EXPERT SYSTEMS FOR FAULT TREE SYNTHESIS

DAVID JOHN HARRIS

Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

November 1990

The University of Aston in Birmingham

Expert Systems for Fault Tree Synthesis

David John Harris **PhD 1990**

SUMMARY

Fault tree analysis, a detailed failure frequency prediction method which forms the basis of quantitative risk assessment, is used widely in a number of different industries. Fault trees present the system failure logic graphically such that system failure methods can readily be assimilated by both engineering and management personnel. Fault trees are useful to the identification of design weaknesses, and may additionally be used as an aid to operator training. The one drawback of fault tree analysis when applied to chemical process plant is that, due to the sheer size and complexity of modern process plants, fault trees are difficult to generate by hand.

The work in this thesis proposes a computer-based method to aid the development of fault trees for chemical process plants. The aim is to produce concise, structured fault trees that are easy for analysts to understand. Standard plant input-output equations for major process units are modified such that they include ancillary units and pipework. This results in a reduction in the nodes required to represent a plant.

Control loops and protective systems are modelled as operators which act on process variables. This modelling maintains the functionality of loops, making fault tree generation easier and improving the structure of the fault trees produced.

A method, called event ordering, is proposed which allows the magnitude of deviations of controlled or measured variables to be defined in terms of the control loops and protective systems with which they are associated. This method enables the user to define the relative controllability of failure event chains, and allows the system to accurately represent plant failure logic.

When combined, the plant reduction and event ordering methods are capable of producing small, well structured fault trees that are comparable to those that might be created by a human expert. In addition, the event ordering approach, which expects the user to make certain key decisions about the way that failures propagate through the plant, is capable of modelling almost all plant configurations and control strategies because it is not limited by any inherent assumptions.

A computer package which implements these methods as an expert system has been developed using a production language, OPS5. This package is tested using an ammonia let down plant area and using two different levels of plant control. The results of these tests show that, using this package, it is possible to generate fault trees by computer which are on a par with those generated by human experts.

Key words: Fault Tree, Expert System, Fault Diagnosis, Cause and Symptom Equations, Object Oriented

ACKNOWLEDGEMENTS

The author wishes to express his profound gratitude to the following people, without whom, this thesis would not have been a success.

- Dr. M.C. Jones for supervising this research from October 1986 to August 1987.

- Dr. J.P. Fletcher who took over the supervision from September 1987.

LIST OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

1.INTRODUCTION

This thesis describes a method of analysing the ways in which chemical process plant can fail, in order to reduce the risk of hazardous events occurring.

The conversion of natural resources into useful products in the chemical process industries usually proceeds via a number of steps, each of which may involve extremely hazardous materials. For efficient operation of some processes, high temperatures and pressures are required. The consequences of failure of chemical process plant involving hazardous materials under such operating conditions are potentially catastrophic. A number of accidents involving multiple fatalities have occurred. Some of these, reported in references [1] and [2], are listed in Table 1.1.

Trends have been set over the past twenty years that necessitate more rigorous methods for safety analysis of chemical plants [3]. These trends are:

1. An increasing preponderance of single train installations;
2. The use of larger processing units;
3. More complete process integration for energy recovery and waste recycling;
4. Reduction of intermediate storage capacity;
5. Centralization of control;
6. Growth of computer-based operations;
7. Complexity of equipment;
8. Location of plants closer to population centres.

The incentive for these trends is increased profitability through economies of scale, tighter control of process variables and reduced transportation costs.

Implicit within these trends are the use of complex integrated

Table 1.1: Major Accidents That Have Occurred in the Chemical Process Industries Since the First World War

Year	Location	Nature of Incident	No of Fatalities
1926	St. Albans, France	Release of Chlorine	19
1928	Hamburg, Germany	Release of Phosgene	11
1939	Zarnesti, Rumania	Release of Chlorine	60
1944	Cleveland Ohio, USA	Fire involving liquefied natural gas	128
1947	Rauma, Finland	Release of Chlorine	19
1948	Ludwigshafen Germany	Explosion involving dimethyl ether	207
1952	Walsum, Germany	Release of Chlorine	7
1964	Antwerp, Belgium	Explosion involving ethylene oxide	4
1966	Feyzin, France	Fire involving propane	17
1967	Lake Charles Louisiana, USA	Explosion involving isobutane	7
1968	Pernis, Netherlands	Explosion involving hydrocarbons	2
1971	Amsterdam Netherlands	Explosion involving butadiene	8
1974	Flixborough, UK	Explosion involving cyclohexane	28
1975	Antwerp, Belgium	Explosion of ethylene	6
1975	Longview Texas, USA	Explosion of ethylene	4
1975	Beek, Netherlands	Explosion of propylene	14
1977	Umm Said, Qatar	Explosion and fire	6
1984	Mexico City, Mexico	Explosion and fire	144
1985	Bhopal, India	Release of methyl isocyanate	3000

control strategies designed for more economic operation near inherent process constraints; increased demands on operator competence; and on-line maintenance and testing policies to avoid process shut-down. As plant sizes have increased and Lees [4] reports that plant sizes have typically increased by a factor of 10 since 1945, so too has the total capital investment per plant. This is accompanied by increased potential for losses if serious process failures should occur.

Loss of containment of process material is the most common type of accident. This may result in the release of toxic substances, or the production of a vapour cloud sufficiently large to cause a serious fire or explosion, as was the case in the Flixborough disaster [5,6].

The impact of such loss of containment may be measured not only in terms of the risk to human life. Release of toxic material into rivers, seas and the atmosphere may have a serious impact on the local environment. In an age of increasing environmental awareness, the chemical process industry's image is considerably tarnished by such incidents. In the wake of public opinion in the U.K., penalties for companies negligently causing environmental damage are rising: this trend looks set to continue over the coming years.

Concern over the operation of plants usually develops following a major accident. In the United Kingdom there was no safety legislation aimed specifically at the development and operation of chemical process plants until after the Flixborough disaster in 1974. Previously, plant safety was governed under the Factories Act 1961 which was concerned solely with the safety of employees, and then the Health and Safety at Work Act 1974 [7], which also made provisions for prevention of risk to the general public. After the Flixborough disaster, an Advisory Committee on Major Hazards was set up by the Health and Safety Commission to consider safety problems associated with large scale industrial premises conducting potentially hazardous operations.

The Advisory Committee sat for nine years and produced three reports during this time [2,8,9]. Perhaps the most important of several recommendations made in these reports was that special legislative

measures, over and above the Health and Safety at Work Act 1974, were required in order to further protect the public. As a result of these, the Control of Industrial Major Accident Hazards Regulations (CIMAH regulations) were passed by British parliament in 1984. The CIMAH regulations require that all manufacturers prove to the Health and Safety Executive that they have identified existing major accident hazards, adopted appropriate safety measures, and provided on-site workers with suitable information, training and equipment to ensure their safety.

There are two approaches to disaster limitation. The **protective systems approach** [10] is to contain any loss using fire walls, sprinkler systems, containment walls, emergency cooling systems and explosion limiting devices. It may also be possible to separate parts of the plant so that, for example, fire in one area does not cause explosion elsewhere. This is an "after the fact" approach; it may reduce the consequences of hazardous incidents, but it does little to prevent their occurrence.

In contrast to this, the **systems safety approach** aims to reduce the likelihood of a hazardous incident occurring. This approach involves some kind of systems analysis, leading to determination of failure pathways of the plant. It may then be possible to improve the intrinsic plant safety through re-design, and also to specify additional control and protective equipment where it is most needed.

Perhaps the first systems failure analysis method to be used in the chemical process industry was **failure modes and effects analysis** (FMEA) [11,12,13]. This method involves the identification of all possible failure modes for each item of equipment, the determination of the effects these failures may have on overall system performance, and an assessment of the seriousness of any hazards that may result. Although the method provides a systematic methodology for the analysis of systems, it is limited by the fact that identification of all possible failure chains for hazardous events is almost impossible for large plants [10,11,14].

The failure analysis method recommended by the Advisory Committee on Major Hazards, and subsequently widely adopted in the U.K., is **Hazard and Operability Studies** (HAZOP) [15,16,17]. The concept of HAZOP is to

provide a formalized framework within which a team of experts of different disciplines can jointly analyse plant from an operational safety point of view. The team attempts to foresee every possible fault that can occur in every plant item. This is achieved by carrying out a systematic study of the design for each vessel and line, using certain guide words to stimulate thought about the way in which deviations from the intended operating conditions can occur, and in turn lead to hazardous situations.

The HAZOP study is split into several sections. Plant units and lines are selected systematically, and their intended operating states declared. The guide words are then applied to these process intentions to generate undesirable states, or deviations. For each deviation, the team attempts to identify every possible cause and all undesirable occurrences that may result. Finally, the team records the actions that it feels should be taken to reduce the risk of any hazards it has identified.

HAZOP forms an effective and thorough means of analysing process plant. It does, however, have certain drawbacks. The systematic nature of the study, whilst necessary to maintain completeness, can make the task rather repetitive. In addition, many of the team members are unoccupied most of the time. Long studies can become a tedious chore, this could potentially result in serious omissions of failure pathways.

Often, the choice of which safety measures to adopt is based on economics. It is necessary to strike a balance between the economic consequences of failure and the cost of adopting a safety measure. In order to determine this balance, a suitable systems failure analysis method should be used to calculate failure probabilities. Unfortunately, HAZOP studies cannot directly be used to arrive at quantitative information such as probability estimates.

A failure analysis method which allows quantitative assessment of failure probabilities is **fault tree analysis** (FTA). FTA was first developed in 1962 [18] as a method to establish and illustrate failure pathways, and has since been extensively used in the electronics and aerospace industries. A fault tree is a graphical structure which consists of an undesirable event, usually a system hazard or off-specification product,

linked to its possible causal events through intermediate causal events in a tree-like structure. Logic gates are used to define the combinations of these causal events which may produce the event under investigation.

Fault trees provide a clear graphical representation of failure pathways and may be analysed qualitatively or quantitatively to identify any weaknesses in the process or protective system. It is also possible to use fault trees to help identify which, if any, extra protective loops are required, and to investigate the relative effectiveness of different control strategies. Further applications of fault trees are fault diagnosis and computer-based alarm analysis.

Despite these considerable merits, FTA is not commonly used in the chemical process industry. The main reason for the under-use of FTA is that the synthesis of fault trees is extremely labour intensive: fault trees constructed by the United States Atomic Energy Commission for part of a nuclear plant required more than twenty five man years to complete [19]. In addition, when design changes are made, considerable extra effort is required to modify previously produced fault trees. In order to overcome these problems, automated systematic methods of fault tree generation have been developed. Despite the success of methods for electronic systems, no method for the automatic generation of fault trees for chemical processing systems is in general use. Several methods have been proposed and computer codes produced, but these suffer from a variety of problems. Computer-based methods tend to produce fault trees which are larger and more complex than hand generated trees, making their interpretation difficult. Most of the methods proposed are also quite inflexible, rendering them unable to handle many types of problems.

There have been many great advances in computer technology over the past few years, both in hardware and software. In hardware terms, computers have increased greatly in speed with the development of sixteen and thirty-two bit microprocessors and the advent of parallel processing, they now also have much greater available memory due to the development of cheaper memory chips of larger capacity. One technology which has benefited greatly from these hardware advances is artificial intelligence. Artificial intelligence techniques attempt to simulate the way in which

humans tackle certain types of problem. One such artificial intelligence technique is the representation of domain-specific "expert" knowledge using **expert systems.**

Expert systems provide a framework for the representation and use of knowledge about a specific subject. This knowledge is used by the system to make deductions from an input set of data. It is possible to guide the system toward a goal or set of goals. For example, in fault tree synthesis it is possible to store knowledge about chemical plants, the failure modes of plant items and groups of items, and about the properties of fault trees. The system can be programmed to act upon input data, in this case describing the plant under investigation, to chain through causes of failure events until a fault tree has been constructed.

Writing expert system programs has become much easier with the advent of **expert system shells.** These shells provide a ready-made framework for the storage of expert knowledge and data, and a pre-defined inference method. Although expert system shells may not always provide the ideal solution for any one specific problem, they are very useful for investigating different problem solution strategies.

The research described in this thesis has been conducted using such an expert system shell, OPS5, with an aim to develop methods for overcoming the problems of computer-based methods for the synthesis of fault trees for process plants. The methods make use of causal equations as described by Lihou [20] as the basis of models of plant items which are stored as "expert" rules. Two methods, one which aims to simplify the problem, and one which aims to provide the flexibility necessary for a system to cope with a wide range of process control strategies, are then combined to generate fault trees from these models. The methods have been implemented on a VAX 8650 computer system, the results are presented herein.

The results achieved show that it is possible to produce good fault trees, whilst retaining sufficient generality to allow a wide range of real-world problems to be solved. An important conclusion of this work is that it is possible, using expert system techniques, to represent chemical

process plant at a number of levels of specificity. Models may either be built up from smaller generalized models, or models for specific plant configurations may be represented explicitly. Explicit representation improves the quality of fault trees, but general representation allows the method to be applied to many plant configurations.

CHAPTER TWO

2.SAFETY AND LOSS PREVENTION

2.1:Introduction

Every company has a moral obligation to safeguard the health and welfare of its employees and the general public. There is also a financial consideration: should an accident occur, damaged plant must be replaced, third party claims met, and earnings will be hit due to lost production.

Most manufacturing processes are hazardous to some degree, but chemical processes involve special hazards associated with the materials and operating conditions used. The process designer and operator must be aware of these hazards, and ensure that the risks are reduced to acceptable levels. Hazard reduction should involve the following steps:

1. Identification and assessment of hazards;
2. Control of the process: prevention of hazardous deviations in process variables by the provision of automatic control systems, trips, alarms, interlocks and good operating procedures;
3. Control of the hazards by, for example, containment of toxic and flammable materials;
4. Loss limitation by the use of containing walls and fire-fighting equipment.

2.2 Hazards that Occur in the Chemical Process Industries

Most of the materials used in the manufacture of chemicals are poisonous to some extent. The hazard of a material depends on its toxicity and the frequency, duration and extent of exposure to it. A highly toxic material, such as chlorine, causes immediate injury whereas less toxic materials, such as carcinogens, when concentrations are low cause damage only after prolonged exposure. The precautions against leakage of these two classes of materials will be very different: slight leakage of less toxic materials is permissible, provided that the concentration is less than the defined **threshold limit value** for that material. Exposure to highly toxic

material may prove fatal, even at low concentrations, consequently every possible step is taken to prevent leakage. It can be seen from Table 1.1 that several of the fatal accidents that have occurred in the chemical process industry have involved the loss of containment of toxic material, often chlorine.

Highly flammable materials also represent a hazard. The extent of the hazard posed by a material depends upon a number of factors. **The flash point** of the material defines the temperature at which it will ignite from an open flame. Storage of materials at temperatures above their flash point should be avoided, particularly in the vicinity of possible spark sources, such as equipment with moving parts. The material's **autoignition temperature** is the temperature at which it will ignite spontaneously in air, without any external ignition source. If a process involves a material above its autoignition temperature, then some sort of gas purging system should be provided to prevent the material coming into contact with oxygen. The **flammability limits** of a material define the upper and lower limits of concentration of the material in air at which a flame will propagate through the mixture. The material's **heat of combustion** is a measure of how much energy is released when the material burns. Materials with a high heat of combustion are very hazardous as fires involving such materials can spread very rapidly.

The consequences of fire in a process plant may be severe. The fire may spread over a large area of the plant, causing considerable damage to equipment and producing a knock-on effect which could lead to other incidents, such as explosion. Certain materials, for example acetylene, can decompose explosively in the absence of oxygen; these are especially hazardous. Explosions may also occur without fire, such as the explosion through overpressure of a steam boiler.

An **unconfined vapour cloud explosion** results from the ignition of a considerable quantity of flammable gas or vapour which has been released into the atmosphere. Such an explosion can cause considerable damage, and pose a threat to both on-site personnel and the general public. An example of an unconfined vapour cloud explosion is that which occurred at Flixborough [5].

2.3: Loss Prevention

In order to minimise the risk of hazards such as those described above, methods for hazard prediction and quantification have been developed. These include **hazard ranking** methods, such as the Dow Fire and Explosion Index [21] and the Mond Fire, Explosion and Toxicity Index [22,23], which are used to estimate the overall hazard a plant represents, and also to identify which plant areas are potentially the most hazardous. Such methods are useful at an early stage in the design procedure for comparing different processes, and are also used for evaluation of the maximum potential accident cost for insurance purposes. Hazard ranking methods cannot, however, be used to identify specific hazards and their causes, nor can they be used to evaluate the probability that a specific hazardous event will occurr.

More detailed **systems analysis** methods are used to identify actual hazards and their possible causes. Such methods are usually used during the later stages of the design procedure to help the designers make improvements to the process and control systems. They may also be used when an existing process is to be modified.

Systems analysis for chemical process plant is often a difficult task. Although some design faults may be identified intuitively, many are not readily obvious. This is largely due to two factors. The sheer size of most modern process plants means that the interactions between different areas and units of the process may be very complex. Such interactions are made very much more difficult to assess by non-linearity of the variables involved: a small change in a process state somewhere in the plant may, if uncorrected, cause a large, potentially catastrophic, condition elsewhere. This is an example of the "butterfly effect": such behaviour may seem unpredictable unless every possible interaction leading to a hazard is accounted for. In order to identify every such failure chain, a systematic systems failure method must be applied to analyse every plant item and the interactions between items in detail. Some methods are described in the rest of this chapter.

2.3.1. Failure Modes and Effects Analysis

Failure Modes and Effects Analysis (FMEA) [11,12,13] is a systems safety analysis method used to determine the possible failure modes of a process. The technique involves the following steps:

1. Identification of all system components;
2. Determination of all failure modes for each component;
3. For each failure mode, determination of the effects on other system components;
4. Identification of the overall effects of the failure mode on the system performance, including any hazardous events;
5. Estimation of the seriousness of the failure.

Simultaneous failures may be handled in the same fashion.

The results of FMEA may be represented graphically by **event trees.** Event trees consist of an initiating event, or combination of events, linked to possible consequences of this event or events. Each consequence is linked in turn to its consequences, until the tree cannot be further developed. Each of the final events obtained in this way may therefore result from the initiating event or events.

FMEA is a **forward chaining** method: event sequences are generated from initiating events to final events. Forward direction of fault propagation is inefficient as there is no guarantee that propagation of any one component failure will lead to a system failure of importance. In order to identify every system failure and its causes, it is necessary to investigate every combination of component failures. For large plants this is practically impossible, even using computers.

2.3.2: Hazard and Operability Studies

Hazard and operability study, commonly known as HAZOP, is a formal and systematic procedure the aim of which is to aid in the identification and correction of potential hazards and operating difficulties for process

plants [15,16,17]. HAZOP provides a formal framework within which a multidisciplinary team of experts can come together for a series of meetings to review the safety of a newly designed or existing plant. It is based on the assumption that most problems missed or incorrectly defined are due to the complexity and size of the problem, rather than because of lack of knowledge on the part of the design team [15].

HAZOP is usually carried out at the detailed design stage of new plant design. There may, however be several reasons for conducting the study:

1. To check the design of the whole plant, and to identify any improvements that may be made;
2. To decide whether and where to build a plant;
3. To check the operating and safety procedures;
4. To review or update the process equipment, safety devices or instrumentation on an existing plant;
5. To generate data for fault diagnosis methods.

2.3.2.1: The Study Team

The HAZOP team usually consists of a multidisciplinary group of experts. The reasoning behind HAZOP is that each team member will function more effectively as part of the study group when the members work together, rather than working separately with pooling of results. The exact composition of the team depends on the type of process under investigation and the stage at which the study is carried out. Kletz [24] recommends that the composition of the study team should approximately match those shown in Tables 2.1 and 2.2, depending upon whether a new or an existing plant is under investigation.

Table 2.1: HAZOP Study Team for New Plants

Team Member	Comment
Design Engineer	Usually a mechanical engineer, responsible for minimising the cost, but not for hazards and operating problems.
Process Engineer	Usually the chemical engineer who drew up the flowsheet.
Commissioning Manager	Usually a chemical engineer who will start up and operate the plant.
Instrument Manager	Required for plant with sophisticated control, alarm or trip systems.
Research Chemist	Required if new chemistry is involved.
Independent Chairman	A specialist in HAZOP, rather than the plant. Should ensure that the team follow the procedure. Leader of the team.

Table 2.2: HAZOP Study Team for Existing Plants

Team Member	Comment
Plant Manager	Responsible for plant operation.
Process Foreman	Knows what actually happens, rather than what is supposed to happen.
Plant Engineer	Responsible for plant maintenance and therefore knowledgeable about many of the faults that occur.
Instrument Manager	Responsible for instrument maintenance including testing of alarms and trips.
Process Investigation Manager	Responsible for investigating technical problems.
Independent Chairman	A specialist in HAZOP, rather than the plant. Should ensure that the team follow the procedure. Leader of the team.

2.3.2.2: HAZOP Methodology and Procedure

HAZOP uses certain words to stimulate thought. **Property** words are used to define the design intention of plant items. Typical property words are flow, temperature, level, pressure, composition, and also such properties as heat transfer and reaction rate. **Guide words** are used to define possible deviations from the defined intentions. The guide words and their meanings as defined by the Chemical Industries Health and Safety Council [25] are shown in Table 2.3.

The information required for the study depends on the scope of the investigation. It is possible to carry out a preliminary study by using a description of the design, but for a detailed study the flowsheets, piping and instrumentation diagrams, equipment specifications and layout drawings are required. For an existing plant, such information should already exist, but it is important that it is up to date, any modifications made during operation must be included. For a batch process, information defining the sequence of operation is also required. Such information may also be required to define sequential operations, such as startup and shutdown, for continuous processes.

The HAZOP team examine the process systematically, using the guide words to detect possible deviations from the design intentions. Every vessel, auxiliary item and line is examined in turn. All possible causes of deviations are identified, along with any consequences of the deviation that may arise. The importance of these consequences, in terms of plant safety and operability, is investigated. This procedure may uncover potential hazards or operating problems; actions that should be taken to prevent or reduce the scale or likelihood of these are then investigated.

2.3.2.3: Recording of Results

During the team study it is possible that not all of the significant remarks will be recorded. It is therefore necessary for team members to review the report made of the session, and for the team to come together for a review session to fine tune the report. Such a review process may uncover errors and omissions. The success of this depends to a large

Table 2.3: HAZOP Guide Words and Meanings

Guide Word	Meaning	Comments
NO or NOT	Complete negation of the design intent	No part of the intention is achieved
MORE	Quantitative increase	Refers to proerties such as flowrate and pressure as well as activities such as "heat" or "react"
LESS	Quantitative decrease	Refers to properties or activities (as "MORE")
AS WELL AS	Qualitative increase	All design intentions are achieved, together with some additional activity
PART OF	Qualitative decrease	Only some of the design intentions are achieved
REVERSE	Logical opposite of the design intent	Mostly applicable to activities, such as flow or reaction
OTHER THAN	Complete substitution	Something quite different from the design intention occurs

extent upon the recording scheme used for the original study. The usual form of recording HAZOP information is to tabulate the property words, guide words, causes, consequences and actions [15], alternatively checklists may be used [26].

2.3.2.4: Drawbacks of HAZOP

A major problem associated with HAZOP is the time it takes to complete a full study. HAZOP studies require many hours of meetings to consider every plant item. As the study team consists of several experts, this means that HAZOP can be rather expensive. This problem is compounded by the fact that when modifications to the plant design are proposed by the team, more meetings are required to assess the impact of these on the overall plant operability and safety.

Due to the systematic nature of HAZOP, the fact that some team members may be mostly unoccupied, and the considerable time it may take to carry out a study, the motivation or concentration of some team members may suffer. It befalls the team chairman to involve each member in the discussion. Some teams may fail to pick up all eventualities if the chairman fails to keep each member interested.

During a long study, it is sometimes difficult for a team to maintain a sense of proportion, especially when assessing the seriousness of identified hazards and the qualitative likelihood of their occurrence.

Another problem is the presentation of data resulting from the study in a form such that it is easy to comprehend. The presentation of this information is important when evaluating the likelihood of hazards, and when assessing the best ways of improving plant design. HAZOP tables and check lists accumulate the information well, but do not present it in a form which is conducive to easy backtracking through causes or gaining an overview of the results. Lawley [15] proposed the use of a logic diagram, in the form of a **fault tree**, to present this information in a more accessible form. Fault trees aid the visualization of causal relationships between deviations, and can also be used to estimate the quantitative likelihood of each deviation and fault event. It is, however, a difficult and

time consuming task to obtain fault trees from HAZOP tables. Care must be taken not to miss any relevant causal events. Fault trees and methods for their construction are described in more detail in Chapter 3.

2.3.2.5: Computerisation of HAZOP Recording

Computer aids may help to overcome some of the problems associated with HAZOP. Such aids could be used to store the findings of the study team and to automatically draw fault trees from the study data, aiding the team in their understanding of the causal relationships between the faults and deviations considered. The information would also be more readily available should operating difficulties occur, or if the process is modified. Such aids do exist [15,16], but further, more advanced aids may be envisaged.

The use of a computer to generate causes and consequences of deviations under investigation during the study could save a considerable amount of time. Such a computer program would probably require that the team study causal and consequential relationships between deviations **within each of the major plant items**, although libraries for common units could be built up. Such library models could be used directly, or modified to meet the specific requirements of the process. Using these data as a basis, the computer could generate both mini-fault trees, representing the immediate causal relationships between deviations, and complete fault trees for important failures. Having fault tree information available during the study would help to speed up the HAZOP process, could result in less errors and omissions, and, particularly if probability calculations are included, would help the team to evaluate the likelihood of failures.

2.4: The Use of Dynamic Modelling as a Systems Safety Analysis Method

It is possible to use dynamic modelling methods to predict the reaction of systems to initiating failure events. An example of this is the work done on emergency cooling systems for nuclear reactors [11]. Dynamic modelling involves the writing of general unsteady state differential equations for mass, energy and momentum transport, and their solution using computers.

This level of description has the advantage that the system behaviour can be accurately predicted. There are, however, some disadvantages [3]:

1. The models are complex and require a great deal of information on the properties of the species and equipment;
2. These models are often difficult to solve;
3. These types of models are often constrained to a specific operating region and to a specific mode of failure.

Dynamic modelling is a useful method for investigating how a process reacts to specific disturbances, but it is not really useful as a hazard identification method because those failure modes that can be analysed need to be identified and analysed before they can be modelled. It is well documented that potentially the most hazardous failure modes are those that are unforeseen [27]. These failure modes are best identified using a systems safety analysis technique such as HAZOP or fault tree analysis. When more detailed analysis of specific failure modes is required, however, dynamic modelling is the only effective strategy.

2.5: Discussion and Conclusion

In an age of large integrated process plants, operating at high temperatures and pressures, and involving toxic and reactive chemicals, there is a requirement for detailed systems safety studies to be carried out. Successful safety study requires the use of a comprehensive formalized systems safety analysis method. Unfortunately, such methods tie up teams of experts for many hours in an environment which is often not conducive to high levels of concentration.

Much work has been undertaken in recent years in an attempt to lighten this workload on safety study teams. This work has mostly taken the form of developing computer tools for such tasks as recording the results of HAZOP studies. More ambitious projects have concentrated on the automatic construction of fault trees; fault trees are useful both as an aid to HAZOP study, and in their own right for the representation of plant

failure chains and as a means to failure probability estimation. The next chapter examines fault tree construction and evaluation in more detail, and includes a description of computer methods for the generation and analysis of fault trees.

CHAPTER THREE

3. FAULT TREE ANALYSIS

3.1: Introduction

Fault tree analysis, usually abbreviated to FTA, is a backward chaining method of systems safety analysis. FTA is used to deduce the possible causes of an undesirable event, and to estimate the likelihood of the event's occurrence. The method makes use of fault trees, which provide a graphical means of showing the failure logic of the system under investigation.

The fault tree begins with a statement of an undesirable condition, known as the **top event.** Usually this event is a system hazard, but it can be any failure to meet the design intention, such as off-specification product. Below this event, any events that could cause the top event are written, along with logic symbols which describe how these causal events must be combined in order for the higher event to result. Each of these causal events in turn becomes a higher event, and their causal events are identified and recorded in the same way, resulting in a tree structure, with each higher event connected to its causal events. This process is repeated until no more causal events can be identified for any event in the tree.

The boon of fault tree representation is that the logical structure of sequences and combinations of causal deviations and failures are clearly laid out for the analyst to trace causes of undesirable plant conditions. Fussell [28] infers that this results in a number of benefits:

1. The analyst is encouraged to deduce failures in a systematic way, and to concentrate on one failure at a time;
2. Fault trees may be used to point out the aspects of the processing system which are important with respect to the failure under consideration;
3. Fault trees are a graphical aid which help to give an appreciation of the design, and of any associated problems or improvements

recommended, to management personnel not directly involved in the design process;

4. Fault trees may be used as the basis for quantitative systems reliability analysis;

5. Fault trees help to give the analyst a genuine insight into system behaviour.

In addition, FTA has also proven useful to operator training; the definition of start-up, shut-down and normal operating procedures; alarm analysis and fault detection [29,30].

3.2: Some Terms Used in Fault Tree Analysis

Some terms specific to FTA are used to describe fault tree building blocks and their properties. Some of these are described below.

A **top event** is the undesirable event under investigation, drawn at the top of the tree. A top event is either an **accident** or a **system failure.** An accident is an occurrence which is hazardous to personnel or the general public, or which causes damage to plant equipment, for example the rupture of a pressure vessel under its internal pressure. A system failure is the inability of the system to meet a design requirement. An example of this is product contamination.

Basic, primary, or **primal** events are terms used to describe a fundamental failure that requires no further development into causal events. Primary events are usually defined to be either unit failure or human error, but under some circumstances it is useful to develop these events into further causes, such as utility failure or factors that may lead to human error. Deviations that derive from an area of plant outside of the analysis may also be defined as primary events.

An **intermediate** or **secondary** event is any event in the fault tree, other than the top event, which requires further development into its causes. Intermediate events are usually deviations of process variables, properties or states from the values intended by the plant designers. Examples of these are high temperature, reverse flow, no heat transfer and

high control loop signal output.

A **repeated** event is an event which also occurs elsewhere in the fault tree. A special type of repeated event is a **common cause** or **common mode** event which causes multiple intermediate events to occur. Common mode events are of particular interest when they cause two or more inputs to an AND gate to be satisfied. For example, if one common mode event satisfies all of the inputs to an AND gate, then only this one event is necessary to produce the output event. Loss of utilities, such as steam supply, may often result in common cause events. Another example is the failure of a control valve which forms part of both a control and a trip loop, thus causing simultaneous failure of both loops.

The symbols used to define how causal events must be combined in order to produce the event above are known as **logic gates.** A logic gate defines the input condition that must be satisfied in order for a failure sequence to propagate up the fault tree toward the top event. There are two basic logic gates, although more may sometimes be used [31]. An AND gate provides an output if, and only if, **all** of the input events are simultaneously satisfied. An **OR** gate transmits the output event if **any one, or more than one,** or the input events are present. The graphical symbols used in fault trees, together with their meanings, are shown by fig. 3.1.

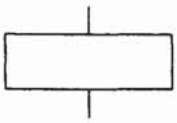
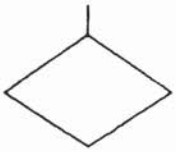
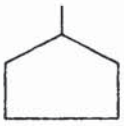
3.3: An Example to Illustrate the Use of Fault Tree Analysis

Fig. 3.2 shows a poorly designed trip system as described by Kletz [32]. The vessel's internal pressure is measured by a pressure transmitter (PT) and controlled by a pressure indicator controller (PIC), which adjusts the setting on a control valve (PCV). If the control system fails to work, and the pressure rises above the set point, the high pressure switch (PS) will close the control valve. Correct trip action in this way will prevent the pressure relief valve from unseating.

Part of the fault tree for the top event "high vessel pressure, causing pressure relief valve unseating" is shown in Fig. 3.3. This fault tree shows clearly the deficiencies of the system. Although both the

Fig. 3.1: Symbols Used in Fault Trees

EVENT REPRESENTATIONS

SYMBOL	MEANING
	AN EVENT WHICH RESULTS FROM COMBINATIONS OF FAULT EVENTS THROUGH THE INPUT LOGIC GATE. AN INTERMEDIATE OR TOP EVENT.
 OR 	A PRIMARY FAULT EVENT THAT REQUIRES NO FURTHER DEVELOPMENT.
 OR 	A FAULT EVENT THAT IS CONSIDERED BASIC IN A GIVEN FAULT TREE. THE TREE MAY BE DEVELOPED ELSEWHERE, OR THE EVENT MAY BE UNIMPORTANT.
	A SWITCH TO INCLUDE OR EXCLUDE PARTS OF THE FAULT TREE THAT MAY OR MAY NOT APPLY TO CERTAIN SITUATIONS.

