [4] blames the pessimism about the AI methods, which existed in the 1960s, on the fear of automation.

Nevertheless, one can not deny the successes of the researchers in the AI field in the 1960s. As a matter of fact, in the beginning of the 1970s, it was believed that the capability to produce a program which has human-like intelligence compatibilities was just imminent [5]. But difficulties were encountered in generalizing these successes into flexible, intelligent programs. Even with the 1970s computers, which had larger memory and much increased computer speed, it was soon found that AI programs exhausted computer memory or the execution time became too long. This directed the AI field to produce more efficient methods for solving AI problems, which in turn led to the introduction of the first commercial product, the expert system. An expert system is defined as a program that contains knowledge

about a certain field and, when interrogated, responds much like a human expert [4]. A new era in AI history began, augmented by the development of MYCIN, one of the first expert systems. MYCIN was developed at Stanford University to help physicians diagnose illnesses [5]. Mostly due to the success of MYCIN, the possibilities of developing expert systems in many other fields of technology have been or are being explored.

Another language, called Prolog (Programming in Logic), was created in 1972 by Alain Colmerauer in Marseilles, France, in an attempt to create an AI language that was more efficient for representing knowledge and drawing inferences [4]. Like LISP, Prolog is a language designed to solve AI-related problems; but unlike LISP, it has several special features, such as a built-in database and a rather simple syntax [4]. Before 1981, LISP was the language prominently used by the AI field in the United States and Prolog was used in Europe. This situation was altered in 1981 with the announcement by the Japanese that Prolog will be used as the language for the “fifth-generation” computers [4].

With all the research done to this day, the AI field still has not yet accomplished its stated task of creating programs that exhibit reasoning and learning processes similar to humans. However, it must be noted that the shortcomings of the AI field are mostly blamed on the lack of understanding of how the human brain actually works. To this day, AI researchers have had to guess how a human brain works. It has been stated [6],

“Although the AI field has been an active one for more than 25 years, AI researchers still have no idea how to create a truly intelligent computer. No existing programs can recall facts, solve problems, reason, learn, or process language with anything approximating human facility. This lack of success has occurred not because computers are inferior to

human brains but because we do not yet know how human intelligence is organized.”

On the other hand, it is ironic that the psychology field has actually taken some of the AI concepts to develop psychological theories, as is also stated in Anderson [6].

“... observing how we could analyze the intelligent behavior of a machine has largely liberated us from our inhibitions and misconceptions about analyzing our own intelligence.”

The field of Artificial Intelligence today is separated into four areas, natural-language processing, vision and pattern recognition, robotics, and expert systems. The source for the description of these areas is Schildt [4]. Development of natural-language processing is claimed to be the most important task in the AI field, since if this is fully accomplished then a direct human-computer communication link is possible. Natural-language processing is not fully developed due to the sheer size and complexity of the human language.

Along with natural-language processing, for a computer to be able to completely understand the world around it, some sort of vision capability is required. Often the term “image processing” is used to describe the fairly broad field of vision and pattern recognition, and enhancement. The reason for this field being so large is that it encompasses two major subdivisions: two-dimensional processing and the three-dimensional processing (sometimes called real-world processing).

The robotics field is basically a combination of other fields of AI. Natural-language processing is used, so a robot can communicate with humans in a human language. Vision and pattern recognition enables a robot to visualize and to have

between algorithms and heuristics. Algorithms are procedures guaranteed to result in the correct solution of problems. For example, the procedure for multiplication is an algorithm. By following the same procedure, one would obtain the correct solution for any two numbers. However, concluding that it will rain because it is cloudy is a heuristic. In other words, algorithms are a brute force method of solving problems which if followed exactly will result in the correct answer, whereas heuristics are short cuts for solving problems but do not always result in the correct solution (sometimes it does not rain on a cloudy day). Then why use heuristics and not algorithms? The answer lies in the fact that there is not an algorithm for every problem. For example, there is not a systematic way of predicting rain which always gives the correct forecast.

Human experts rely on heuristics in their decision making, and in order to be able to mimic them, expert systems contain a knowledge base which can hold information in the form of heuristics, or other forms of knowledge used by humans. Furthermore, expert systems contain an inference mechanism which controls the execution and draws inferences from the the knowledge base [7]. Some of the most important features of an expert system that are cited [8] indicate that an expert system should:

- cover a limited domain of expertise,
- explain its train of reasoning,
- detach facts and inference mechanism,
- allow modularity,

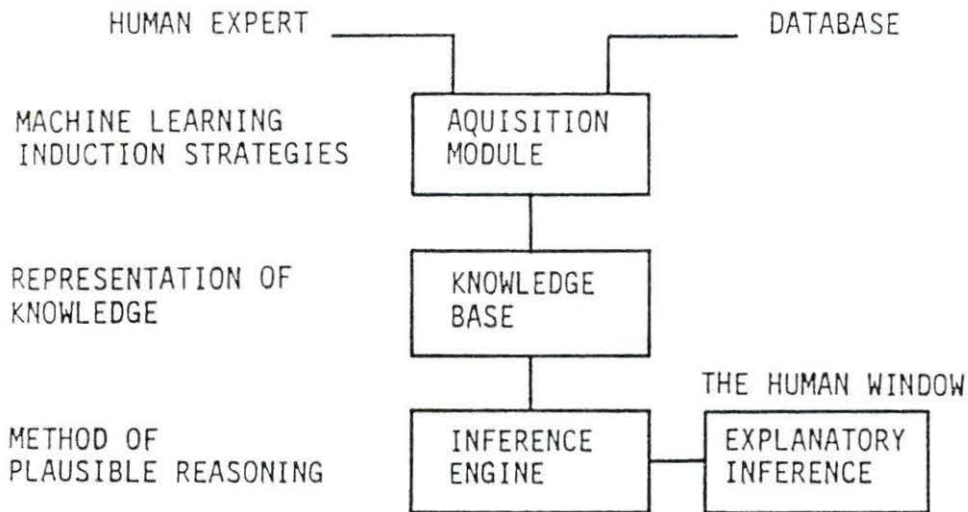


Figure 1.1: Expert system framework

- incorporate rules of thumb that human experts use,
- deliver advice as its output,
- and most importantly, make money.

A typical block structure of an expert system is presented in Figure 1.1 [6]. As depicted in this figure, the kernels or core components of an expert system are the knowledge base and the inference engine. However, the component “the human window” should not be overlooked. Most expert system languages contain excellent graphics capabilities and window systems. Since expert systems are competing with the human experts’ interactive capabilities, full exploration of this aspect is crucial.

1.4.1 Knowledge Base

The knowledge base stores information about the subject domain. Common difficulties encountered while building a knowledge base are knowledge representation (how to encode information so the computer can use it), and knowledge acquisition [7]. Knowledge representation in an expert system must contain [8]:

- domain terms, the jargon used by experts in the field
- structural relationships, the interconnection of component entities
- causal relationships, the cause-effect relations between components

Knowledge acquisition consists of two parts: acquiring the knowledge which the human expert uses in solving a problem, and collecting the data the human expert needs to solve a problem. Acquiring the knowledge poses a problem, since in most cases the human expert's knowledge on how to solve a problem (e.g., heuristics) is ill-defined and not well understood even by the expert. In general, there are four main methods of storing knowledge symbolically [8]:

- Production rules, which have an IF (A) THEN (B) format, where the condition (A) specifies some pattern and the conclusion (B) may be an action or assertion. Production rules are used for representing heuristics or other rules in the form of (IF condition THEN conclusion).
- Semantic domains, which are knowledge linked together in the form of a tree or graph. The human brain also stores information in the form of a semantic domain. For example, when one thinks about rain, the brain immediately links it with snow, cloud, etc.

- Frames, which are generalized record structures that may have default values and may have actions coded as the value of certain fields or slots.
- Horn clauses, which are a form of predicate logic on which Prolog is based.

1.4.2 Inference Engine

The inference engine is used to reach conclusions and to control the reasoning process [7]. In simpler words, it is the part of the expert system which thinks through the knowledge it has acquired. Inference engines are of either a deterministic or a probabilistic type. When one deals with a knowledge area where assertions are certain, deterministic inference engines are used. On the other hand, probabilistic inference engines deal with the type of knowledge such that an assertion is associated with an assigned probability. The majority of knowledge areas are probabilistic, but for many of these areas, the uncertainty is statistically insignificant, so the deterministic inference engine is mostly used [7]. Probabilistic or deterministic inference engines use either forward or backward-chaining methods.

Forward-chaining is the term used to describe the process of working forward from the evidence to the conclusions. Forward-chaining is sometimes called data-driven because the inference engine uses information that is provided by the user to move through a network of logical ANDs and ORs until it reaches a terminal point, which is the object. If the inference engine cannot find an object by using the existing information, then it requests additional data. In summary, a forward-chaining inference engine moves through the rules that define the object and creates a path that leads to the next rule; therefore, the only way to reach the object is to satisfy all of its rules [7].

Backward-chaining is working from hypothesis to evidence, the reverse of forward-chaining. It starts with an object and requests information to confirm or deny it. Thus, it is sometimes called object-driven because the system begins with an object and tries to verify it.

2 LITERATURE REVIEW

The main purpose of this review is to demonstrate the differences between ESAS and other programs which perform fault tree analysis and to give a description of some expert systems used for diagnosing systems in nuclear power plants. However, to give an overall picture of the use of AI in the nuclear industry, a summary of a paper titled "Application of Artificial Intelligence in the U.S. Nuclear Industry" [9] is presented.

2.1 AI in the Nuclear Industry

In the publication titled, "Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry" [10], a broad collection of papers on the application of AI programs to the nuclear industry's problem areas is presented. In the summary article in the publication, Uhrig [9] cites the problem areas where implementation of expert systems have been successful.

According to Uhrig, the development of expert systems in the nuclear industry in the United States is conducted by a broad list of organizations, including nuclear equipment vendors, architect-engineer firms, universities, national laboratories, federal agencies, the electric power utility industry, and small entrepreneurial groups. However, the largest effort is the program undertaken in 1983 by the Electric Power

Research Institute (EPRI). Special interests are in the areas of fault recognition and diagnosis, fault recovery, task planning, intelligent operator interfaces, and intelligent systems control.

Uhrig pays special attention to the use of expert system in operator assistance. He argues that the use of expert systems in assisting reactor operation is most appropriate, since great quantities of numeric, symbolic, and quantitative information are handled by the reactor operators even during routine operation. Processing large amounts of information in an abnormal situation is even more difficult and at the same time crucial. Also, the performance of operators at a nuclear power plant is affected by the stress caused in handling such large amounts of information. This causes a great deal of guesswork on the part of the operators. Expert systems can avoid this problem by providing expert advice and rapid access to a large information base. Uhrig states:

“The application of AI technologies, particularly expert systems, to the control room activities can reduce operator error and enhance plant safety and reliability. Furthermore, a large number of non-operating activities (e.g. testing, routine maintenance, outage planning, equipment diagnostics, fuel management, etc.) exist where expert systems can increase the efficiency and effectiveness of overall plant operation.”

In regards to the computers used in the control rooms, he notes that microcomputers are preferred by the operators, since they are easy to work with. Expert systems can be developed for the special AI computers, such as LISP machines, and the executable version of the programs can be transferred to the microcomputers in the control room. The major problem with implementing expert systems for assisting in operation of nuclear power plants is the reluctance of utilities to introduce to regulatory review, a new technology that involves a great deal of uncertainty. Until

they are convinced that the benefits gained warrant the effort involved, any further development of these expert systems will be slowed down.

Uhrig then cites some expert systems typical of the ones being developed in the U.S. which are actually in use at nuclear power plants today. They are briefly itemized here [9],

- Reactor Emergency Alarm Level Monitor (REALM), developed by Technology Applications, Inc., implemented to assist in deciding the response to each level of emergency (i.e., unusual event, an alert, a site area emergency, or a general emergency).
- An EPRI-developed tracking system for emergency operating procedure. It is an on-line expert system that requires no input from the operator, and can explain the conclusions reached upon the request of the operator.
- An expert system called CLEO, developed by Westinghouse Hanford Company, and used for refueling the fast flux test facility (FFTF). It is able to generate a list of necessary refueling moves in less than 30 seconds, given the present and the future core configurations of FFTF.
- An expert system, developed by the Oak Ridge National Laboratory (ORNL), used for advising operators of the ORNL's 100 MWt high flux isotope reactor.
- An expert system developed at Idaho National Engineering Laboratory for reactor safety assessment. Its main task is to aid an NRC reactor safety team to maintain a "big picture" of a transient-in-progress.
- Trip Buffer Expert System (TRIBES), developed by Middle South Utilities for

trip analysis caused by the core protection calculator and the control element assembly calculator. The calculators form a group of six digital computers that monitor nuclear power plant parameters and control parameter positions. These core protection systems initiate a trip to prevent violation of fuel design limits. After a trip, the analysis of the output of these calculators is required before the plant can be restarted.

- An expert system, developed by Stone and Webster, used for analyzing the limiting conditions of operation (LCO) and technical specifications in a nuclear power plant.
- PLEXSYS (plant expert system), for presenting piping and instrumentation drawings, and electrical on-line schematics. PLEXSYS is integrated with a conventional program called TAGS (Tagout Administration and Generation System). This integrated system can recommend a “safety tagout boundary” which allows the maintenance to be performed without danger of tripping the plant. TAGS was developed by Southern California Edison for their San Onofre Nuclear Power Plant.
- GenAID, an on-line generator diagnostic system, to diagnose 15 conditions with damage potential to the generator and to recommend corrective action for each condition. Developers were Texas Utilities and Westinghouse.
- Generic Diagnostic System (GDS) software shell of Combustion Engineering, which is used for developing expert system for power plant diagnostics.
- At Ohio State University, an expert system to diagnose operational problems

in nuclear power plants even in the presence of some incorrect and/or conflicting data is being developed. It uses classification techniques that can diagnose a large percentage of anomalies found in most mechanical systems.

It is difficult to pinpoint the problem areas where the development of expert systems have been mostly emphasized. The areas which the Department of Energy (DOE) has supported through the Small Business Innovation Research (SBIR) program are the following [9],

- “An expert system operator aid for nuclear power plant maneuvers,” Applied Research Associated, Inc.
- “An expert system decision aid for reactor trip reduction and post trip analysis,” Expert-EASE Systems, Inc.
- “Signal validation by combining model-based and evidential reasoning approaches,” Expert-EASE Systems, Inc.
- “The design of a reliable fuzzy fault-tolerant automatic control,” Technology International, Inc.
- “A bayesian diagnostic system: an expert system to aid reactor operation,” Pickard, Lowe and Garrick, Inc.
- “Residual heat removal advisor,” Odetics, Inc.

Uhrig concludes his report by forecasting an inevitable demand for automation of most functions of the nuclear power plants due to demands for increased safety

margins, lower environmental impacts, increased performance, and greater investment protection. Also, he recognizes the fact that AI and expert systems must play a major role in assuring the regulators and the public of plants being properly designed, built, operated, and maintained.

2.2 Review of Computer Programs Used for Fault Tree Analysis

In this section, a brief description of some expert systems and conventional programs that were developed to perform or to assist in performing fault tree analysis is presented. Readers, not familiar with the method of fault tree analysis, may wish to refer to Chapter 3 for a better understanding of the programs described next.

An expert system was developed by Forgnier to assist in the construction of fault trees [11]. This expert system can help a fault tree analyst to produce a high quality fault tree by alleviating the four major difficulties associated with fault tree construction, namely [11]:

- The drawing and input of the initial set of fault trees is time-consuming.
- As the risk analysis proceeds, more and more effort is required to update the fault trees and to ascertain that they remain consistent.
- There is a problem of coordination when more than one person is involved in the drawing and updating of the fault trees.
- As fault trees for subsystems are assembled into an overall plant fault tree, logical loops frequently occur. These loops must be broken before the trees can be analyzed by conventional fault tree codes.

The use of work-stations promotes the success of this expert system in improving continuity in the thought process of the fault tree development. The key contribution of this expert system is the ability to very effectively resolve logical loops in the fault trees. Knowledge Engineering Environment (KEE) was the expert system shell used for developing this expert system. KEE is a product of IntelliCorp.

Another expert system in the area of PRA, called EXPRESS, is used for the automation of reliability studies [12]. Automation was first applied to the construction of the fault trees used to assess the reliability of static thermal hydraulic systems. The term "static thermal hydraulic system" suggests that there are not any changes in the system configuration during the accident. For example, valves which are open initially, remain open. In the second phase of automation, a fault tree for electric power systems was built by trying to maintain the same approach (knowledge and reasoning) as the approach used for the thermal hydraulic systems.

The knowledge representation of EXPRESS is based on its two inference engines, ALOUETTE and LRC developed by Direction des Etudes et Recherches at EDF. ALOUETTE is an inference engine which utilizes forward chaining with a knowledge representation based on production rules (if...then...) and facts which are represented as triplets (object, relation, object). LRC is a language used to represent knowledge in prepositional zero order logic. In zero order logic, clauses contain at most one conclusion. In EXPRESS, the backward chaining inference capability of LRC is used for building a fault tree starting from undesirable events.

The methods implemented by EXPRESS are based on the basic observation that failures relating to the components of a thermal hydraulic system can always be grouped under a few large categories, for example, fluid flow interruption (blockage),

and loss of fluid (external leak). Based on this observation, EXPRESS, in the first phase, reduces the number of components of the system by grouping them into larger component categories (macro-components), according to the consequences of their failures (blockage or leakage). In the second phase the failure consequences are deduced for each component in terms of path losses, according to the macro-component(s) to which the component studied belongs. The result of these two phases is a rule based program in the LRC language. By adding a hand written rule base for undesirable events and boundary conditions, the LRC program can then be used to build a fault tree for the system studied. EXPRESS has successfully been used to study some static thermal hydraulic systems in the PALUEL nuclear power plant.

In summary, EXPRESS must have access to both backward and forward-chaining inference mechanisms. The former is required for constructing fault trees and the latter for resolving them. This is typical of most expert systems developed for fault tree analysis, which unfortunately causes complexities.

In Vessely [1], conventional computer codes for performing fault tree analysis are categorized into qualitative and quantitative analysis. Qualitative analysis includes computation of cutsets which only depends on the structure of the fault tree. In contrast, probabilistic assessment is called the quantitative evaluation of the fault tree. The division between qualitative and quantitative aspects developed naturally because probabilistic analysis often involves repeated evaluation of the tree [1]. The major difficulty with qualitative analysis, using computer codes, is computer storage and time requirements, even for analyses of medium size fault trees. This problem is partially handled by limiting the maximum number of components in a cutset.

For example, in WASH-1400 [2] only the single and double cutsets were considered, since cutsets containing more than two components become insignificant due to their low probability of occurrence.

The Efficient Logic Reduction of Fault Trees ELRAFT , written in Fortran IV for the CDC 6600, is capable of finding cutsets for up to six basic events for the top event and other specified intermediate events. It uses the unique factorization property of the natural numbers to find the cutsets of a fault tree. It assigns a unique prime number to each basic event. A bottom-up algorithm is used to process the tree. Cutsets, for the gates at successively higher levels, are represented by the product of the numbers associated with their input events. The major drawback of ELRAFT is that, for large trees, the product of the prime factors can soon exceed the capacity of the machine.

Another code for finding cutsets is ALLCUTS [1] developed by the Atlantic Richfield Company. It is written in Fortran IV and COMPASS (assembly language) for the CDC 6600 computer. ALLCUTS can compute the top event probability, sort and print up to 1000 cutsets in descending order of probability, and select cutsets in a specified probability range. It handles up to 175 basic events and 425 gate events. ALLCUTS is coupled with a graphics program which produces a plot of the fault tree based on the fault tree input description. Also, there is another program which can be used to check the input and cross reference the gates and input events used in ALLCUTS. ALLCUTS uses a top-down algorithm. In a top-down algorithm, cutsets are calculated by successive substitution into the gate equations beginning with the top event and working down the tree until all gates have been replaced by basic events. For an elaborate description of top-down and bottom-up algorithms

refer to Chapter VII of Vessely [1].

Programs which conduct a qualitative analysis are given the manually constructed fault tree as an input. The Nuclear Technology Systems Division (NTSD) of Westinghouse Electric Corporation [13], developed an interactive graphic fault tree editor called GRAPHER which could convert fault tree data into the appropriate computer code input file. With the aid of a computer-aided design (CAD) system, NTSD developed an approach for representing fluid systems within a CAD database using associated software capable of converting the database information into a fault tree model. This fault tree model is then supplied to the automated fault tree (AFT) software, which is provided with Individual Plant Evaluation (IPE) rules. The IPE consists of detailed fault tree guideline for PWRs. By using these guidelines and the fault tree model, AFT is then able to construct the fault tree.

The input for computer codes which perform quantitative evaluations of fault trees consist of two parts: (1) cutsets, and (2) failure rates. Given the above inputs, several types of quantitative results may be computed including: numerical probabilities (probabilities of system and component failures), quantitative importance (quantitative ratings of contributions to the system failure), and sensitivity evaluations (effects of changes in models and data, error bounding).

2.3 Expert Systems Developed for Diagnostics

In general, expert systems developed for diagnostics either use event-oriented, or a function-oriented knowledge base, or both. The event-oriented knowledge base is based on a fault model which requires some relationship between cause and symptom. The function-oriented knowledge base includes an understanding of the func-

tion of each component in the system being diagnosed. In the event-oriented knowledge base a component is considered either failed or operable, on the other hand, in the function-oriented knowledge base, the input and output parameters of each component is studied to decide whether a component is functioning properly or not. In the following descriptions of the expert systems, some of the forms of these two types of knowledge bases are illustrated.

A knowledge based system for plant diagnosis, has been developed by Kiguchi et al. [14]. This expert system basically performs three tasks, suspect pick up, suspect discrimination and, if necessary, test generation and evaluation. First the event-oriented knowledge, the causality relationships, is used to identify the cause or possible suspects. If the cause is found then the expert system turns to the guidance phase. Otherwise, it looks for function-oriented knowledge, which describes structure, behavior and status of the plant, to further discriminate between the suspects. Thus, two types of knowledge are used, a causality description which is a cause and consequence relationship, and a system description which includes an intended structure and expected behavior of the system. The former uses knowledge of anomalous situations and thus is event-oriented, whereas, the latter uses the knowledge of normal situations and is function-oriented. In comparison, the former is more efficient because it is direct, but requires all anomalous situations to be covered. The latter is less efficient because it is indirect, but it is more powerful, since it is much easier to describe how the system should work.

The function-oriented knowledge is represented in the form of frame structures corresponding to real plant schematics and event-oriented knowledge is embedded around these frames. This expert system is written in LISP and Fortran, on an

“improved” LISP interpreter which has a fast memory management capability. Fast data transfer between two languages, LISP and Fortran, is required, since time is an important factor in diagnostics. Numerical calculations are done by Fortran programs and LISP is used for writing the programs which perform the symbolic manipulations. The function and event-oriented knowledge bases require forward and backward-chaining inference engines respectively which are provided by this expert system.

Kiguchi et al. [14] describe an application study for diagnosis of a BWR by numerical simulations and show that it is possible to diagnose multiple events in the time sequence of their occurrence. Appropriate guidance can be given well in advance due to the good predication capability. When something unexpected happens that cannot be explained by its knowledge, the method tells what cannot be explained, which would support human decision making.

A unique expert system which uses a model-based display, is being developed by Beltracchi [15]. In this expert system a model-based display is identified, discussed, and illustrated. The model used in the display is based on the Rankine Cycle, a heat engine cycle. The individual control panels within a control room of a nuclear power plant contain meters, indicators, gauges, control stations, and switches, etc. The processing of the data of a heat engine cycle acquired from the panel could be presented in a Rankine cycle, which is depicted in a temperature versus entropy plot. In this plot the entropy values are not significant to the operator; however, the temperatures are useful. The data required for the Rankine cycles of all anomalies possible, depicted on a temperature versus entropy plot, could be obtained by assimilating anomalies in a thermal hydraulic system and storing them.

During operation, the anomaly is identified by matching the current Rankine cycle with the already stored ones, using different pattern matching techniques. Beltracchi claims that the Rankine cycle of each anomaly is unique, so that this method is feasible for diagnostics. In developing ESAS, methods such as the one used by Beltracchi could have been employed in order to decrease the number of suspects. However, the primary objective was to not specialize the analysis to a system or system type.

A part of the ESAS's knowledge base is the semantic network representation of the system being analyzed. The benefits of this type of knowledge representation is demonstrated in the expert system developed by Kitamura et al. [16]. In this knowledge representation the physical connections of components in a system are known. Also as its knowledge base, this expert system is provided with a list of causes of all possible malfunctions (primal events). The failure diagnosis consists of two stages: a signal processing stage for anomaly detection and an inference stage or primal events identification. The signal processing stage analyzes the differences between predicted and measured values of process signals to detect subtle changes in the signals induced by an anomaly. After verifying the signals, the causes of an anomaly are found by scanning through the knowledge database to determine the reasonable hypothesis consistent with observations.

Kitamura et al. [16] state that this exhaustive search might seem tedious and impractical, but that the search is performed quite rapidly, owing to the simple structure of the semantic network. They note that the applicability of the semantic network representation of plant architecture as a basic technique for failure diagnosis was confirmed through simulation studies of a PWR reactor. This expert system

is written in Prolog and implemented on an ACS-1000 computer (NEC Co.) of Tohoku University Computer Center and utilizes an upgraded Prolog Processor named SHAPEUP.

The knowledge contained in PRA models can aid in emergency response decision making, as noted by Dixon and Ferns [17] who state,

“PRA techniques are used by the nuclear industry to model the potential response of a reactor subjected to unusual conditions. The knowledge contained in these models can aid in emergency response decision making.”

PRA models can assist the operations personnel in both short and long term decision making. Given the current plant conditions, an on-line PRA could identify quickly which cooling sources and flow paths are available or unavailable. This would permit the operator to identify quickly which resources to use based on the priority list. In addition to the identification of current success paths, a PRA model can effectively identify the weak links in those paths, thus predicting the occurrence of a possible accident in the longer term [17].

Long term accident planning is useful for two main reasons [17]: (1) the current status of a component may degrade over a period of time (e.g., if ventilation systems are currently failed, components of other systems will overheat and fail over time) and foreseeing such failures without consulting PRA models can be easily overlooked, and (2) the identification of any single failure conditions supports comprehension of sudden changes in plant trends; namely, due to failure of a component or components, all success paths may depend on a single component. Identifying this component is crucial.

Dixon and Ferns [17] are developing an integrated set of computer programs

which utilizes AI techniques to develop, cultivate, and harvest information from PRA models composed of interrelated fault trees, event trees, response trees, and other forms. The integrated system is called FORESTER. Existing components of FORESTER include the Integrated Fault/event TRee Engineering (IFTREE), the SeQUence IMPortance calculator (SQUIMP) and System Un-Maintainability Assessment Code (SUMAC). IFTREE supports the graphical construction and assessment of fault trees and event trees coupled to fault trees. SQUIMP is a specialized code partially integrated with IFTREE. SQUIMP supports development and assessment of large event tree networks. SUMAC is a post processor which accepts cutsets and event probability and repair time distributions as input. SUMAC then produces event, cutset, and un-maintainabilities and un-maintainability importances for indicated repair periods. Un-maintainability is defined as the probability that the system, subsystem, or component of interest cannot be repaired in time T given that it is currently failed. The most recent development in FORESTER, is a specialized analysis capability being integrated into IFTREE called "subset assessment". A subset assessment of a model involves the specification of the events of interest and any cutoff parameters. The assessment results obtained from the model are those which meet the cutoff criteria and involve one or more of the events of interest. The cutoff criterion is basically a setting of a minimum probability for the event or events which could occur. This capability locates specialized groups of cutsets in an efficient and potentially real-time manner. According to Dixon and Ferns [17],

"There are many applications for subset assessment, including identification of risk associated with localized fires, terrorist attacks, maintenance errors, and any other problem involving a small group of components

in a large, complex facility. In the area of emergency response, subset assessment can identify the particular failure modes associated with equipment which is confirmed unavailable. It can also support (what if) activities.”

ESAS, even though the cutsets are obtained using a different method, uses a method similar to subset assessment in providing the user a “what if” resource.

3 A DESCRIPTION OF FAULT TREE ANALYSIS

As was mentioned earlier, ESAS calculates cutsets without performing the fault tree analysis. However, for the purpose of defining some terms used for describing ESAS's method of finding cutsets, a brief description of fault tree analysis is required. The description of fault tree analysis given here is taken from the Fault Tree Handbook [1]. The Fault Tree Handbook was written to serve as a text for the system safety and reliability course, given to over 200 nuclear regulatory commission personal and contractors and to make available to others a set of otherwise undocumented material on fault tree construction and evaluation.

3.1 Construction of a Fault Tree

In a fault tree analysis, one must properly define the system, decide which component failure causes an accident in the system (select the top event), form the fault tree for the top event selected, and resolve the fault tree for finding the cutsets corresponding to the chosen top event. Cutsets of a system are one or a combination of components whose failure results in the failure of the system.