LOGIC OPERATIONS

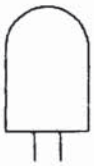
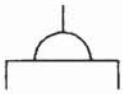
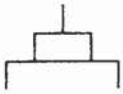
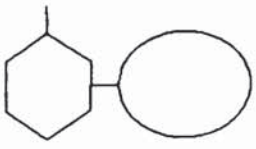
SYMBOL	MEANING
 OR 	AND GATE DESCRIBES THE LOGICAL OPERATION WHEREBY THE COEXISTENCE OF ALL INPUT EVENTS IS REQUIRED TO PRODUCE THE OUTPUT EVENT.
 OR 	OR GATE DEFINES THE SITUATION WHEREBY THE OUTPUT EVENT WILL EXIST IF ONE OR MORE OF THE INPUT EVENTS EXIST.
	INHIBIT GATE: THE INPUT EVENT DIRECTLY PRODUCES THE OUTPUT EVENT IF THE INDICATED CONDITION IS SATISFIED. THIS MAY BE USED TO DEFINE A STATE OF THE SYSTEM THAT ALLOWS THE FAULT SEQUENCE TO OCCUR.

Fig. 3.2: A Poorly Designed Trip System

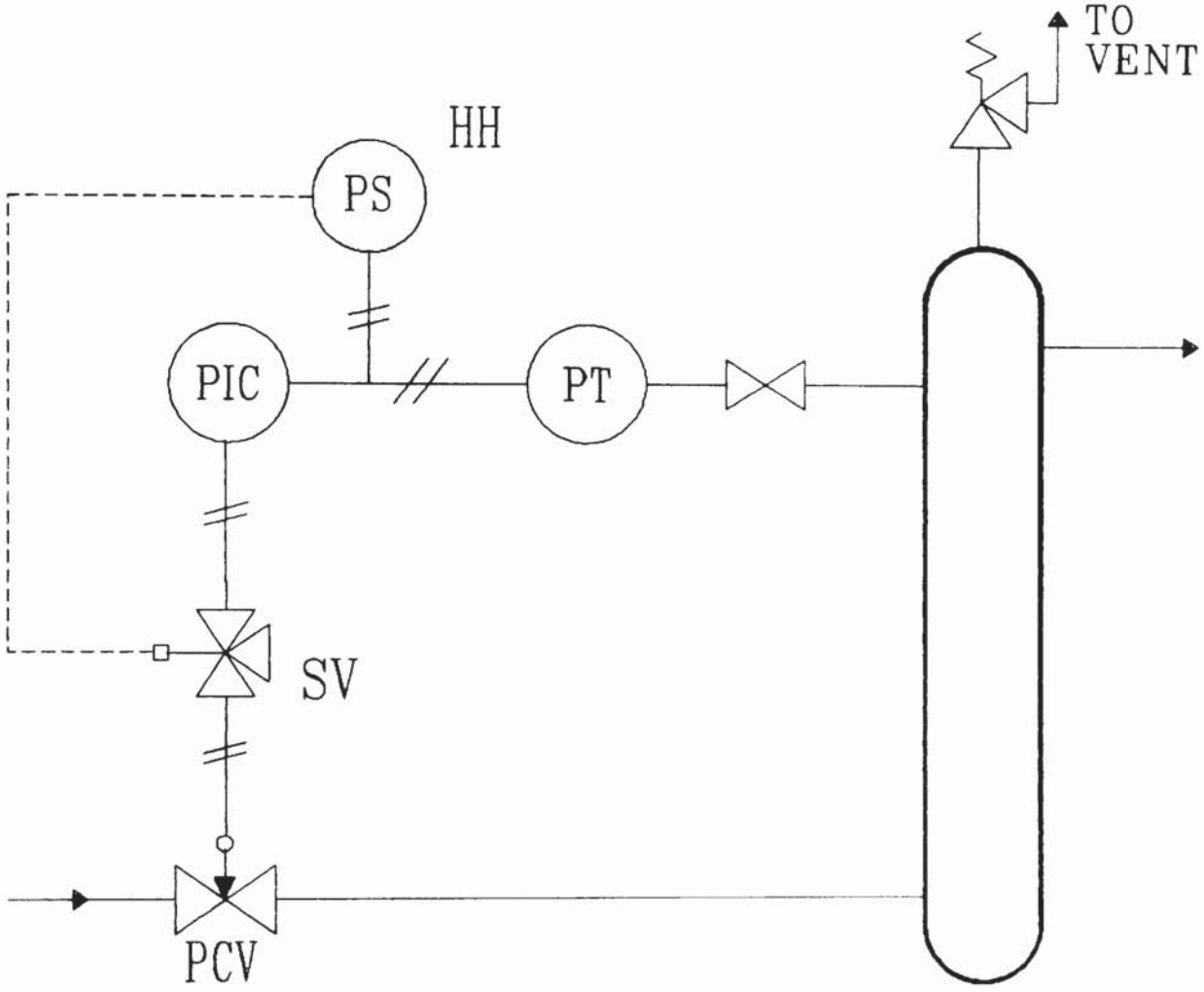
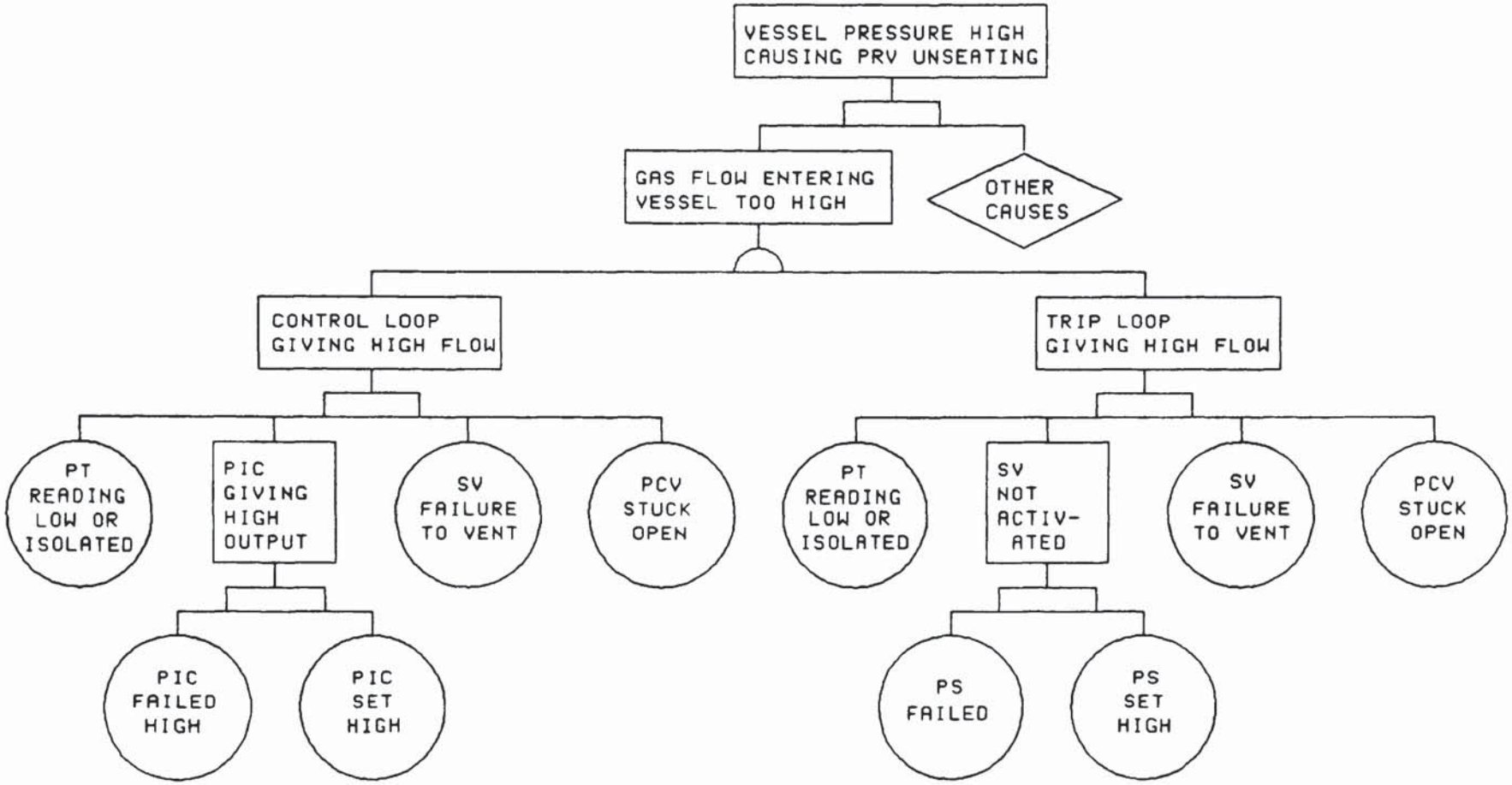


FIG. 3.3: PART OF A FAULT TREE FOR THE POORLY DESIGNED TRIP SYSTEM



control loop and the trip loop must fail for the top event to occur, there are three common mode events: "CV stuck open", "PT reading low" and "SV failure to vent". Each of these common mode events satisfies both inputs to the AND gate. If any one of these failures should occur then the top event could result. If the example were analysed quantitatively, it would be seen that the trip system is almost useless. The probabilities of control valve or pressure transmitter failure, both of which are on plant, greatly outweigh the probability of controller failure as this is in the clean atmosphere of the control room.

An improved system, with high reliability, is shown in Fig. 3.4, and its fault tree in Fig. 3.5. In this case, the control and trip loops operate independently, so there are no common mode events.

This is a typical example of the way in which fault trees may be analysed qualitatively in order to identify design problems. Browning [33] has studied some accidents that have occurred, and shows how fault tree analysis could have been used to identify the flaws which caused these accidents.

3.4: Automated Fault Tree Synthesis

Given the power of fault tree analysis, it may appear strange that the technique has not been adopted on a wide scale in the chemical process industry. There is one overwhelming reason for this: there exists no good automated, systematic methodology for fault tree generation for chemical process plants. Fault trees are currently mostly generated by hand, an arduous and expensive task when one considers the combinatorial aspects of large scale, integrated, complex processing systems. With hand generation of faults trees, the likelihood of missing critical failure pathways is increased, and a large investment in time and manpower required. In order for such a task to be practicable, there demands the use of some form of automatic, computer-based method for the synthesis of fault trees.

There are several requirements that any such automatic method must meet in order for it to be suitable for industrial use:

Fig. 3.4: A High Reliability Trip System

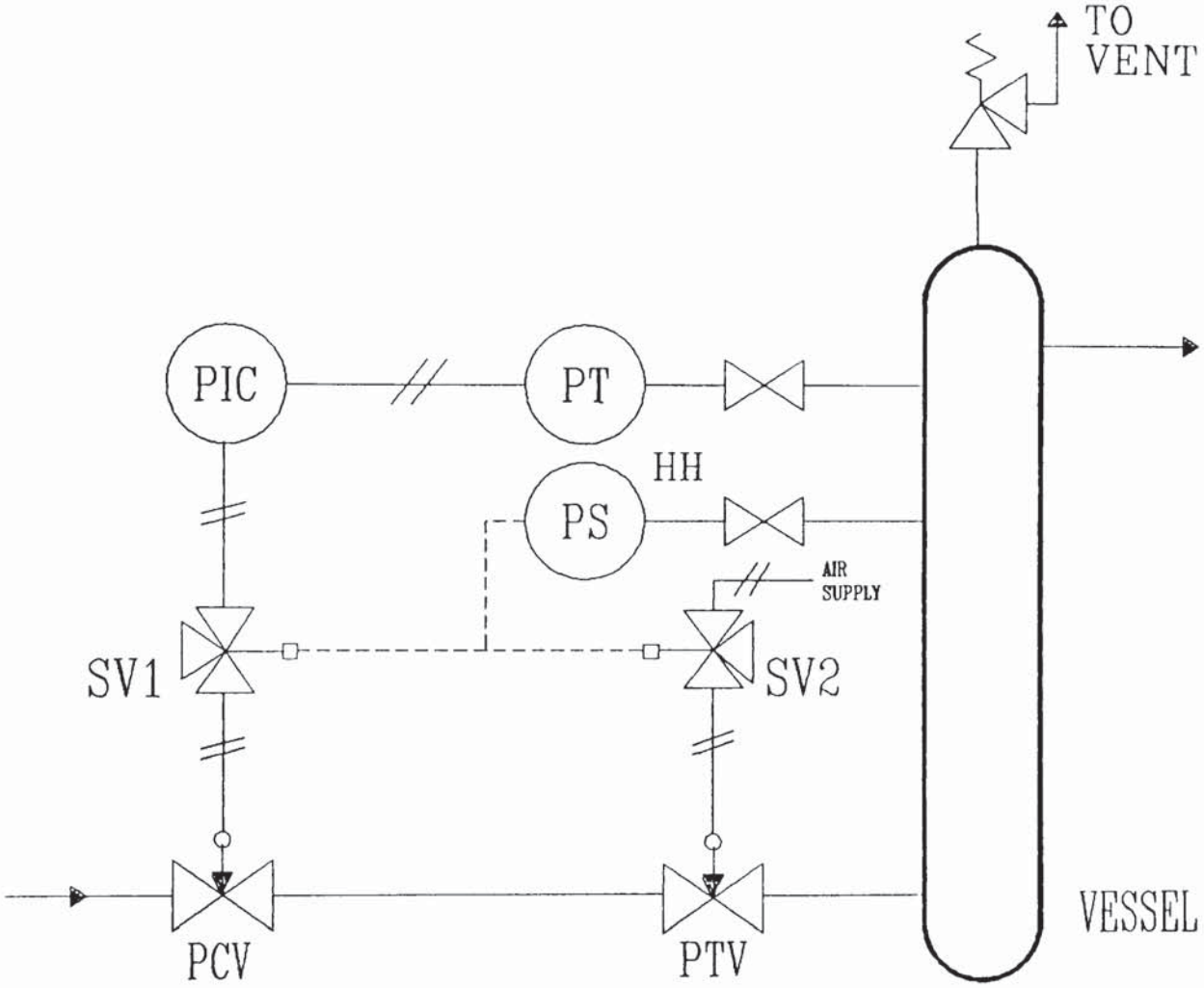
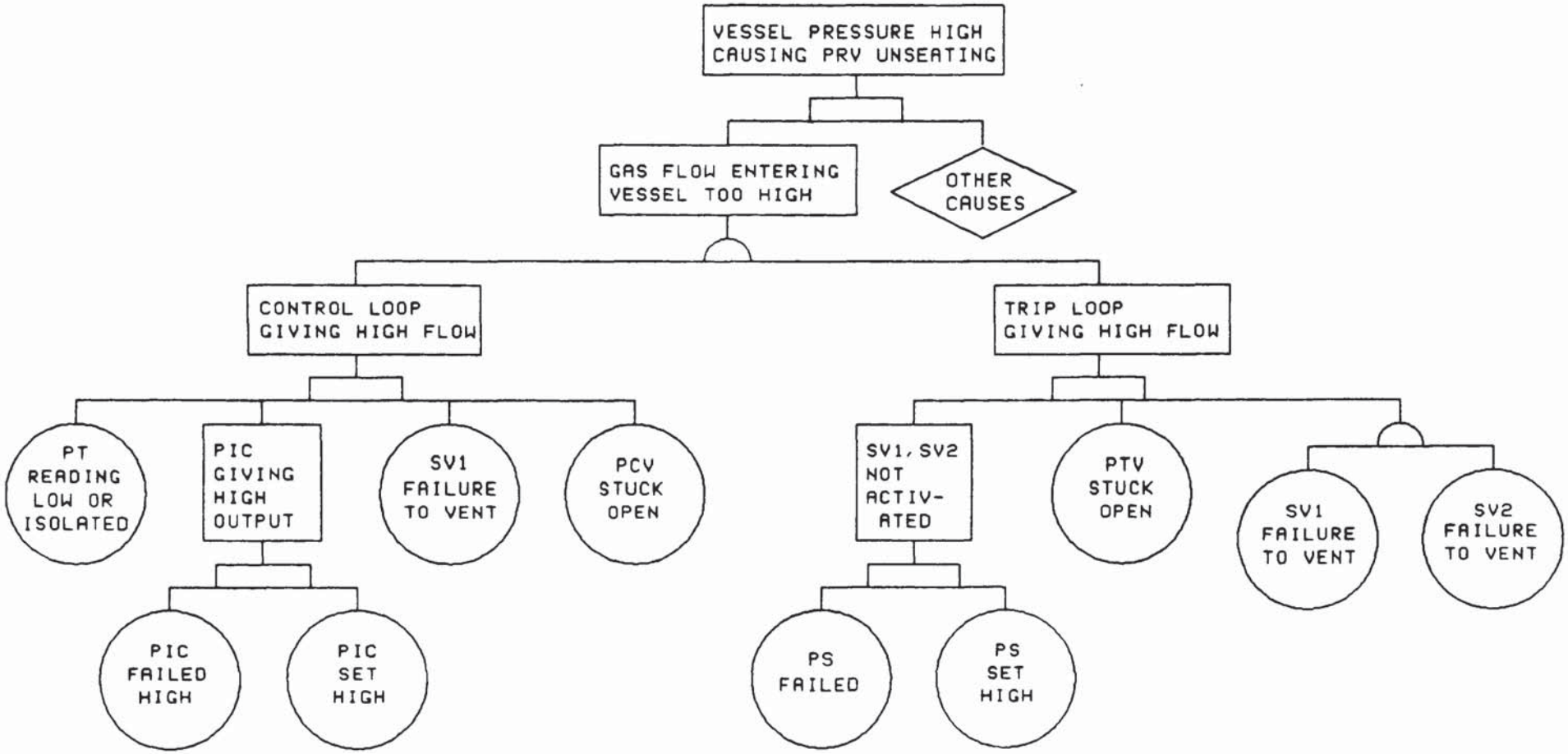


FIG. 3.5: PART OF A FAULT TREE FOR THE HIGH RELIABILITY TRIP SYSTEM



1. The method should produce fault trees that are of a manageable size, and with some apparent structure. It is a general criticism of automatic methods that they generate large, unstructured fault trees that are difficult to analyse.
2. The data required for input should be in a form which does not take too much effort to produce, else the labour-saving capabilities of the method will be considerably reduced;
3. Most automatic methods make some kind of assumptions about process plant. For example, it is commonly assumed that a pressure relief venting system will be capable of protecting a vessel against overpressure. In some situations, particularly when an existing plant has its capacity increased, this may not be the case; a combination of pressure relief venting and trip systems could be used. It is important, therefore, that any inherent assumptions about process plant the method makes must not limit its applicability to a wide range of problems;
4. The method must use multivalued logic; not just LOW and HIGH values of variables, but also NO, FULL, REVERSE and any other deviations that may be used by a systems hazard analysis method such as HAZOP. The method should also be able to handle two-way fault propagation through lines, loops and vessels; one-way fault propagation is unsuitable for chemical processing systems;
5. The method must be totally reliable; the fault trees produced should be complete and error-free.

It is not easy to design a method which meets all of these requirements, especially as some of them are somewhat contradictory. It is, for example, almost always necessary to make some concessions in terms of fault tree size and complexity in order for the method to maintain its applicability to a wide range of problems.

A number of automatic methods have been proposed, some of these implemented on computer. A review of some of these methods follows in the ensuing sections.

3.4.1: Process Unit Modelling Methods

A prerequisite for fault tree synthesis is the existence of an adequate description of system behaviour. A general process flowsheet may be considered to be a network of units interconnected in a specified fashion. This suggests a modular approach to modelling behaviour: system decomposition into basic units; development of a behaviour model for each module; and finally, linking the models together in a network representation. This method of process representation is called process unit modelling.

A unit modelling method for the automatic synthesis of fault trees for electrical circuits was developed by Fussell [34]. This method, called the Synthetic Tree Model, used failure transfer functions for each system component as a starting point. Component failure transfer functions are simply mini-fault trees: each output failure is related to its possible causes, these causes being either input failure states or actual failures of the component. For example, no current may be passed by a fuse if there is no voltage difference applied across it due to, say, a power supply failure; this is an input failure state. Alternatively, if the cause is fuse failure then this would be a component failure. The overall system fault tree is constructed by combining the relevant component failure transfer functions in a manner governed by a model of component connections, known as the Component Coalition Scheme.

In electrical systems, components often have only two states and system dynamic lags are rarely significant. In chemical processing systems, however, there are more variables, each of which may take several different states, system dynamics may be significant, faults may propagate in both directions through a unit or line; feedback loops are commonly used. These present major problems for component modelling methods for fault tree generation. For these reasons, Fussell's method is unsuitable for chemical systems, but it does form the basis of many of the fault tree synthesis methods for chemical process plant that have subsequently been developed.

Powers and Tompkins [3,10,35] were the first to develop a unit

modelling approach to fault tree synthesis for chemical processing systems. To describe exactly the behaviour of a unit it is necessary to define the mass, energy and momentum relationships that exist within the unit. There are several possible levels of description for these relationships. At one extreme, the unit may be modelled by writing general unsteady state differential equations for the relationships within the unit. Although this method enables accurate prediction of movements of the process from safe operating states to hazardous states, Powers and Tompkins identified several disadvantages of this modelling. These disadvantages were discussed in section 2.4.

Powers and Tompkins proposed the use of less detailed **information flow** models for process representation. Information flow is a simple way of indicating how the variables which characterize each unit are connected in a process network. Information is passed from component to component within a processing system by variables common to several components, the output from one component being the input to a connected component. The simplest information flow model indicates only that a particular output variable depends on several input variables. Approximation to the quantitative dependency between the variables is possible. A useful method is to indicate the sign of the dependency. A positive dependency indicates that an increase in the input variable results in an increase in the output variable, conversely a negative dependency means that an increase in the input variable results in a decrease in the output variable. The scope of these models can be further increased by including the magnitude and dynamics of the dependency.

Powers and Tompkins used information flow models incorporating signed dependencies as the basis of their fault tree synthesis method. The method involved searching for possible causes of events through the information flow structure, resulting in a fault tree which is useful for finding possible failure modes which may previously have been overlooked.

A similar method was used by Martin-Solis et al [36]. The starting point was a set of functional equations for each of the process components. A functional equation shows the output variable on the left hand side, and those input variables which affect it, along with the sign

of the dependency, on the right hand side. The functional equations used for a control valve, such as shown in Fig. 3.6, are shown in Table 3.1. Mini-fault trees for the component in question are derived, in part, from the functional equations. In order to complete the mini-fault tree, information about the possible failure modes of the component is required. This is called event/fault information, also shown in Table 3.1. The derived mini-fault tree for the control valve is shown in Fig. 3.7. The overall plant fault trees are produced by combining the mini-fault trees; a list of plant component connections is used to guide this process.

Fig. 3.6: A Control Valve

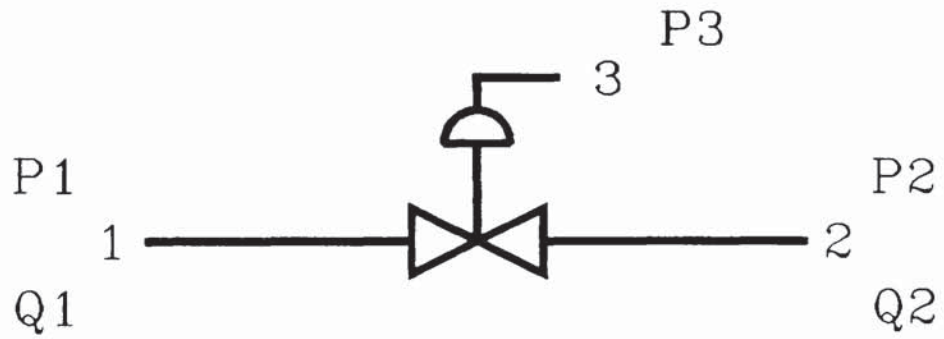
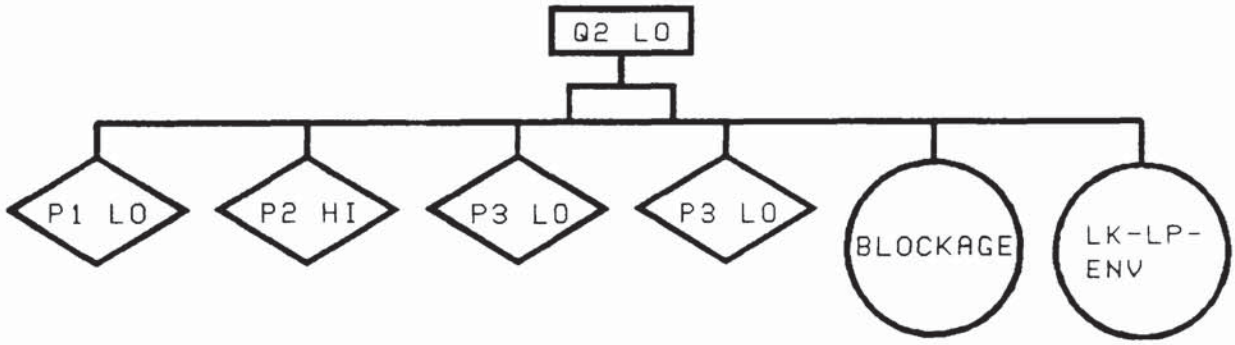


Fig. 3.7: A Derived Mini-fault Tree for a Control Valve



The methods of Salem et al [37,38,39,41], and Berenblut and Whitehouse [42] use decision table models of process components. A decision table catalogues all possible combinations of input and internal failure states on the left hand side, and relates them to values of the output variables on the right hand side. The decision table used by Salem et al for a control valve such as that in Fig. 3.6 is shown in Table 3.2.

Table 3.1: Functional Equation Model and Event/Fault Information for a Control Valve

Functional Equations:

$$\frac{dP}{dt} = f(Q1, -Q2)$$

$$Q2 = f(P1, -P2, P3)$$

Event/Fault Information:

BLOCKAGE: Q2 LO

LK-LP-ENV: Q2 LO

DUMMY: Q2 LO

(DUMMY is the top event of a separate minitree which has inputs P1 LO and VALVE STUCK)

Table 3.2: Decision Table Model for Control Valve

Q1	P3	P2		INTERNAL STATES		P1	Q2
1	-1	-1		-1		1	1
2	3	-1		-1		2	2
-1	1	-1		-1		2	1
-1	2	-1		-1		1	2
-1	-1	2		-1		2	1
-1	3	1		-1		1	2
-1	-1	-1		101		1	2
-1	-1	-1		102		2	1
2	-1	-1		103		2	2
-1	-1	1		103		1	2

1 = LO
2 = HI
3 = unchanged
-1 = don't care
101 = fails OPEN
102 = fails CLOSED
103 = fails STUCK

Each of the methods described above have their merits, but they all suffer similar problems. Component modelling results in a large number of process nodes at which variables are defined. This, in turn, results in fault trees which contain a large number of events, making them difficult for the analyst to assimilate. Fault trees produced by such methods also tend to lack any identifiable structure, and are unable to accurately model control loops and protective systems. It is not possible to represent the true functionality of control loops as the sum of their component parts, process unit modelling methods which attempt to do this will fail to produce adequate fault trees.

3.4.2: Digraph Methods

The first fault tree synthesis method to treat control loops separately from other plant units was developed by Lapp and Powers [43,44,45]. Again, the starting point is process unit modelling; the input-output models used are effectively signed information flow models as used by Powers and Tompkins, with an indication of the magnitude of the dependencies. Table 3.3 shows the input-output table for a control valve as in Fig. 3.6 for the pipe output flow rate Q2. The magnitude of the relationship is indicated by 1, which represents a normal effect, or 10, which represents an intense effect. A **directed graph** or **digraph** is then derived for each unit from the input-output models. A digraph is an information flow diagram; the digraph nodes represent process variables, defined at the input and output nodes for the unit, and unit failure modes. The digraph nodes are connected by signed edges; these show the sign and magnitude of the dependency between the connected variables, this dependency only applies in the direction of the arrow on the edge. The digraph for a control valve as in Fig. 3.6 is shown in Fig. 3.8. An edge from P3 to Q2 with a sign of +1 means that Q2 will increase as P3 increases. An edge may also be conditional upon some process state, the edge "0,stuck" from P3 to Q2 means that if the control valve fails STUCK, then P3 has no effect on Q2. Conditional edges are used to represent combinatory disturbances which will result in AND gates in the final fault tree.

Fig. 3.8: A Digraph for a Control Valve

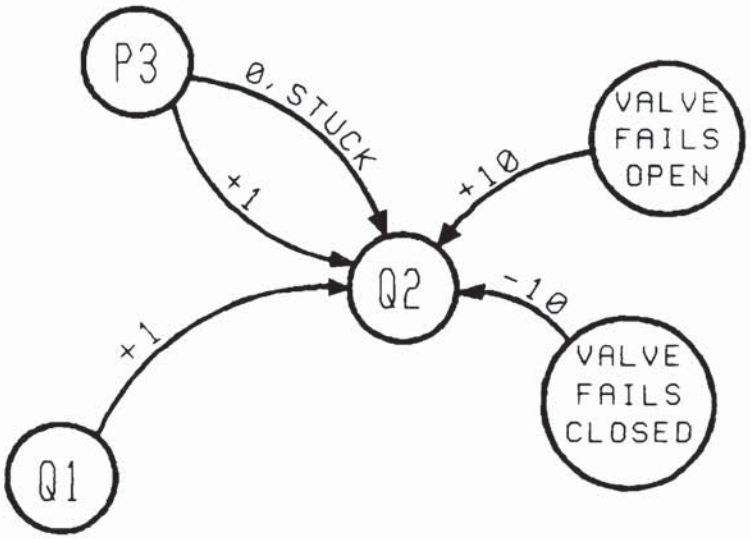


Table 3.3: Input-Output Model for Control Valve

Output: Q2

Input	Gain
Q1	+1
P3	+1
Fails OPEN	+10
Fails CLOSED	-10
Fails STUCK	0

1: normal effect
10: intense effect
+: ouput same sign as input
-: output opposite sign as input

The algorithm used to derive fault trees from digraph models of each plant unit consists of the following steps:

1. Generate the digraph for the entire plant from the unit digraphs and a connection table;
2. Identify all negative feedback loops and feedforward loops in the plant digraph;
3. Select the digraph node which corresponds to the top event of the fault tree;
4. Determine the local causes of this event by noting the inputs to the node;
5. Delete any local causes which violate consistency checks;
6. Select the appropriate operator, depending on whether there is a loop which passes through the current node. This operator is used to provide the logical connections between the local causes identified. If no loop is present, then the connecting logic is an OR gate. If a loop is present, then the appropriate feedforward or feedback loop operator is applied and the event is stored for later consistency checks. The operators used for feedback and feedforward control loops are shown in Figs. 3.9 and 3.10.;
7. Select a node corresponding to an undeveloped event in the fault tree, and repeat steps 4 to 7. If only primary events remain undeveloped, then the tree is complete.