To define a system, first the purpose of the system analysis must be determined. For example, the analyses of whether a system fails in a hazardous way, or whether the system will prove more costly than originally anticipated, require

different system descriptions. For the purpose of fault tree analysis, a system is defined as, “a deterministic entity comprising an interacting collection of discrete elements [1].” The term deterministic suggests that the system is identifiable. For example, the solar system is an identifiable system, but the entire universe is not. Furthermore, to define the interacting collection of discrete elements of a system properly, one must decide upon the boundary and resolution of the system. In deciding on boundaries of a system, one determines the comprehensiveness of the fault tree analysis, whereas the determination of resolution limits the detail of the analysis. Also, the interaction of elements in the systems of a nuclear power plants usually consists of a flow of electrical current, steam/water, etc. through all the components of a system. For the sake of simplicity, we will refer to the term “homogeneous flow” for describing a system with is a flow of only one type through the components of a system.

A fault tree depicts the logical interrelationships of basic events that lead to the undesired event, which is the top event of the fault tree. Table IV-1 in Vessely [1] contains the main components of a fault tree. Primary events are those events which are chosen not to be developed any further in a fault tree. Four types of primary events are, (1) basic events, (2) undeveloped events, (3) conditional events, and (4) external events. An intermediate event is a fault event which occurs because one or more primary or intermediate events cause it; in the representation, it propagates through the logic gates. In other words, a combination of intermediate events, combined and propagated by the usage of gates, result in another intermediate event or the top event. The logical relationship between the input events of a gate is decided by the gate type. Namely, an occurrence of the output event of an AND-

gate, requires occurrence of all its input events, whereas, in the case of an OR-gate the occurrence of the output event depends on the occurrence of any of the input events.

To better understand what primary and intermediate events are, let us distinguish between faults and failures. Faults and failures of components have different meanings in fault tree analysis. A failure is used when a component fails completely, whereas, a fault is used when a component does not function properly. Thus, all failures are faults but not all faults are failures. In forming a fault tree, one chooses a top event, and determines the events leading to it through a logic gate. The faults which lead to the top event become top events themselves. This procedure is repeated until the top event becomes the fault corresponding to the failure of a component. The failure of a component is referred to as a primary event. Therefore, the top event and all intermediate events are faults and the primary events are failures. It must be noted that a component can fail in different ways. These are called failure modes. The rules for constructing fault trees are [1]:

- Write the statements that are entered in the event boxes as faults; state precisely what that fault is and when it occurs.
- If this component fault consist of a component failure, classify the event as a “state-of-component”, otherwise, as “state-of-system”.
- If “state-of-component”, is the event add an OR-gate below the event. If “state-of-system” is the fault event, it may require an OR-gate, AND-gate, or no gate at all.
- If the normal functioning of a component propagates a fault sequence, then

it is assumed that the component functions normally (no miracle rule).

- All inputs to a particular gate should be completely defined before further analysis of any one of them is undertaken.
- Gate inputs should be properly defined fault events. Namely, gates should not be connected with other gates directly, and similarly events should not be connected with other events directly.

In summary, in a fault tree the events are termed “faults” if they are initiated by other events and are termed “failures” if they are the failure of a component. The gate output is the “higher” fault event under consideration and the gate inputs are the more basic “lower” fault or failure events. So, in constructing a fault tree, one proceeds from “higher” faults to the more basic or “lower” faults.

3.2 Evaluation of a Fault Tree

For resolving fault trees, it is assumed that the reader has a background in Boolean algebra. Readers wishing more information on Boolean algebra can refer to Vessely [1] or other common sources. Because gates relate events in exactly the same way as the Boolean operations, there is a one-to-one correspondence between the Boolean algebra representation and the fault tree representation. This process results in a Boolean expression. To find the list of cutsets, one could simplify this expression into sets of primary events related by OR-operators, and primary events within the sets are related by AND-operators. The simplification is done by following the rules of Boolean algebra depicted in Table VII-2 of Vessely [1].

In Figure 3.1, a typical pumping system is presented. The following construction and evaluation of the fault tree for this system, is obtained from Vessely [1]. The fault tree constructed for this system is presented in Figure 3.2. The symbols for the primary events of this system are shown in Figure 3.2. The Boolean expression derived from this fault tree is,

$$T = (A + B + C) . (C + A . B)$$

(Note: ANDs are represented by (.) and ORs by (+))

By following the Boolean algebra rules, one could simplify this expression into a list of cutsets, as is suggested in the following steps.

$$\begin{aligned} T &= (A + B + C) . (C + A . B) \\ &= A . C + A . A . B + B . C + B . A . B + C . C + C . A . B \\ &= A . C + A . B + B . C + A . B + C + A . B . C \\ &= C + (A . B) \end{aligned}$$

Note that (C) and (A . B) are the cutsets. The cutsets found imply, in order to not have any flow to the reactor from the tank, either valve C or pumps A and B must fail.

In a PRA study of a system, one is interested in the actual probability of occurrence of a top event. After obtaining the cutsets, the failure probability of the system or probability of occurrence of the top event, could be calculated by simply replacing ANDs and ORs with multiplication and addition respectively and assigning a failure probability to each primary event. For example, assuming that the probability of the failure of the tank and pump are 0.0001 and 0.02 respectively, then the failure probability of the pumping system depicted in Figure 3.1 is 0.0005.

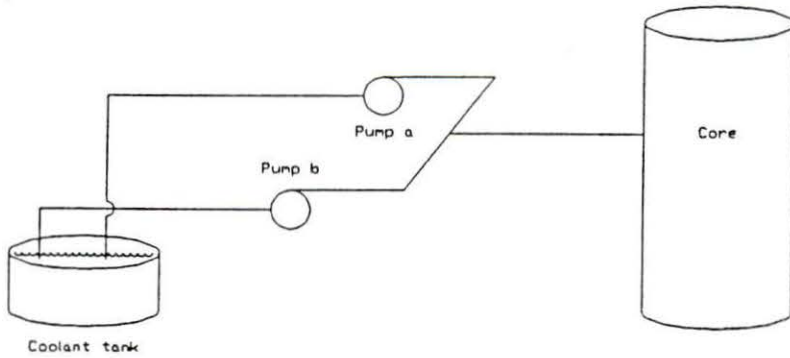


Figure 3.1: Configuration of a pumping system

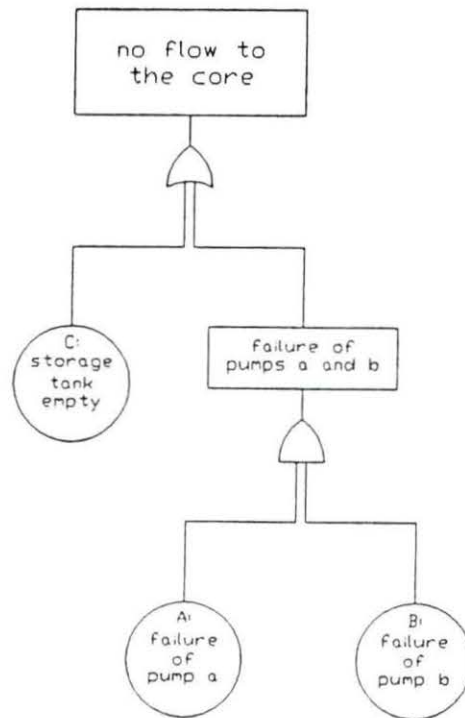


Figure 3.2: The fault tree for the pumping system

4 BASICS OF PROLOG

“Prolog (Programming in Logic) is a computer programming language that is used for solving problems that involve objects and relationships between objects” [18].

Prolog is based on Horn clauses. Horn clauses are developed to convey logic-based ideas in a written form. For many applications of logic, it is sufficient to restrict the form of clauses to those containing at most one conclusion. Clauses containing at most one conclusion are called Horn clauses, because they were first investigated by logician Alfred Horn [18]. Horn clauses are a subset of a formal system called predicate logic. Horn clauses are said to be of order zero, since at most one conclusion is allowed in them [19]. The reader can refer to Stanat and McAllister [20], for more on predicate logic and Horn clauses. It can be shown that any problem in logic can be expressed in a Horn clause form or as stated by Kowalski [19],

“The majority of formalisms for computer programming bear greater resemblance to Horn clauses than they do to ”non-Horn” clauses. In addition, most of the models of problems which have been developed in artificial intelligence can be regarded as models for problems expressed by means of Horn clauses.”

In Section 4.1, knowledge representation (facts and rules) in Prolog, which are represented in the form of Horn clauses is described. Iterative techniques in Prolog

are discussed in Section 4.2. Turbo Prolog [21] is the software used to develop ESAS. This software has some added features which are described in Section 4.3. There are many topics about Prolog which could be covered; but unfortunately the scope of this paper does not allow it. However, the material given here contains enough background information to help the reader understand the development of ESAS. The reader can also refer to Schildt [4], Shafer [22], Marcus [23], Malpas [24], and Clockstin and Mellish [18] for a more elaborate discussion on Prolog. The source for the the following is the Turbo Prolog 2.0 User's Guide [21]. Before beginning the discussion on Prolog it should be noted that example programs in Prolog appear in *italics*.

4.1 Knowledge Representation in Prolog

Knowledge representation in Prolog consists of declaring some facts about objects and their relationships, and defining some rules about objects and their relationships. These facts and rules are in the form of Horn clauses.

Facts consist of a predicate and its arguments. A predicate could be an attribute of or a relationship between arguments. For example, the following predicates are attributes of their arguments:

English	Prolog
Gold is valuable.	<i>valuable(gold).</i>
A fast car is fun.	<i>fun(fastcar).</i>
Valve a is open.	<i>open(valvea).</i>

In the *valuable(gold)*, *valuable* is the predicate and *gold* is its argument. Examples of some Prolog predicates which describe relationships of their arguments are illustrated:

English	Prolog
Bill likes Jane.	<i>likes(bill,jane).</i>
Bill likes Ann.	<i>likes(bill,ann).</i>
Bill likes John.	<i>likes(bill,john).</i>
Tom is Jack's brother.	<i>brother(tom,jack).</i>
Pipe a is connected to pump b.	<i>connected(pipea,pumpb).</i>

Prolog provides a concise notation to combine several objects into one. In the example above the predicates *likes*, which are in the form of Prolog facts, can be written in the form, *like(bill,[jane,ann,john])*. Square brackets are used to represent a list. Each item in the list is known as an element. The first element of the list is called the head and the tail is a list comprising all the subsequent elements.

Rules are the other form of knowledge representation. In the above examples, assertion of the facts is unconditional. Rules enable predicates to be conditional by usage of an “if” clause. For example, the Prolog rule for “Bill likes wine if the wine is white” is, *likes(bill,wine) if wine(white)*. In other words, a rule is a conclusion that is known to be true if one or more other conclusions or facts are found to be true.

A brief discussion of Prolog's syntax will lead to a formal procedure for developing facts and rules in Prolog. All predicates and their arguments are a continuous string of characters. Also, there are no size limitations on the length of predicates or arguments. Prolog was developed to easily convert logic-based ideas into a written form. The procedure commonly used in forming facts and rules is, (1) eliminate all unnecessary words from sentences, (2) transform the sentence, placing the relationship or attribute first, and (3) group the objects after the relationship.

Rules and facts can be generalized by using variables. They are represented by arguments which have an uppercase letter as their first character. For example,

```

valuable(gold).
valuable(diamond).
costsalot(Something) if valuable(Something)

```

implies that in order to prove that something costs a lot, first prove that it is valuable. Prolog operates by trying to match all predicates with a given predicate (the goal).

```

goal: costsalot(What).
Prolog's response: What = gold.
                  What = diamond.
                  True.

```

To prove *costsalot(What)* is true, first the *valuable(What)* has to be proven to be true. In other words, *valuable(What)* becomes the new goal. In the attempt to match the variable *What* with the argument found in each predicate *valuable*, Prolog will search from the top of the program to the bottom. When it finds a predicate that matches the goal, it binds the value to the variable so that the goal and the predicate are identical; the goal is said to unify with the predicate. This matching operation is called unification. After a unification, Prolog continues to search for more unifications, until there are no predicates to match, this is called backtracking. As can be seen, Prolog found two values for the variable *What*, since two *valuable* predicates were declared as facts. The relentless search for all solutions in Prolog is due to its backtracking capability.

Now that variables and unification in Prolog have been introduced, we will re-examine lists to review in more detail the terms head and tail of a list. Instead of separating elements with commas, one can separate the head and tail with a (*|*). For example, *[a,b,c]* is equivalent to *[a|[b,c]]*. The following examples demonstrate how Prolog performs list matching (list unification),

List 1	List 2	Unification of lists 1 and 2
$[X, Y, Z]$	$[jane, ann, john]$	$X = jane, Y = ann, Z = john$
$[car]$	$[X Y]$	$X = car, Y = []$
$[1, 2, 3, 4]$	$[X Y]$	$X = 1, Y = [2, 3, 4]$
$[a, b, c]$	$[Important _]$	$Important = a$

To conclude the discussion on knowledge representation, we must note that for proving rules, the value of some arguments of a predicate is not needed. Prolog allows this by use of the character $(_)$. Suppose in the predicate $letters([a,b,c])$, we are only interested in the head of the list. This is performed by attempting to unify $letters([Important|_])$ with $letters([a,b,c])$. Prolog replies with $Important = a$ and assumes that any element in the tail satisfies this unification.

4.2 Iterative Techniques in Prolog

Most conventional programming languages allow repetition by using FOR, WHILE, and REPEAT statements. Prolog does not contain iteration capabilities in these conventional forms. This discourages some Pascal, C, Basic, and Fortran programmers from using Prolog. Even though there is not a direct way of expressing iteration in Prolog, the power of this language is not restricted. Prolog has two kinds of repetition tools: backtracking, in which it searches for multiple solutions in a single query, and recursion, in which a procedure calls itself. Recursion is often said to be a “memory eater”. Prolog’s solution to this problem is to provide a special type of recursion called tail recursion which is compiled into an iterative loop in machine language. This means that although the program logic is expressed recursively, the compiled code is as efficient as it would be in Pascal or Basic. It has been stated in Reference [21],

“... recursion is, in most cases, clearer, more logical, and less error-prone than the loops that conventional languages use.”

In what follows, some techniques for conducting repetitive processes by backtracking, recursion, and tail recursion are described.