Fig. 3.9: Feedback Control Loop Operator

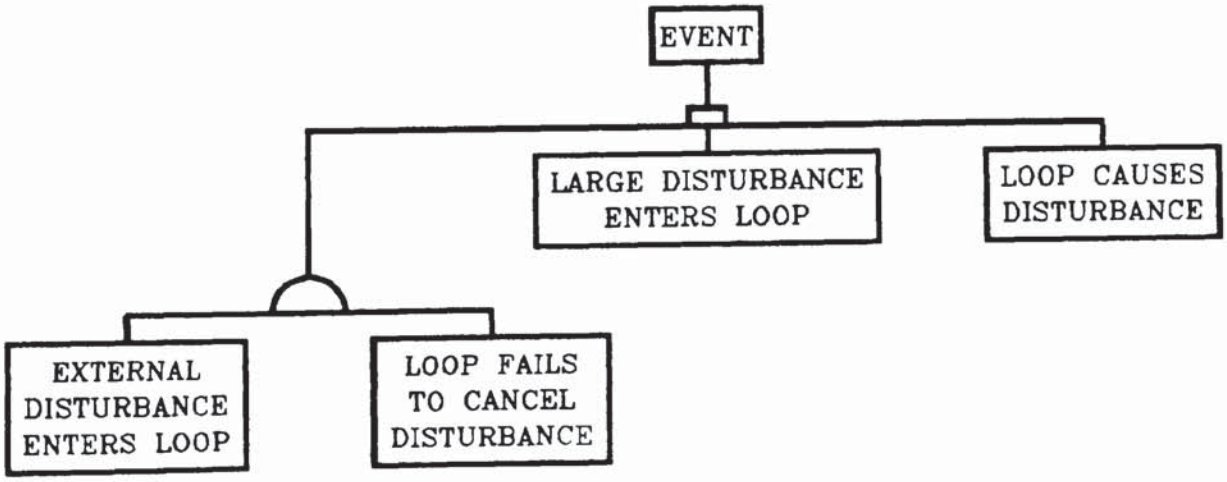
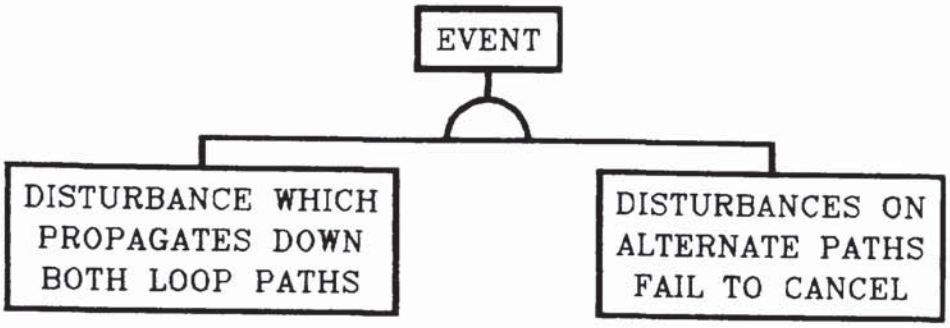


Fig. 3.10: Feedforward Control Loop Operator



Allen and Madhava Rao [46] used a digraph technique, similar to that of Lapp and Powers, to generate fault trees. It was argued that using ±10 to represent a gain beyond the control loop's controllable range leads to problems when there are multiple control loops present if some, but not all, of the loops are able to handle the disturbance. Instead, they do not define the magnitude of the gain; events beyond a control loop's controllable range are included in a listing of circumstances under which specific control actions will fail.

The digraph methods described in this section produce more structured fault trees than simple process unit modelling methods because they handle loops separately. They do, however, produce voluminous fault trees, because they use process unit modelling. Digraph methods do have some problems with multiple control loops; neither method provides an effective interface for the definition of uncontrollable events. A problem of signed information flow methods, such as digraphs, is that they only represent LOW and HIGH deviations of process variables. In order for a complete analysis, NO, REVERSE and FULL deviations should also be defined. Additional problems with these methods are discussed by Lambert [47] and Yellman [48] who dispute the validity and requirement for an XOR gate used by Lapp and Powers.

3.4.3: Modelling Based on Control Loop Structure

Shafaghi et al [47] proposed a fault tree synthesis method based on control loop structure. The method was developed as a manual technique;

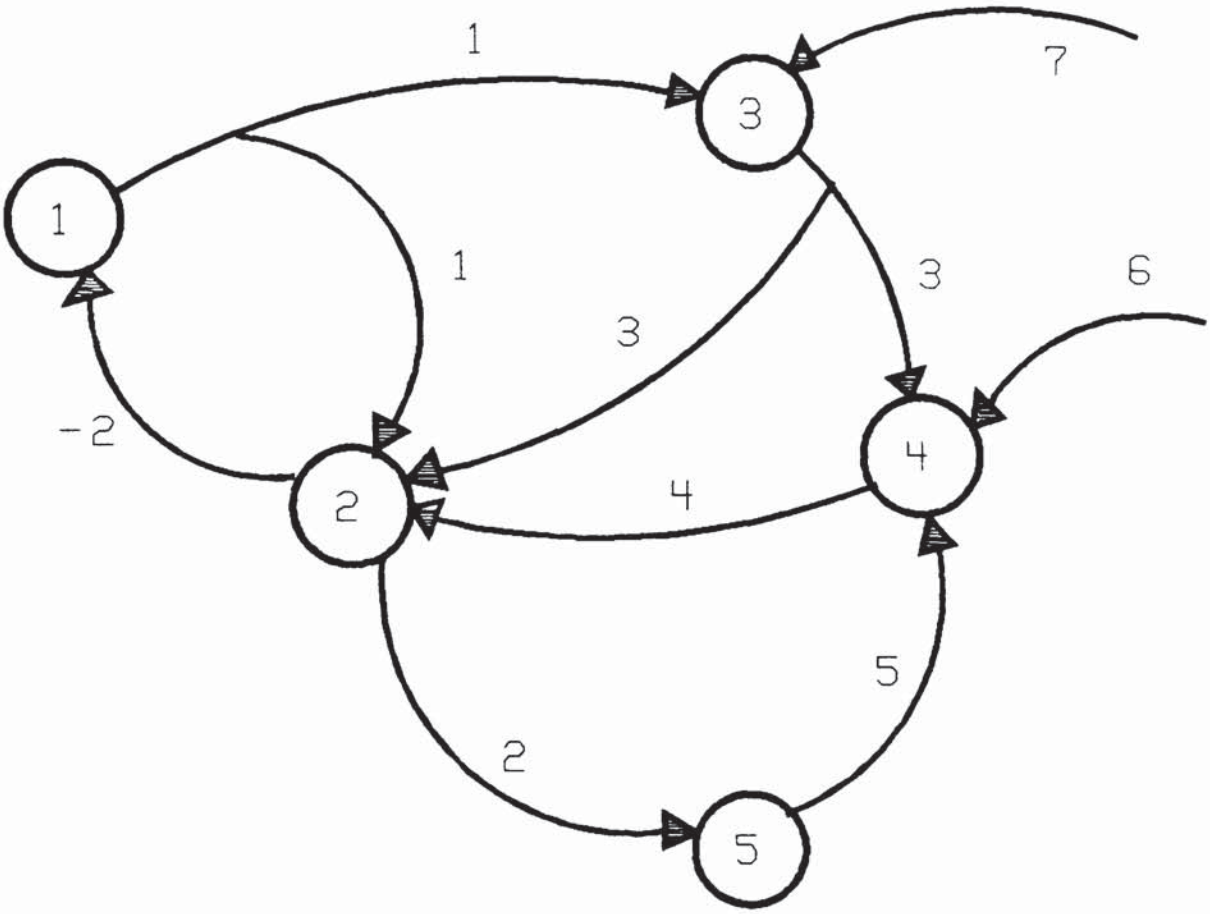
it has not been implemented on a computer, although there appears to be no fundamental reason why this should not be possible. The concept of the control loop structure approach is to provide a framework for the fault tree using the control loop connection structure of the plant. It is argued that fault trees structured in such a way are easier for the user to understand than trees produced by unit modelling and digraph methods.

In structure modelling, the plant is decomposed into a set of functional subsystems which generally consist of a number of units. In Shafaghi's method the subsystems considered are the plant's control loops; the subsystems include both control loop instruments, such as sensors, controllers and control valves, and other items of equipment which are sources of loop disturbances, such as vessels, heat exchangers and pumps. Failure of the control loop is defined as deviation of the controlled variable outside of the defined limits. The control loop has a single output, the controlled variable, and a number of inputs which can perturb that variable. In addition, there are other inputs which pass through the loop uninfluenced. For example, an input temperature deviation passes through a flow control loop uninfluenced as an output temperature deviation.

Protective systems are handled in two separate ways. If the protective system acts to prevent the deviation of a variable which is not controlled by a control loop, it is treated as a trip loop. Trip loops are treated in the same way as control loops, except that they have a single failure mode, namely that of protection failure. If the protected variable is controlled by a control loop, however, the protective system is treated as part of that control loop.

A loop digraph is used to describe the plant's information flow. The loop digraph consists of nodes which represent loops; edges are drawn between nodes where the output of one loop passes as an input to another. The edge number is simply the number of the source loop, if this is negative then it implies a disturbance that is uninfluenced by the target loop. A typical loop digraph is shown in Fig. 3.11. Additional edges are used to define inputs from across the plant boundary.

Fig. 3.11: A Typical Loop Digraph



From the digraph, a connection table is derived. Each row in the table represents a loop; the elements in that row represent the inputs to that loop. The connection table corresponding to the digraph shown in Fig. 3.11 is shown in Table 3.4.

From the connection table, a generalized fault tree is derived. This tree provides the structure for the final fault tree. The top event of the fault tree is selected; this is the deviation of a controlled variable of a particular control loop, and is placed at the top level of the generalised tree. The inputs to that control loop are obtained from the connection table, and placed below the top event. Each of these input loops is expanded into its inputs at the next level, until no more expansion is possible. The generalized fault tree obtained in this way from the connection table in Table 3.4 is shown in Fig. 3.12.

Fig. 3.12: A Generalized Fault Tree

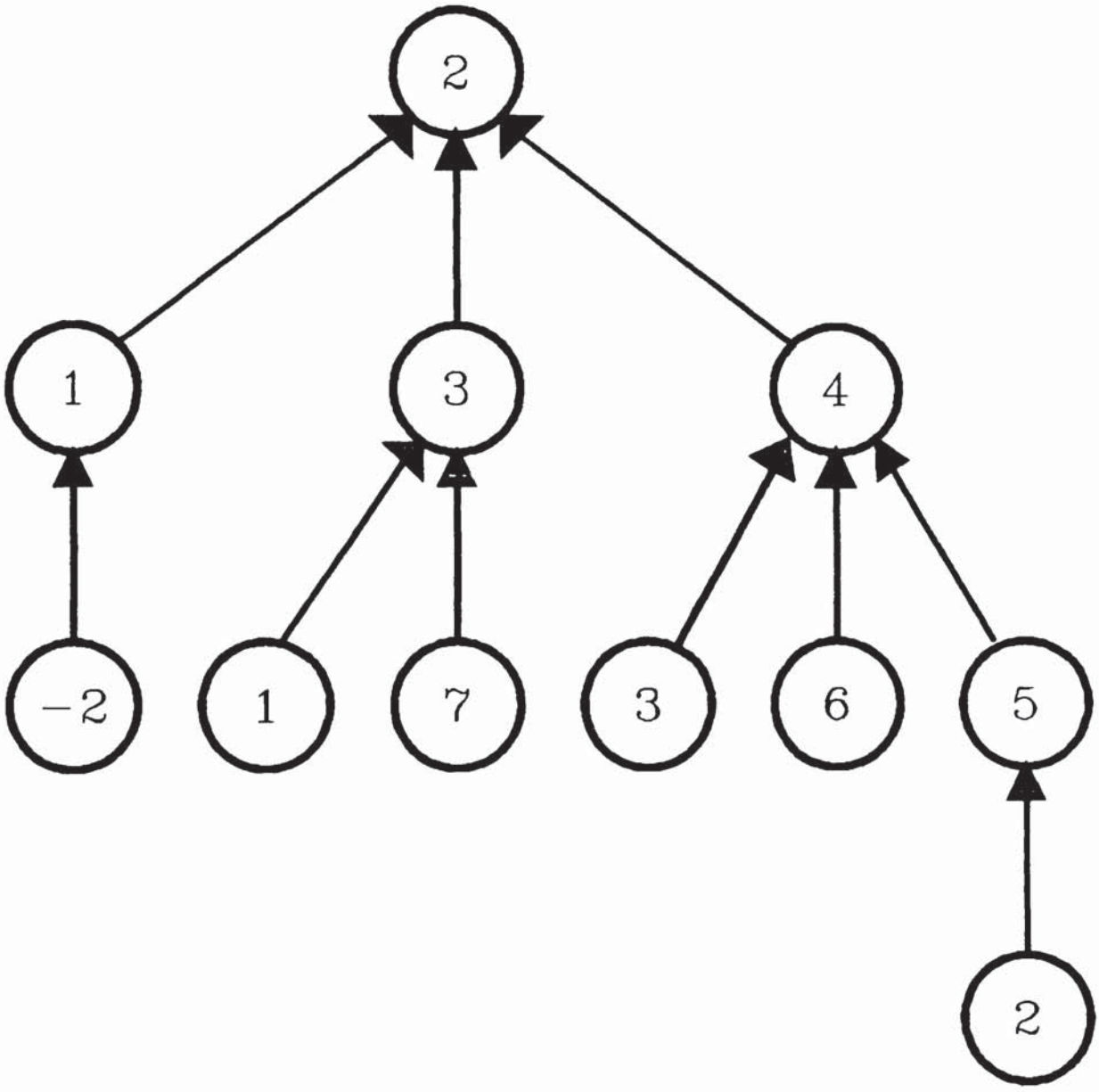


Table 3.4: Connection Table for Plant Digraph

LOOP	CONNECTIONS		
1	-2		
2	1	3	4
3	1	7	
4	3	5	6
5	2		

The complete fault tree is obtained from the generalized fault tree by replacing each loop, represented by a circle in the generalized tree, with the appropriate loop failure structure. Shafaghi identifies four modes of control loop failure: loop element failure; uncontrollable external disturbances; induced failure; invariant failure. In addition, Shafaghi combined control loop failure with protection failure by an AND gate where control and trip loops act in combination.

The fault trees obtained by this method bear out the authors' claims that the use of structure modelling may result in clearer fault trees than other methods. The method as described does have some limitations, however. Fault events are restricted to failure modes within control and trip loops; failure of plant items not included in loops is not considered. The method is thus incomplete: not all failure event chains will be identified. Several other types of failure mode are not considered: trip loop or pressure relief fail safe; the effects of loop failure modes and correct loop response to input deviations on the controlling variable. In addition, the method lacks a means of representing models of plant units and loops. This means that much of the work must be done by the user in identifying loop failure modes and connections.

3.4.4: Generation of Fault Trees from HAZOP Data

The methods described above all generate fault trees using a plant topology or information flow model and some form of process modelling. An alternative route is to use data generated by Hazard and Operability Studies as a starting point. Fault trees generated from HAZOP data can be used to present the results of a HAZOP study and for fault diagnosis and alarm analysis during plant operation.

Fault trees can be synthesized from HAZOP data, as demonstrated by Lawley [15] and Kletz [16]. Events in fault trees generated by these methods are stated in sentences or short phrases; this requires a lot of computer memory and physical space in the fault tree printout. The methods also require that relevant information is first extracted by the user from the HAZOP data, this is often a difficult and time consuming process.

Lihou [48,49] showed that construction of fault trees from HAZOP data is considerably easier if a fault coding system is used in conjunction with cause and symptom equations. In Lihou's coding system, faults and deviations are written as an item identification number, followed by a set of numbers or letters in brackets. A deviation from normal conditions in a line or vessel is identified by unique numbers which identify, in sequence, a property word, a guide word, and, where relevant, a component. For example, undesirable transport of synthesis gas in line 2 of an ammonia let down plant might be represented as L2(143); LIC1(-1) would represent a low set point to LIC1.

Deviations in pipelines or equipment are written at the left side of a cause equation, also known as an input-output equation, followed by an "=" sign, used to mean "is caused by". On the right of the equation are the causes, separated by a "+" sign to mean OR. Where causes are conditional on, for example, a safety system failure, they are bracketed together and connected to the conditional failure by a "*" sign to mean AND.

Symptom equations are used to show how major vessels respond to single deviations in lines connected to them or to malfunctions of the vessel itself. Symptom equations differ from cause equations in that the

cause is on the left, followed by a "-", and then a string of nodal deviations within the vessel, separated by "*".

Fault trees generated from HAZOP data stored in this way are quite good; they are reasonably concise, although slightly more verbose than trees synthesized directly by human analysts. In terms of structure, trees generated from HAZOP data are generally quite similar to those a human analyst might produce.

3.5: Qualitative Evaluation of Fault Trees

As well as providing a graphical representation of plant failure modes, fault trees may be used to estimate the likelihood of hazardous events occurring. Such estimates may be based either on qualitative or quantitative evaluation of fault trees.

It is possible to identify many weaknesses in plant design simply by inspection of fault trees, without the need to conduct a full quantitative evaluation. Most failures leading to a hazardous event should be combined with other independent events by at least one AND gate between the failure and the top event. Primary events which are combined by less AND logic than other primaries will be weak links in the system. "Less AND logic" used here is a relative term; this could mean that the event is combined by less AND gates, or that the events it is combined with are significantly more likely than those combined with other primary events.

It is also often possible to identify many common mode events during qualitative analysis; often these are the result of poor design, and may be rectified prior to further, quantitative, fault tree evaluation. The example of poor trip loop design used earlier in this chapter shows a case where qualitative fault tree evaluation could be used to identify design faults.

3.6: Quantitative Evaluation of Fault Trees

One of the major benefits of fault trees is the fact that they can be analysed quantitatively to estimate the probability of hazardous events. Hazard probabilities are particularly useful when assessing what control or

protective systems to incorporate. Such decisions are often based on economics; hazard probabilities, particularly those calculated for plants with and without the protective measures under scrutiny, aid the assessment of the value of specific measures. Hazard probabilities are also useful for comparing different control strategies or plant designs. By calculating the probabilities of all the major hazards, it is possible to estimate the overall hazard a plant represents. Comparison of probabilities for several hazardous events allows effective siting of safety measures where they are most likely to be needed.

Quantitative fault tree evaluation may sometimes be used to improve plant safety at no extra cost. Certain process variables may be "**over-protected**", the contribution that deviation of the variable makes to a hazardous event's probability is negligible when compared to another "**under-protected**" variable. In such cases it is often possible to reduce the protective measures against deviation of the over-protected variable whilst increasing the measures against deviation of the under-protected variable in order to reduce the overall failure probability at no extra cost.

The first step in the quantitative evaluation of fault trees is to estimate the probability of occurrence of all of the primary events. Some methods that are used to estimate primary event probabilities are described in the next section.

3.6.1: Event Probabilities

Random events, such as the failure of equipment, occur at unpredictable points in time. In order to quantify the likelihood of random occurrences, **probabilities** are used. The probability of an event of interest, where its absence, or an alternative event, could be observed is estimated from equation (3.1).

$$P(A) = \frac{\text{number of observations in which event A occurred}}{\text{total number of observations}}$$

$$= \frac{n_A}{n} \qquad \ldots(3.1)$$

where: P(A) is the probability of event A

In order to obtain a good estimate of probability, a large number of observations should be made. The probability is defined by:

$$P(A) = \lim_{n \to \infty} \frac{n_A}{n} \qquad \ldots(3.2)$$

Probability density distributions are used to represent the probability of a continuous variable which is the outcome of a random process. Failure distributions are functions of time f(t) that, when integrated over a specified time interval t_1 to t_2, provide the probability of at least one failure during the interval. This is known as the **cumulative failure probability**, F(t), which is defined by equation 3.3.

$$F(t_1 t_2) = \int_{t_1}^{t_2} f(t)dt \qquad \ldots(3.3)$$

Conversely, **reliability**, R(t), is the probability that an equipment unit, person, or system will continue to perform according to specification over any period of t years. The relationship between reliability and cumulative failure probability is shown by equation (3.4).

$$R(t) = 1 - F(t) \qquad \ldots(3.4)$$

There are many functions used to represent probability density distributions. Those most commonly applied to equipment failure probability estimation are the **negative exponential** and the **Weibull** distributions. The negative exponential distribution, defined by equation (3.5), contains one parameter, the failure rate, which is assumed to be independent of time, age, or environmental influence. The failure rate, r, defined by equation (3.6), is the reciprocal of the Mean Time Between Failures (MTBF).

$$f(t) = r \exp(-rt) \qquad \ldots(3.5)$$

$$r = \frac{N}{\sum_{i=1}^{N} t_i} = \frac{1}{\mu} \qquad \ldots(3.6)$$

where:

r = the failure rate

μ = mean time between failures

N = the number of failures that have occurred

t_i= the intervals between successive failures

The Weibull distribution, equation (3.7), contains two parameters: the scale factor, also known as the characteristic age of the equipment, and the shape parameter. Lihou [50] summarises the meaning of different values of these parameters, and describes methods for their calculation. A high value of the scale factor indicates a well designed system, good quality control, large safety factors or derated equipment. The shape parameter defines the distribution type. A value less than one is often observed for early failures, soon after the equipment is installed, and failures that occur shortly after major shut-downs; this indicates poor maintenance. Equipment that fails due to gradual deteriation has a shape parameter of between 3 and 3.5. Such a shape corresponds to the normal distribution: the probability density function will be nearly symmetrically distributed about the MTBF. A shape factor of between 1 and 3 is found for failures caused by a combination of gradual deteriation and random sudden overloads. It can be seen from equations (3.5) and (3.6) that the negative exponential distribution is a special case of the Weibull distribution, with a shape factor of 1 and a scale factor of MTBF.

$$f(t) = (\beta/\sigma)(t/\sigma)^{\beta-1} \exp\{-(t/\sigma)^{\beta}\} \qquad \ldots(3.7)$$

where:

σ = the Weibull scale parameter;

β = the Weibull shape parameter.

The probability that an equipment experiences a failure per unit time, given that it was repaired and as good as new at time zero and has survived to time t is known as the failure rate of the equipment, r(t).

The failure rate is obtained from the relation:

$$r(t) = \frac{f(t)}{1-F(t)} \qquad \ldots(3.8)$$

If the equipment failure rate is known, the failure probability and failure probability density function may be evaluated from equations (3.9) and (3.10). For the negative exponential distribution, F(t) and r(t) are defined by equations (3.11) and (3.12); the corresponding values for the Weibull distribution are equations (3.13) and (3.14).

$$F(t) = 1 - \exp \{ - \int_0^t r(t)\, dt \} \qquad \ldots(3.9)$$

$$f(t) = r(t) \exp \{ - \int_0^t r(t)\, dt \} \qquad \ldots(3.10)$$

$$F(t) = 1 - \exp (-rt) \qquad \ldots(3.11)$$

$$r(t) = r = 1/MTBF \qquad \ldots(3.12)$$

$$F(t) = 1 - \exp \{ - (t/\sigma)^{\beta} \} \qquad \ldots(3.13)$$

$$r(t) = (\beta/\sigma)(t/\sigma)^{\beta - 1} \qquad \ldots(3.14)$$

Typically, cumulative failure probabilities are estimated using the negative exponential distribution, equation (3.11). But care should be taken: Anyakora et al [51] showed that this may underestimate the probability by up to a factor of four times under poor conditions. Probability estimation from the Weibull distribution gives better results than the negative exponential distribution, but data for the scale and shape parameters is not widely available.

Standby safety equipment, such as trips, pressure relief valves and non-return valves, and redundant process equipment, such as duplicated pumps, may fail, and the failure remain undetected until the device is called upon. In order to detect failure of such items, they are inspected and tested periodically. If the inter-inspection interval is T (years), the proportion of time for which the equipment will remain in the failed state during any test interval is known as the **Fractional Dead Time** (FDT). For

the purpose of evaluating fault trees involving standby equipment, the FDT is used as a measure of the failure probability of the equipment. The FDT is defined by the relation:

$$FDT = (1/T) \int_0^T F(t)\, dt \qquad \ldots(3.15)$$

If failures of the equipment are assumed to be random, equation (3.11), which defines F(t) for the negative exponential distribution, may be substituted into equation (3.15) to obtain:

$$FDT = 1 - \frac{1 - \exp(-rT)}{rT} \qquad \ldots(3.16)$$

Expansion of equation (3.16) gives:

$$FDT = \frac{rT}{2!} - \frac{(rT)^2}{3!} + \frac{(rT)^3}{4!} - \ldots \qquad \ldots(3.17)$$

Typically, $rT \ll 1$, so equation (3.17) can be reduced to

$$FDT = rT/2 \qquad \ldots(3.18)$$

A period of time, known as the proof testing period, t', is required to inspect and test the equipment. The fraction of the inter-inspection interval, t'/T, is considered as part of the fractional dead time, because the equipment is unavailable. During the test, human operators are required to monitor and perform certain operations to simulate the function of the safety devices. Suppose that the number of operations involved is n, and that for each operation, i, there is an associated error probability, e_i. If the test is incorrectly performed then the FDT for that period becomes unity. Equation (3.18) can be rewritten to include these additional factors:

$$FDT = rT/2 + t'/T + \sum_{i=1}^{n} e_i \qquad \ldots(3.19)$$

3.6.2: Fault Tree Calculations

Once the primary events have been assigned probabilities, it is possible to evaluate the probabilities of the intermediate and top events. There are two main approaches to evaluation of non-primary event probabilities: bottom-up structural evaluation, and direct evaluation from minimal cut sets. Both of these approaches rely on two Boolean probability rules: the **Intersection Rule** and the **Union Rule.** The calculations involved in the quantitative evaluation of fault trees are usually very complex, thus computers are used to perform these calculations unless the fault trees are very small.

If an event T, which occurs only if all its causal events C_i occur, then the logical expression for T is

$$T = C_1 \cap C_2 \cap \ldots \cap C_n \qquad \ldots(3.20)$$

For all C_i independent of each other, the probability of T is given by

$$P(T) = P(C_1) * P(C_2) * \ldots * P(C_n)$$

$$= \prod_{i=1}^{n} P(C_i) \qquad \ldots(3.21)$$

This is the intersection rule.

Conversely, if a system fails if any one of several components j fail then, in terms of reliability:

$$R(System) = R_1 . R_2 \ldots . R_n = \prod_{j=1}^{n} R_j \qquad \ldots(3.22)$$

The component reliabilities are multiplied because each of the components must be working for the system to work. The probability of system failure is the complement of its reliability, thus:

$$P(T) = 1 - \prod_{i=1}^{n} R_j \qquad \ldots(3.23)$$

where T represents the system failure mode. In terms of probability, if any event C_i could cause T, then

$$P(T) = 1 - \prod_{i=1}^{n} (1 - P(C_i))$$

$$= \sum_{i=1}^{n} P(C_i) - \sum_{i=2}^{n} \sum_{j=1}^{i-1} P(C_i) * P(C_k)$$

$$+ \sum_{i=3}^{n} \sum_{j=2}^{i-2} \sum_{k=1}^{j-1} P(C_i) * P(C_j) * P(C_k)$$

$$- \ldots\ldots\ldots$$

$$+ \ldots\ldots\ldots$$

$$+ (-1)^{n-1} \prod_{i-1}^{n} P(C_i) \qquad \ldots(3.24)$$

This is the union rule. If all C_i are mutually exclusive, that is no two or more can simultaneously coexist, then this can be simplified to

$$P(T) = \sum_{i=1}^{n} P(C_i) \qquad \ldots(3.25)$$

It is, however, generally considered bad practice to add probabilities.

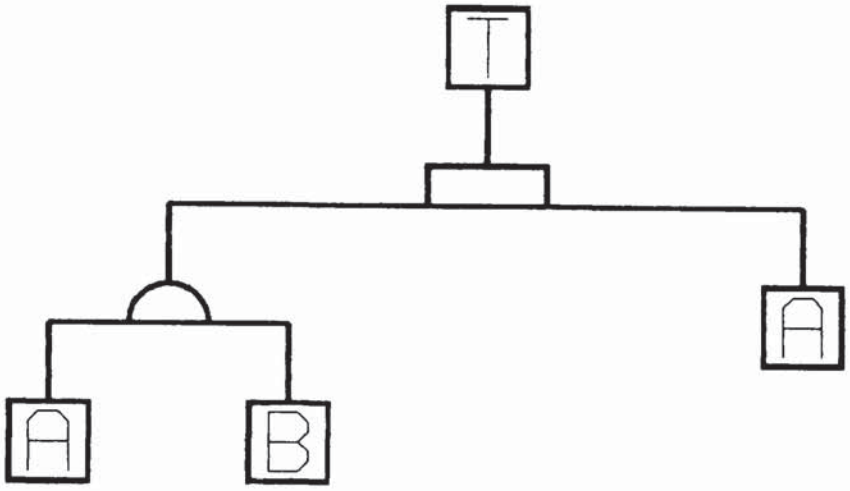
3.7: Probability Evaluation of Fault Trees with Repeated Events

The structural approach to the quantitative evaluation of fault trees involves traversal of the fault tree from bottom to top, calculating the probabilities of all the intermediate events from their input events' probabilities, using the Union and Intersection rules of combined probabilities described above. Initially, at the bottom of the tree, these inputs will be primary events. As the traversal proceeds upwards, previously calculated intermediate events also provide known inputs. This procedure is repeated until the top event is reached. In this method of calculation, however, there are two implicit assumptions:

1. The primary events for a given tree are independent, failure of one component does not depend on any other failure;
2. There are no two identical primary events in the fault tree.

When the second assumption is invalid, repeated events are present and normal probability calculation will produce an erroneous result. As an example, consider the fault tree in Fig. 3.13.

Fig. 3.13: A Fault Tree with Repeated Events



Using primary probabilities P(A)=0.1 and P(B)=0.2, and evaluating P(T) using the Union and Intersection rules

$$P(T)=1-(1-P(A))(1-P(A)P(B))$$
$$=1-0.9(1-0.02)$$
$$=0.118$$

It can be seen that this is incorrect: if A occurs, T occurs regardless of B; if A does not occur, T does not occur, regardless of B. Thus P(T)=P(A)=0.1.

There are two main sources of repeated events in fault trees: common cause failures and process loops. Common cause failures result when any one condition or occurrence is responsible for multiple component failures. Examples include: utility failure which may result in the failure of many dependent items of equipment; a design or manufacturing fault common to a number of, say, valves, pipe whip or water hammer could cause simultaneous failure of several such susceptible items. If the common mode events are fully explored, the common cause

will be repeated in the tree. Repeated events may also be produced by process loops, either recycle or control loops. A fault at one point in the process may propagate both upstream and downstream, it is thus possible that a given primary failure may lead to the top event by more than one sequence of events, in combination with more than one set of associated failures. To calculate probabilities for fault trees involving repeated events, special techniques have to be used, a review of some of these techniques follows.

3.7.1: The Conditional Probability Method

Lihou and Jones [52] proposed the use of a Bayesian conditional probability method. Using this method, the tree is evaluated conditionally for each repeated event. For example, for a given top event T, having a repeated event A in its fault tree, the tree is evaluated separately with two assumed conditions:

1. It is assumed that A does occur. The probability that T occurs if A occurs is evaluated.
2. It is assumed that A does not occur. The probability that T occurs if A does not occur is evaluated.

The unconditional probability of T is evaluated by adding the conditional probabilities, thus

$$P(T) = P(T:A).P(A) + P(T:\overline{A}).P(\overline{A}) \qquad \ldots(3.26)$$

where:

P(T) = The true probability of T
P(T:A) = The probability of T given that A occurs
$P(T:\overline{A})$ = The probability of T given that A does not occur
P(A) = The probability that primary event A occurs
$P(\overline{A})$ = The probability that primary event A does not occur

For the tree in Fig. 3.13, using equation (3.26):

$$P(T)=1*0.1+0*0.9$$
$$=0.1$$

This gives the correct result.

The method may be extended to cope with larger numbers of repeated events. For two such events A and B:

$$P(T) = P(T:AB).P(A).P(B) + P(T:A\overline{B}).P(A).P(\overline{B})$$
$$+ P(T:\overline{A}B).P(\overline{A}).P(B) + P(T:\overline{A}\overline{B}).P(\overline{A}).P(\overline{B}) \quad ...(3.27)$$

Clearly the number of terms in the expression is 2^n where n is the number of repeated events. This exponential increase in required calculation precludes the direct use of this method for trees containing large numbers of repeated events. The method could be used to give an approximate result by using Monte Carlo simulation methods, or by considering only those repeated events of most importance, and assuming all other events are independent. Some form of sensitivity analysis could be performed to select those repeated events of greatest significance for inclusion.

3.7.2: Methods Involving Minimal Cut Sets

Another approach involves the expression of every non-primary event as a sum-of-products. The **minimal** cut sets for the top event are obtained; from these it is possible to calculate its probability. A cut set for an event is a set of events which, if fully satisfied, will produce the event in question. The minimal cut sets define the minimum requirements, in terms of primary events, that must exist for the event in question to result. Several computer algorithms for determining the minimal cut sets of fault trees have been developed including Fussell's MOCUS program [53,54] and related methods [55,56,57,58,59]. These methods use a "top down" approach: the algorithm starts at the top event and works down toward the primary events. These algorithms suffer one common drawback: only the minimal cut sets of the top event under investigation are determined by one traversal of the tree. In order to obtain the minimal cut sets for intermediate events in the tree, each such event must be

treated as if it were a top event, and the algorithm applied again. Top down methods are thus extremely inefficient if failure probabilities of secondary events are required.

Many "**bottom up**" algorithms, such as those by Semanderes [60], Wheeler et al [61] and Bennetts [62], have also been developed. These start at the primary events and work up toward the top event. It is therefore possible to obtain failure probabilities for intermediate events, as well as for the top event, although this is not true of Bennetts' algorithm. Generally, all minimal cut sets generation algorithms involve complicated processing, but many improvements, in terms of speed and efficiency, are have been made in recent years.

The probability of a minimal cut set is calculated using the probability intersection rule, equation (3.21). If the minimal cut sets are s-independent, that is no primary event appears in more than one cut set, the top event probability may be calculated by regarding each minimal cut set as if it were a primary event and using the union rule, equation (3.24).

Whether or not the fault tree contains repeated events, there is no guarantee that the minimal cut sets will be s-independent. The fault tree in Fig. 3.14, contains no repeated events, but the minimal cut sets are not s-independent:

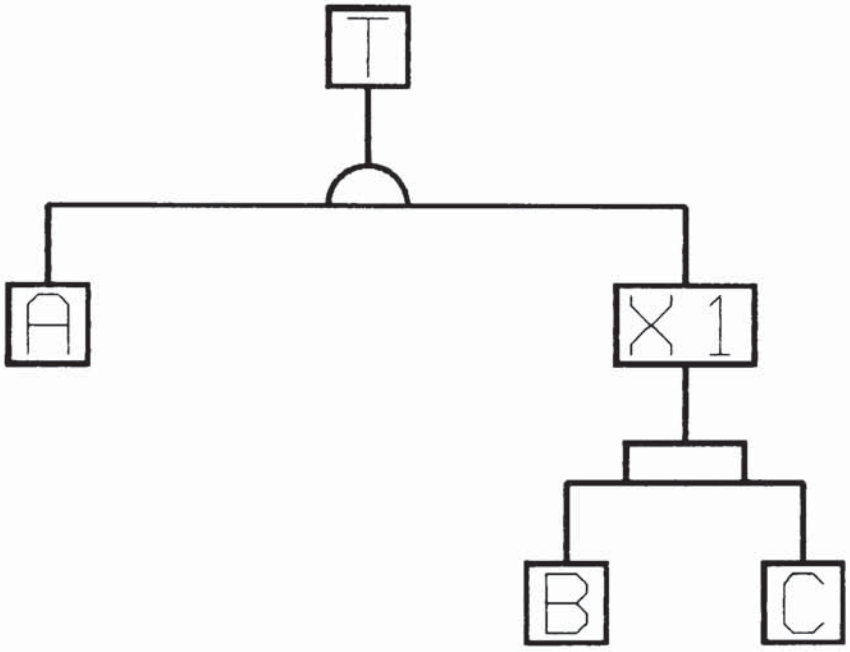
minimal cut sets: (A, B) ; (A, C).

In this case, the minimal cut sets may not be regarded as primary events, and the top event probability calculated by the union rule. In order to obtain the correct result, the probability of X1 must be evaluated by applying the union rule and the top event probability calculated by applying the intersection rule between A and X1.

There are two methods of overcoming the problem of non s-independence of minimal cut sets in probability calculation. The first method, developed by Vesely [63,64] and incorporated in a computer package known as KITT [65], is to manipulate the interaction of minimal cut

sets when applying the probability union rule so that primary events appear only once in each product. The basic assumption made is that the primary events occurring in the cut sets are independent; it is then possible to consider an intersection of minimal cut sets as a cut set or failure mode. The probability of such an intersection is simply the product of the unique primary event probabilities, where the primary event occurs in at least one of the minimal cut sets.

Fig. 3.14: A Fault Tree with S-dependent Cut Sets, But Without Repeated Events



$$P(M_i \cap M_j \cap \ldots\ldots \cap M_m) = \prod^{i,\ldots,m} q \qquad \ldots(3.28)$$

where: q is the probability of the primary event found in the minimal cut sets.

$\prod^{i,\ldots,m}$ is the product of basic event quantities where the basic event occurs in at least one of the failure modes i,..,m

This equation may be used to evaluate the intersection terms in the equation which results from applying the intersection rule between A and X1. This method allows the exact probability of the top event to be calculated. This technique is also used in the methods of Wheeler [61] and

Caldarola [56]. The computer time required to evaluate exact probabilities increases exponentially with the number of minimal cut sets. Thus for large, complex fault trees, which involve a large number of minimal cut sets, probability calculation using this method can be quite slow.

The products of minimal cut sets when using the probability rule can be eliminated if the minimal cut sets are arranged such that they are **disjoint** or mutually exclusive. This reduces the processing requirement: if there are N cut sets, then the number of terms to be evaluated is also N, as opposed to 2^N-1 using the previous method. Bennetts [62,66] developed a technique to generate disjoint minimal cut sets to take advantage of this; the method works for non s-independent minimal cut sets. Pairs of terms in the sum of products expression are compared to determine their "disjointness". The pair are disjoint if and only if one term contains at least one Boolean variable, the complement of which appears in the other term. If the terms are not disjoint, then one of them must be modified to make it disjoint relative the other.

If a pair of terms M_i and M_j are found to be non-disjoint, their relative component is defined as the non-empty set $\{y_1, y_2, \ldots\ldots, y_n\}$, where all y's are the uncomplemented Boolean variables that occur in M_i but not in M_j. By applying the Boolean Reduction Law, the expression for the union of M_i and M_j can be written as:

$$\begin{aligned} M_i \cup M_j = M_i &\cup (\overline{y}_1 \cap M_j) \cup (y_1 \cap \overline{y}_2 \cap M_j) \cup \ldots\ldots \\ &\cup (y_1 \cap y_2 \cap \ldots\ldots \cap y_{r-1} \cap \overline{y}_r \cap M_j) \\ &\cup \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &\cup (y_1 \cap y_2 \cap \ldots\ldots\ldots\ldots \cap \overline{y}_n \cap M_j) \end{aligned} \quad \ldots\ldots(3.29)$$

The probabilities of disjoint minimal cut sets produced in this way may simply be added to calculate the top event probability. It has been found, however, that generating minimal cut sets and then applying the disjoint algorithm can use a considerable amount of computer time. Works by Jionsheng [67] and Ramadaan [68] have shown that computer processing time can be reduced by using the disjointness principle to produce disjoint sum-of-products directly from the fault tree without prior

determination of the minimal cut sets.

3.7.3: The Top Down Recursive Algorithm

Page and Perry [69,70] developed a top down recursive algorithm, known as TDPP, for the direct computation of fault tree probabilities. In this method, the top event probability is calculated as a function of the input events' probabilities, the method of combination depending on the logic connecting the input events. For primary input events, the probability is inserted; otherwise further calls are made to evaluate the probabilities of the events in terms of their input events. This process is repeated recursively until all of the input events are primaries; the top event probability is then calculated. The algorithm includes a routine which checks that combined input events are s-independent; a disjointness method being used to handle repeated events.

The TDPP method is the only method of calculating exact fault tree probabilities for non s-independent fault trees which is really suitable for implementation on microcomputer. All of the other methods described above require large amounts of memory for the generation, storage and processing of minimal cut sets or conditional probabilities for large fault trees. The TDPP algorithm is compact and relatively simple, making it ideal for microcomputer implementation. One problem with the TDPP algorithm is that, as stated, the probabilities of intermediate events are not directly calculated. Fisal [71] developed a modified version of the TDPP algorithm which calculates intermediate event probabilities, and implemented this algorithm on a microcomputer.

3.8: The Use of Fault Trees for Alarm Analysis and Fault Diagnosis

As well as providing a good technique for hazard analysis, fault trees may be used as an aid to real-time fault diagnosis. Because fault trees clearly show the causal logic and event sequences that produce failure states, they are a very useful aid to process plant operators for carrying out fault diagnosis. Fault trees enable operators to trace causality much more easily than they could using any other method. The method used by Martin-Solis [36] is capable of producing fault trees in

real-time such that they may be used for fault diagnosis.

A further application of fault trees is to use them as a source of data for computer programs which trace the causes and possible effects of events that cause process alarms. Such programs may be used as on-line operator aids. An alarm analysis program using fault trees has been developed by Ulerich and Powers [72]. The program monitors many process variables and states, as well as alarm states, in order to verify event chains in fault trees. Once an event chain has been verified, two actions are taken:

1. The base events of the chain are identified to the plant operator as the causes of certain process alarms;
2. New conditional probabilities, based on the verified events, are calculated for the top events of fault trees. If the conditional probability of a hazardous event exceeds a certain limit, appropriate actions, such as plant shutdown, are taken.

An important consideration when attempting to validate event chains is that indicated values may differ from reality. If a failure event is indicated by process measurements, an AND gate may be defined, thus:

FAILURE STATES EXISTS = FAILURE STATE INDICATED
* INDICATOR FUNCTIONING CORRECTLY
...(3.30)

Many process alarms are in fact false alarms. It is therefore important to use several combined plant readings which verify that a particular fault event exists. Ulerich and Powers' method uses fault trees constructed from digraphs as a starting point. These are known as **prior**, or ***a priori***, fault trees, because they are calculated before any event is assumed to have occurred. Event chain validation is achieved using a **fault detection tree** which is derived by constructing an AND gate at each primal event. Inputs to this gate are the primal event itself and real-time data which would verify the event. The real-time verification data inputs are obtained by tracing the event up its causal branch in the prior fault tree.

A second derived tree, the **hazard-calculation tree**, is used to evaluate conditional, or ***posteriori***, probabilities of hazardous events. Cut sets are used to evaluate top event probabilities.

3.9: Discussion and Conclusion

Fault trees provide a powerful means for the graphical representation of systems failure mechanisms. Analysis of fault trees, either qualitatively or quantitatively, enables weaknesses in the system design to be identified; fault trees can also help the designer to make improvements to the system. The use of fault trees to estimate failure probabilities is an important feature: failure probabilities are useful when carrying out risk assessment, particularly when evaluating whether a risk is acceptable and when making an economic comparison between different levels or types of control and safety equipment and measures. Fault trees provide a means of representing HAZOP information that is useful to both the study team and to those to which the team report. Fault trees also provide a powerful means for process plant fault diagnosis, and are used in recent computer-based alarm analysis methods.

The generation by hand of fault trees for large chemical processing systems is prohibitively complex and time consuming; hand generation from HAZOP data is easier, but still rather laborious. These factors have lead to a great deal of interest in computer-based fault tree generation, but although quite good fault trees may be produced by programs acting on HAZOP data, no good methods for the generation of fault trees directly from process plant diagrams and data have yet been developed. Those computer-based methods that have been produced generally create large, unstructured fault trees, are incomplete, and are applicable to few real-world chemical process plants. The lack of a good computer package for the generation of fault trees is the major factor in the relative under-use of fault tree analysis in the chemical process industry.

The other barrier against wide scale use of fault tree synthesis software is mistrust of automatic computer-based systems when used for such an important and complex application as process plant safety analysis.

It is possible that this wariness could be reduced if a semiautomatic system were developed; such a system would do most of the fault tree synthesis, as for fully automatic systems, but would expect the user to solve certain problems which may be encountered. The advantages of such an approach are twofold. The user may have to consider the problem in a different way to which he had previously, and in so doing he may perhaps detect flaws in the plant. In addition, the need for the computer program to make limiting assumptions about plant operations is reduced. It is for these reasons that the author believes the search for fully automatic fault tree synthesis methods to be misguided, and instead favours a semiautomatic approach in which the program prompts the user to tell it how the processing system would react to certain failure situations.

The quantitative evaluation consists of two critical stages: estimation of the probabilities of primary events, and calculation of the top event probabilities for fault trees from these estimates. It is reasonable to assume that all equipment of a given type will have the same inherent reliability, but the way they are used may be totally different; their mean failure rate can vary from 0.25 to 4 times the average failure rate for those equipments. Such environmental factors may be accounted for by using the Weibull distribution to predict mean failure rates, but this does require further data to be made available for the estimation of parameters. Many applications of fault tree analysis however only require a comparison of failure probabilities to be made. Relative probability calculations are less sensitive to inaccuracies in primary event failure rates. For such applications, the negative exponential distribution may be used in conjunction with standard unit failure rate data, to obtain good estimates of relative probabilities.

Most methods to calculate top and intermediate event probabilities accurately for fault trees containing repeated events involve complex processing, making them unsuitable for microcomputers. The top down recursive algorithm developed by Page and Perry, however, is compact; this method brings quantitative evaluation of large fault trees involving repeated events within the realm of microcomputers.

CHAPTER FOUR

ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS

In this chapter, artificial intelligence techniques useful to the representation of chemical process plant for the purpose of computer-based synthesis of fault trees are introduced. The emphasis here is on rule-based "expert systems", and methods for their implementation, notably the OPS5 production language. A number of alternative techniques: programming languages such as LISP and PROLOG, and expert system shells, are described briefly.

4.1: Artificial Intelligence

There have been many attempts to define "artificial intelligence" (AI) but, in simplistic terms, artificial intelligence is the study of how to make machines or computers perform tasks at which, at the moment, people are better. AI researchers have studied many such tasks, such as game playing, theorem proving, general problem solving, perception of vision and speech, natural language understanding, and expert problem solving. Although the techniques used for the solution of these tasks may vary widely, one factor is common to all hard AI problems: it has been found that intelligence requires knowledge [73]. This knowledge presents some problems:

1. it is voluminous;
2. it is hard to characterize accurately;
3. it is often difficult to obtain;
4. it may be constantly changing.

Most AI research is based on the **physical symbol system hypothesis** as described by Newell and Simon [74]. They define a physical symbol system as follows:

> *"A physical symbol system consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure). Thus, a symbol structure is composed of a number of instances (or tokens) of symbols related in some physical way (such*

as one token being next to another). At any instant of time the system will contain a collection of these symbol structures. Besides these structures, the system also contains a collection of processes that operate on expressions to produce other expressions: processes of creation, modification, reproduction and destruction. A physical symbol system is a machine that produces through time an evolving collection of symbol structures. Such a system exists in a world of objects wider than just these symbolic expressions themselves."

They then state the physical symbol hypothesis as:

> *"A physical symbol system has the necessary and sufficient means for general intelligent action."*

The importance of the physical symbol system hypothesis is twofold. It is a significant theory of the nature of human intelligence; it also forms the basis of the belief that machines may be programmed to perform tasks presently performed by people, and generally considered to require "intelligence".

4.2: Problem Solving

There are two fundamental steps that are required to build an AI system to solve any particular problem:

1. Definition of the problem. This must include precise specifications of initial conditions, and what final conditions constitute acceptable problem solutions;
2. Problem analysis. A few important features of the problem may have an immense impact on the appropriateness of various problem-solving strategies.

Most AI problem-solving techniques involve definition of the problem as a **state space search.** A problem's state space is some form of structure by which the starting and final conditions, and all intermediate conditions required to bring about the solution, may be defined. For example, a fault tree's state space involves a structure for the representation of fault trees. The problem definition for fault tree synthesis is to move from the initial state (top event) to a complete set of solutions (primary events), through any number of intermediate states

(intermediate events). In order to find the solutions, the rules for fault tree construction must be employed.

The process by which a system moves from its initial state toward the solution(s) is known as **searching**. Search forms the basis of many intelligent processes, thus it is useful to structure AI programs in such a way as to facilitate the search process. There are two basic methods of knowledge representation which are used to guide state space search. One method, rule-based representation in production systems is described below. The other method is the use of **frame-based** systems. Frames are objects which have labelled **slots**. Slots may contain either values or some means for obtaining values, such as a **procedural attachment** which is a set of instructions that compute the slot's value, or a set of rules that conclude values for other slots in the frame, or procedural knowledge in the form of a LISP program, or an inheritance relationship with another frame. One of the major benefits of frames is their ability to allow class relationships which may be defined to represent hierarchical information.

4.2.1: Production Systems

An alternative structure to that offered by frame-based systems is provided by **production systems**. A production system consists of:

1. A set of rules, each consisting of a left side that determines the applicability of the rule, and a right side that describes the action to be performed if the rule is applied:

Fig. 4.1: A Generalized Production Rule

```
(RULE name
      IF (condition 1 is TRUE)        left side
      AND(condition 2 is TRUE)
      AND...
      THEN(execute action 1)          right side
          (execute action 2)
          ...
      ...)
```

The left side conditions may include NOT conditions, but OR statements are not allowed;

2. A database that contains whatever information is appropriate to the the particular task. Some parts of the database may be permanent, while other parts may contain information pertinent only to the solution of the current problem. The database may be structured in a way which is appropriate to the type of problem to be solved;

3. A control strategy that specifies the order in which rules will be compared to the database, and a means of resolving the conflicts that arise when several rules are satisfied simultaneously;

Fig. 4.2 shows the control cycle employed by production systems.

An important factor in the development of a production system to solve any particular problem is the control strategy employed. The control strategy selects the next rule to be executed. There are two criteria that a control strategy should satisfy:

1. It should cause motion. In order to achieve this, different rules should be selected, depending on the current state of the database.

2. It should be systematic. The selection criteria must be well defined and known to the programmer, else the system may fail.

Two common, but complementary, search control strategies are **breadth-first search** and **depth-first search**. In breadth-first search, branches from the current node are searched in preference to movement down to a lower node. When depth-first searching is employed this preference is reversed: traversal down the tree is of higher priority than branch search. This is illustrated by Fig. 4.3 which shows a search tree with the nodes numbered in the order in which they are searched by a breadth-first strategy; Fig. 4.4 shows the same tree, numbered to show the order of a depth-first search.

In order to solve hard problems, characterized by large search spaces, it is often preferable to compromise the requirements of mobility

Fig. 4.2: The Production System Control Cycle

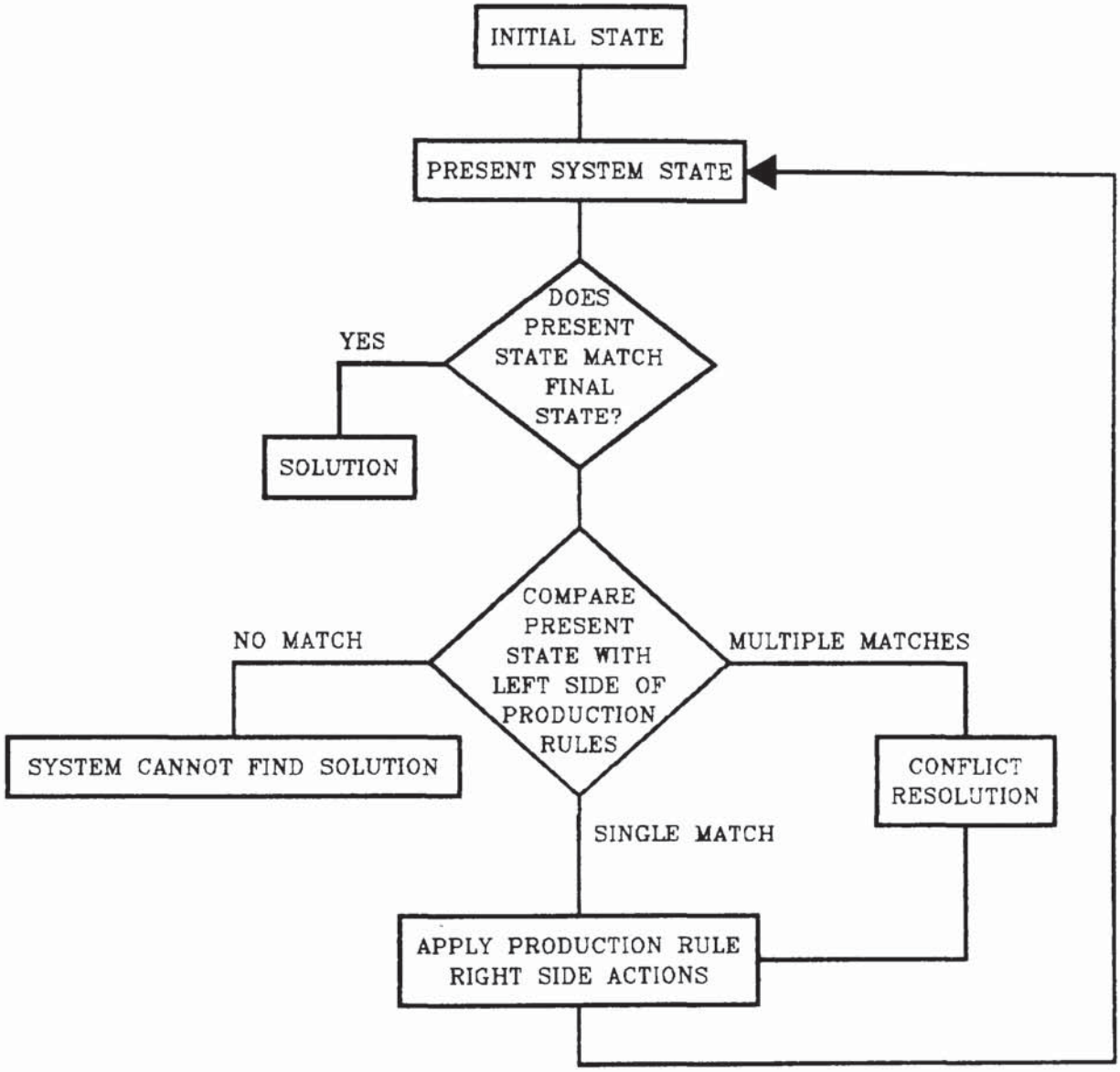


Fig. 4.3: A Breadth-first Search Tree

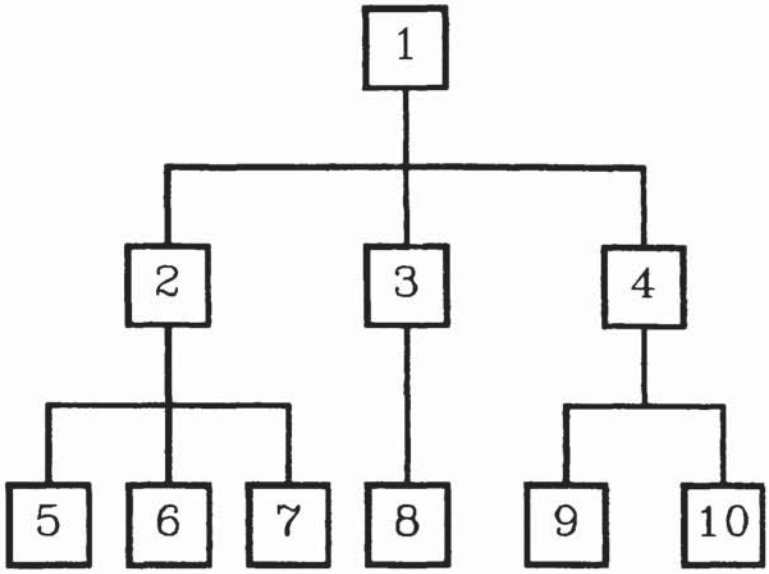
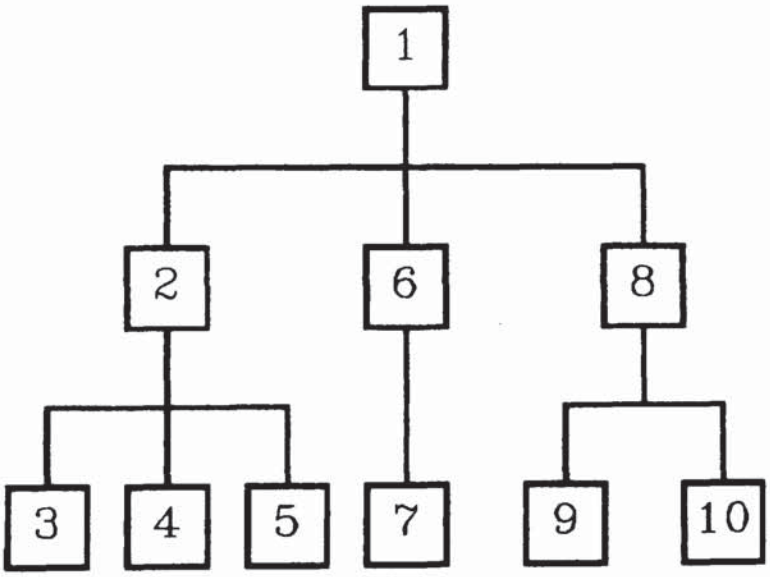


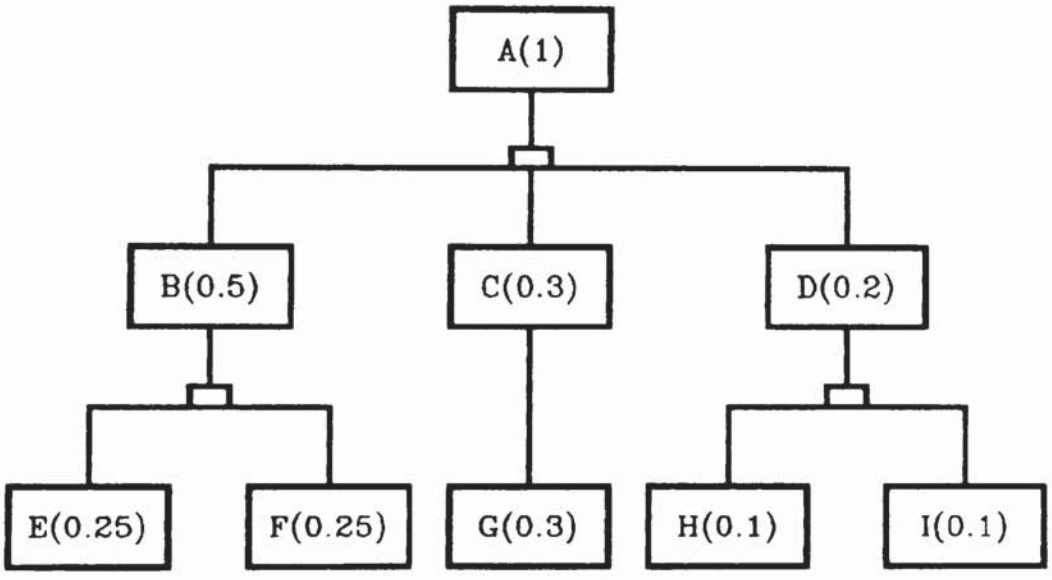
Fig. 4.4: A Depth-first Search Tree



and systematicity to construct a control structure that is no longer guaranteed to find the best answer, but will almost always find a very good answer. Such a structure utilises **heuristics.** A heuristic aims to improve the efficiency of the search process, but may make a sacrifice in terms of completeness in order to achieve this. There are many general-purpose heuristics, but it is often possible to improve on these by constructing special-purpose heuristics that exploit domain-specific knowledge to solve particular problems.

An example of a heuristic that could be used in real-time alarm analysis utilising fault trees, is that when an alarm occurs, possible causes could be generated in a depth-first manner, with the node chosen for expansion being that with the highest conditional probability. Such a system would attempt to generate the most likely causes first. This search process is illustrated by Fig. 4.5. The primary events would be generated in the order E, F, G, H, I. It can be seen that this is non-optimal: event G is the most likely causal event. A better heuristic might also take branching into account.

Fig. 4.5: A Heuristically-driven Search Tree



The use of heuristics as a driving mechanism in a system which will accept the first viable solution it finds may result in a non-optimal solution. Simon [75] argues that in most cases this will be acceptable, this is being due to two factors:

1. There is good evidence that humans, when problem solving, are not optimizers but rather will accept the first viable solution encountered, regardless of whether a better solution may exist;
2. Although the approximations produced by heuristics may not be very good in the worst case, worst cases rarely arise in the real world.

If good heuristics are selected, it will normally be possible to find an acceptable solution to a hard problem much more quickly than the time it would take to find the optimal solution.

Another important factor to consider when building a production system is the direction of reasoning to be employed. It is possible to search either from the initial states to the goal states, **forward reasoning**, or from the goal states to the initial states, **backward reasoning**. Forward reasoning is accomplished by matching the current database, initially the starting state, against the left sides of production rules. This process is repeated until the goal state is reached. This is the reasoning mechanism assumed in Fig. 4.2. Backward reasoning, also known as goal-directed reasoning or backchaining, starts with the goal configuration(s). The search proceeds by finding all of the right sides which match the goal. These are the rules which, if applied, would generate the goal state. The left sides of the rules are then applied to generate the nodes at the next level of the tree. This process is repeated until the initial state is achieved.

Rich [76] describes three factors that influence the choice between forward and backward reasoning:

1. The number of start states and goal states. It is preferable to move from the larger to the smaller, consequently easier to find, set of states;

2. The branching factor. This is the average number of nodes that can be reached from a single node. It is preferable to move in the direction with the lower branching factor, as this will reduce the space that needs to be searched in order to find a solution;
3. Will the program be required to justify its reasoning process to a user? If so, it is important to proceed in a direction that corresponds more closely with the way the user will think.

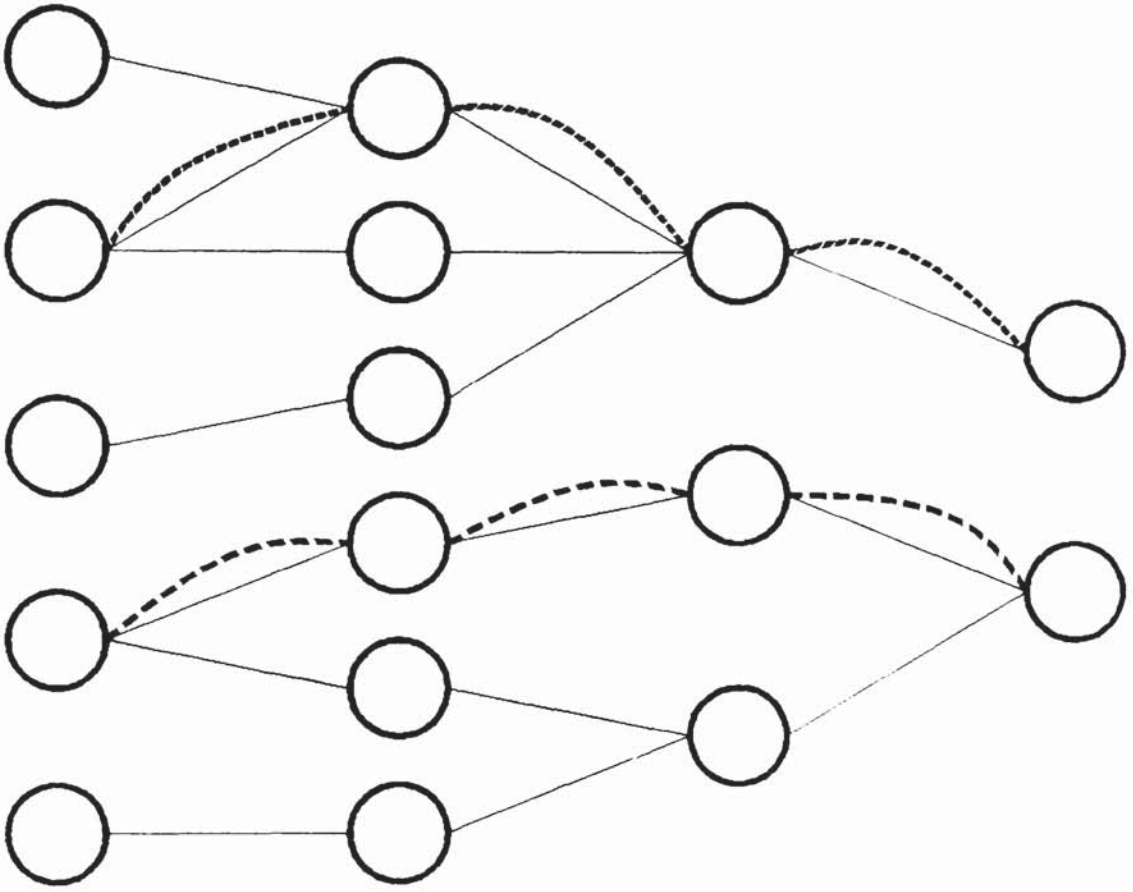
A third option is to use a **bi-directional search**, in which the system works simultaneously forward from the start state and backward from the goals until the paths meet somewhere in between. For an undirected search this method works quite well [77], but bi-directional heuristic search has proven less successful [77,78]. This phenomenon is illustrated in Fig. 4.6. The forward and backward searches may pass each other, resulting in a less efficient search than would have been required by a unidirectional heuristic search process. Bi-directional search has, however, proven more successful when implemented by a program which is carefully constructed to exploit each of the search directions in exactly the situations in which they work best. This facility is provided by the language PLANNER [79], but this language has never been fully implemented.