4.2.1 Backtracking

As was mentioned, a procedure backtracks when it looks for another solution to a goal that has already been satisfied. One can force backtracking by using the built-in predicate *fail*, which causes the rules containing it to always fail. It seems, in any logic problem, the objective is always to satisfy premises in order to be able to reach a conclusion. Why have rules which always fail? The following example explains the reason for this. Suppose facts are declared:

```
computerlanguage(basic).
computerlanguage(pascal).
computerlanguage(prolog).
computerlanguage(lisp).
computerlanguage(fortran).
```

The rule *printlanguages* utilizes the predicate *fail* to create a loop which prints all languages declared as facts above.

```
printlanguages if
    computerlanguage(X) and
    write(X) and
    fail.
printlanguages.
```

If the goal *printlanguages* is given to Prolog, it first looks for a solution for *computerlanguage(X)*, in other words, it tries to find a value for *X* by unifying the predicate *computerlanguage* with the available facts, then it writes the value for *X* and fails.

Since *write(X)* does not have any other solutions, it backtracks to *computerlanguage(X)*. This is repeated five times since five computer languages are declared as facts. After all the alternatives for *computerlanguage(X)* have been explored, the first *printlanguages* will fail. At this time prolog tries again and finds the second clause which is unconditionally true. In summary, the role of *fail* in this procedure is to tell Prolog to assume that a solution to the original goal has not been reached, and thus force it to backtrack and look for an alternative.

Another built-in predicate is *cut*, which has a function opposite of *fail*. This predicate is used extensively in recursive procedure. Where *fail* is used to promote backtracking, *cut* is used to stop it. Its value is always true, so it has no effect on the logic of the procedure. It acts like a diode in an electrical circuit. It allows flow of logic from top to bottom, however, if there is a predicate in a clause which is not true, it stops backtracking from bottom to top. When it is a waste of time and storage to look for alternative solutions or when the logic of the program requires the prevention of the consideration of alternative sub-goals, *cut* is used. For example in the rule, *rule if a and b and cut and c*, we are telling Prolog that we are satisfied with the solution it finds to the sub-goals *a* and *b*. Although Prolog is able to find multiple solutions to the call *c* through backtracking, it is not allowed to backtrack across the cut to find an alternative solution to the calls *a* and *b*. Also, it is not allowed to backtrack to another clause that defines the predicate *rule*.

4.2.2 Recursion

The other way of expressing repetition is through recursion. This procedure lends itself to solving problems where a more complicated case of the input argu-

The same procedure is followed until the problem is reduced to $factorial(1, Fact1)$. Now the unification with the initial condition is successful and the variable $Fact1$ gets the value of 1. The last clause which multiplies the $Fact1$ and 2 is reached which returns a value for $Fact2$. This is continued until a value for $Fact5$ is obtained and therefore $Answer$ is unified with a value. This procedure is possible because Prolog allocates memory to each argument of $factorial$ in all intermediate steps.

As illustrated above, the logic of recursion is easy to follow if one is not concerned with how the computer works. Prolog is so different from other languages that ignorance of what computers actually do to solve a problem is often an asset to the Prolog programmer. There is one drawback in using recursion: it is very memory intensive. The use of tail recursion eliminates this memory restriction.

In the example given above, the reason Prolog allocates memory to arguments of $factorial$ is to be able to conduct the last multiplication clause. If somehow the recursive call was made at the end (tail) of the procedure, there would be no reason for keeping track of intermediate steps. In tail recursion the procedure calls itself at its last step. In other words, the call is made in the very last sub-goal of the clause, and there are no backtracking points earlier in the clause. Following is a procedure for $factorial$, written using tail recursion.

```

factorial(N,FactN) if
    factorialaux(N,FactN,1,1).
factorialaux(N,FactN,I,P) if
    I <= N and NewP = P x I and
    NewI = I + 1 and cut and
    factorialaux(N,FactN,NewI,NewP).
factorialaux(N,FactN,I,FactN) if
    I > N.

```

Suppose the goal, $factorial(5, Answer)$ is given. In order to satisfy $factorial(5, Answer)$,

first Prolog must satisfy $factorialaux(5, FactN, 1, 1)$. The values for the factorial of I is P when the rule $factorialaux$ is called for the first time. In other words, factorial of I is P , is the initial condition. In this case I and P are one. Then the call to $factorialaux$ for the first time first calculates the factorial of one $NewP = P * I$, increments I by one, and finally calls itself. The second time $factorialaux$ is called the factorial of two is calculated. This process is repeated until I is incremented to 6. When the new goal becomes $factorialaux(5, FactN, 6, 120)$, I which is six becomes larger than N which is five and the first clause for $factorialaux$ fails. However, the second $factorialaux(5, FactN, 6, 120)$ succeeds and $FactN$ gets the value of 120. The use of *cut* insures no backtracking after the sub-goal $I \leq N$ fails thus the first clause for $factorialaux$ fails. The advantage of this *factorial* is that the rule calls itself as its last sub-goal, therefore, there is not a need to store the information obtained in the intermediate steps of the recursion.

4.3 Turbo Prolog

Prolog software used to develop ESAS is Turbo Prolog 2.0 developed by Borland International, Inc. [21]. Minimum hardware requirements for this software are:

- IBM PC, XT, AT, PS/2 or true compatible
- 284k RAM internal memory minimum, though 640k is recommended
- PC-DOS or MS-DOS operating systems, version 2.0 or later

This software has some added features which make it more attractive than many other expert system tools. Some of these features used in developing ESAS are:

- External database system, with over 30 built-in predicates for developing and maintaining large databases
- Window management tools, with several built-in predicates which create and manage virtual or normal windows
- High resolution video support
- Turbo linker which makes module programming possible
- Turbo Prolog Toolbox which is a collection of already written modules in Prolog and C language used for some common tasks
- Most importantly, low cost

5 DEVELOPMENT OF ESAS

The acquisition and representation of knowledge and methods used for constructing the rules ESAS uses to perform its tasks are discussed in this chapter. Tasks ESAS is capable of accomplishing are as follows. Given the input and output components, ESAS is capable of finding paths through a system. In a search for paths, components could be specified as failed and thus bypassed in the search. This feature of ESAS enables it to accomplish its most important task, which is finding the cutsets of a system. As previously mentioned, cutsets are a combination of components whose failure causes the failure of the system. Given the failure rate of each component and knowing the cutsets, ESAS is then able to calculate the failure probability of the system. Also, by use of the cutsets found and stored, ESAS is able to diagnose a system by forming a failure menu tree. The failure menu tree contains the components whose failure causes the accident depicted in the root of the tree. Finally, as part of system diagnostics, "what if" scenarios are provided, where the user can ask whether there is a path through the system if components or a group of components have failed given the input and output components.

The acquisition and representation of the information about a system stored as the database is described in the Section 5.1. Facts used and rules developed for finding paths through a system, finding cutsets, and diagnosing a system are

described in Sections 5.2, 5.3, and 5.4, respectively. Finally, in Section 5.5, the interactive features of ESAS with the user are discussed.

5.1 Acquisition and Representation of Information

The knowledge acquired by ESAS for performing its tasks is encoded into Prolog predicates in the form of facts. Knowledge of programming in Prolog is not required for using ESAS. Information is acquired in a user friendly fashion and the database is developed and saved in the data files. Editing the database stored in the data files is also made available to the user by ESAS. It is strongly recommended that the user avoid editing the database directly and only edit the database by using ESAS, since files storing the database can easily become invalid if not manipulated by ESAS. A description of the information acquired and its representation, encoding facts into Prolog predicates, are discussed in this section.

5.1.1 Semantic Network Representation of a System

The major component of the knowledge acquired by ESAS is the semantic network representation of a system. In such a network objects are represented as nodes and the relation between them as links. A semantic network is in the form of a tree or a graph. Trees are formed from nodes connected to one another in a fashion such that the branches cannot be connected, whereas in graphs any node can be connected to any other nodes or to itself. Graphs and trees can be either directed or non-directed. In the directed case, a link specifies the direction of the relationship between the nodes it connects, whereas in the non-directed graph or tree a link relates nodes in both possible directions.

In general, systems in a nuclear power plant could be represented in a directed graph form, where nodes are components and links are the physical connections between the components where there is a flow of some type. For example, in a pumping system a pump and a pipe could be specified as nodes related by a flow of coolant in the direction, pump to pipe. Therefore, ESAS when presented the semantic network representation of a system, makes two assumptions. It assumes, all the links of the semantic network representation of a system constitute a flow of some type. For example, in a thermal hydraulic system the flow of steam/water and in an electrical system a flow of electrical current must be present throughout all the links of the semantic network representation. Also, it assumes that one and only one type of flow is allowed in a system.

However, by making these two assumptions, components such as sensors and relays, which are not linked to the system by a "flow connection", can not be included in the semantic network representation of a system. To incorporate the components which can not be linked to the system by the flow connection, "non-flow connection" links are introduced. Even though, components linked by non-flow connections do not directly influence a flow path through a system, they manipulate other components, linked by flow connections, which do influence the flow path of a system. In other words, it is crucial to include the components linked to the system with non-flow connection, since they play an important role in the function of a system. The generic term "activator" is used for components which are linked by non-flow connections in this thesis and in developing ESAS.

The following examples will hopefully clarify the concept of using the non-flow connections. If a pump is functioning properly and is activated by a sensor, there

could not be any flow through the pump if the sensor has failed. In other words, there is a need for non-flow connection links for properly representing systems containing a sensor. Another example for activator components is the relays which are used in electrical systems. The function of relays, which is changing the operational mode of the switches in the system, can be simulated by use of non-flow connection links. We will address the mode of operation of switches and valves in Section 5.1.2.

For properly representing a system in a semantic network form, some considerations must be made on deciding on the boundaries and limit of resolution of the system. Inside the boundaries of the system all components must be linked to the system by either flow or non-flow connections. Moreover, as was previously mentioned, flow of one and only one type through all components linked by flow connections must exist. Systems in nuclear power plants which have more than one type of flow can be represented by deciding on the boundaries of sub-systems where there is one and only one type of flow. For example, a system which is a combination of a thermal hydraulic and an electric systems (i.e., the semantic network representation requires specifying links describing the flow of both (steam, water) and electrical current), could be presented as two separate sub-systems and analyzed by simply deciding on the boundaries of the two sub-systems. For finding the cutset of a system, sometimes there is a need for including an output node which physically does not represent a component in a system. The rules for scenarios where such an output node is required are discussed in the Section 5.3.

Components in a system are represented by nodes. Usually, a component of a system itself consists of sub-components which in turn consist of sub-parts. Therefore, nodes in a semantic network actually represent macro-components which

consist of components which in turn consist of sub-components. In an analysis of a system, by deciding on the limit of resolution, the size of the macro-components are determined. If the limit of resolution is high, the analysis can be done with high specificity; however, the analysis is more time consuming and tedious. On the other hand, if the limit of resolution is low, the analysis can be done faster but less accurately. By studying the importance of the components in a system, one can compromise between the specificity of the analysis and the time it takes to perform an analysis. For example, in a thermal hydraulic system, one can reduce the number of components in a system by excluding the pipes from the semantic network representation or by considering them as a part of the components whom are connected to. Even though this measure is justifiable, due to the low failure probability of pipes relative to other components, the analysis can not account for the accidents caused by failure of pipes. Therefore, in deciding on the limit of resolution, one must consider the desired specificity of the analysis of the system. Readers wishing for a more elaborate description on the limit of resolution can refer to Vessely [1].

In summary, to represent a semantic network representation of a system, one must decide on the boundaries of the system, and this decision must be made considering that flow of one and only one type must exist through all components linked by flow connection. Also, those components in the system which are not linked to the system by a flow can be linked to the system with the non-flow connections. Finally, considering the specificity of the analysis and time required to conduct an analysis, one must decide on the limit of resolution. The resulting semantic network representation of a system then should be a deterministic entity comprising

an interacting collection of discrete elements [1].

5.1.2 Other System Information

Other than the semantic network representation of the system, additional information acquired by ESAS includes the mode of operation of the “gate type” components (e.g., valves and switches), the failure modes and failure rates, and the advisory text for diagnostic purposes.

In a semantic network representation of a system, a component could either fail or succeed. This implies that in a search for paths they are avoided if failed or passed if not failed. However, components such as switches in electrical systems and valves in thermal hydraulic systems, if failed could still allow flow. For example, if a failed switch is closed prior to the system operation, it would still allow flow of electrical current, since it is “failed-closed”. Therefore, in order to properly take the failure of gate type components into account, ESAS acquires the two modes for each gate type component. A mode specifies whether or not flow is allowed through a gate type component. For example, specifying an open mode of a switch implies that the flow is not allowed, whereas specifying a closed mode implies that the flow is allowed through the switch.

Two modes are acquired for each gate type component, initial and operational. The initial and operational mode specify the mode of the gate type component prior to and during the operation of the system respectively. The rule used for deciding which specified mode to use is: if the gate type component is assumed failed then the initial mode is used else the operational mode shall be used where each mode can be specified either allowing flow or not allowing flow. In the case where the

modes of operation of the switches are altered by relays, the initial and operational modes correspond to the mode enforced by relays when they are energized and de-energized, are used, respectively. For example, in the case where a manual switch is closed to begin the operation of the system it is connected to, the initial and operational mode of the switch can be specified as open and closed, respectively.

The semantic network representation and the gate type component specifications are used for the analyses, including finding paths, searching for cutsets, and diagnostics. Therefore, it is crucial to provide them to ESAS for it to be able to perform any tasks. The failure modes, and standby and operational failure rates, described next, are only used for calculating the failure probability of the system; thus if not presented to ESAS, the analyses, finding paths and cutsets and performing diagnostics can still be conducted.

The standby and operational failure rates are defined as rates at which a component can fail when the system is not operating and operating respectively [1]. These rates are used for calculating the failure probability of a component. The method used for calculating the failure probability of a system is described in Section 5.3.

In Chapter 3, we defined the failure modes of a component. Failure modes are the different ways a component can fail. For each failure mode, an operational failure probability rate must be specified. A table containing different failure modes and their corresponding operational failure rates for most common components in a nuclear power plant is provided to the user by ESAS. The data for this table are obtained from WASH-1400 [2]. Data required to calculate the failure rate of a system for components not included in this table could be entered separately.

The information described to this point is acquired from the user and stored in

the database. After finding the cutsets, ESAS stores them in the form of facts for diagnosing the system and calculating the probability of occurrence of an accident as part of the system database. Also, for diagnostic purposes, a text explaining the corrective actions of each component can be entered, and is stored in the database. ESAS then provides the text entered for a component when that component is found to be the cause of an accident in the diagnostic mode. Turbo Prolog's text editor is used for entering the advisory text.