4.3: Expert Systems

The only major breakthrough, in terms of industrial usefulness, made by artificial intelligence researchers is in the development of **expert systems**, also known as **intelligent knowledge based systems**. There are many tasks that require a great deal of specialized knowledge that most people do not possess. These tasks can only be performed by experts who have accumulated the required knowledge; but such experts are usually in short supply. Expert systems are computer programs that codify a great deal of such expert knowledge, and use this knowledge to perform useful tasks that could, otherwise, only be performed by human experts.

The most important characteristic of an expert system is that it contains a large database of knowledge. The method used to represent this knowledge is of critical importance to system performance. Many

Fig. 4.6: A Search Tree to Illustrate the Problems Encountered by Heuristically-driven Bi-Directional Search



SHOWS BACKWARD ELEMENT OF SEARCH

SHOWS FORWARD ELEMENT OF SEARCH

early expert systems were composed of sets of fairly unstructured LISP functions. It is now widely recognized, however, [80] that a more organized approach to knowledge representation is important. In addition, it is important to separate a system's knowledge base, which should be able to grow and change with use, from its program part, which should be relatively stable. For these reasons, the most widely used way of representing domain knowledge in expert systems is as a set of production rules. The operation of these systems is then controlled by a simple procedure, the exact nature of which depends on the nature of the knowledge being used and the information required.

4.3.1: Probabilistic Reasoning

One important factor in the development of expert systems is the treatment of probabilistic knowledge. In some applications, knowledge may be certain: a set of facts imply other facts. In other applications, however, knowledge may be uncertain; in such cases the system must be designed to take these uncertainties into account. R1 [81], later known as ZCON, an expert system that is used to configure DEC VAX computer systems, is an example of an expert system in which probabilistic information is unnecessary: in all situations it is possible to state the correct action to be taken with absolute certainty. An example rule from the R1 system is shown in Fig. 4.7. This system only needs to find one viable configuration, thus the first solution found will suffice. The system works mainly by forward chaining, the rules have well-specified left sides which means that most of the work is done in choosing the correct rule to execute at each step, rather than in searching.

In contrast to this approach, MYCIN [82], a program that diagnoses infectious diseases, makes use of uncertain knowledge. A number between zero and one is used to represent the certainty of the inference contained in the rule. A rule from MYCIN is shown in Fig. 4.8.

MYCIN is able to combine several pieces of inconclusive information to make a conclusion about which it is almost certain. The search is directed backwards from its goal of determining that there are disease-causing organisms that must be treated. MYCIN must find all

possible diseases that might account for a patient's symptoms, thus a complete search is conducted. The use of heuristic information when all solutions must be found is rarely of value as all paths must eventually be searched; thus MYCIN depends heavily on search.

Fig. 4.7: A Production Rule From the R1 Expert System

IF: the most current active context is distributing massbus devices;
AND there is a single-port disk drive that has not been assigned to a massbus;
AND there are no unassigned dual-port disk drives, and the number of devices that each massbus should support is known;
AND there is a massbus that has been assigned at least one disk drive and that should support additional disk drives;
AND the type of cable needed to connect the disk drive to the previous device on the massbus is known;
THEN: assign the disk drive to the massbus.

Fig. 4.8: A Production Rule from the MYCIN Expert System

RULE577
IF: the infection which requires therapy is meningitis;
AND the strain of the organisms is known;
AND the morphology of the organism is known;
AND the patient has been seriously burned;
THEN: there is weakly suggestive evidence (certainty: 0.3) that Pseudomonas-aeruginosa is one of the organisms (other than those seen on cultures or smears) which might be causing the infection.

4.3.2: Human Interaction with Expert Systems

In order for an expert system to be an effective tool, the user must be able to interact with it easily. Many expert systems prompt the user for further information during the course of their reasoning process. Unless the reason for this information is very obvious, it is helpful if the

system is designed such that the user can interrogate it to find out why the information is required. It is also important in some problem domains that the system explains its reasoning. In medical diagnosis, for example, a doctor will place little trust in a program that does not explain itself.

One approach to the problem of explaining the reasoning process undertaken by an expert system is to chain backward through the rules that have been used to reach the conclusions in question. At each step the system may tell the user what it already believes, before chaining back a further step to identify from whence those beliefs arose. This is the explanation method used by the TEIRESIAS knowledge base of the MYCIN system [83]. In order to backtrack, the system must keep a record of the entire tree traversed. This is a good method for systems which employ a relatively simple reasoning structure, but it is very unwieldly for systems which manipulate large quantities of data: such systems may require several hundred or more rule executions to reach a solution!

A method more applicable to complex systems is described by Swartout [84]. This method does not attempt to track each piece of data individually, but rather uses general knowledge about what each program part does to answer questions about the program as a whole. This method is useful for making general answers about a program's reasoning process, but is unable to explain which rule was responsible for the generation of a particular piece of information, and would perhaps not satisfy sceptical users.

4.3.3: Knowledge Acquisition

An important, and often difficult, part of expert system construction is the elicitation of knowledge from a human expert. Many experts make use not only of knowledge which can be precisely defined, but also of "rules of thumb". Obtaining such knowledge from an expert is a difficult process which may satisfactorily be carried out only by a trained knowledge engineer. Knowledge elicitation is generally considered to be a difficult and time consuming process.

The objective of the TEIRESIAS system was to reduce the role of the

system builder in the knowledge acquisition process by allowing the domain expert to interact directly with the system. Although TEIRESIAS was developed specifically as a front end for MYCIN, the ideas and techniques are widely applicable. TEIRESIAS is concerned with three key issues:

1. Comprehensibility: in order that the system be comprehensible to the end user, TEIRESIAS aims to provide a system in which the user can understand what is going on in his own terms;

2. Debugging: this is a serious problem in expert system construction. Initially, the knowledge base may contain errors or omissions. It is important that these be ironed out before the system is put into general use. Unfortunately, it is often difficult to discover such errors: a large system may execute any one particular rule infrequently. If that rule contains an error it may not be picked up unless very comprehensive testing is carried out;

3. Knowledge elicitation: TEIRESIAS includes not only the initial building of the system, but also the addition or modification of knowledge at a later stage.

TEIRESIAS provides explanations to the user of how results are obtained and the motivation for conclusions and advice. The user can debug the knowledge base by querying results and demanding explanations. He can then use the knowledge transfer facilities to add or alter knowledge in the database. In order to perform these functions, TEIRESIAS requires more knowledge than MYCIN: knowledge about itself. This knowledge is represented by a set of meta-rules containing information about how it reasons and how much it knows. Such a meta-rule is represented in Fig. 4.9.

The system also has "rule models" which are abstract descriptions of rule subsets which have characteristics in common. These models are used to form expectations about rules; the system will check with the user if these expectations are not realised. The models are organised in a tree form, TEIRESIAS starts at the top, descending until it finds the closest fit.

If no close fit is found then the rule is beyond the expectations of the system. Natural language processing facilities, in the form of keyword matching, are used to improve interaction with the user.

Fig. 4.9: A Meta-rule from the TEIRESIAS Knowledge Base

IF (1) the infection is a pelvic-abscess
(2) there are rules that mention Enterobacteriaceae in their premises
(3) there are rules that mention gram positive rods in their premises

THEN there is suggestive evidence (0.4) that rules dealing with Enterobacteriaceae should be invoked before those dealing with gram positive rods.

4.4: Programming Languages for Expert Systems

There were two separate approaches taken during the development of AI languages. The first approach, adopted largely within the artificial intelligence research community, was to abandon the principle of generality and to develop processing techniques which were effective for specific problems within defined problem domains. With these techniques, processing power is largely derived from the large amount of domain specific knowledge coded into a system. This effectively models the behaviour of humans, who rarely use very general procedures but gain effectiveness from their knowledge about the problem domain. This approach to expert problem solving, the development of "knowledge-based" systems, is usually implemented using the list processing language LISP, or higher-level tools, many of them based on LISP, which have also been developed.

The second approach, undertaken mainly by mathematicians and workers in logic programming, attempted to maintain generality by working with predicate calculus and identifying proof procedures. Many such methods were extremely inefficient, but the programming language PROLOG was developed from this research. PROLOG possesses an efficient control

structure, and has become a widely used general-purpose language for knowledge programming.

4.4.1: LISP

LISP, which stands for LISt Processing language, is based on work done by John McCarthy during the 1960s on non-numeric computation [85]. Since its conception, LISP has undergone much development, it now forms the basis of many AI programs.

The fundamental structure of LISP is the list. Lists are made up of a combination of abstract symbols, known as **atoms**. Atoms represent the most fundamental program object, and therefore have no component parts. Atoms are assigned names, and may have attributes or properties attached to them. The most important attribute of an atom is termed its **value**.

A list is made up of a combination of atoms, enclosed in parentheses. For example, the expression

(WRITES-PROGRAMS STEVE LISP)

is a list consisting of three atoms, "WRITES-PROGRAMS", "STEVE" and "LISP". Lists may contain additional lists embedded within them, for example

(WRITES-PROGRAMS (PROGRAMMER STEVE) (AI-LANGUAGE LISP)).

Each unit of the list is termed an **element**. In the above example the first element is an atom, the second and third elements are lists, each of two atoms. Lists are terminated by the **null** or **empty** list, NIL. This list terminator is not shown in normal list syntax.

The processing of LISP expressions, called **S-expressions**, may be achieved using only the five basic functions listed below. Although LISP systems provide many more than these functions, many can be built from these five.

CAR: gives the value of the first element of a list, for example the CAR of (WRITES-PROGRAMS STEVE LISP) is the atom WRITES-PROGRAMS.

CDR: gives the value of a list without its first element, for example the CDR of of (WRITES-PROGRAMS STEVE LISP) is the list (STEVE LISP).

CONS: joins two expressions to form a list, for example the CONS of the atom PROGRAMMER and the list (STEVE) is the list (PROGRAMMER STEVE).

ATOM: gives the value "true", T, when applied to an atom, otherwise gives the value "false", NIL.

EQUAL: gives the value "true" if two expressions are equal, otherwise gives the value "false".

LISP programs consist of sets of functions. The control structure of a program is largely "applicative", in that it is guided by the application of functions to arguments, which may themselves be functions. In order to define useful functions, two higher level constructs are required: DEFUN is used to create the actual function definition; COND represents a conditional. Each line of code following COND may be read as a separate "IF....THEN...." statement. COND evaluates by moving down the list of statements, evaluating the "IF" part. The first that evaluates to something other than NIL, or "false", causes the evaluation of its corresponding "THEN" part. Fig. 4.10 gives an example of a function which could be used to check whether an individual is a programmer. The function looks for the atom WRITES-PROGRAMS as the first element of a list, such that if the list were (WRITES-PROGRAMS STEVE) then the function would return the value true. The second "IF" statement will always be true; the "THEN" part delivers "false".

LISP is the most common language used by AI programmers. Expert systems may use LISP constructs to represent domain specific knowledge, the most usual method being the use of property lists. In a property list, an atom may have "properties" associated with it. These properties are

defined by the programmer and provide a convenient way of representing associations between objects. A "property" is defined as a "property/value" pair linked to an atom. For example, the atom "STEVE" may be given the property "INTERESTED-IN" and the value "FUNGI". The generality of the property list structure has proved valuable for constructing specialized forms of knowledge representation, such as semantic networks and frame structures. Property lists are also commonly known as **object-attribute-value (O-A-V) triplets**; an object has a certain attribute which may itself take a value. An O-A-V triplet is really a simple and convenient form of **semantic net.** Semantic nets consist of nodes and labelled links between nodes that describe their relationship. Such a structure is very general and provides a good way of representing information for making inferences and for representing hierarchical relationships.

Fig. 4.10: A LISP Function

```
(DEFUN PROGRAMMER (FACT)
  (COND
    ((EQUAL (CAR FACT) 'WRITES-PROGRAMS) T)
    (T NIL)      ))
```

4.4.2: Expert System Shells

The programming of expert systems using LISP is a major process: function definition; conflict resolution strategy; control structure design; knowledge acquisition and representation must all be decided upon and implemented. The benefit of expert systems is that they may be used for the solution of hard real-world problems, but the difficulty of expert system construction using LISP is a major hurdle to their widespread use. In order to ease the burden of expert system construction, several "empty" expert systems, or **expert system shells**, have been developed.

Although most expert systems have a highly domain dependent control structure, some attempts have been made to separate the control structures of successful systems from their knowledge bases. The result

is an empty expert system which may be used to form the control structure for expert systems for different knowledge areas, but with similar characteristics. This approach was adopted to extract the EMYCIN domain-independent structure from MYCIN [86]. EMYCIN has been successfully applied to a number of domains, such as respiratory intensive care, engineering structure calculations, pregnancy advice, blood disorders and ventilation management.

An alternative approach to the generation of expert system shells is to design and build a shell which is applicable to a wide range of domains. This approach has resulted in a number of expert system shells which have been successfully applied to industrial domains, notably OPS5. A considerable reduction in development time is possible by applying such a tool which has been well debugged and applied to several industrial domains. A review of some of the commercially available expert system shells follows, these tools are described more fully in [87].

ES/P ADVISOR is a tool tailored to facilitate "text animation": an application in which expertise that is already recorded as instructions, regulations or procedures can easily be converted into a small expert system. This has been used to create small advisory systems in domains such as tax and benefit regulations, building regulations, safety standards and office procedures. The designer of an ADVISOR system begins with a textural regulation and separates the actual advice from the conditions that define when the advice is applicable. Rather than entering specific rules, the designer enters attribute-value pairs together with commands that tell the system how to relate them and how to handle the information. By itself, ES/P ADVISOR is a very narrow tool for developing small programs, but by interfacing the system with a PROLOG environment a more powerful expert system building tool may result.

Expert-Ease is a small system building tool available for personal computers. The tool is termed a "decision-making spreadsheet": it is quite good at developing a decision tree that will sort through a number of possible descriptions to determine which one the user actually faces. In effect, the knowledge forms just one rule; this imposes a restriction on the types of problem to which the program may be applied.

The **M.1** system is a consultation type system similar to EMYCIN. Knowledge is represented by attribute-value pairs with accompanying confidence levels. Heuristic knowledge is represented by production rules. A feature of M.1 is its support of "variable" rules; this technique makes it easy to enter a lot of similar rules by writing one generic rule together with "look-up table" type information. Like EMYCIN, M.1 is a back-chaining system, but "when-found" rules allow some limited forward chaining.

Knowledge Engineering System (KES) is a family of products that support the development of consultation-style knowledge systems. Facts are represented as attribute-value pairs with associated confidence factors. Attributes and values may be arranged in hierarchies. Relations among facts are represented by production rules, with a choice of two inference techniques:

1. Statistical pattern classification using a Bayesian prior probability approach to assessing the likelihood that a fact is true;
2. The hypothesis test cycle, which is a strategy for cycling through alternative hypotheses about a problem until one is determined with sufficient certainty.

KES allows goals to be defined; the inference engine back-chains through rules in order to obtain a value for the goal.

S.1 is a tool that is designed to build knowledge systems that solve diagnosis/prescription problems that can be organized and classified. A number of commercial systems have been produced using S.1 by its developers, Teknowledge. Facts are stored as object-attribute-value triplets with associated confidence factors. Objects may be classified, with all objects of a certain class inheriting a set of common attributes. Attributes may be arranged hierarchically when one attribute of a class is a precondition of another. Similarly, hierarchical relationships among variables may be declared. Production rules are used to represent other relationships between objects. A powerful feature of S.1 is that it allows existential and universal quantifiers to be included in the premises of

rules to test whether any or all of a set of objects have the specified properties. Inference proceeds by back-chaining, facilitated by **control blocks.** Control blocks are explicit procedural statements which regulate when classes are instantiated, how the values of attributes are sought, whether default values are assigned and how results are displayed. Control blocks may contain conditional statements which enable lines of reasoning to be pursued or avoided, depending on intermediate results. S.1 is unusual among rule based systems in that procedural control aspects are explicitly defined; most such systems rely on ordering of the rules in the rule base or on specifically written rules to declare goals.

The Intelligent Machine Model (TIMM) is a knowledge system building tool designed to be used by problem domain experts rather than knowledge engineers. TIMM functions somewhat like Expert-Ease in that it accepts a set of examples which form a matrix. Each matrix is, in effect, a rule. The rule base is created as the expert enters examples. Unlike Expert-Ease, however, TIMM allows several rules to be created and for these to be linked together to form a hierarchical structure. Knowledge is represented in TIMM as attribute-value pairs, a certainty factor associated with facts, and a reliability factor associated with each set of examples and their conclusions. Knowledge is obtained from the expert in the following fashion:

1. TIMM interrogates the expert about what attributes are important with respect to a particular domain;
2. TIMM requests the outcome of an example in the problem domain. It then probes to see what values are associated with the attributes for that case;
3. TIMM generalizes rules and optimizes a decision tree based on the examples entered by the expert.

TIMM is a useful tool for the extraction of knowledge, but is not really suited to building large, complex systems.

OPS5, unlike the programs described above, is more a programming language than a tool, but is most commonly used as a forward chaining production system. System development is considerably easier using OPS5

than using LISP, but OPS5 retains considerable flexibility to be applied to a wide range of problem domains. OPS5 has been used for several major system development efforts, including R1.

Objects with attributes and values are used to represent facts in OPS5. Rules are represented as if-then productions. OPS5, in common with most programming languages, does not have narrow, highly refined and specific constructs. Rather, it has a few generic constructs which may be applied in a variety of ways. For example, no inherent confidence factors exist in the OPS5 language, but an attribute to achieve this may be defined.

The inference engine used by OPS5 is also very simple. A "recognize-act" cycle is used to implement a forward chaining mechanism. The condition elements of rules are compared with the elements in working memory until a rule executes and working memory is modified according to the actions of the rule. The conflict resolution strategy employed encourages depth-first search: rules matched by the most recently produced working memory elements are satisfied preferentially. Those rules with more complex condition elements will also execute in preference to those with more general condition elements, and no rule will execute more than once with the same satisfying working memory elements. From these simple conflict resolution strategies it is possible to implement more complex control strategies such as goal-directed reasoning, rule subset partitioning and the use of demons. A more detailed account of the OPS5 language appears in Appendix A.

4.4.3: Large Hybrid System Building Tools

There are a number of large, hybrid system tools, such as KEE and LOOPS. These programs are generally very expensive, will only run on large workstations and are often difficult to implement and debug.

Hybrid systems use frames to unify procedural and declarative knowledge expressions. A choice of backward or forward chaining is often available. Hybrid systems generally make use of graphics to help explain the representation, reasoning and behaviour of the knowledge system.

Graphical models of physical objects, meters and gauges may be constructed. Gauges and meters may be used to show the status of active values. Active values are the values of certain key slots which may be monitored as the system runs. These active values may be modified by the user to investigate a range of different situations within a problem.

4.4.4: PROLOG

PROLOG [88] is a production rule language in which programs are written as rules for proving relations between objects. PROLOG implements a simplified version of predicate calculus and is considered a true logical language.

Programming in PROLOG takes the following form:

1. Some facts about objects and relationships are specified;
2. Rules are written governing objects and their relationships;
3. Queries are made to the system about objects or relationships.

In a sense, PROLOG computation is simple logical deduction. Facts are stated, and PROLOG will tell the user of any logical deductions that it can make from these facts. PROLOG is the best current example of logic programming, but it cannot begin to handle all the deductions that are theoretically possible in predicate calculus. In order for PROLOG to be useful for developing large expert systems, additional methods for implementing complex control strategies must be made available. As it stands, PROLOG is of limited value only in the field of expert system development.

4.5: The Applicability of Expert Systems

Most of the applications of expert systems developed to date have been in areas of consultation or diagnosis. Expert systems are good in this sort of domain where there is a large amount of sound "expert" knowledge available. A requirement for the use of expert system techniques in a particular domain is that domain knowledge is tractable, and is capable of being represented by "if...then" statements. Methods

have been developed to deal with uncertain knowledge; this has opened up many applications to possible representation by rule bases in expert systems.

Until recent years, much A.I. programming was done using LISP. It is, however, a time consuming process to construct expert systems using LISP; expert system "shells" provide a much easier route to expert system production. Many such shells are rather limited in their applicability, but others, such as OPS5 and the more complex hybrid systems, are very flexible in nature and may be applied to a wide range of problem domains. PROLOG is an application of logic which may lead to a "fifth generation" of computer languages. Problem solution strategies have been developed to improve PROLOG's applicability to real-world problems, but as yet such strategies are relatively poor.

Expert systems may well be a tool of the future, although they may be superseded by newer A.I. concepts, such as neural networks. The use of expert systems in industrial applications is on the increase, due to increased awareness of their capabilities and the improved development environments which are now available. Expert systems are now widely recognized as a powerful technique for the solution of difficult problems in many domains which involve large amounts of knowledge. The days of using expert systems merely as large databases to perform consultation or diagnosis may now be over.

4.6: Choosing A Computer Tool for The Development of a Fault Tree Synthesis System

The generation of fault trees for process plant by a method which uses models of plant units is, because of the amount of "expert" knowledge involved, an ideal application for expert systems. Fault trees are generated by a search process, in this case a complete search with no need for heuristic knowledge to guide it. If, however, fault trees were to be generated in real time for fault diagnosis or alarm analysis applications, some "most likely cause" heuristic might be useful.

Another stimulus for using expert system techniques in a

computer-based fault tree synthesis program is to attempt to improve the readability of fault trees by modelling the way in which a human expert might create fault trees. It may be possible to embody the informal rules that experts use when structuring and simplifying a problem in such a program using "if...then" rules in order to reduce the size and improve the structure of the fault trees produced. Methods of embodying such knowledge are described in chapter 6.

The main requirement for expert knowledge is in the creation of input-output models and failure modes of process units: this is the sort of knowledge which is reasonably well defined, and thus could probably be obtained from an expert with relative ease. A second level of expert knowledge which could be used is that of more general plant failure modes: a flow measurement device may record wrongly, for example, if there is entrained spray with a vapour flow. This type of knowledge is rather more difficult to come by, but would improve the performance of a fault tree synthesis program considerably.

Having decided to implement the fault tree synthesis procedures described in chapters 6 to 8 as an expert system, another question arises: what programming language or development tool should be used? There are several factors that influence this decision:

1. Is the method of knowledge representation and the synthesis algorithm known beforehand? If so, them a programming language such as LISP might be the best approach: time can be spent perfecting knowledge representation and control and inference procedures before the system is developed. On the other hand, if experimentation is required to arrive at the most successful synthesis method, a simple tool is probably the best approach: inference and representation capabilities are fixed, so time is spent on prototyping rules, running examples and modifying the system;
2. Is a forward chaining or a backward chaining method required? Many tools support only backward chaining, only a few support forward chaining, very few support both;
3. Are there extensive procedural sequences required? Frame-based systems support procedural sequences better than semantic net

derivatives, such as object-attribute-value triplets;

The exact methods of knowledge representation and fault tree synthesis are rarely known beforehand when researching a method: considerable modification and rebuilding is normally required. It is therefore preferable to use a tool rather than a programming language.

Although fault tree synthesis is a backward chaining technique, the methods of automating the synthesis process described in this thesis are better suited to a forward chaining approach. This is due to the use of operators to model feedback control loops which are more easily applied in a forward chaining mechanism. In addition, internal consistency checks are easier with a forward chaining approach.

It is possible, indeed preferable, to write a fault tree synthesis method which does not require many procedural elements. A semantic net type system is thus selected in preference to a frame-based system.

As a forward-chaining semantic net derivative system, OPS5 meets all of these requirements. OPS5 falls somewhere between being a shell and a programming languages, offering many of the benefits of both: it is easier to use for system building than LISP or PROLOG, but more flexible than most shells. The inference strategies employed by OPS5 are sufficiently advanced to allow large systems to be constructed. For these reasons, OPS5 is used to implement all of the programs presented in this thesis.

CHAPTER FIVE

5. A PLANT MODEL: THE BASIC AMMONIA LET DOWN SYSTEM

5.1: Introduction

This chapter introduces a problem which will be used extensively in the ensuing chapters to illustrate the methods for fault tree synthesis described therein. The basic plant, including a simple control strategy, is described in this chapter. A more complex control strategy is also modelled, this is described in chapter 10.

5.2: The Basic Ammonia Let Down Problem

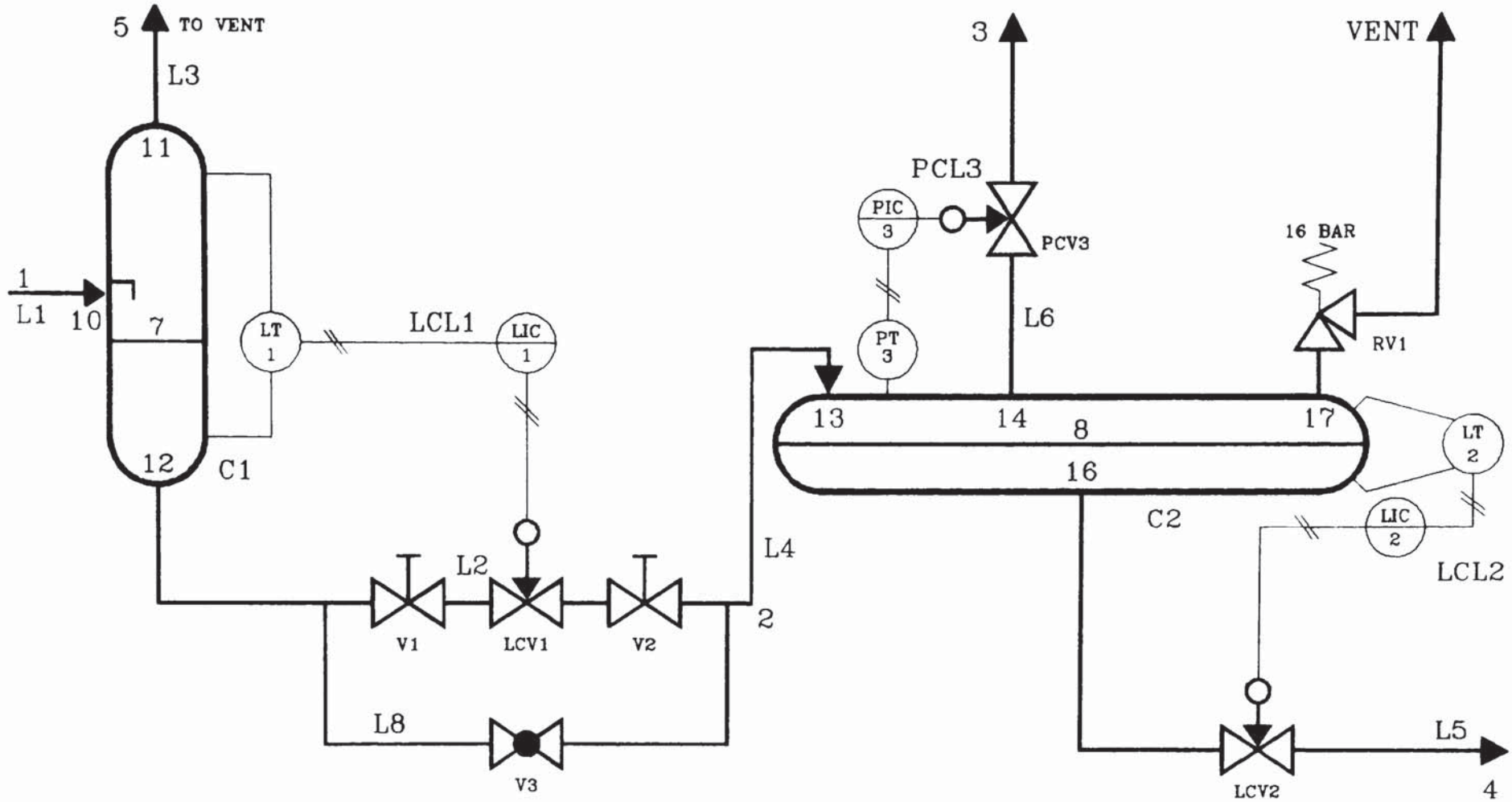
The basic ammonia let down problem, as shown by the flowsheet in Fig. 5.1, is described by Lihou [48,89]. The purpose of the plant is to separate a mixture of liquid ammonia and synthesis gas. There are two stages to this process:

1. Separation of a two-phase mixture of cooled liquid ammonia and synthesis gas;
2. Flashing to allow synthesis gas to desorb from the ammonia liquid product from stage 1.

The two-phase mixture of cooled liquid ammonia and synthesis gas, containing some ammonia gas, enters the plant area at node 1. The mixture is at -1°C and 148 bar gauge. The liquid flowrate is 36400 Kg/h with 0.8% dissolved synthesis gas. The gas flowrate is 27600 Kg/h with 3% ammonia gas and 97% synthesis gas, with traces of methane.

A vertical cylindrical vessel, C1, separates the two-phase flow from line L1. The synthesis gas, plus some ammonia gas and the methane, is discharged via line L3. Liquid ammonia containing dissolved synthesis gas leaves via line L2. The level in C1 is controlled at about 40% by the control loop LCL1. C1 is designed for 160 bar gauge at -6.5°C. The pressure drops from 148 bar gauge to 14 bar gauge across the level control valve, LCV1, the capacity of which is 60000 Kg/h.

Fig. 5.1: Flowsheet of a Basic Ammonia Let Down Plant



Line L2/L4 is a 50mm diameter line which transfers 36400 Kg/h of liquid ammonia at -1°C, containing 0.8 mole % synthesis gas in solution, to vessel C2. The flowrate in L2 is adjusted by the controller LIC1 to maintain a liquid seal in C1. This line is split into two for the purposes of this analysis. This allows the siting of a node, node 2, in the line.

Line L3 returns 27600 Kg/h of synthesis gas, containing 3% ammonia gas, to the synthesis gas recirculator.

The horizontal cylindrical vessel, C2, allows synthesis gas to desorb from the ammonia liquid entering via line L4. Flashing occurs in C2 at an operating pressure of 14 bar gauge and temperature of -5°C. The pressure is controlled by the loop PCL3. The pressure control valve, PCV3, has a capacity of 750 Kg/h. The liquid level in C2 is maintained by the loop LCL2; the level control valve, LCV2, has a capacity of 45000 Kg/h. C2 is designed for 16 bar gauge and -6.5°C. A relief valve, RV1, with a 25mm inlet pipe and a 50mm tail pipe leading to the vent stack, is set at 16 bar gauge.

Line L5 is a 2.5 inch line which transfers degassed ammonia liquid, containing 0.1% dissolved synthesis gas, to storage at a flowrate of 36118 Kg/h.

Line L6 transports the flashed gas, containing 27% ammonia, from C2 to a fuel system at a rate of 282 Kg/h.

5.3: Representation of the Basic Ammonia Let Down Problem

In order to represent the ammonia let down problem for analysis by the program described in chapter 9, the following are defined: major process units C1, C2; minor process units V1, V2, V3; control loops LCL1, LCL2, PCL3; RV1, a pressure relief system; control units LT1, LIC1, LCV1, LT2, LIC2, LCV2, PT3, PIC3, PCV3, RV1; lines L1, L2, L3, L4, L5, L6; the bypass line L8.

Nodes are defined at each plant boundary, internally to each unit, and in line L2/L4, which is divided into two sections: line 2, an output line

from C1; line L4, an input line to C2.

Input-output and failure mode equations, based on those used by Lihou, for the gas-liquid separation and flash vessels are presented as general models for these types of vessels with the configurations shown in Figs. C.1 and C.3. These models are presented in general terms so that they may be applied to any vessels with the same configurations, except that they are defined in terms of components specific to the ammonia let down problem. More widely applicable models would use generalized component types, this is discussed in section 11.2.1. The equations represented by these models are shown in Tables C.1 and C.2.

Two hazardous events of interest are identified by Lihou:

1. Overpressure of C2;
2. Gas breakthrough into liquid ammonia storage.

These are used as top events for fault tree development.

CHAPTER SIX

6. A METHOD OF SIMPLIFYING THE REPRESENTATION OF CHEMICAL PROCESS PLANTS

6.1: Introduction

The requirements for a useful computer-based fault tree synthesis method applicable to process plants were discussed in section 3.4; in summary these are:

1. The fault trees produced should not be unmanageably large, and they should possess some identifiable structure;
2. The data required for input to the system should be easily obtainable and of a relatively simple nature;
3. The method should not produce spurious fault tree events;
4. The method should be capable of reliably producing all possible causal event chains.
5. The method should be able to cope with all types of plant configurations and control systems.

It was stated in section 3.4 that existing fault tree synthesis methods have met one or two of these requirements, but no method has yet been developed which even comes close to satisfying them all. It has been found during this research that compromises must be made in order to meet these requirements in combination. For example, methods that are able to cope with a wide range of problems tend to require complex data for input and to produce large fault trees.

The approach taken in this research has been to develop a hybrid method which attempts to meet all of the above requirements. Two techniques have been used: one of problem reduction, this being a large step toward producing smaller, more structured fault trees; the other of problem representation which allows the system to handle a full range of chemical plant models including complex, non-standard control strategies. These two techniques have been integrated in a fault tree generation package, Ordered Fault Tree Synthesis (OFTS). A feature of the OFTS

package is that it prompts the user to answer questions about the ability of control and trip systems to handle certain failure event chains. Such a step toward a less automatic, more interactive, approach is considered by the author to be necessary if a widely applicable fault tree synthesis method is to be developed. It is otherwise necessary to make assumptions about the controllability or otherwise of deviations. Such assumptions limit the applicability of fully automatic methods.

The OFTS package has been implemented as an expert system, as discussed in section 4.6. The techniques described in this thesis use the capabilities of expert systems to model human representational processes in order to produce relatively small, structured fault trees. This is achieved by representing knowledge which is applied intuitively by human experts as "operators" which act on more complex input-output models representing the major plant units. These operators act to increase the coverage of unit models such as to reduce and structure the problem.

This research has produced methods which are generally applicable to chemical process plant, but the application of which requires detailed modelling of unit operations. The applicability of these methods is demonstrated by an example, with the models required to solve this particular problem. The example used is the basic ammonia let down plant, as described in chapter 5, and variations on this involving more complex control systems. No attempt has been made to produce a wider range of unit models, nor are cascade or manual control strategies modelled in the current implementation of the methods. Methods for modelling these strategies are discussed in chapter 11.

6.2: Methods of Representing Chemical Process Plant

The traditional approach to process plant representation for the purpose of automatic fault tree synthesis is to produce a connection diagram between all plant units. "Nodes", at which variables are defined, are created at the input and output of all plant units, with additional nodes internal to some units. From this diagram, and a set of operational and failure modes for each of the units, an intermediate fault connection stage is produced from which fault trees may be generated. The form of

this intermediate stage represents the major difference between these methods; it has taken the form of failure transfer functions [28], information flow models [3], decision tables [37,42], digraphs [43] and mini-fault trees [36].

Shafaghi [47] developed a different approach: to model the plant by control loop structure rather than by unit models and connections. This method produces more structured fault trees because of the structural basis of the modelling. A second benefit of this method is that the fault trees generated are smaller than those produced by the other methods mentioned above. This fault tree reduction is due to a decrease in the number of nodes the method uses to represent the plant: the only nodes used in Shafaghi's method are those which represent each control loop. Unfortunately Shafaghi's method does not consider those deviations arising outside of control loops, nor does it cater for complex control strategies such as that described in section 7.1. A further problem with Shafaghi's approach is that a considerable amount of data preparation is required prior to fault tree generation.

There are merits to both of these types of approach. A unit modelling method allows standard libraries of input-output equations and failure modes for chemical processing units to be used. Such a library of information can be utilised in an expert system to provide much of the data required for fault tree construction. Shafaghi's structural approach loses this convenience, but it does provide improved results, due both to problem structuring and reduction in terms of the number of nodes.

The problem reduction method described in the next section attempts to combine these two approaches so as to retain their inherent advantages, but to minimise their disadvantages.

6.3: The Functional Approach to Problem Reduction

The functional approach to problem reduction is a method of reducing the number of nodes the system uses to represent the plant, whilst accessing a library of input-output equations for process units as the major source of data.

The functional basis of the method is:

1. To expand the coverage of models of major process units to include their ancillary items, such as pipework, valves and pumps. This is known as **model expansion.**
2. To model control loops, trip loops, operator actions, interlocks and pressure relief systems as individual entities. These are known as **loop operators.**

The starting point for model expansion is a standard model of a process unit, consisting of input-output and failure mode equations. The nodes at which input to or output from the unit occur are created only as temporary nodes; no deviations at these nodes are explicitly defined. Line operators are used to convert input-output equations, originally defined in terms of these temporary nodes, into equations defined in terms of a larger plant section. Additional failure modes, those corresponding to failures in lines or ancillary plant items, are included in the expanded model. The effect of model expansion is to reduce the number of nodes used to represent the plant state; this produces a corresponding decrease in the size of fault trees generated from expanded models.

Loop operators represent the actions and failure modes of control loops and protective systems. Loop operators are defined in terms of their function: to protect against the deviation of a certain variable by manipulating a controlling variable. One or more measurements may act as inputs to a loop. The modes of failure for each loop are created in terms of the units which make up that loop. The effects of loop failure modes are represented by operators which produce AND gates with loop STUCK or FAIL-DANGER as inputs to deviations of controlled variables, as well as creating additional inputs to events, corresponding to the loop failure modes FAILED-HI, FAILED-LO or FAIL-SAFE. The actions of loops on their controlling variables, both in terms of loop failure modes and correct loop responses to measured variable deviations, are also represented by operators. The reason for using loop operators to represent control loops and protective systems is to explicitly represent their functionality in order to produce accurate, well structured fault trees. An additional

benefit of loop operators is a further reduction in the number of nodes the system uses to represent the plant, allied with a corresponding reduction in fault tree size.

The effect of these approaches is to reduce, simplify and structure the problem. This is illustrated by an example. The functional unit representing the process units and lines of the gas-liquid separation area of the ammonia let down problem described in chapter 5 requires only 4 nodes, as shown in Fig. 6.1. A typical process unit modelling method would require 18 nodes to represent the same plant, as shown by Fig. 6.2. The mini-fault tree produced by using the functional method for the event 2 FLOW LOW is shown in Fig. 6.3. Application of a standard process unit modelling approach similar to that described by Martin-Solis et al [36] produces fault trees containing long chains of events such as that shown in Fig. 6.4. This fault tree, for the event 18 FLOW LOW, is equivalent to the fault tree for 2 FLOW LOW shown in Fig. 6.3. The aim of model expansion is to eliminate these long failure chains by reducing the number of nodes the method uses to model the plant. This results is much simpler fault trees, such as that of Fig. 6.3.

6.4: Rules for the Representation of Chemical Process Plant Using the Functional Approach

Plant definition requires that the plant topology, operating states, and components be described. Major, minor and control units, and the lines which connect them must be defined, along with node names and the components that exist at each node.

During plant definition, certain rules about the siting of nodes, and the definition of certain units as major or minor units should be observed. These rules are mostly not completely binding. This gives the user some flexibility when creating a representation of the plant. It should be pointed out, however, that in most situations where a choice of representations is possible, the fault trees produced will be accurate whichever representation is used, although they may differ slightly in size and structure.

Fig. 6.1: The Gas-liquid Separation Area as Represented by Functional Modelling

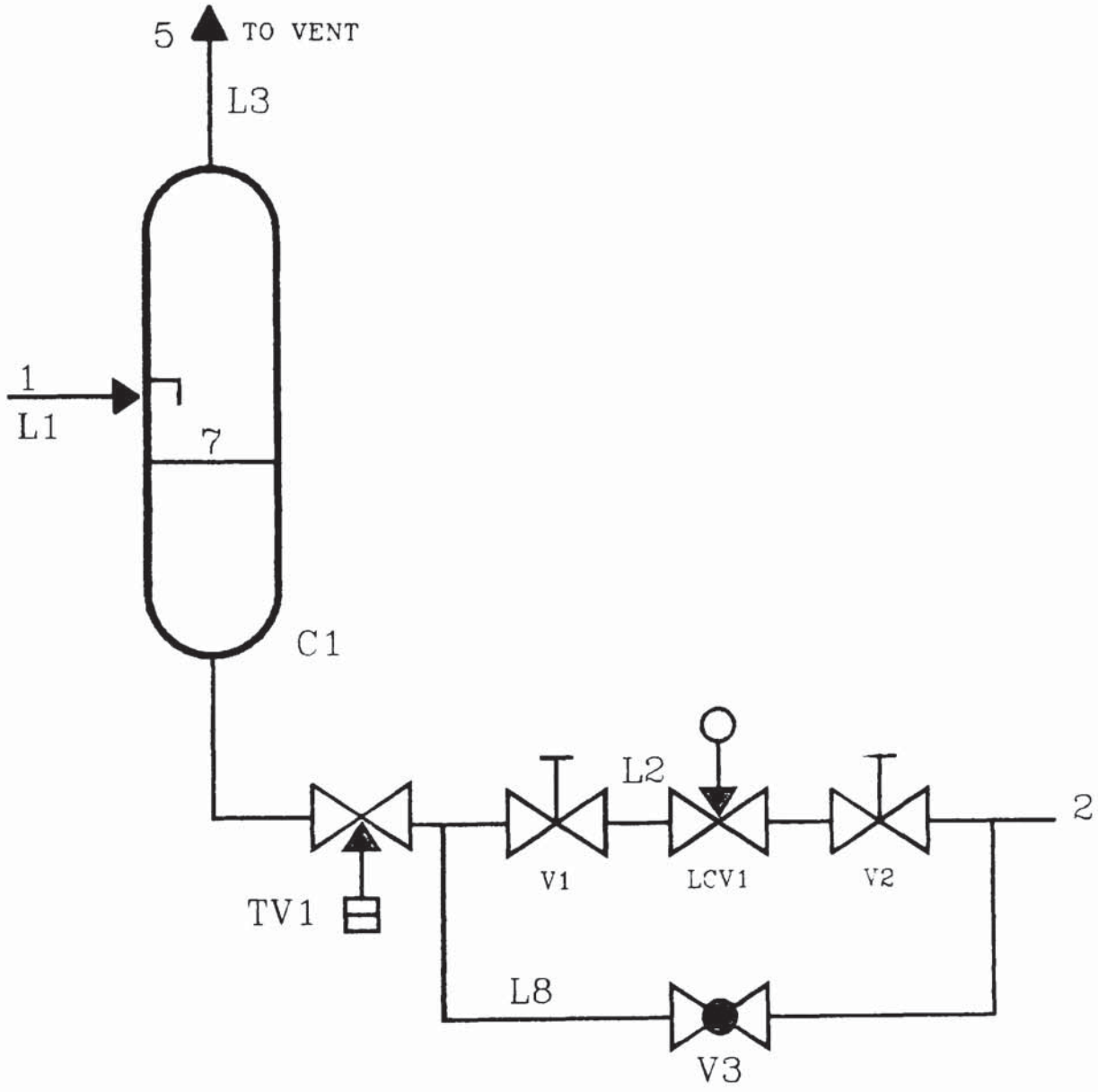


Fig. 6.2: The Gas-liquid Separation Area as Represented by Process Unit Modelling

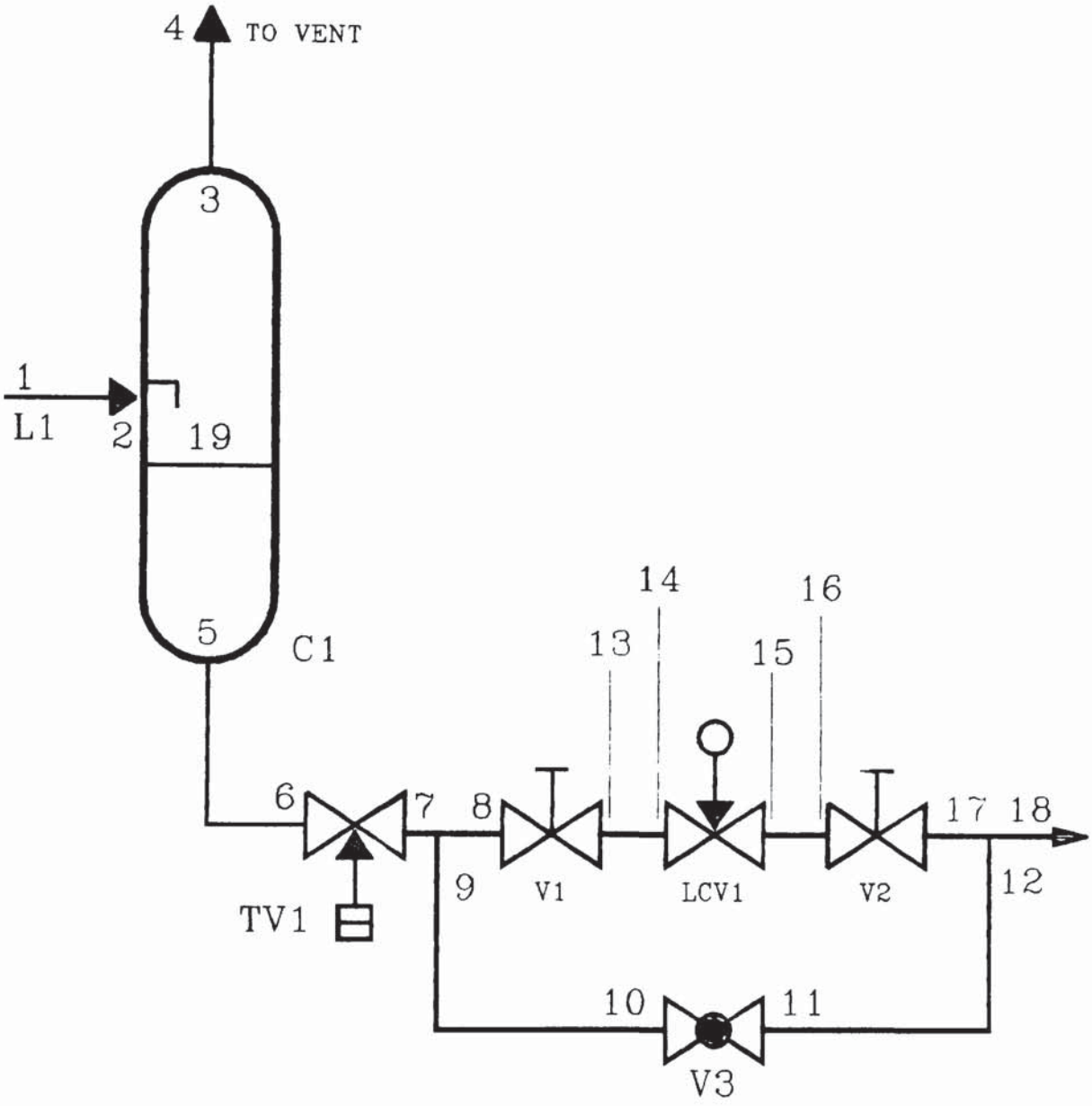


Fig. 6.3: A Mini-fault Tree for the Gas-liquid Separation Area Using Functional Modelling

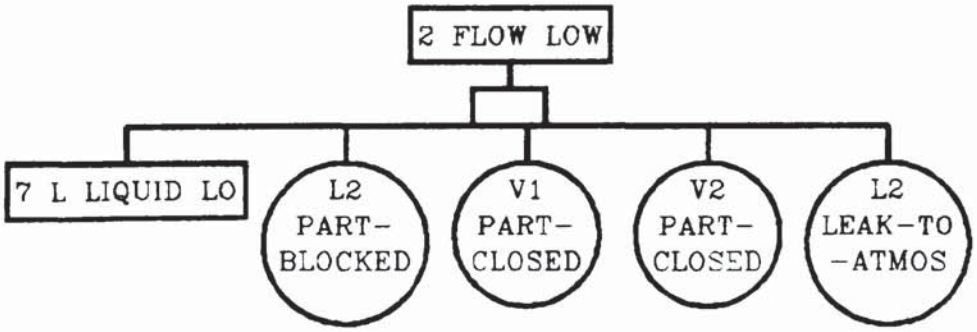
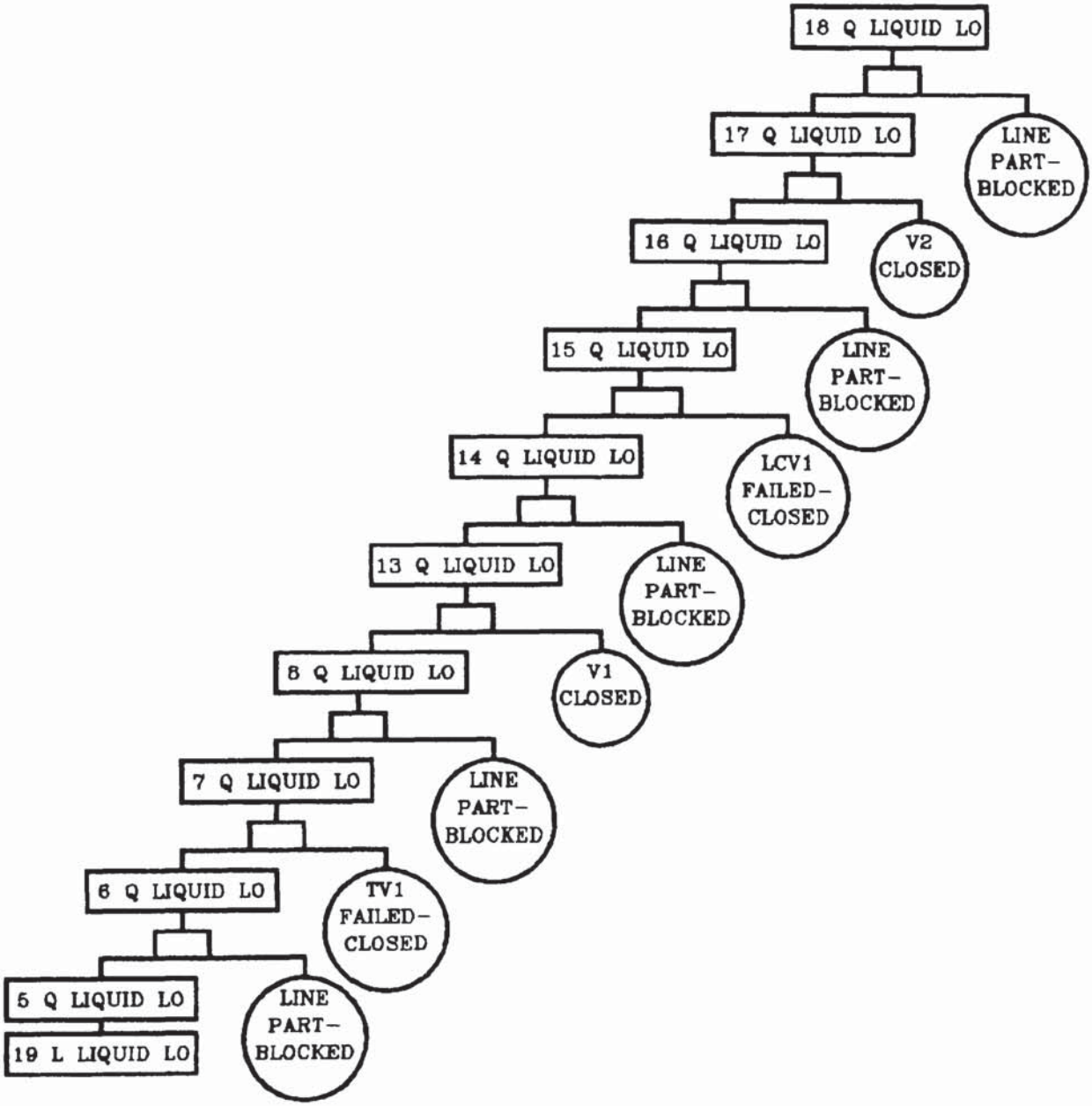


Fig. 6.4: A Mini-fault Tree for the Gas-liquid Separation Area Using Process Unit Modelling



The Functional approach sub-divides plant units into three classes: major process units, minor process units and control units. **Major process units** are defined as those units which have at least one relationship between their input and output variables which is not directly dependent. A directly dependent relationship occurs where there is an input-output equation of the form:

output deviation = input same variable, phase, component, and deviation;
or,
input deviation = output same variable, phase, component, and deviation.

That is, the deviation is simply transmitted unchanged through the unit. If one or more deviations are changed, or produce other deviations, when passing through a unit, then this unit should normally be defined as a major process unit.

Often, major process units require the definition of at least one internal node. Internal nodes for a major process unit should be defined if:

1. Variable deviations at internal nodes are necessary to create the correct failure logic for the unit; or
2. Variable deviations at internal nodes are helpful for showing the failure structure logic for the unit; or
3. A variable inside the unit is measured or controlled by a control loop or protective system.

If an internal node is required for a unit, then that unit must be defined as a major process unit.

Minor process units are those units, excluding final control elements, in the process stream all of whose input-output equations are directly dependent. In fact, what this means is that a minor process unit may be modelled by the same input-output equations as a line, although different failure mode equations may be used. Internal nodes cannot be defined for minor process units.

Control units are those units which form part of a control or trip loop, interlock, alarm equipment or pressure relief system, including final control elements, but excluding any other units in the process stream between the measured, controlled and controlling variables. Examples of control units are transducers, switches, controllers, alarms, solenoid valves, control valves and trip valves.

Consider the ammonia let down plant in Fig. 5.1. The major units are C1, a gas-liquid separator, and C2, a flash vessel. The control units are the transducers, switches, controllers, control valves, trip valves and the relief valve. The minor process units are V1, V2, and V3.

Pipework is grouped to form lines of three types:

1. Input lines to a major unit;
2. Output lines from a major unit;
3. Bypass lines.

Other line types are not modelled in the current program implementation. The failure and operating modes of the modelled line types are described by low-level operators, known as **line operators.** Line operators allow the failure modes of minor process units in a line to be incorporated, without there being any requirement to define more than two nodes, one at each end of the line.

Each line is also defined as "open" or "closed". An open line is one in which there are no closed valves, including trip valves but not control valves, preventing flow; a closed line contains at least one closed valve. The status of lines and valves depends on the operating condition under investigation; if alternative operating conditions require investigation then these must be defined as a separate problem. In the basic ammonia let down problem the following lines are defined:

L1 (open input line);
L2 (open output line);
L3 (open output line);
L4 (open input line);

L5 (open output line);
L6 (open output line);
L8 (closed bypass line).

In terms of blockage or leakage failure modes, minor units in a line are considered to be part of that line. Thus the failure mode "blocked valve" is included in the general line failure mode "blocked line" where the valve in question is part of the line. In terms of all failure modes other than blockage or leakage, however, minor units are separate from the lines they form part of.

The rules for node placement are as follows, although the exact composition of lines and placement of nodes is left to the discretion of the user:

1. For each major unit, as many internal nodes may be defined as are necessary to fully describe the relationship between input and output variables;
2. For each major unit at each line entering or leaving the unit a temporary node must be defined;
3. In each section of pipework between two major units a node must be defined. This node actually splits the pipework into two lines, an input line and an output line;
4. At each side of a split or junction of pipework a node must be defined, unless the splitting or joining line is a closed bypass line, or any other line type which is described by a high-level operator.

In the ammonia let down example, the following nodes are defined: 1, 5, 7, 2, 8, 3, 6, 4. The following are defined only as temporary nodes: 10, 11, 12, 13, 14, 15, 16.

In the ammonia let down example, the units V1, LCV1 and V2 are included as part of L2 rather than L4. Ultimately, the choice of where to divide L2 and L4 makes little difference to the fault tree and no difference at all in terms of failure probability calculations, but it is natural to include the minor units here as part of the C1 output line, and thus part of the C1 functional area, as they form part of the C1 level control loop

LCL1. If, however, the node is placed between V1 and V2, and no other nodes defined between C1 and C2, the representation would fail: the event L8 SOME flow would not produce high flow at this node and so the failure event chain linking high flow in L2 and L4 due to V3 being open would be missed.

Generally, minor process units should be grouped in lines in such a way as to most accurately represent their functionality. The above case of node placement between V1 and V2 is a situation where the representation will fail. This is an example of a specific node placement rule; other such rules may be identified as further plant models are analysed.

For convenience, pressure and level deviations are defined only at internal nodes. Thus a flow deviation is not directly related to pressure gradient deviations in a line, but the result is the same because line flow deviations are related to pressure and level deviations where the line meets a major unit.

All control loops and protective systems must be defined as loop operators, their measured, controlled and controlling variables defined, along with the units that comprise the loop. In addition, the operating modes of the loop should be defined. Operating modes are defined in terms of the measured, controlled and controlling actions of the loop. The **measured action** of a loop describes the relationship between the measured and controlled variables. The measured action is defined as positive if an increase in the measured variable tends to cause an increase in the controlled variable, otherwise it is negative. If the measured variable is the same as the the controlled variable, the measured action is, by default, positive. The **controlled action** of a loop describes the effect of the controlling variable on the controlled variable. The controlled action is positive if an increase of the controlling variable tends to increase the controlled variable. The **controlling action** of a loop describes the action of the loop on the controlling variable. The controlling action is defined as positive if the loop acts to increase the controlling variable when an increase in the measured variable occurs.

6.5: Expansion of Process Unit Models

The process unit models required are input-output equations, failure modes and their consequences for major process units only. The input-output equations used are similar to Lihou's cause and symptom equations [48], but are rewritten so that all equations take the form:

deviatory event = cause (* complementary cause...) + ...

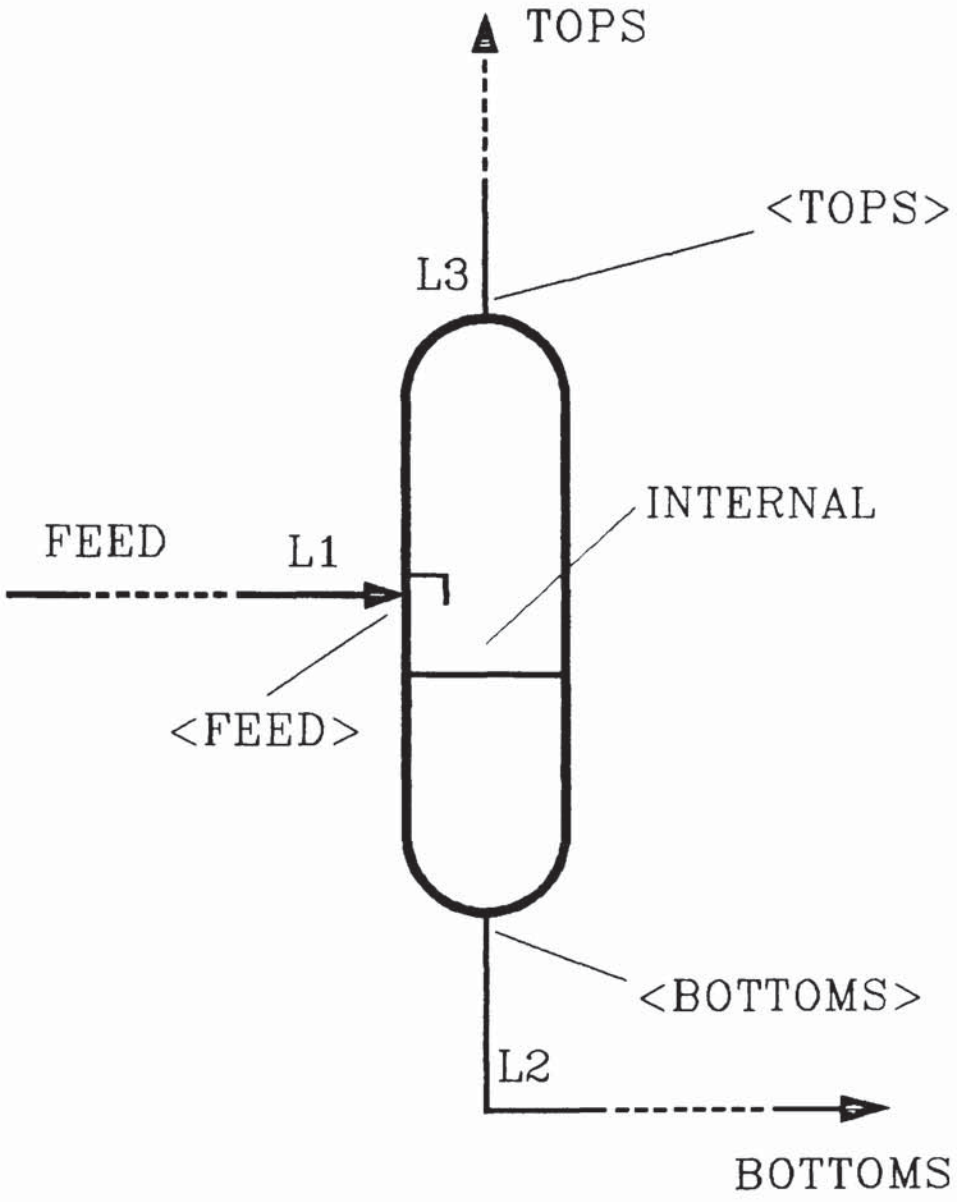
where the deviatory event is a deviation from the normal state of any variable at any node in the unit; cause is any other deviatory or failure mode event of the unit which, in conjunction with any complementary causes, may cause the left-hand-side. Complementary causes are joined by an AND gate, here shown as "*"; "+" represents an OR gate. The input-output equations and failure modes for gas-liquid separators and flash vessels used to solve the ammonia let down problem are listed in Figs. C.2 and C.4.

The problem reduction approach involves the replacement of temporary nodes, defined as the unit nodes which join input and output lines, with the nodes at the other end of the adjoining lines. The input-output equations for the gas-liquid separation unit in Fig. 6.5, for example, are originally written for the nodes <FEED>, INTERNAL, <TOPS> and <BOTTOMS>. These are then converted to equations involving the nodes FEED, INTERNAL, TOPS and BOTTOMS, with additional line and minor unit failure mode causes being added. Nodes on both the left and right hand sides of input-output and failure mode equations may be replaced in this way. Process unit models produced by this node reduction are called **expanded process unit models.**

6.5.1: Line Operators

The operators that generate expanded process unit models are called the **left hand side node replacement operator,** shown in Fig. 6.6, and the **right hand side node replacement operator,** shown in Fig. 6.7. Additional causes where left hand side node replacement occurs are listed in Table 6.1, these are added as shown in Fig. 6.8. Table 6.2 shows the additional causes where the right hand side is replaced, these are added as shown in Fig. 6.7. These tables only include line failure modes, minor unit failure

Fig. 6.5: Nodes Defined for the Gas-liquid Separation Area Prior to Problem Reduction



NODE – functional node

<NODE> – temporary node

Fig. 6.6: The Left Hand Side Node Replacement Operator

<Temporary node> event = causes

becomes:

<Node connected by a line to temporary node> event = causes
+ additional line failure modes (as Table 6.1)

where the event is the same deviation, but at a different node, and the causes remain unchanged

Table 6.1: Additional Line Failure Modes When Using Left Hand Side Node Replacement Operator

LHS (Output)	RHS Additional Line Failure Mode Cause
OUT Q NO	BLOCKED + LEAK-TO-ATMOS
OUT Q LO	PART-BLOCKED + LEAK-TO-ATMOS
OUT Q HI	LEAK-FROM-ATMOS
OUT Q REV	LEAK-TO-ATMOS
OUT T HI	EXT-FIRE
IN Q NO	BLOCKED + LEAK-FROM-ATMOS
IN Q LO	PART-BLOCKED + LEAK-FROM-ATMOS
IN Q HI	LEAK-TO-ATMOS
IN Q REV	LEAK-FROM-ATMOS

"IN" refers to input lines to the major process unit;
"OUT" refers to output lines from the major process unit.

Only one of LEAK-TO-ATMOS and LEAK-FROM-ATMOS may be defined for any one line.

Fig. 6.7: The Right Hand Side Node Replacement Operator

consequence = <Temporary node> event
becomes:
consequence = <Node connected by a line to temporary node> event
+ additional line failure modes (as Table 6.2)

where the event is same, but is defined at a different node; the consequence event remains unchanged

Table 6.2: Additional Line Failure Modes When Using Right Hand Side Node Replacement Operator

RHS (Input)	RHS Additional Line Failure Mode Cause
OUT Q NO	BLOCKED
OUT Q LO	PART-BLOCKED + LEAK-FROM-ATMOS
OUT Q HI	LEAK-TO-ATMOS
OUT Q REV	LEAK-FROM-ATMOS
IN Q NO	BLOCKED + LEAK-TO-ATMOS
IN Q LO	PART-BLOCKED + LEAK-TO-ATMOS
IN Q HI	LEAK-FROM-ATMOS
IN Q REV	LEAK-TO-ATMOS

"IN" refers to input lines to the major process unit;
"OUT" refers to output lines from the major process unit.

Only one of LEAK-TO-ATMOS and LEAK-FROM-ATMOS may be defined for any one line.

modes are added separately. If both sides of an equation are defined in terms of temporary nodes, then both of these operators should be applied.