5.1.3 Representation of Facts in ESAS

The predicates used for storing the information ESAS acquires, mentioned in the previous two sections, are listed below.

```

component(component name, component type, list of failure modes,
           list of operating failure probability rates,
           standby failure probability rate, component number)
connected(input component, output component)
activator(component name, list of components whom the activator
           manipulates, input component for the activator, output
           component for the activator)
gate(component name, initial mode, operational mode)
advisory(component name, failure mode, text)
cutset(type, list of output components, list of input components,
        cutset)
found(type, list of output components, list of input components,
        maximum number of components in a cutset search for)

```

For all of the components in a system, the predicate *component* is stored in the knowledge base. The first argument of the predicate *component*, *component name* is a string used as the key for searching the database. In order to be able to properly search the database, in acquiring the *component names*, ESAS does not allow multiple uses of the same name (i.e., each name must be unique). Also, there

is a limit on the number of characters of the argument *component name*, namely 20, to reduce the database search time.

The argument, *component type* could be used to describe the component type or other information. This argument is also specified as a string; however, the limit on the number of bytes this string could have is 64K, which is the maximum allowable size of a string type arguments specified by the Turbo Prolog compiler [21]. The argument, *failure modes* is a list of strings which contains the failure modes of the component. For each failure mode, the corresponding operational failure rates are stored in the argument *operational failure probability rate*, specified as a list of real numbers. The *standby failure probability rate* is then used to store the standby failure probability rate, declared as a real number. The argument *component number* is an integer which is used for sorting the *component* predicates in the database.

The predicate *connected* is used to represent the flow connection links. Its first argument is the *component name* of the input of the link and the second argument the output. The predicate *activator* is used for specifying components which are linked to the system by non-flow connections. The third and fourth arguments of this predicate are used for representing relays and they contain the name of components which are the power source and the ground for the relay, respectively. Gate type components are represented by the predicate *gate* which stores the initial and operational mode as its arguments. The text used after diagnosing a system for advisory purposes is stored in the predicate *advisory*. This text is stored as the third argument of the predicate *advisory*, which is declared as string and thus has a maximum size limit of 64K bytes.

The cutsets found are stored in the predicate *cutset*. The predicate *found* is stored in the database for informing ESAS that a search for cutsets has been performed and therefore preventing a redundant search even if no cutsets were found in the search. The information contained in the arguments of these components is described in Section 5.3.

The pumping system depicted in Figure 5.1 is identical to the one in Figure 3.1; however, to demonstrate the representation of an activator and a gate type component, components *sensor* which activates *pump a* and *pump b* and *valve* which is connected to the output of *pump a* and *pump b* and the input of the core, specified to be closed in the initial mode and open in operational mode, are included. In presenting the information for the pumping system, the failure rates were not given, thus, the value of zero is assigned by ESAS. The predicates developed for storing the information entered for the pumping system depicted in Figure 5.1 are presented below.

```

component(tank,storage tank,[tank is empty],[0],0,1)
component(pump a,pump,[pump failure.failure due to no power],[0,0],0,2)
component(pump b,pump,[pump failure.failure due to no power],[0,0],0,3)
component(valve>manual valve,[failed closed],[0],0,4)
component(output,core,[],[],0,5)
component(sensor,low level sensor,[sensor failure],[0],0,6)
connected(tank,pump a)
connected(tank,pump b)
connected(pump a,valve)
connected(pump b,valve)
connected(valve,output)
activator(sensor,[pump a,pump b],none,none)
gate(valve,closed,open)

```

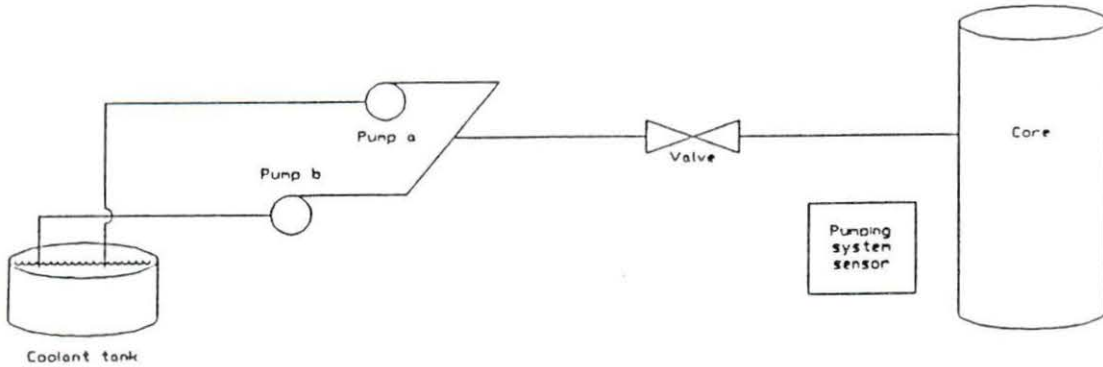



Figure 5.1: The configuration of the modified pumping system

5.2 Rules for Finding Paths Through a System

As was mentioned in the previous section, we represented the semantic network of a system in the form of a directed graph. In defining a graph, branches could be connected. This suggests that there is a possibility of the existence of a loop in a graph. A loop is defined as a path in which a node is visited more than once. Therefore, tracing a graph as it is represented will not be successful in the cases where a loop exists. However, in trees, no loops are allowed by definition, since branches cannot be connected. Thus, by transforming a graph representation into a tree representation of a system, the search for a path could be done without the fear of encountering loops. This transformation is demonstrated in Figure 5.2. Kowalski [19], could be referred to for the topic of graph transformation.

There are several techniques used for tracing through a tree, one of which is called depth-first search. The essence of this technique is to pick some alternative at every node visited and to work forward from that alternative. Other alternatives

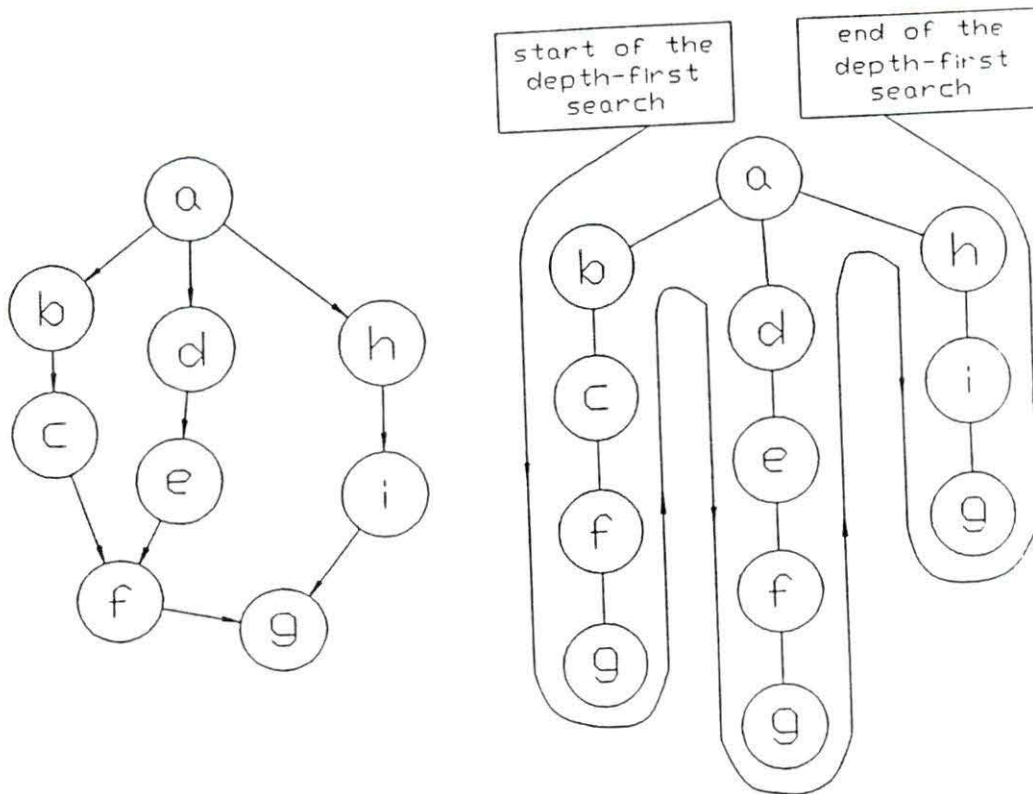


Figure 5.2: A tree representation of a graph

at the same level are ignored completely as long as there is a hope of reaching the destination by use of the original choice. The starting point for this search is the root of the tree or the top node and going forward constitutes moving down and left as far as possible. For example, in finding a path between nodes (a) and (g) of the tree depicted in Figure 5.2, this method searches through the tree, as is demonstrated. Paths, in the order discovered, are (a,b,c,f,g), (a,d,e,f,g), and (a,h,i,g).

The task of transforming the graph into a tree could be accomplished by avoiding re-visitation of a component in a search for a path. This simple rule of thumb, to not allow the revisiting of a component in exploring a path, could easily be programmed into a Prolog rule which conducts a depth-first search. This rule uses

the directed graph representation of a system and finds paths without falling in the traps of loops which can exist in graphs. ESAS's Prolog procedure for tracing through a directed graph is,

```

findpath(_,Dest,[Dest|Tail],[Dest|Tail]).
findpath(Start,Dest,[Last|Tail],List) if
    connected(Last,X) and
    not(member(X,Tail)) and
    opengate(X) and
    findpath(Start,Dest,[X,Last|Tail],List).

member(X,[X|_]).
member(X,[_|Y]) if member(X,Y).

```

The rule *findpath* is written in a tail recursive form; thus, it is not memory intensive. Using a tail recursive procedure for finding paths is essential, since systems with a large number of components and connections soon exhaust the stack memory storage required for ordinary recursive procedures. Refer to Chapter 4 for a description of tail recursive procedures.

The first and second arguments of the predicate *findpath* are the start and the destination nodes, respectively. The third argument is a list of nodes already visited in the path being explored and other nodes specified to be avoided in the search for a path. By searching the database for connected predicates, the next node to be visited *X* is found. The existence of the node *X* in the path is then examined by using the procedure *member*. The procedure *member* is also tail recursive. Its task is basically to see if the node *X* specified in its first argument is a member of the list in its second argument which contains the list of nodes already visited. By doing so, circling in loops of a graph in a search for a path is avoided.

The procedure *opengate* is not listed here since it is rather lengthy. The task of this procedure is to check whether the node specified as its argument is a gate

type. This is accomplished by checking the database for predicates *gate*. If X is not a gate type component then *opengate* automatically succeeds. However, if it is a gate type component then *opengate* decides whether the gate type component is in the mode which allows flow or vice versa. The *opengate* procedure is written to cover various scenarios possible concerning their connections with the system. In developing ESAS, it was intended to not specialize on a type of system. For accomplishing this, all possible forms a gate type component functions had to be considered. However, it is impossible to cover all scenarios, which in turn resulted in restricting the use of ESAS to systems that contain gate type components which are considered in *opengate* procedure. The scenarios covered concerning gate type components are valves or switches, valves which are connected to sensors, switches which are connected to relays, and switches whose mode of operation is altered by activator components other than relays.

The path finding procedure calls itself until the boundary condition displayed in the first predicate of this procedure succeeds. This predicate simply checks whether the destination is reached. If the destination is reached then it returns the path found by its fourth arguments. If the destination is not reached, the second predicate is called again. The backtracking feature of Prolog enables the path finding procedure to explore all the possibilities for existence of a path in a depth-first search manner.

5.3 Rules for Finding Cutsets and Failure Probability of a System

In this section the methods used for developing the rules for finding the cutsets are discussed. Two methods are proposed for finding the cutsets of a system. The

two methods are derived from the simple observation that systems in nuclear power plants fail in two ways. In the first case, no flow from input component(s) to the output component constitutes an accident, and in the second case the opposite constitutes an accident. Finding the cutsets in the first case is called the "no-flow analysis" and the second case "flow analysis."

Generally, the no-flow analysis is performed for thermal hydraulic systems where no flow to the output component causes an accident and flow analysis is performed for electrical systems, consisting of a combination of relays and switches, designed to block the flow of electricity to the output component to stop its operation. Also, an analysis which performs both flow and no-flow analyses could be performed for thermal hydraulic systems which have a test loop. Thus, in normal operation a flow to the normal output component with no flow to the output component of the test loop are desired. These analyses for finding cutsets cover most accident scenarios possible in nuclear power plant systems. The use of ESAS for finding cutsets is restricted to the above analyses, even though it was intended to make the use as general as possible.

When the type of analysis is acquired from the user, ESAS asks for the output component of the system. For performing either flow or no-flow analysis, specification of one and only one output component is allowed by ESAS. In the analysis where the combination of both flow and no-flow is desired, specification of two output components are allowed (i.e., one and only one output component to which flow and another to which no flow is desired). As is commonly known, most systems in nuclear power plants have more than one output component. However, the semantic network representation of those system with more than one output component

can be manipulated to have only one output component, by considering the type of analysis desired. Only four cases emerge. In the case where no flow to all of the output components of a system causes an accident, no-flow analysis could be performed by specifying each output component separately and performing the no-flow analysis for all of the outputs component individually. Then the accumulated cutsets obtained from the no-flow analysis for each output component is the list of cutsets desired. In the second case, where no flow to any of the output components of a system causes an accident, a node which physically does not represent any component in the system can be specified as the output node by linking it as an output of all the actual output components. The third case, where flow to all output components of a system causes an accident, is analogous to the second case and therefore the same manipulation of the semantic network could be applied. And finally, the fourth case where flow to any of the output components causes an accident is analogous to the first case. Thus, the same procedure could be followed.

The task of specifying the input components to the system is less complex. For no-flow analysis, normally all the components which are not an output of any other components are the input components of the system. For performing flow analysis of electrical systems, normally, power sources can be specified as the input components of the system.

In specifying the input and output component(s), a great deal of care must be used since it could affect the result of the analysis performed for finding cutsets. Afterwards, acquiring the type of analysis desired, the output component, and input components ESAS begins the actual search for finding the cutsets.

First, ESAS examines the type of analysis selected to be performed. If no-

flow or flow analyses are selected, assuming all the components are functioning properly, there must be a path or no paths between the specified input components and the output component of the system. If these conditions are not met then ESAS terminates the search for cutsets immediately; otherwise it continues with the analysis.

Cutsets are a list of components whose failure causes the failure of the system. This definition is used to introduce a rule of thumb by which ESAS finds cutsets. In the case of performing no-flow analysis, assuming an arbitrary list of components have failed in the system; if there are no paths between the input components and the output component, then by definition, this arbitrary list of components is a cutset. On the other hand, in performing flow analysis, if there is a path, then the list of the arbitrary components is a cutset. This rule of thumb, although very elementary, is extremely successful for finding the cutsets as will be demonstrated in the next chapter where four systems which are similar to most systems in nuclear power plants are analyzed.