For an example of how the left hand side node replacement operator works consider equation 6.1, which is the initial input-output equation for the deviation "low flow of liquid at node <BOTTOMS>" for the gas-liquid separator process unit of Fig. 6.5:

<BOTTOMS> Q LIQUID LO = INTERNAL L LIQUID LO ...(6.1)

The left hand side node replacement operator replaces the temporary node <BOTTOMS> with the node BOTTOMS at the other end of line L2, adding line failure modes. Equation 6.2 is the result:

BOTTOMS Q LIQUID LO = INTERNAL L LIQUID LO + L2 PART-BLOCKED
+ L2 LEAK-TO-ATMOS ...(6.2)

Line L2 is an output line, so the line failure modes added are those of line 2 from Table 6.1. "L2 LEAK-TO-ATMOS" is only added if the line pressure is greater than one atmosphere.

For an example of the right hand side node replacement operator's action, consider equation 6.3 which gives the causes of "low liquid level at the INTERNAL node" of the gas liquid separator. The nodes <FEED> and <BOTTOMS> are replaced with FEED and BOTTOMS respectively, and failure modes for lines L1 and L2 are added to produce equation 6.4. Any minor units that may exist in L2 are not considered until the next section.

INTERNAL L LIQUID LO = VESSEL LEAK-TO-ATMOS
+ <BOTTOMS> Q LIQUID HI
+ <FEED> Q LIQUID LO ...(6.3)

```
INTERNAL L LIQUID LO = VESSEL LEAK-TO-ATMOS
                     + BOTTOMS Q LIQUID HI
                     + L2 LEAK-TO-ATMOS
                     + FEED Q LIQUID LO
                     + L1 PART-BLOCKED
                     + L1 LEAK-TO-ATMOS          ...(6.4)
```

Lines 3 and 6 from Table 6.2 were used to create this equation. The "LEAK-TO-ATMOS" failure modes of L1, L2 and the VESSEL itself are only generated where the pressures are greater than one atmosphere.

6.5.2: Incorporating Minor Process Units Into Line Operators

Minor process units are considered as a part of lines, but their failure modes are added separately. Only the failure modes of minor process units are considered: these are incorporated into operators, just as line failure modes are incorporated into the left-hand side and right-hand side node replacement operators. An example is the failure mode PART-CLOSED for a gate valve. This failure mode may cause low flow in a line, so it is added as a possible cause of low flow at a non-internal node at either end of the line.

This treatment of minor process units assumes that the input-output equations for the unit are the same as those for lines. Some units may be classified as minor process units, however, even if they do not conform to this rule. An example of this is the non-return valve, for which the line equation

```
input Q REV = output Q REV                       ...(6.5)
```

does not hold true. Non-return valves may be treated as minor process units, however. Links, generated by the node replacement operators, representing the above equation are simply broken if a non-return valve is in the line, and replaced by an AND gate:

```
input Q REV = output Q REV * NON-RETURN-VALVE FAILED
                                                 ...(6.6)
```

Generally, however, the definition that input and output deviations for minor process units are directly dependent does hold true. This definition makes the integration of minor process units into input and output lines very easy. A list of the failure modes of minor process units and their consequences is used, events representing these failure modes are created, and then linked to their consequences as possible causes.

This process may be complicated where combinations of process units occur in a single line. For example, if more than one closed gate valve exists in a line, the link for SOME flow involves an AND gate:

input or output Q SOME = V1 PART-OPEN
* V2 PART-OPEN
* ... ...(6.7)

Rules may be written to deal with specific combinations of units such as this. Often, however, even groups containing several minor process units do not cause any complications, in such cases the effects of each unit are simply added separately. Wherever possible, units that may be defined as minor process units should be treated as such, but as a last resort, if difficult circumstances do prevail, any process unit may be defined as a major process unit.

An example minor process unit is a gate valve, the failure modes of which are shown in Table 6.3. The valve's status governs which of these failure modes are possible. Where an open gate valve exists in a line, then the failure mode "PART-CLOSED" is added as a cause of low flow into or out of the line; "CLOSED" is added as a cause of no flow into or out of the line. In a line isolated by a closed gate valve, the failure mode "PART-OPEN" is added as a cause of some flow into or out of the line. If more than one gate valve isolates a line, then "PART-OPEN" failure modes are joined by an AND gate, as in equation 6.7.

Closed gate valves isolating a bypass line are treated slightly differently. Here "PART-OPEN", or a group of valves "PART-OPEN" joined by an AND gate if there is more than one valve isolating the line, cause

high flow, rather than some flow, at nodes before or after the bypass. Bypass lines provide an example of situations in which a choice of representational specificity is available. A closed bypass line may be specified as such, and rules written about closed bypass lines. Alternatively, a closed bypass line may simply be defined as a closed line with a line division at one end and a junction at the other. Either way, the failure logic comes out the same, but resulting fault trees will be simpler if the more highly specified approach is used. This is another example of problem reduction, and is discussed more fully in section 11.1.3.

Table 6.3: Gate Valve Failure Modes and their Consequences in a Line

VALVE STATUS	FAILURE MODE	CONSEQUENCE
CLOSED	OPEN	Q SOME*
OPEN	PART-CLOSED	Q LO
OPEN	CLOSED	Q NO

*If more than one valve of closed status exists in a line, then the failure mode "Valve OPEN" for each valve should be combined by an AND gate.

6.5.3: Justification of the Process Unit Model Expansion Methodology

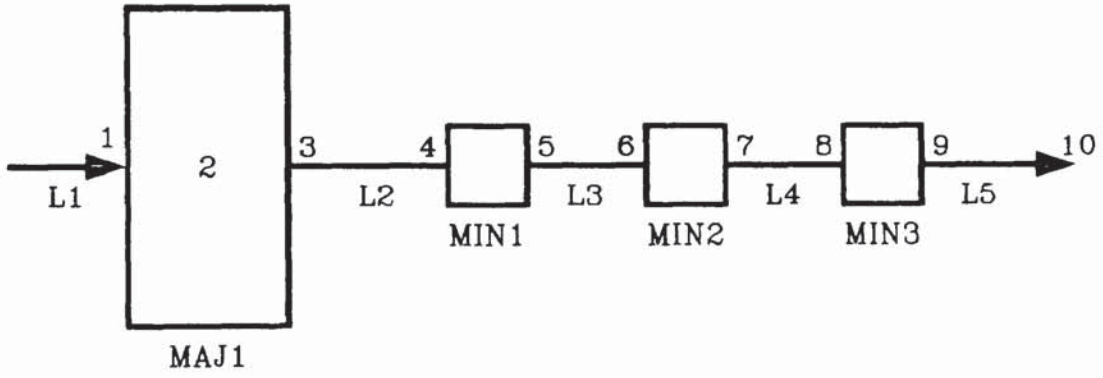
It was stated at the start of this chapter that any fault tree synthesis method must be totally reliable: the method should be capable of producing fault trees that are both accurate and complete. The functional method involves simplification of the plant representation; it is important that this simplification is achieved without any concession in terms of accuracy or integrity of the fault trees produced.

The justification for model expansion is that causal failure events may be added under an OR gate. Any minor units, all of whose input-output equations are directly dependent, forming part of a line may be modelled in terms of their failure modes only, without any loss of functionality. This is due to the fact that input-output equations, all of which are directly dependent, may be summed into input-output and failure mode equations for the entire line, including the minor process

units in the line. The resulting expanded input-output equations for the entire line will also be directly dependent.

This is illustrated by Fig. 6.8, which shows a generalised output line containing three minor process units, MIN1, MIN2, MIN3. The major unit from which the line leaves is MAJ1. If the line is modelled by standard process unit modelling: lines L2, L3, L4 and L5 comprise the functional output line; nodes 3 to 9 are placed at the input to and output from each unit.

Fig. 6.8: A Generalized Output Line Containing Three Minor Process Units



The input-output equations for MAJ1 are not all directly dependent; for a single output event, "V1,P1,C1,D1", at node 3 they are of the general form:

2 or 1 V0,P0,C0,D0 = 3 V1,P1,C1,D1 + MAJ1 FAIL1 ...(6.8)

or:

3 V1,P1,C1,D1 = 2 or 1 V2,P2,C2,D2 + MAJ1 FAIL2 ...(6.9)

where:

Vi is a variable
Pi is a phase
Ci is a component
Di is a deviation
FAILi is a set of unit failure modes

Equation 6.8 is a **backward equation:** it applies in the opposite direction to

the flow. Conversely, equation 6.9 is a **forward equation**, it applies in the direction of flow. The following analysis considers only forward equations, but it is true also of backward equations because both forward and backward equations are directly dependent for the lines L2, L3, L4 and L5, and the units MIN1, MIN2 and MIN3.

The input-output equations for each of the lines, L2, L3, L4 and L5 are directly dependent, therefore an output deviation may only be caused by the same input deviation, and vice-versa. The general forward equations for these lines are:

$$4\ V1,P1,C1,D1 = 3\ V1,P1,C1,D1 + L2\ FAIL3 \quad ...(6.10)$$

$$6\ V1,P1,C1,D1 = 5\ V1,P1,C1,D1 + L3\ FAIL3 \quad ...(6.11)$$

$$8\ V1,P1,C1,D1 = 7\ V1,P1,C1,D1 + L4\ FAIL3 \quad ...(6.12)$$

$$10\ V1,P1,C1,D1 = 9\ V1,P1,C1,D1 + L5\ FAIL3 \quad ...(6.13)$$

The group of line failure modes, FAIL3, causing the event V1,P1,C1,D1 is the same for each line because each line may be modelled in the same way.

The input-output equations for each of the minor process units, MIN1, MIN2, MIN3, are directly dependent, but for each of these units a different set of failure equations is used. The general forward equations for these units are:

$$5\ V1,P1,C1,D1 = 4\ V1,P1,C1,D1 + MIN1\ FAIL4 \quad ...(6.14)$$

$$7\ V1,P1,C1,D1 = 6\ V1,P1,C1,D1 + MIN2\ FAIL5 \quad ...(6.15)$$

$$9\ V1,P1,C1,D1 = 8\ V1,P1,C1,D1 + MIN3\ FAIL6 \quad ...(6.16)$$

By combining equations (6.9) to (6.16) the following relationship is obtained:

$$\begin{aligned} 10\ V1,P1,C1,D1 = {} & 2 \text{ or } 1\ V2,P2,C2,D2 + MAJ1\ FAIL2 \\ & + L2\ FAIL3 + L3\ FAIL3 + L4\ FAIL3 + L5\ FAIL3 \\ & + MIN1\ FAIL4 + MIN2\ FAIL5 + MIN3\ FAIL6 \end{aligned} \quad ...(6.17)$$

If the lines L2, L3, L4 and L5 are grouped into one single output line, L6,

then equation 6.17 may be shortened to:

$$10\ V1,P1,C1,D1 = 2\ or\ 1\ V2,P2,C2,D2 + MAJ1\ FAIL2 + L6\ FAIL3 + MIN1\ FAIL4 + MIN2\ FAIL5 + MIN3\ FAIL6 \quad ...(6.18)$$

This equation describes the output line operator for forward equations. The output line operator for backward equations for the same example is:

$$2\ or\ 1\ V0,P0,C0,D0 = 10\ V1,P1,C1,D1 + MAJ1\ FAIL1 + L6\ FAIL7 + MIN1\ FAIL8 + MIN2\ FAIL9 + MIN3\ FAIL10 \quad ...(6.19)$$

The corresponding equations for input line operators are of the same form as equations 6.18 and 6.19.

It is possible to create minor process units even if not all of their input-output equations are directly dependent, but care must be taken to ensure that the units' full functionalities are modelled, both when they appear in lines singly and in combination with other units.

6.6: Representation of Control Loops and Protective Systems by Loop Operators

Control and trip loops and pressure relief systems are treated as separate "loop" entities, which are composed of individual control units. The failure and operating modes of loops may be defined by loop operators, which represent the effects of each mode on controlled and controlling variables. Loop operators may themselves be defined in terms of the operating and failure modes of the individual control units that make up the loop.

A single loop, such as a feedforward control loop, may have several measured variables; such loops are treated as separate entities, with the controlling and measured actions, and the control units defined discretely for each measured variable.

Two types of loop failure mode are identified: **general failure modes** and **specific failure modes.** General loop failure modes result from failures of control units that make up the loop. It is possible to classify control unit failure modes, and the loop failures that result. Tables 6.4 and 6.5 show the general failure modes and their unit causes for control loops and trip loops.

Table 6.4: General Failure Modes and Their Unit Causes for Control Loops

UNIT TYPE	MEASURED ACTION	CONTROLLED ACTION	CAUSE OF LOOP STUCK	CAUSE OF LOOP FAILED-HI	CAUSE OF LOOP FAILED-LO
TRANSDUCER	POSITIVE	--	STUCK	FAILED-LO	FAILED-HI
TRANSDUCER	NEGATIVE	--	STUCK	FAILED-HI	FAILED-LO
CONTROLLER	--	--	STUCK	FAILED-HI	FAILED-LO
ACTUATOR	--	POSITIVE	STUCK	FAILED-OPEN	FAILED-CLOSED
ACTUATOR	--	NEGATIVE	STUCK	FAILED-CLOSED	FAILED-OPEN

Actuator here refers only to control and trip valves

Specific loop failure modes are those that arise from other plant deviations. Examples of specific loop failure modes are:

1. Loops which tend to increase flow of the controlling variable fail if flow is reduced by a blockage in the line.
2. Loops which tend to decrease flow of the controlling variable may fail if flow is increased by a leakage in the line.
3. Where liquid entrainment in a gas flow occurs this may produce an increase in the overall density of the flow, causing flow transducers to record incorrectly.

Specific failure modes may be included in the list of failure modes for the loops or units to which they apply, along with the general failure modes. Specific loop failure modes are not explicitly modelled in the current

Table 6.5: General Failure Modes and Their Unit Causes for Trip Loops

CONTROL UNIT	CONTROLLED DEVIATION	MEASURED DEVIATION	CONTROLLED ACTION	CAUSE OF TRIP LOOP FAIL-DANGER	CAUSE OF TRIP LOOP FAIL-SAFE
TRANSDUCER	--	HI, SOME	--	FAILED-LO, STUCK	FAILED-HI
TRANSDUCER	--	NO, LO	--	FAILED-HI, STUCK	FAILED-LO
CONTROL VALVE	HI, SOME	--	POSITIVE	FAIL-OPEN, STUCK	FAIL-CLOSED
CONTROL VALVE	HI, SOME	--	NEGATIVE	FAIL-CLOSED, STUCK	FAIL-OPEN
CONTROL VALVE	NO, LO	--	POSITIVE	FAIL-CLOSED, STUCK	FAIL-OPEN
CONTROL VALVE	NO, LO	--	NEGATIVE	FAIL-OPEN, STUCK	FAIL-CLOSED
TRIP VALVE	HI, SOME	--	POSITIVE	FAIL-OPEN	FAIL-CLOSED
TRIP VALVE	HI, SOME	--	NEGATIVE	FAIL-CLOSED	FAIL-OPEN
TRIP VALVE	NO, LO	--	POSITIVE	FAIL-CLOSED	FAIL-OPEN
TRIP VALVE	NO, LO	--	NEGATIVE	FAIL-OPEN	FAIL-CLOSED
SOLENOID VALVE	--	--	--	FAIL-TO-VENT	OPEN-TO -VENT

implementation of the OFTS program, although the effects of such failures may be catered for during causal ordering.

6.6.1: The Effect of Loop Failure Modes on the Controlled Variable

Shafaghi [47] identified three modes of control loop failure that affect the controlled variable. In this research, trip loops and pressure relief systems are also modelled in this way, although Shafaghi only identified one mode of failure for trip loops.

Loop **invariant failure**, represented by STUCK for control loops or FAIL-DANGER for trip loops or pressure relief systems, describes a failure mode whereby the loop does not react to input deviations. In order for a controllable disturbance to produce a large deviation of the controlled variable, any loops capable of protecting the controlled variable against this input disturbance must fail in this way. Loop operators which model invariant failure modes generate combinatory inputs to events representing deviations of the controlled variable. Invariant loop failure is caused by invariant failure of one of the loop elements, for example, a control valve's stem may stick in one position.

Loop **activation failure**, represented by control loop FAILED-HI or FAILED-LO, trip loop or pressure relief system FAIL-SAFE, describes an error of loop activation. Control loop activation failure occurs when the loop produces incorrect output: the value of the controlled variable is too high or too low. Trip loop or pressure relief system activation failure occurs when the loop operates falsely when the measured variables are within their design limits. Loop activation failure operators generate additional causes of deviations of the controlled variable. For example, control loop FAILED-HI is a possible cause of a high deviation of the controlled variable. Loop activation failure modes are caused by a failure of one of the loop elements, for example, a transducer which records too high will cause a negative feedback control loop to fail low.

Loop failure due to an **induced failure** occurs when a control element is susceptible to failure due to a specific input condition. When this input condition occurs, the control element may fail, causing loop failure.

Induced failure is thus a combinatory failure mode: an element must be susceptible to a particular event, and that event must occur, in order for failure to be induced. Induced failure modes are usually specific to certain types of equipment, and are not modelled in the current computer implementation of this work.

Shafaghi modelled the direct effects of these loop failure modes on the controlled variable. In this research, however, loop activation failure modes are modelled in terms of their effects on the controlling variable. The effects of loop activation failure modes on the controlled variable are modelled indirectly through the controlling variable. This modelling simplifies the fault tree synthesis algorithm, and represents the failure event connections more accurately.

6.6.2: The Effect of Loops on the Controlling Variable

Controlling variables represent the output of loops, as such they are affected both by normal loop actions and by loop failures. Previous automatic fault tree synthesis methods do not include specific controlling event operators, thus overlooking two classes of failure propagation. The two loop operators identified are:

1. Activation failure of loops will affect the controlling variable, producing an erroneous output. The effects of specific loop failure modes on the controlling variable are described in section 9.11.
2. Correct loop response to a deviation of the measured variable will result in a value of the controlling variable which is different from the design value. A causal link is created such that deviations of the measured variable form inputs to controlling variable deviations; a condition of this link being that the loop is functioning normally. Normal conditions are ensured by introducing boundary conditions. This is described in section 9.11.

The above operators allow the effects of controlled variable deviations and loop failures to propagate through the system. Other failure modes that may result in deviations of the controlling variable are unaffected.

6.7: Discussion

The methods of plant reduction described in this chapter are designed to reduce the size of fault trees, whilst maintaining complete integrity. They do not, however, solve any of the problems relating to the method's applicability to a wide range of problems and control strategies. Methods for solving such problems are discussed in the next chapter.

A fault tree synthesis program developed using the plant reduction strategies described in this chapter, and loop ordering strategies described in chapter 7 has been developed. Results obtained by using this method are presented and discussed in chapter 10. The conclusion drawn from these results is that the plant reduction strategies used do produce a significant reduction in fault tree size.

CHAPTER SEVEN

7. A METHOD WHICH USES LOOP OPERATORS TO FACILITATE MODELLING OF COMPLEX PLANT CONTROL STRATEGIES

7.1 Introduction

One of the major difficulties in the construction of automatic methods for fault tree synthesis is that of defining the controllability, or otherwise, of events by control loops and protective systems. Lapp and Powers [43,44,45] addressed this problem by defining the magnitude of input-output dependencies by the use of a gain. A gain of 0 represents no dependency; ±1 represents a normal dependency, within the bounds of controllability for control loops; ±10 represents a gain beyond the capacity of control loops. Allen and Rao [46] argued that there were problems with this representation: it is difficult to handle multiple control loops if some, but not all, loops are able to handle certain disturbances. The only way that multiple control loops could be handled by this representation would be to define each dependency in terms of its controllability by each control loop: a prohibitively time-consuming task for all but the smallest of problems. Another problem with this method is its inability to handle trip loops. If trip loops algorithms were included in this method then a further gain value of, say, ±100, would be required to define dependencies in terms of their controllability by trip loops.

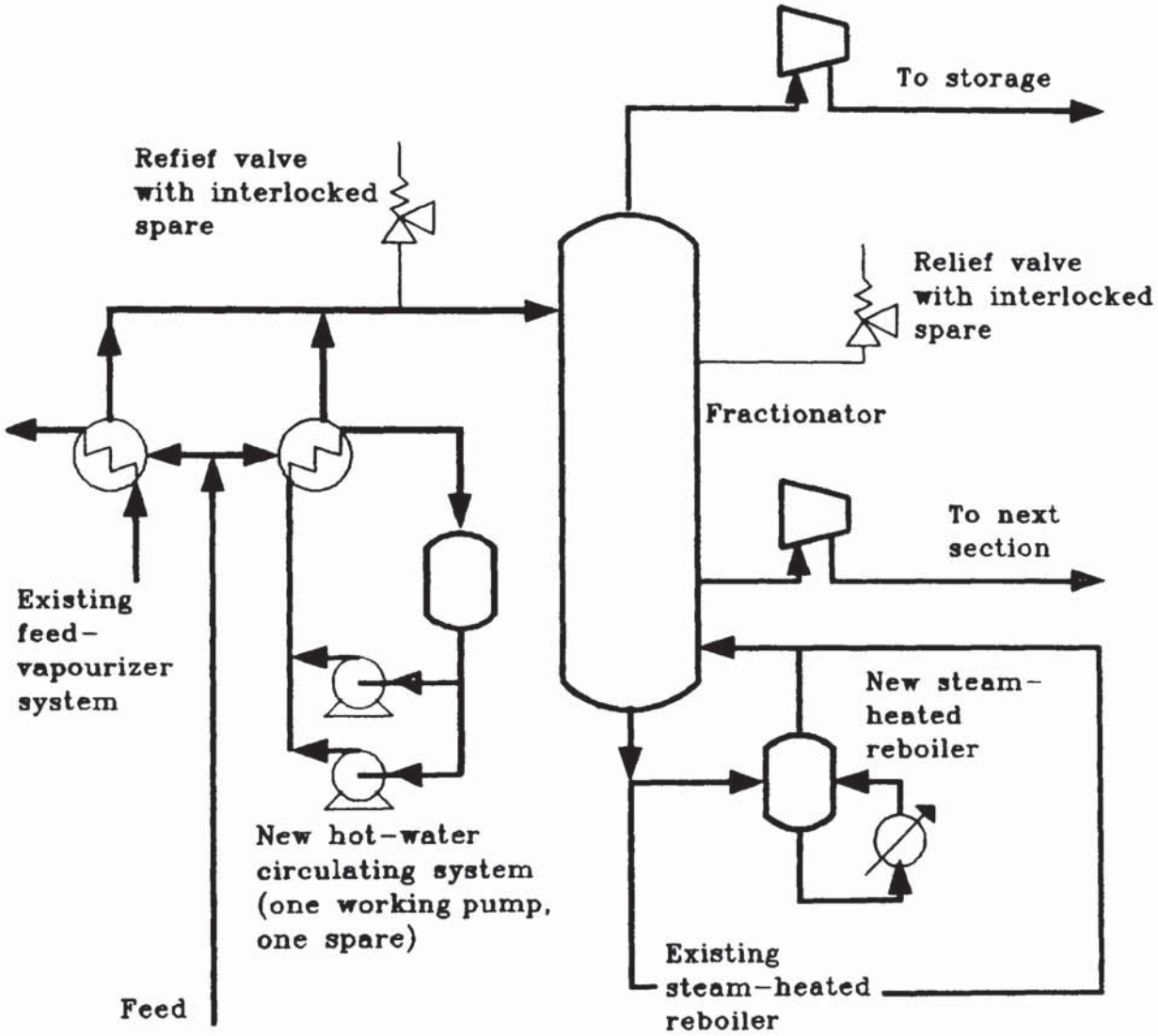
In light of these problems, Allen and Rao used a different method for the representation of uncontrollable events. Events that were beyond the ability of a specific control loop to deal with were included in a list of circumstances under which specific control action will fail. This method could be extended to cover trip loops. There is however, a problem with this representation: magnitudes of events are not defined. It is possible that a small disturbance of an event may be controllable, whereas a larger disturbance of the same event, but caused by different input logic, may be uncontrollable. If the input events are to be analysed sufficiently for the actual causes of uncontrollable events to be identified, then fault trees may as well be constructed by hand. This method does not alleviate the problem of defining every dependency in terms of every control loop.

Shafaghi's method of dealing with uncontrollable events [47] was to classify each input disturbance to a control or trip loop in terms of its controllability by that loop. This method does not take magnitudes of events or dependencies into account: uncontrollable events for loops are simply general events that cannot be handled by certain types of loop.

A second problem involving the handling of control and trip loops in fault tree synthesis is that of dealing with atypical plant arrangements. Fig. 7.1 shows a plant uprating scheme as described by Lawley and Kletz [90]. Two new heat sources were required on an existing plant: a feed vapourizer heated by pump-circulated hot water, and a new steam-heated reboiler. Calculations indicated that the two existing relief valves would not afford adequate protection in the uprated section against a downstream block-in. Because of limitations in the relief-header system, additional protection could not be achieved by resizing the existing relief valves. Two alternatives were identified: to install a third relief valve with its own header; or to install a high-pressure-trip system, in parallel with the existing relief valves, to trip the proposed new hot-water circulating pump and its spare, and to shut off steam supply to the new reboiler in the event of abnormal rise in system pressure. Various trip systems were analysed. It was shown that a one-out-of-two, separate channel, high-pressure-trip system would be capable of safeguarding against vessel rupture, with a reliability ten times greater than conventional relief protection, as long as adequate proof testing was maintained.

This plant uprating scheme is an example of the sort of atypical control strategy with which automatic fault tree synthesis methods have difficulty. For adequate protection against vessel rupture due to high vessel pressure, it is necessary that one of the relief valves lifts correctly, and one of the trip loops operates correctly. Automatic fault tree synthesis methods would commonly assume that one of or that all of these protective systems must operate correctly in order to prevent vessel rupture; this is clearly not the case. It is therefore necessary that, in order to model such plant arrangements, some means of defining protection logic is incorporated into the fault tree synthesis method.

Fig. 7.1: A Plant Uprating Scheme



A method, known as **event ordering**, which attempts to solve the dual problems of event controllability and protection logic definition, has been developed during this research. The event ordering method is described during the remainder of this chapter.

7.2: The Event Ordering Method for Modelling Complex Plant Control Strategies

The basis of event ordering is to define the magnitude of deviations in terms of loop controllability. The magnitude of an event is described by the event's **order**. The order of a deviation of a controlled variable defines that event's controllability by certain classes of loop. The order of a deviation of a measured variable defines the ability of certain classes of loop to protect controlled variables against that deviation. Only controlled and measured events are defined in this way.

7.2.1: Defining an Order Scale for Controlled and Measured Events

Refer to the ammonia let down plant section in Fig. 7.2. The variable "liquid level at node 7" is controlled by the control loop LCL1, and the low-level-trip loops LLTL1 and LLTL2. The magnitude of deviations of this variable may be defined in terms of the actions of these loops. This is illustrated by Fig. 7.3. Consider the low deviation of this variable. The variable may fluctuate from its set point due to deviations of other variables; the action of the control loop is to reduce the scale of these fluctuations, maintaining the value of the variable within certain limits. A deviation of the controlled variable within these limits is termed a **zero-order deviation.** When the level falls below the lower limit of the control loop, the low-level-trip systems will operate. The level at which this occurs represents a **first-order deviation.** The action of these trip systems is to prevent the level from falling further. A **second-order deviation** may be defined: this represents failure of the trip loops to regulate the deviation within certain limits. A second-order deviation results from an uncontainable input event, or failure of both of the trip loops.

In the above example, the deviations were ordered with respect to

Fig. 7.2: Section of an Ammonia Let Down Plant

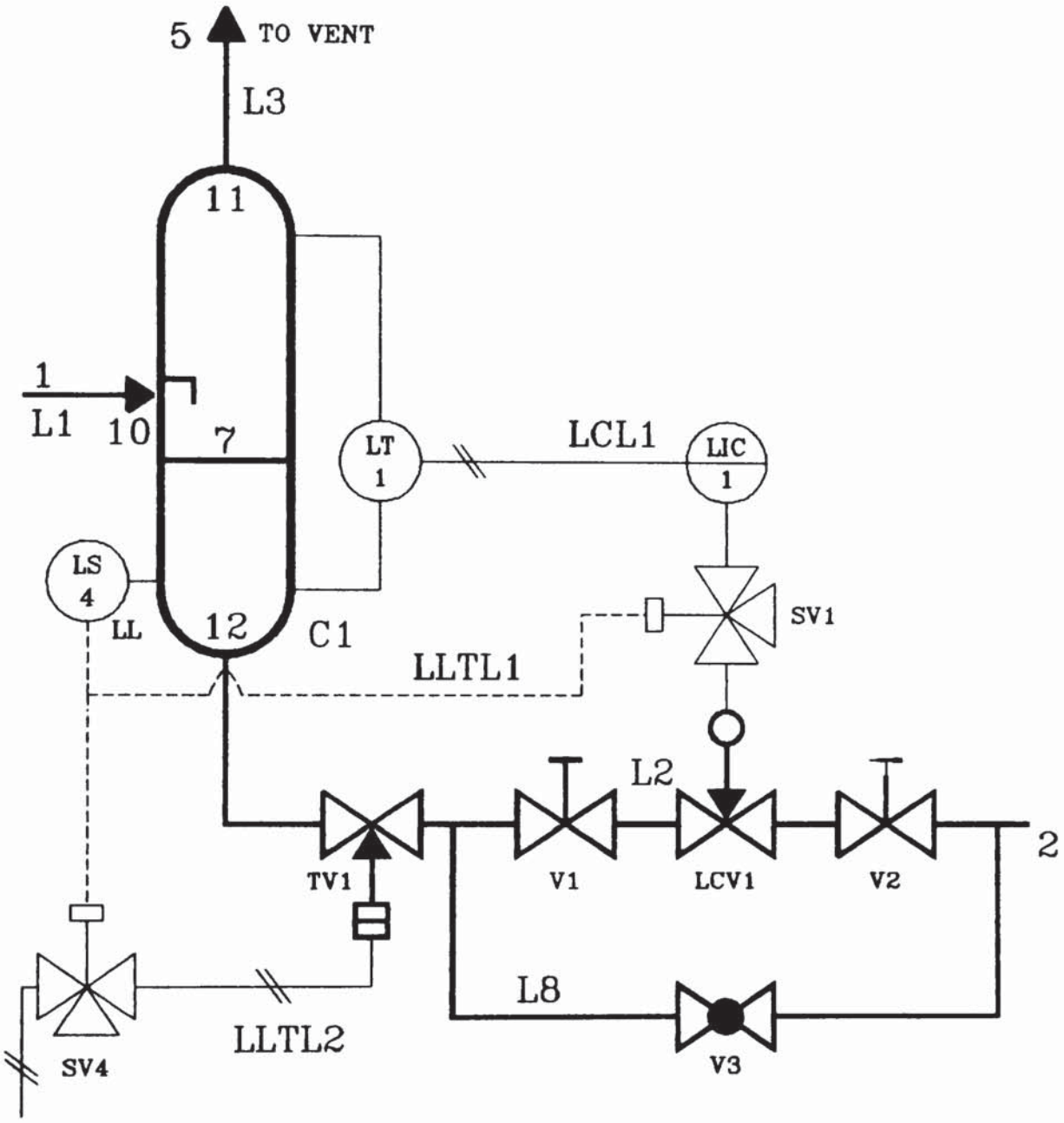
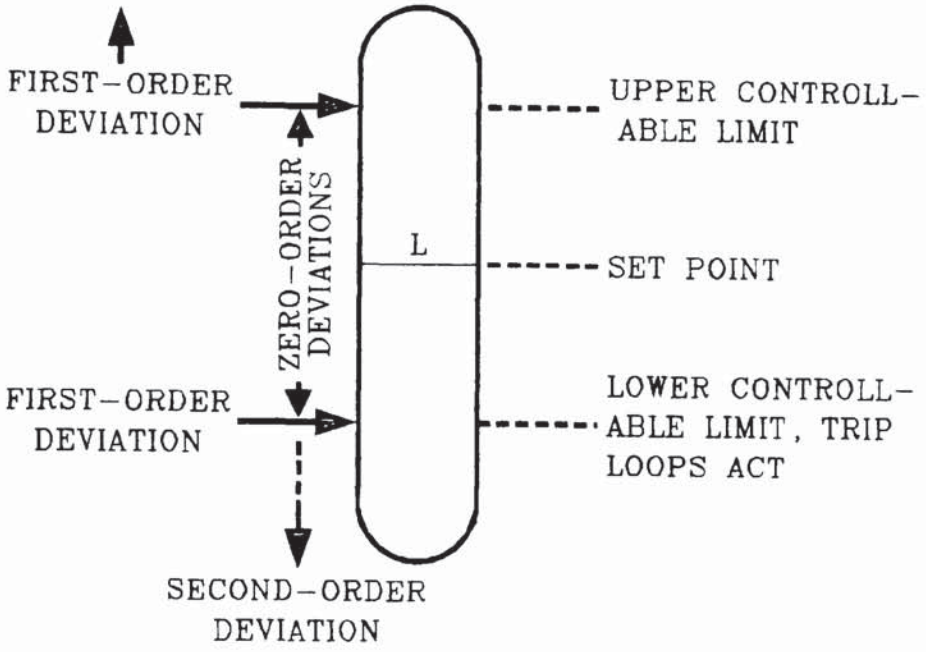


Fig. 7.3: Representation of an Event's Order Scale



the control loop, LCL1, and the trip loops, LLTL1 and LLTL2. The control loop was assigned an order of 1, the trip loops were assigned an order 2. The definition of a loop's order is:

> *"The order of deviation of the controlled variable which the loop acts to prevent."*

In this example, the trip loops were assigned the same order as they are activated by the same level switch. Trips activated by different level switches could, if necessary, be assigned different orders.