When the path finding procedure is called to explore the possibility of the existence of a path, the list of assumed failed components, except the gate type and the activator type components, are transferred by use of the third argument into the *findpath* procedure. As was mentioned in the previous section, in a search for a path the components included in this list are avoided. In Section 5.1, the activator components were defined to be components which do not directly influence the flow path of a system, however, they manipulate other components which do. To simulate the failure of an activator component, the list of components, excluding the gate type components, which are manipulated by the activator component are included

in the list of components to be avoided in a search for a path. In transferring the list of components assumed failed, we include all components, except the gate type components, since failure of gate type components require a different treatment. Namely, to simulate the failure of the gate type components, the specified initial mode is used to decide whether there could be a flow through this component as was explained in Section 5.1.

ESAS is capable of finding cutsets containing up to six components. It first begins by searching for cutsets containing only one component. If there are N components in the system then there are $N-1$ suspects containing only one component, since the output component is not considered as a possible cutset. By calling the path finding routine $N-1$ times, ESAS can then find all the cutsets containing one component. Then, in the search for cutsets consisting of two components, there are $(N-1)!$, factorial of $(N-1)$, lists of components as suspects. However, the number of suspects are reduced by eliminating all those on the list which are a super set of the cutsets containing only one component previously found. The definition of the super set is that set A is the super set of set B if all of the elements in set B are a member of set A . The elimination of these lists are obvious since, if component A is found to be a cutset then the failure of the combination of component A and any other components in the system can also cause the failure of the system; however, this combination is not a proper cutset. Also, lists containing a component twice are eliminated, since they are the same as the list with only one component which has already been considered as a possible cutset. Thus, after forming the lists containing two components and eliminating the improper ones, the path finding routine is called to find the actual cutsets.

In forming lists composed of components in the system as suspects for being a possible cutset, all of the components (except the output component) are included for no flow analysis. However, in flow analysis where a flow to the output component constitutes an accident, only the activator and gate type components are included in the list of suspects. This becomes apparent by realizing that when a flow to the output component constitutes an accident, components such as a wire which should allow flow at all times can not be assumed failed. This is referred to as the "no-miracle rule" in Vessely [1]. Thus, only the components which can enforce no flow under normal operation can be considered to fail in the flow analysis.

The same procedure is followed for finding lists containing three to six components as proper suspects. The path finding routine is called and the actual cutsets are found. As can be seen, the number of suspects increases factorially, as the number of components in the system or in the list of suspects increases. In order to decrease the analysis time, it is imperative to properly decide on the limit of resolution, which in turn determines the number of components in the system. Also, for reducing the analysis time, in most analyses the failure probability of components in the cutsets with more than two components becomes insignificant; thus ESAS enables user to specify the maximum number of components in a cutset desired in order to avoid searching for cutsets containing higher number of components.

The cutsets found are then stored. Thus a search for cutsets is only performed once and the user can search for cutsets containing a higher number of components without repeating the search for the cutsets previously found and stored. Cutsets must also be stored for diagnostic purposes where the response time becomes a major factor.

A no-flow analysis was performed on the pumping system depicted in Figure 5.1, specifying components tank and core as the input and output components of the system, respectively. In this analysis, ESAS was asked to search for lists of component(s) whose failure cause no flow from the storage tank to the core. That the failure of the components, tank or valve or sensor or (pump a and pump b), causes the system to fail was concluded from this analysis. The Predicates used to store the Prolog facts representing the cutsets found are:

```

cutset(no-flow,[output],tank,[tank])
cutset(no-flow,[output],tank,[valve])
cutset(no-flow,[output],tank,[sensor])
cutset(no-flow,[output],tank,[pump a,pump b])

found(no-flow,[output],tank,2)

```

After finding the cutsets, the failure of the system can be calculated. ESAS acquires the operational and standby time of the system. Then the product of the sum of the operational failure rates and the operational time added to the product of the standby failure rate and standby time of the components in a cutset is the failure probability of the components in the cutset. By adding the probabilities obtained for all cutsets, the failure probability of the system is obtained. Even though, in developing ESAS, the major task was to develop rules for finding cutsets, developing a procedure for finding the failure probability of the system turned out to be more challenging. This is due to the fact that problem areas where symbolic manipulations are required lend themselves to programming in Prolog but not to numerical manipulations. This is the major factor for making the quantitative analysis as simple as possible.

5.4 Rules Developed for Diagnostics

As was cited in Chapter 2, cutsets found could be used for systems diagnosis. By definition, cutsets are a list of components whose failure causes an accident. When an accident occurs, the user is interested in knowing the possible causes, which in turn are the cutsets. ESAS provides the cutsets in the form of a menu tree with items displaying the cutsets which contain one and two components. The user then can select an item to be displayed, and will be shown the failure modes and the text entered for explaining the suggested corrective actions. This could be used as an on line tool for quick reference to technical specifications for corrective actions in the case of a failure of a component (provided this information is previously entered into ESAS).

Also as part of diagnostics, to examine the importance of successful operation of component(s) in the system, the path finding procedure could be directly called by the user. ESAS acquires the input and output components, and the components which are assumed failed, and displays the paths found (if any). The components assumed failed are treated as was mentioned in the previous section. This feature becomes attractive for studying complex and large systems. For example, the large drawings of reactor trip systems which depict hundreds of components such as relays, switches, diodes, etc. could be used once to enter the semantic network representation of the system. Then ESAS could be used to trace through the system searching for paths from any component to any other component with the option of bypassing component(s).

5.5 Interacting with ESAS

A useful expert system must possess a non-intimidating and user friendly interactive capability. As a matter of fact, in theory, there should not be a need for a user's manual when employing an expert system. All actions taken by an expert system must be explained, warnings must be given to the user when necessary, while the expert system is in use. In other word, the manual for using an expert system must be provided by the expert system itself, so that the expert system is comparable to human experts in interacting with the user. In this section, a description of how ESAS can interact with the user is provided. However, by no means can this brief description be used as a manual, since ESAS is developed to display the explanatory texts, warnings, and other necessary information while being used.

In order to make ESAS user friendly, it is developed to be menu driven. The diagram depicting the various options of ESAS is presented in Figure 5.3. When scanning the menu, a text explaining tasks performed by selecting the option under consideration is presented to the user. The actions of ESAS are partitioned into manipulating the database, finding cutsets, and performing diagnostics. Manipulating the database consists of tasks, CONSULTing, CREATing, EDITing, DELETing, and VIEWing the database. Needless to say, before editing or viewing a database, finding cutsets, or performing diagnostics, the database of the system must be created, or if already created, it must be consulted.

When creating a new database, the first information acquired by ESAS is the system name. The system name is used as the title of the files created by ESAS for storing the database. Thus, the database of a system is referred to by the assigned

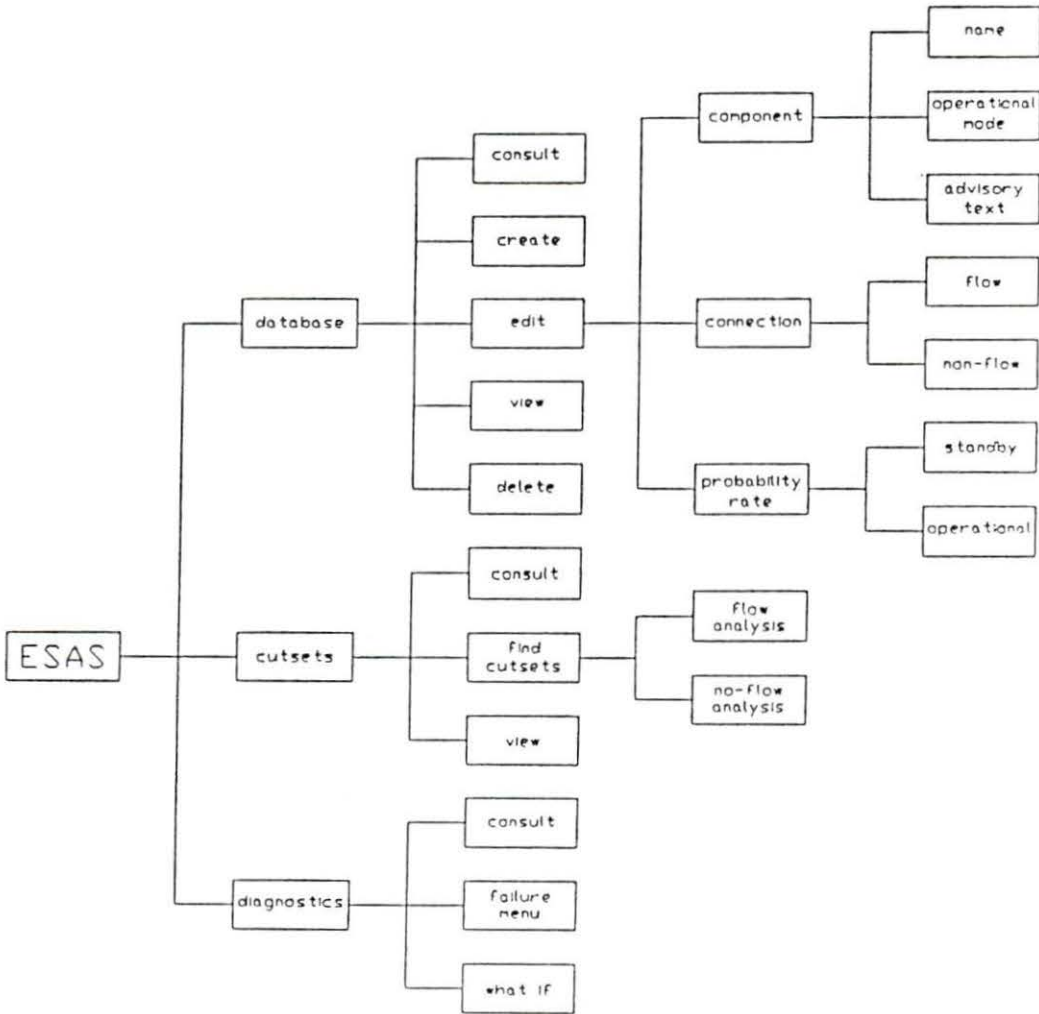


Figure 5.3: The menu system of ESAS

system name. ESAS acquires information by use of virtual windows, where page editing is provided, or by use of menus. For example, in acquiring the component names, ESAS creates a virtual window which contains N fields, where N is the specified number of components in the system. The user edits the entire window when entering the component names. After all component names have been entered, ESAS searches for improper entries (e.g., a component name entered twice). Then, ESAS encodes the component name into the corresponding predicates to be stored in the database, in the form of Prolog facts. The component names need to be entered only once, since a menu containing the component names is created to be used for acquiring other information such as specifying gate type components or connections.

In general, when using ESAS, the user rarely needs to enter information without use of menus and when menus are not provided page editing capabilities are provided. In addition to displaying the database on screen, ESAS is able to write the database into data files or produce printed copies of it. ESAS always asks for information from and replies to the user in English sentences, not Prolog predicates.

The tasks performed by choosing the second or third option (i.e., finding cutsets or performing diagnostics), discussed in previous sections, are systematic and self explanatory.

6 ANALYSIS OF NUCLEAR PLANT SYSTEMS

Four systems were chosen to demonstrate the features of ESAS and to illustrate the success of ESAS in finding cutsets.

6.1 An Emergency Core Cooling System

A simplified emergency core cooling System (ECCS), depicted in Figure 6.1, obtained from Lewis [25], was analyzed by ESAS. The pipes in this system were excluded from the analysis for simplifying this demonstration. As can be seen, the semantic network representation of this system includes nine components, with the coolant tank and core as the input and output components of the system and the valves a, b, c, and d as gate type components which are specified as closed prior to and open during the operation of the system. The protection system sensor is specified as an activator type which activates pumps a and b. Considering the accident of having no flow to the core from the coolant tank, ESAS was asked to find cutsets containing a maximum of four components and found the cutsets to be, (coolant tank) or (protection system sensor) or (pump a and pump b) or (pump a and valve c and valve d) or (pump b and valve a and valve b) or (valve a and valve b and valve c and valve d). These cutsets can easily be verified by inspecting the ECCS. Also, they were verified by performing a fault tree analysis for the top event,

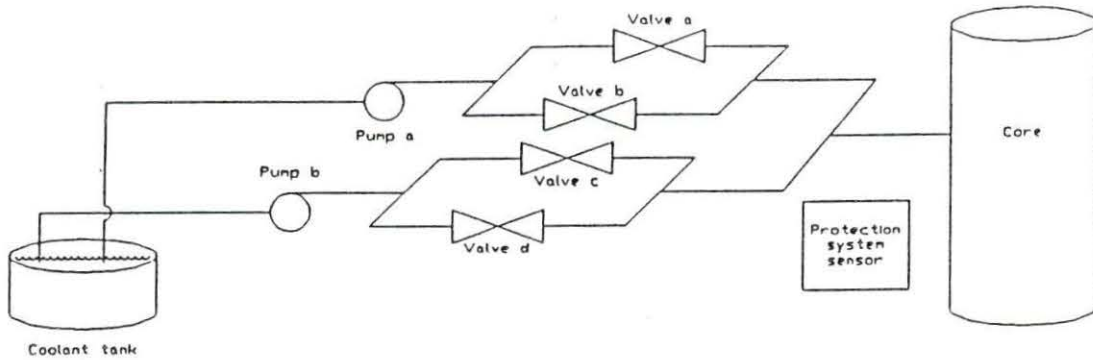


Figure 6.1: The configuration of ECCS

no flow to the core. The fault tree constructed for ECCS, obtained from Reference [26], is depicted in Figure 6.2.

To demonstrate the diagnostics features of ESAS, a search for paths between the components coolant tank and core was done, assuming the failure of valves a and d. Paths found were (coolant tank, pump a, valve b, core) and (coolant tank, pump b, valve c, and core).

To perform diagnostics for the accident, no flow to the core, ESAS formed the failure menu tree depicted in Figure 6.3. From this menu, the user can choose the failure of a single component and will be presented with the failure modes of that component previously entered in the database. For each failure mode, an advisory text can be put in the database; by consulting this, the user can take corrective actions. For example, this text can contain the technical specifications related to the failure of a component.

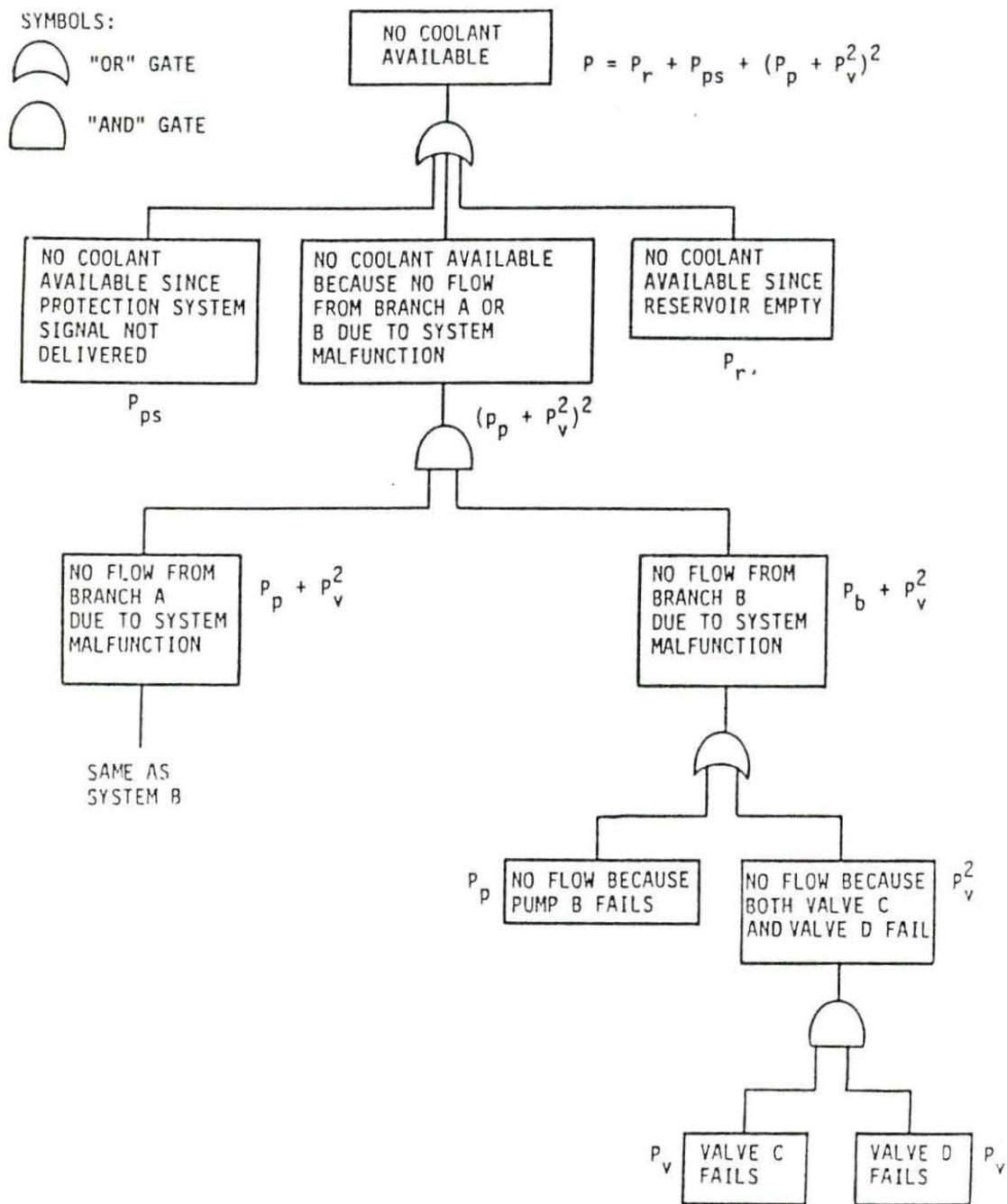


Figure 6.2: The fault tree constructed for ECCS

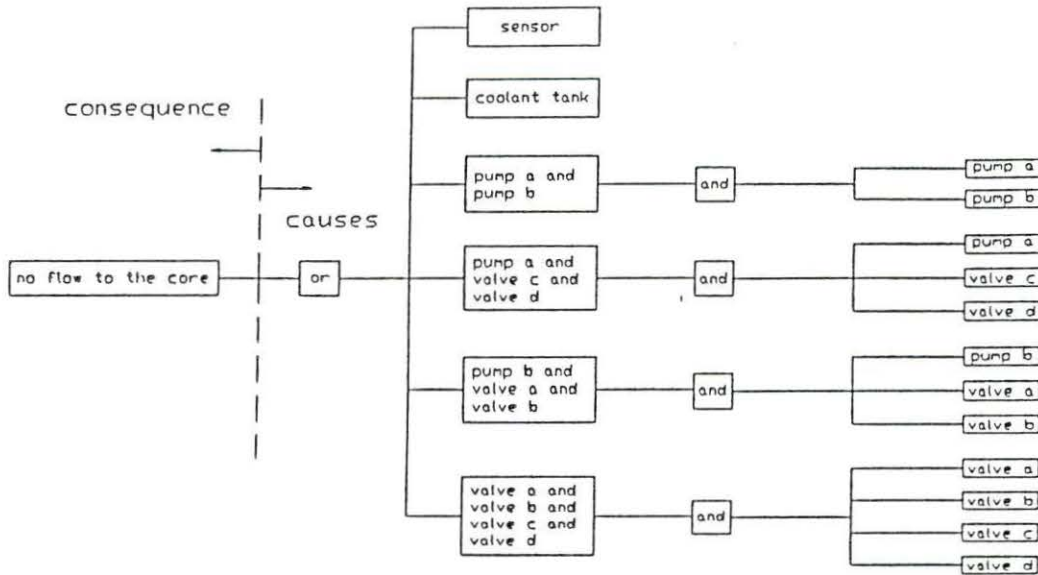


Figure 6.3: The failure menu tree for ECCS

6.2 A Pressure Tank System

Figure 6.4 shows a pressure tank system (PTS), obtained from Vessely [1]. The function of this system is to regulate the operation of the pump. This pumping system provides coolant from an infinite reservoir to the pressurized tank. The pressure switch has contacts which are closed when the tank is empty. When the threshold pressure has been reached, the pressure switch contacts open, de-energizing the coil of relay K2 so that relay K2 contacts open, removing power from the pump.

Initially, switch S1 contacts are open, relay K1 contacts are open, and relay K2 contacts are open; i.e. the control system is de-energized. In this de-energized state the contacts of the timer relay are closed.

System operation is started by momentarily depressing switch S1. This applies

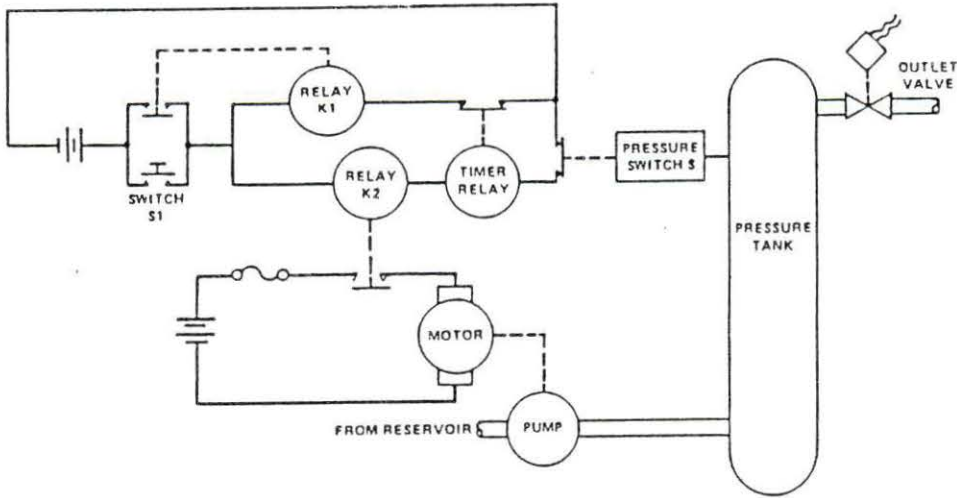


Figure 6.4: The configuration of the pressure tank system

power to the coil of relay K1, thus closing relay K1 contacts. Relay K1 is now electrically self-latched. The closure of relay K1 contacts allows power to be applied to the coil of relay K2, whose contacts close to start up the pump motor.

The timer relay has been provided to allow emergency shut-down in the event that the pressure switch fails closed. Initially the timer relay contacts are closed and the timer relay coil is de-energized. Power is applied to the timer coil as soon as relay K1 contacts are closed. This starts a clock in the timer. If the clock registers 60 seconds of continuous power application to the timer relay coil, the timer relay contacts open, breaking the circuit to the K1 relay coil and thus producing system shut-down.

The undesired event chosen is rupture of pressure tank after the start of pumping. The fault tree developed for this undesired event is presented in Vessely [1]. In the fault tree analysis performed in Vessely, the failure of switches connected to

relays are not taken into account, or their failure are included with the failure of the relays they are connected to.

The semantic network representation of this system includes, batteries 1 and 2, switches (S, S1, SK1, SK2, and SK3), relays (time relay (TR), K1, and K2), and motor, where SK1, SK2, and SK3 are the names assigned to switches connected to relays K1, K2, and TR, respectively. As can be seen, the pressure tank is excluded from the analysis, since the undesired event can be translated to energizing the motor after the start of pumping. This measure has to be taken, since the pressure tank is not connected to the system by a flow of electrical current, nor can it be linked to the system by non-flow connections. The switches specified as gate type components are specified as closed in the initial mode (or the energized mode if connected to a relay) and open in the operational mode (or de-energized mode if connected to a relay). By doing so, a no-flow analysis can be performed where the battery 2 and motor are the input and the output components, respectively. The cutsets found by performing the fault tree analysis are [1], (pressure tank) or (K2) or (S and K1) or (S and TR) or (S and S1). ESAS, in finding the cutsets, could not find the failure of the pressure tank, since it was not included in the semantic network representation of the system. However, since the failure of switches are included in the no-flow analysis performed by ESAS, they appear in the list of cutsets found. The cutsets found by ESAS are, (K2) or (SK2) or (S and K1) or (S and SK1) or (S and TR) or (S and SK3) or (S and S1).

The cutsets found by performing fault tree analysis are included in the list of cutsets found by ESAS, except the failure of the pressure tank. The analysis performed by ESAS is a more accurate one, since the failure of switches connected

to the relays are also found.

6.3 The PWR Containment Spray Injection System

The PWR containment spray injection system (PWRCISIS), obtained from WASH-1400 [2], is chosen to demonstrate the need for performing a flow and no-flow analysis for finding all of the lists of components whose failure can cause an accident. This system is depicted in Figure 6.5. The function of this system during normal operation is to supply a flow to header nozzles in either subsystems A or B. However, at the same time, no flow through the feedback loops, back to the refueling water storage tank is desired. Thus, flow to the header nozzle sub-systems and no flow to the storage tank are desired.

The components included in the semantic network representation of the system are, manual valves (V4A, V4B), motor operated valves (CS100A, CS100B, CS101A, CS101B, CS101C, CS101D), check valves (VCS-15x(1A), VCW-15x(1B)). These gate type components are initially specified to be closed and during system operation specified to be open. Valves (V2A, V2B, V2C), and check valves (V3A, V3B) are also included in the semantic network representation, but are specified to be open and closed during the initial and operational mode of the system, respectively, since they are a part of the feedback loops. Other components included in the semantic network representation are: refueling water storage tank, filters (1-CS-FL-1A, 1-CS-FL-1B), pumps (1-CS-P-1A, 1-CS-P-1B), and header nozzle subsystems A and B. Finally a node, called output, which does not represent a component in the system, is required to properly represent the output components of the system. See Section 5.3 for a more elaborate description of the need for including this component in the

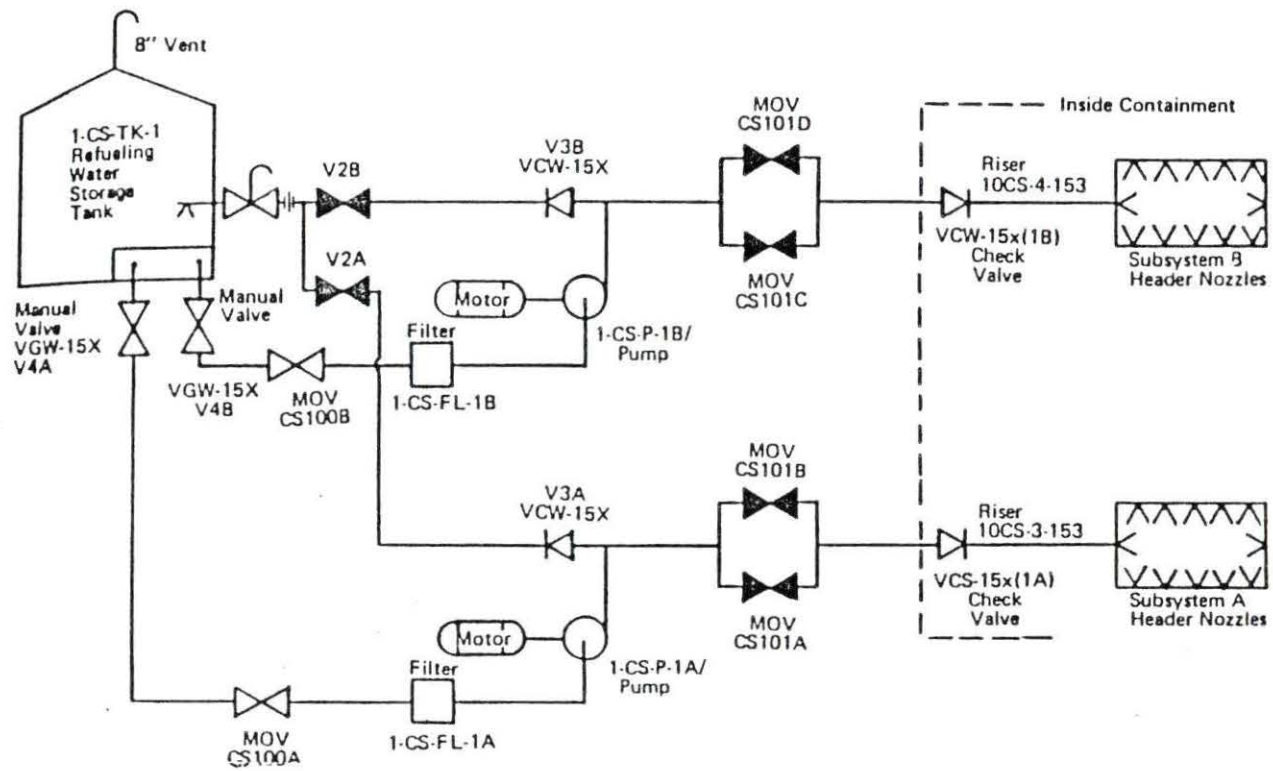


Figure 6.5: The configuration of the PWRCSIS

semantic network representation.

By performing a flow analysis, two cutsets containing three components are found, namely (V3A and V2A and V2C) or (V3B and V2B and V2C). This is apparent, since if all of the valves in each of the cutsets fail then they remain in the initial mode which is specified as open, thus allowing flow back through the feedback loop to the refueling water storage tank.

By performing a no-flow analysis, only one cutset containing one component was found, namely (storage tank). It is readily apparent that two parallel paths to the output components are present; one of which starts with component V4A and ends at subsystem A and the other which starts V4B and ends at subsystem B (called paths A and B, respectively). In each path, two motor operated valves (MOV) are put in parallel. Excluding these MOVs in both paths, there are six components in each path. By searching for cutsets containing two components, 36 cutsets were found, namely, all the non-redundant combinations of all components of path A with path B. For example, cutsets (A and B) and (B and A) are redundant.

The search for cutsets containing three components resulted in 12 cutsets, namely, the combination of parallel MOVs in path A with the six components in path B and parallel MOVs in path B with the six components in path A. Finally, one cutset containing four components was found namely, parallel MOVs in paths A and B. The cutsets found were identical to the ones obtained in WASH-1400 [2].

6.4 A Power Distribution Box

The power distribution box system, depicted in Figure 6.6, obtained from Vesely [1], is analogous to PTS since it is devised to cutoff the power supply to the

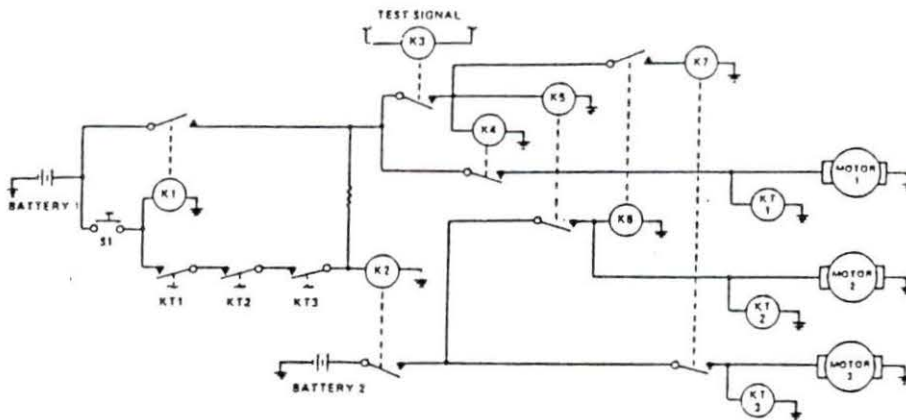


Figure 6.6: The configuration of the power distribution box

output components, motors 1, 2, and 3. However, it is more complex.

With contacts KT1, KT2, and KT3 normally closed, a momentary depression of push-button S1 applies power from battery 1 to the coils of relays K1 and K2. Then the relays K1 and K2 close and remain electrically latched. Next, a 60-second test signal is impressed through K3, the purpose being to check proper operation of motors 1, 2, and 3. Once K3 has closed, power from battery 1 is applied to the coils of relays K4 and K5. The closure of K4 starts motor 1. The closure of K5 applies power from battery 2 to the coil of K6 and also starts motor 2. Finally, the closure of K6 applies power from battery 1 to the coil of K7. Closure of K7 starts motor 3.

After an interval of 60 seconds, K3 is supposed to open, shutting down the operation of all three motors. Should K3 fail closed after the expiration of 60 seconds, all three timers (KT1, KT2, KT3) open, de-energizing the coil of K1, thus shutting down system operation. Suppose K3 opens properly at the end of 60 seconds, but K4 fails closed. In that case, KT1 opens to de-energize K1 and motor

1 stops. KT2 and KT3 act similarly to stop motor 2 or motor 3 should either K5 or K7 fail closed.

The undesired event is the overrun of any motor after test is initiated. The cutsets found in Vessely [1], by performing fault tree analysis for overrun of motor 1 are, (K1 and K4) or (K4 and S1) or (K4 and KT1) or (K1 and K3) or (K3 and S1) or (K3 and KT1 and KT2 and KT3). The cutsets found for overrun of motor 2 are, (K5 and K2) or (K5 and K1) or (K5 and KT2) or (K5 and S1) or (K3 and K1) or (K3 and S1) or (K3 and KT1 and KT2 and KT3). The cutsets found for overrun of motor 3 are, (K1 and K7) or (K2 and k7) or (K7 and KT3) or (K7 and S1) or (K1 and K3) or (K3 and S1) or (K3 and KT1 and KT2 and KT3).

The semantic network representation of this system includes all relays and switches depicted in Figure 6.6, in addition to all switches connected to the relays. By inspecting the cutsets found by performing the fault tree analysis, one can see that Vessely combined the failure of the switches connected to the relays with the failure of the relays they are connect to. Thus, failure of a relay can be due to the failure of the relay or the failure of the switches it is connected to. In the analysis performed by ESAS, the failure of these switches were also taken into account. Similar to the semantic network representation of the PTS, all switches in this system are specified to be initially closed. They then open during the operation of the system. To analyze this accident with ESAS, flow analysis was performed three times where in the first analysis, the battery 1 and motor 1, in the second analysis, the battery 2 and motor 2, and in the third analysis, the battery 2 and motor 3 are chosen to be the input and output components, respectively.

The cutsets found by ESAS included all cutsets found by Vessely, which were

obtained by performing fault tree analysis. Additional cutsets found were composed of switches connected to relays which were excluded from the analysis performed in Vessely. For example, in addition to the cutset (K1 and K4), the cutsets (SK1 and K4), (SK1 and SK4), and (K1 and SK4) were found by ESAS, where SK1 and SK4 are the names assigned to switches connected to relays K1 and K4, respectively. Or in addition to the cutset (K4 and S1), the cutset (SK4 and S1) were also found by ESAS. Thus, the result of the analysis performed by ESAS is more accurate than the one reported in Vessely.

When we state the analysis performed by ESAS is more accurate than the one obtained by performing fault tree analysis, we do not imply that the same accuracy can not be achieved with fault tree analysis. However, we do imply that if the accuracy is increased the fault tree analysis will become even more complex and tedious.

7 CONCLUSIONS

7.1 Restriction on Types of Systems Analyzed by ESAS

In Chapter 4, the advantages of programming in Turbo Prolog [21] were cited. One of the most useful advantages was being able to program in modules. This feature of Turbo Prolog allowed us to develop ESAS in stages. The evolving stages were formed by attempting to analyze new types of nuclear power plant systems in the order they were encountered, based on the fact that the components of these systems have to be related to one another by a flow type. The systems which were used in developing ESAS in the order they were encountered are the systems included in Chapter 6, in Sections 6.1, 6.2 and 6.3, respectively. By studying the functions of these systems and their components, we modified the knowledge base of ESAS (i.e., rules for finding paths and cutsets) so that the function of these systems and their components can be simulated and analyzed by ESAS. This resulted in including new definitions such as “non-flow connections” to incorporate relationships other than “flow” between components etc. in the knowledge base of ESAS as was described in Chapter 5.

Originally, we intended to make the use of ESAS as general as possible. However, we recognize that it is impossible to simulate the function of all of the components in a nuclear power plants with only being able to simulate the function

of components contained in the three systems described in the first three sections of Chapter 6. At the same time, we hope the reader recognizes that these three systems are typical of most systems in nuclear power plants. In conclusion, use of ESAS for finding paths in a system is restricted to those systems where flow of one and only one type is present through all components of the system. Also, the use of ESAS for finding cutsets is restricted to those systems which can be analyzed in terms of flow or no-flow and whose components have functions which can be simulated by the rules in the knowledge base of ESAS.

7.2 Suggestions for Future Work

It was reasoned in the previous section that the use of ESAS can be generalized by incorporating function of the components which are not now included in the rules used by ESAS. By including analyses other than flow and no-flow, use of ESAS can be broadened to a larger variety of systems. However, by expanding ESAS, a memory management problem can arise.

ESAS is written in modules which are compiled and linked to an executable program by the Turbo Prolog compiler [21]. The executable program of ESAS can be run on an IBM XT, AT, or compatible. The size of the operating memory of these computers is 640K bytes minus the memory required for the operating system program DOS itself which requires approximately 75K bytes. The executable version of ESAS requires approximately 400K bytes just for the rules of the knowledge base. When analyzing a system, the memory required for consulting the database of a system must be added to the 400K bytes. Therefore, this constitutes a size limitation on the memory required for the database of a system which in turn trans-

lates to a size limitation on the number of components a system can contain. This size limitation can be increased by overlaying ESAS. Those modules of ESAS which perform a task (e.g., finding paths) can be compiled to an executable program. Then, ESAS will be comprised of several smaller executable programs which can be run by using a main program. By doing so, the executable programs, which are activated by the main program, will be smaller in size thus allowing more space for the consulted database of a system. Also, by overlaying ESAS, we alleviate the problem of memory limitation when expanding it to include rules which incorporate additional component and system functions.

The diagnostics features of ESAS can also be improved by simulating the function of more component types. As was described earlier, the function of gate and activator type components are the only ones simulated by ESAS. Other than the function of these types of components, components either succeed which implies that a flow is allowed through them or vice versa if they fail. Therefore, only a qualitative analysis of components are possible. For example, if a pump fails we can not simulate the coast down period where partial flow is available, or if a pipe ruptures flow is completely cut-off, even though there can be a partial flow through the pipe. The function of different components can be simulated by use of FORTRAN programs which solve equations corresponding to the components. The input of these programs can be the condition of the components and then the output can be the degree of success of the components in allowing flow.

Another diagnostic feature which can be added to improve ESAS is incorporating time dependencies. For example, in electrical systems where time relays are used, even though the relay is energized it does not immediately alter the mode of

the gate type component it is connected to. Enabling ESAS to perform an analysis at each time the system configuration is altered can make it possible to perform these types of analysis.

To improve the interfacing capabilities, graphics programs can be included in ESAS which allow the user to enter a graphic representation of the system when entering the database. Then ESAS can use this graphic representation to point out the cutsets found or, in diagnostics mode, to point out the components found as possible causes of an accident.

In introducing the purpose of this project, we stated that the end product of fault tree analysis are the cutsets and we proposed a method by use of which cutsets can be found without performing fault tree analysis. However, fault trees are sometimes used by the operators of nuclear power plants for diagnostics purposes and for operating assistance. ESAS can be programmed to work backward from the cutsets and form the fault tree which if resolved will result in the cutsets found. This is an involved task which would require a great deal of effort.

In conclusion, ESAS can successfully accomplish the tasks it was originally programmed to perform. However, as is demonstrated in this section, the development of ESAS can not be considered to be completed. It is commonly known, most projects are never completed, since as a goal is reached others are created.

8 BIBLIOGRAPHY

- [1] Vessely, W. E. Fault Tree Handbook. Springfield, Va.: National Technical Information Service, 1981.
- [2] "Reactor Safety Study: An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants". U.S. Nuclear Regulatory Commission, WASH-1400 (NUREG-74/014).
- [3] Knief, R. A. Nuclear Energy Technology. Washington: Hemisphere Publishing Corporation, 1981.
- [4] Schildt, H. Advanced Turbo Prolog. Berkeley, Calif.: McGraw-Hill, 1987.
- [5] Winston, P. H. Artificial Intelligence. New York: Addison-Wesley, 1979.
- [6] Anderson, J. R. *Cognitive Psychology and Its Implications*. 2nd ed. New York: W. H. Freeman and Company, 1985.
- [7] P. Harmon and D. King. *Expert Systems. Artificial Intelligence in Business*. New York: John Wiley and Sons, 1985.
- [8] Forsyth, R. Expert Systems. Principles and case studies. London: Chapman and Hall, 1984.
- [9] Uhrig, R. E. "Applications of Artificial Intelligence in the U.S. Nuclear Industry". p. 11 in *Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry*. (Majumdar, M. C., Majumdar, D., and Sackett J. I., eds.). New York: Plenum Press, 1987.
- [10] Majumdar, M. C., Majumdar, Debu, and Sackett, J. I. *Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry*. New York: Plenum Press, 1987.
- [11] Forgnier, B. "An Expert System for Fault Tree Analysis". p. 747 in *Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry*. (Majumdar, M. C., Majumdar, D., and Sackett J. I., eds.). New York: Plenum Press, 1987.

- [12] Ancelin, C., Le, P., De Saint-Quentin, and Villatte, N. "EXPRESS: An Expert System to perform System Safety Studies". p. 761 in *Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry*. (Majumdar, M. C., Majumdar, D., and Sackett J. I., eds.). New York: Plenum Press, 1987.
- [13] Kelly, R. D., and K. J. Vavrek. "Automated Fault Tree Generation." Nuclear Plant Journal January-February (1988): 46,48.
- [14] Kiguchi, T., Motoda, H., Yamada, N., and K. Yoshida. "A Knowledge Based System for Plant Diagnosis". p. 635 in *ANS International Topical Meeting on Computer Applications for Nuclear Power Plant Operation and Control*. (American Nuclear Society, La Grange Park, IL, 1985).
- [15] Beltracchi, L. "A Model-Based Display". p. 337 in *Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry*. (Majumdar, M. C., Majumdar, D., and Sackett J. I., eds.). New York: Plenum Press, 1987.
- [16] Kitamura, M., Baba, T., Washio, T., Sugiyama, K., and K. Katajuma. "Semantic Network Approach to Automated Failure Diagnosis in Nuclear Power Plant". p. 654 in *ANS International Topical Meeting on Computer Applications for Nuclear Power Plant Operation and Control*. (American Nuclear Society, La Grange Park, IL, 1985).
- [17] Dixon, B. W., and K. G. Ferns. "Using Risk Based Tools in Emergency Response". p. 799 in *Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry*. (Majumdar, M. C., Majumdar, D., and Sackett J. I., eds.), New York: Plenum Press, 1987.
- [18] Clockstin, W. F., and C. S. Mellish. Programming in Prolog. 2nd ed. New York: Springer-Verlag, 1984.
- [19] Kowalski, R. Logic for Problem Solving. New York: Elsevier Science Publishing Co., 1979.
- [20] Stanat, D. F., and D. F. McAllister. "Discrete Mathematics in Computer Science". Englewood Cliffs, N. J.: Prentice-Hall, 1977.
- [21] Turbo Prolog. Scottsville, Calif.: Borland International Inc., 1986.
- [22] Shafer, H. Advanced Turbo Prolog. Berkeley, Calif.: McGraw-Hill, 1987.
- [23] Marcus, C. Prolog Programming. Reading, Massachusetts: Addison-Wesley, 1986.

- [24] Malpas, J. Prolog: A Relational Language and Its Applications. Englewood Cliffs, N. J.: Prentice-Hall Inc., 1986.
- [25] Lewis, E. E. Nuclear Power Reactor Safety. New York: John Wiley and Sons, 1977.
- [26] Danofsky, R. A. Nuclear Engineering 441 Class Notes. Iowa State University, Ames, 1982.