For the purpose of event ordering, pressure relief systems are considered to be equivalent to trip loops. If a loop is assigned to an order n, then Table 7.1 defines the meaning of various controlled event orders in terms of controllability by that loop.

Events may be assigned as many orders as is necessary to represent the effect of loops. Several loops may be assigned to the same order, such as the trip loops LLTL1 and LLTL2 in the above example. Often it is unnecessary to define greater than two orders of loops. The definition of loop orders is at the discretion of the user, who must understand both the workings of the OFTS system, and the plant.

Table 7.1: The Meaning of Controlled Event Orders in Terms of Loop Controllability

ORDER	LOOP CLASS	MEANING
≤ n-1	CONTROL	Deviation within controllable limits
≥ n	CONTROL	Deviation outside controllable limits
≤ n-2	TRIP, RELIEF	Loop not activated
n-1	TRIP, RELIEF	Loop activated, deviation containable
≥ n	TRIP, RELIEF	Loop activated, deviation uncontainable

In addition to controlled variables, those variables that are measured by feedforward control loops, or by trip loops, are also ordered. Measured variables are ordered in terms of the loops they are measured by. The important factor in the ordering of measured events is their effect on the controlled events. Measured events are given an **initiating order** and a **saturating order** for each loop for which they are measured. The initiating order is the relative magnitude of the deviation which will initiate loop action; the saturating order is the relative magnitude of the deviation which will saturate the loop, and is thus beyond that loop's controllable limits. For feedforward control loops, the initiating order will be zero. The meaning of various measured event orders in terms of loops of initiating order, n, and saturating order, m, are defined by Table 7.2. As for controlled events, as many different orders as the user desires may be defined, but it is recommended that the only minimum required to accurately represent the plant's operation are used. An event which is both controlled and measured by separate loops may be ordered on a common scale in terms of both controlled and measured orders.

Table 7.2: The Meaning of Measured Event Orders in Terms of Loop Controllability

ORDER	LOOP CLASS	MEANING
< m	CONTROL	Controllable deviation
≥ m	CONTROL	Uncontrollable deviation
< n	TRIP	Loop not activated
≥ n < m	TRIP	Loop activated, containable deviation
≥ m	TRIP	Loop activated, uncontainable deviation

The use of event ordering not only allows for the definition of event magnitudes, it also adds structure to the fault trees produced. The algorithms that are used to generate fault trees in terms of event orders are described in the subsequent sections.

7.2.2: Definition of Protection Logic for Controlled Events

One of the major benefits of the event ordering method is that it allows any logical combinations of loops that protect a variable to be defined. It is not assumed that all loops must operate correctly to contain a variable, or that only one loop is required to function correctly; any combination is allowed. This allows the modelling of complex control strategies, such as high-integrity-protective-systems (HIPS), and combinations of pressure relief and trip systems.

Combinations of loops that protect a variable are defined in terms of logic, using, where necessary, intermediate **groups** to build a protection tree. Only loops with the same measured event which are active at the same order may be grouped together; if a loop is inactive or saturated at a certain order, then it cannot be considered to protect against that order. Groups are themselves defined in terms of other groups and loops, forming a mini-tree, until all the base events consist of loops. Each group may be related to its input groups and loops by one of three types of logic operator:

1. AND gates, which require that all the input groups and loops must function correctly in order that the output group protection is achieved.
2. OR gates, which require only that one of the inputs function correctly for the output protection to be achieved.
3. OUT-OF, this operator requires an argument, **m**, meaning that m out of the n inputs must function correctly for the output protection to be achieved.

In most cases, it is necessary to define, at most, one intermediate group; often no intermediates are necessary. This method does, however, allow protection of any complexity to be defined.

Having created a mini-tree that describes the logic in order that protection is achieved, a mini-fault tree for the top group's failure to protect (FPROTECT) may be obtained. This is achieved by converting the logic gates in the protection mini-tree as follows:

1. AND gates for protection are converted to a protection failure OR gate: only one input group or loop must fail for the output group protection to fail;
2. OR gates for protection are converted to AND gates in the mini-fault tree: all the input groups and loops must fail for the output group protection to fail;
3. OUT-OF gates in the mini-tree for protection are converted into OR and AND gates in the mini-fault tree, depending on the values of n and m. If, for example, 2 OUT-OF 3 protection is required, then 3 pairs of input events are created as in equation 7.1:

output group protection failure =

(input group 1 failure * input group 2 failure)
+ (input group 1 failure * input group 3 failure)
+ (input group 2 failure * input group 3 failure)

...(7.1)

Mini-fault trees for failure to protect are combined with deviations of the measured or controlled variables by loop-failure-to-protect operators similar to those used by Shafaghi, Lapp and Powers, and Allen and Rao. There are two loop operators:

1. Loops where the controlled variable is the same as the measured variable, including negative feedback control loops. When the group protecting an order, n, fails, then the order n-1 of the controlled event is equivalent to the higher order n. That is,

controlled deviation, order n = (failure to protect order n
* controlled deviation, order n-1)

...(7.2)

2. Loops where the controlled variable is different to the measured variable, including feedforward control loops. The loops normally protect the controlled variable against a certain order of the measured event. Where that order of the measured event appears as an input, no matter how indirectly, to an event which is a protected deviation of the controlled variable, then the measured event should be combined by AND logic with group protection failure. This corresponds to equation 7.3. Several intermediate events may occur between the controlled and measured events.

controlled deviation, order n = [intermediate stages]
= (measured event, order i
* failure to protect order n against order i)
...(7.3)

The failure to protect events in equations 7.2 and 7.3 represent group failure mini-fault trees.

7.2.3: Causal Ordering: Defining the Controllability of Failure Event Chains

The definition of events as controllable or uncontrollable by certain groups of loops is called **causal ordering.** The **causal order** of an input event chain, with respect to a specific controlled or measured event, is the highest order of the controlled or measured event that the input event chain is judged to be able to cause, **assuming that all control and trip loops, and pressure relief systems function correctly.** Thus the causal order of an event chain is the order of the controlled or measured event for which the causal order is defined to which the event chain will form a direct input, without further failure to protect logic in conjunction with it.

A second order must be defined, the **limiting order.** The limiting order of an input event is the maximum order of a controlled output event that it can produce, **assuming that all protection of the controlled variable fails.** This places a limit on the order of a controlled event that the input event can cause, even in combination with total protection failure.

There is one assumption applied to causal ordering: that an event is

equally controllable or uncontrollable by all loops of a certain order. Where this assumption is not valid, then the event should be deemed controllable by that order of loops, but included in the specific failure modes of any loops for which it is uncontrollable. Normally, however, this assumption will be true, except for any events previously defined as specific failure modes of some, but not all, of the loops of the same order.

The important question with definition of event magnitudes and their effects on loops is: which events should be defined with respect to which loops? It is not normally practical to develop a mini-fault tree so deeply as to examine the base events as prior knowledge about the effects of other event magnitudes will be necessary in order to correctly define the tree. A simpler answer would be to order only the immediate input events to the controlled event in question, but this treatment is insufficiently detailed to represent plant failure chains accurately.

The method employed in this research is to generate mini-fault trees for each controlled and measured event, ending tree development at: primary events; events that arise from across plant boundaries; controlled and measured events. Each of the base events has its causal, and limiting if the top event is a deviation of a controlled variable, orders defined by the user in terms of this top event's order scale. Causal and limiting orders are defined for every permissible order of controlled and measured base input events. Failure chains of no or negligible importance may be removed altogether.

The reasoning behind this method is that it is unlikely that events linked more distantly than this will have a direct influence on the loops under investigation. Only controlled and measured events are assigned orders; only ordered events and primary events are assigned causal and limiting orders. This provides a convenient, systematic method for the representation of the effects of event magnitudes on loops.

7.3: Using Causal Ordered Event Branches to Build a Fault Tree Structure

Event branches given causal orders as described above may be used to build structured fault trees. There are two types of input to a

controlled event of order n in a fault tree:

1. Event chains with a causal order of n.
2. A combinatory input: order n protection failure AND order n-1 of the controlled event. This is produced by applying equation 7.2.

This results in fault trees with a general structure as shown by Fig. 7.4. This imposition of a readily identifiable structure to fault trees makes them easier to comprehend.

When generating fault trees in this way it is important to check that the limiting orders of event chains are not violated. If, for example, the limiting order of an event chain is n-1, and its causal order is n-2, then that chain may form an input to order n-2 of the controlled event, which itself forms an input to order n-1. If however, the controlled event of order n-1 forms an input to order n of the same event, then the event chain's limiting order is violated, and it should be removed from the tree.

When a measured event of order m forms an input to a controlled event of order n, and the controlled event order n is protected against order m of the measured event by at least one activated, but not saturated, loop, then a combinatory input is formed, as described by equation 7.3.

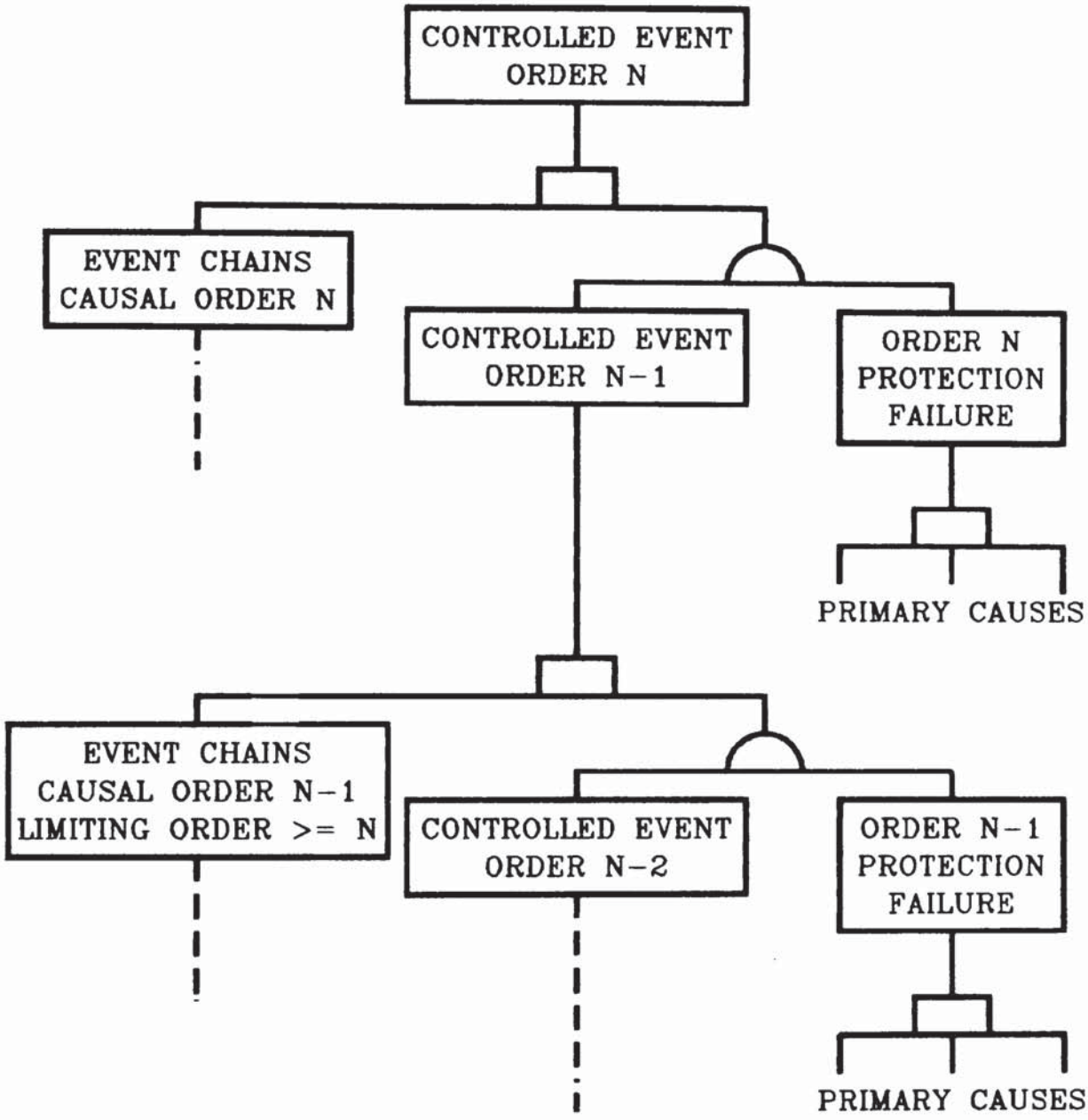
Inputs to measured events that are not also controlled events are made on a direct causal order basis. An input to order n of a measured event may be made by all event chains with a causal order of at least n.

7.4: Discussion

The event ordering method provides a systematic means for the definition of event magnitudes. The basis of this definition is that events are ordered in terms of their controllability by certain groups of control loops and protective systems. This provides a means for the representation of event magnitudes which is meaningful in terms of failure propagation.

The definition of protection mini-trees, and subsequently of

Fig. 7.4: A Generalized Fault Tree Showing Inputs to an Ordered Controlled Event



protection failure mini-fault trees, allows the modelling of complex event protection logic.

Fault trees are given extra structure by the use of event ordering, as can be seen from the generalised fault tree shown in Fig. 7.4.

One drawback of event ordering is that it increases the number of events in fault trees. This is an example of the paradox discussed in section 6.1: the event ordering method provides a means for the representation of complex plant control strategies, and also improves fault tree structure; it does, however, result in an increase in fault tree size. The actual increase in fault tree size due to event ordering is fairly small, however. When combined with a method of reducing the size of fault trees, the event ordering method is capable of producing relatively small fault trees with good structure. A hybrid method for fault tree synthesis, which combines the plant reduction and event ordering methods, is described in the next chapter.

CHAPTER EIGHT

8. A HYBRID METHOD WHICH COMBINES PLANT REDUCTION AND EVENT ORDERING

8.1: Introduction

The plant reduction and event ordering methods described in the previous chapters are incorporated into one hybrid fault tree synthesis method. The aim of this method is to produce small, structured fault trees, whilst maintaining the flexibility necessary to enable a wide range of complex plant control strategies to be modelled. The hybrid method is implemented as an expert system. This chapter describes the hybrid method itself, the computer implementation of the method using the OPS5 programming language is described in chapter 9.

8.2: Choosing a Method of Knowledge Representation

In many artificial intelligence problems the method of knowledge representation is very important to the efficiency of the solution method. This is true for the synthesis of fault trees by expert system methods; in fact, it is true also for the synthesis of fault trees using traditional programming methods.

There are two basic methods of representing failure chains prior to final fault tree generation: mini-fault trees and information flow methods, such as digraphs.

8.2.1: Mini-fault Trees

Mini-fault trees are partial trees developed only as deeply as some predetermined level. Martin-Solis et al [36] use what they call mini-fault trees to represent unit and line models, but these trees are effectively only cause-symptom equations as they are developed to a depth of only one level. These mini-fault trees have the form:

top event = cause1 (* complementary cause ...)
+ cause2 (*...) + ...(8.1)

True mini-fault trees consist of at least two levels. In order to use mini-fault trees as a knowledge representation scheme, a rationale must be developed for determining for which top events mini-fault trees should be produced and at what points mini-fault trees should terminate.

Some work on the use of mini-fault trees as an intermediate step between process unit models and the final fault tree has been undertaken during the course of this research. The top events selected were those that are either controlled, or measured by feedforward control loops or trip loops. The following event types terminated tree development:

1. Primary events;
2. Events that arise from across the plant boundary;
3. Controlled events;
4. Events measured by feedforward control loops or trip loops.

Application of these rules for mini-fault tree development and termination produces mini-fault trees of a reasonable, but manageable size.

8.2.2: Information Flow Networks and Digraphs

The information flow network was pioneered by Powers & Tompkins [3], who used a signed dependency link between plant variables. The digraph technique developed by Lapp & Powers [44] takes this further: the digraph technique represents a signed dependency with a crude measure of the gain involved in the dependency and the inclusion of any conditional events required to bring about this dependency.

Experience with the development and running of fault tree synthesis programs using mini-fault trees and information flow methods has led to several conclusions about the relative merits of these methods of knowledge representation:

1. In a real-time environment, such as alarm analysis, previously

generated mini-fault trees could be used so that fault trees can be generated rapidly. Savings in fault tree synthesis time derive from the fact that the addition of loop failure modes, the actions of loops on controlling variables and much of the inconsistency analysis may be done during generation of the mini-fault trees, rather than at the final synthesis stage.

2. Memory requirements for storing mini-fault trees are significantly greater than for the storage of information flow networks. This is because single events may appear in several mini-fault trees, whereas an event only appears once in a plant event network. Memory restrictions when a large plant is to be analysed may preclude the use of mini-fault trees as an intermediate stage.

3. The generation of mini-fault trees, because of their rigid structure and the rules pertaining to their development and termination, requires a large element of procedural program control, much of which is unnecessary for information flow network methods. Procedural control makes program development more difficult, and is thus avoided wherever possible.

8.2.3: The Plant Event Network

A **plant event network** has been developed during the course of this research. The plant digraph methods used by Lapp & Powers and Allen & Rao [46] are capable of representing only low and high deviations of variables and unit failure modes. The plant event network is generated from input-output equations similar to Lihou's cause and symptom equations [48], and is able to handle the following variable deviations: NO, LO, HI, FULL, SOME, and REV. The meanings of these deviations and, where applicable, their equivalent guide word as defined by the Chemical Industries Association [25] are shown in Table 8.1. There are several differences between these deviations and the CIA guide words. NO means the negation of an intention or part of an intention, such as no flow, or no gas flow. SOME refers to both "some as well as" and "some other than". FULL refers to level only, meaning that the phase in question fills the vessel, rather than only occupying the vessel up to a certain level. These deviations are fully capable of describing all fault conditions in terms of variable fluctuations, are easier to define in terms of

Table 8.1: Deviation Words Used in the Plant Event Network

Deviation Word	Equivalent Guide Word	Meaning
NO	PART OF[1]	A part of the design intention is not achieved
LO	LESS	Quantitative decrease
HI	MORE	Quantitative increase
FULL	--	Refers only to level, represents the complete filling of a vessel by the phase in question
SOME	AS WELL AS[2]	Some additional activity occurs
REV	REVERSE	The logical opposite of the intention

[1] This is not exactly true: NO incorporates aspects of NO or NOT and PART OF

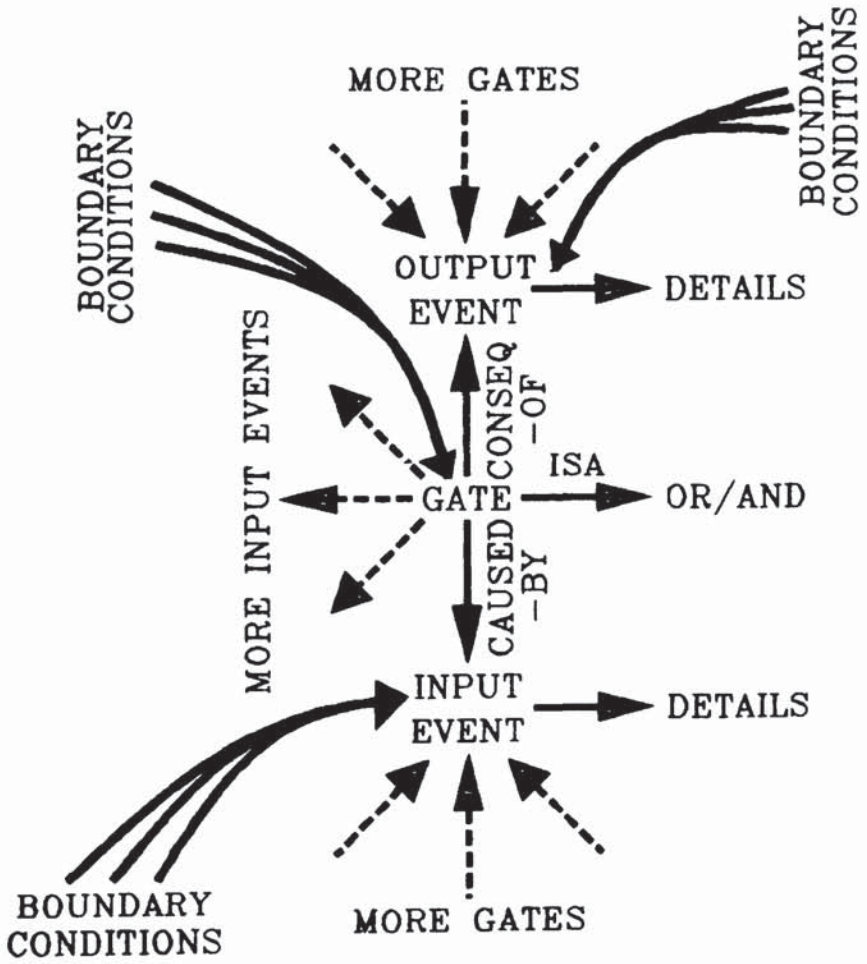
[2] This is not exactly true: SOME incorporates aspects of AS WELL AS and OTHER THAN

input-output equations and easier to form into failure chains than the CIA guide words.

This definition of deviations other than low and high is important: it ensures that all plant failure chains may now be traced. Previous signed dependency methods, including digraphs, were unable to find and trace some failure chains because of this omission.

A connection diagram for the plant event network is shown in Fig. 8.1.

Fig. 8.1: Connection Diagram Representing the Plant Event Network



Failure events form the basic nodes of the network. Gates, either logical OR or AND gates, are used to connect events. Boundary conditions associated with specific events or gates are also included in the plant event network. Boundary conditions are used to prevent the generation of

events in the fault tree which are inconsistent with the rest of the branch. Events represented by boundary conditions cannot, even indirectly, form an input to the event to which they are a boundary. If a boundary condition is associated with a gate then that boundary event may not form an input to any of the events in the fault tree which are derived from that gate.

8.2.4: Protection Mini-Trees and Protection Failure Mini-Fault Trees

Protection mini-trees represent the requirements for protection of a controlled variable to be achieved, rather than to fail. Protection mini-trees thus describe system reliability, rather than system failure. These trees are composed of groups and loops, each of which represent reliability. Groups represent any non-base events in the tree, only loops may be a base event. Any number of groups may be included in the tree, connected by any possible logic that may be described by the AND, OR and OUT-OF gates defined in chapter 7. The use of OUT-OF, as well as AND and OR, gates improves the quality of the user-interface presented by this part of the method. It is considerably easier for the user to specify, say, 2 out of 3 protection, rather than specifying each combination necessary to achieve protection of the controlled variable.

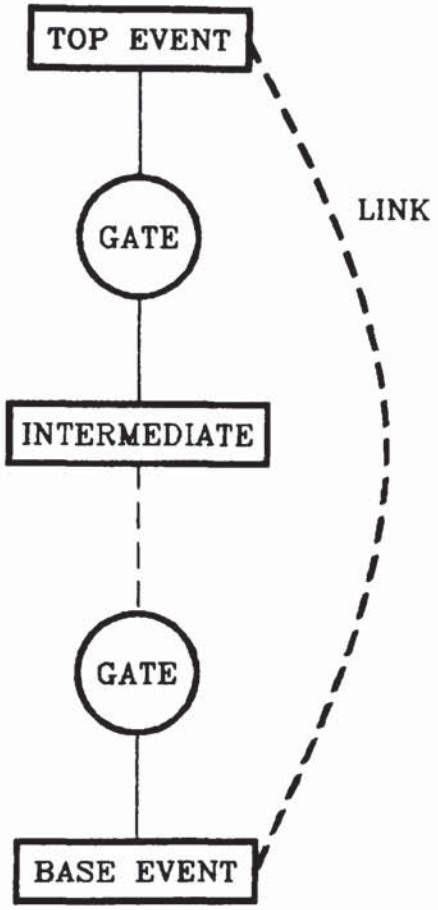
Protection mini-trees are converted to protection failure mini-fault trees as described in section 7.2.2. Protection failure mini-fault trees consist of events representing group or loop failures. Loop failures are further composed of the control unit failure events that act as inputs. Protection failure mini-fault trees are thus incorporated into the plant event network, although they are not linked to form combinatory inputs to controlled variable deviations; this is not done until the fault tree synthesis stage of the method.

8.2.5: Ordered Links

Causal ordered links could be incorporated directly into the plant event network by creating a direct link, with specified causal and limiting orders, between the top and the base events of the failure event chain to which the causal and limiting orders apply. This method is illustrated by

Fig. 8.2. This approach is memory efficient, but it does require a complex fault tree synthesis algorithm: the mini-tree must first be added to the fault tree before being shifted to form an input to the correct order of the output event, or removed altogether if the limiting order is violated. Alternatively, a search algorithm could be used to find inputs of the correct causal order that do not violate limiting order conditions.

Fig. 8.2: Direct Link Method for Representing Causal and Limiting Orders



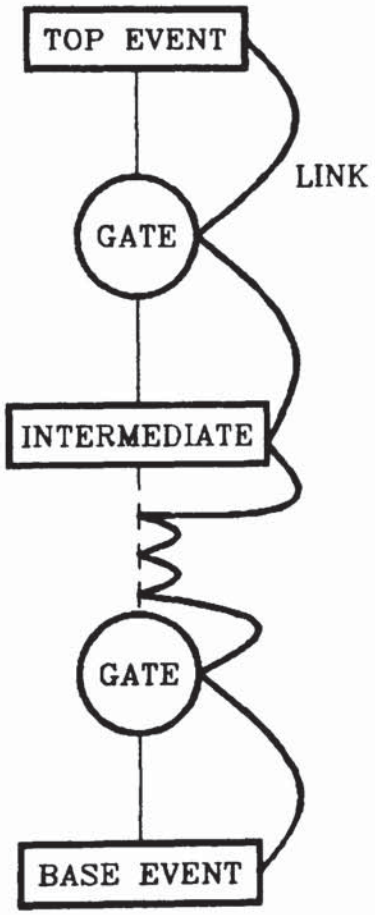
A second problem with this approach occurs when input events are repeated, but with different event paths between them and the top of the ordered mini-fault tree. It is quite possible that one of these input branches may be assigned a different causal or limiting order to the other. It is not easy to represent this using the direct ordered link method, because no details of the intermediate events are recorded.

A third problem with this approach lies in the representation of

combinatory events and branches. To represent these, additional conjunction links would have to be used.

An alternative method is to use a **routed link** between the top and base events of the mini-fault tree. This approach is illustrated by Fig. 8.3. The routed link method includes the intermediate events and gates between the top event and the base event. Routed links may be viewed as an explicit representation of mini-fault tree branches, each branch being assigned causal and limiting orders. Routed links are less efficient than direct links in terms of memory usage, but they do solve the problems described above.

Fig. 8.3: Routed Link Method for Representing Causal and Limiting Orders



The method to represent causal and limiting orders used in the OFTS program is routed links. A method for improving the efficiency of routed links in terms of memory requirement which involves assigning default

values of causal and limiting orders is discussed in section 11.2.4.

8.2.6: Fault Trees

Fault trees could, like the plant event network, consist of tree nodes, with gates used to create the linking structure. This representation is, however, less efficient than a method which includes the linking structure within the nodes themselves. Such a method is applicable to fault trees because each node is only immediately descended from one higher node, whereas it could not be used for the plant event network because each event could act as an input to many other events.

In order that the loop-failure-to-protect operator for measured events be applied, it is necessary to maintain a record, known as the **above-events register**, of all events in the fault tree in a direct path above each expanding event. This register is checked to see whether a measured event forms an input to an event which is protected against that measured event; if this is the case then the loop-failure-to-protect operator is applied.

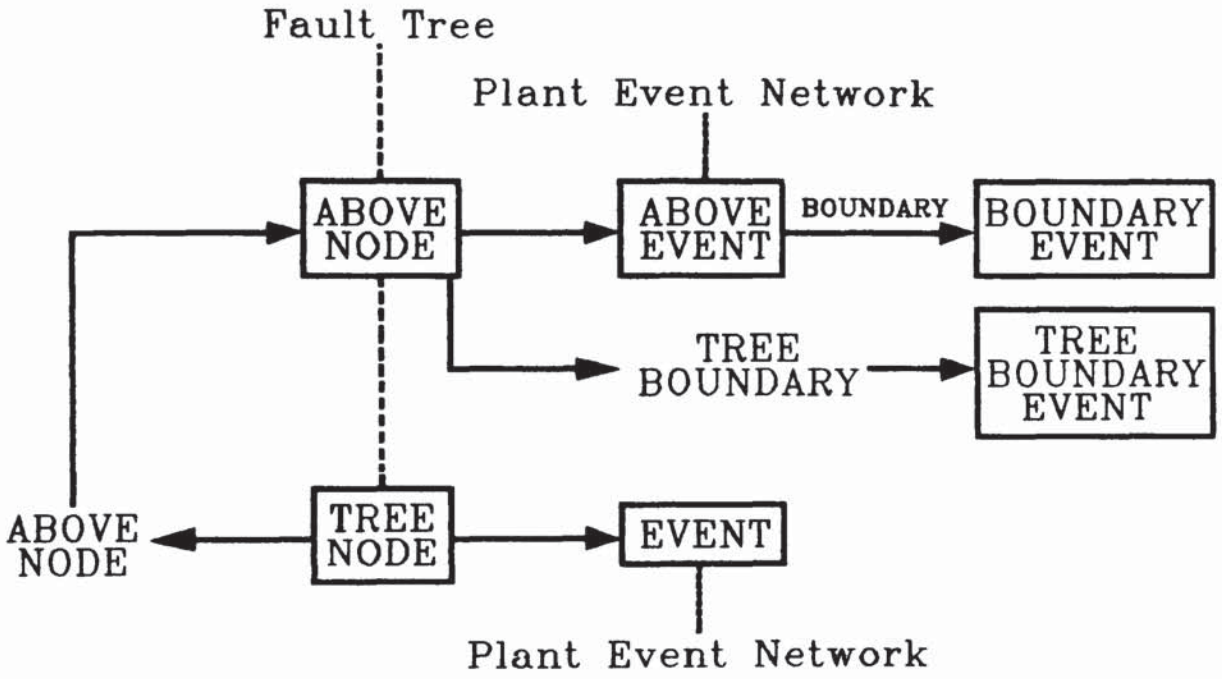
The above-events register is also used to aid the application of boundary conditions for error-checking. As well as events in a direct path from the top event to the event in question, all those combined by AND gates with these direct above events are also added to the register. These are known as indirect above events and are used only for boundary checking. The above events have boundary conditions associated with them; these boundaries are checked against the event in question to ensure its validity. If the event violates any boundaries then it, and any events combined with it by an AND gate, are removed from the fault tree.

Where a gate is used which is subject to certain boundary conditions, these boundary conditions are associated with the tree nodes which are descended from the gate in question. These boundaries are checked for each lower event in any branch of the tree derived from the use of this gate.

The fault tree data structure used to represent fault trees, their

boundary conditions and the above-events register is shown in Fig. 8.4. This figure also shows how nodes in the fault tree point to events in the plant event network.

Fig. 8.4: The Fault Tree Data Structure



8.3: Application of the Hybrid Method

A number of stages to fault tree development using the hybrid method may be identified:

1. Problem definition;
2. Plant Event Network generation;
3. Definition of protection mini-trees, and generation of protection failure mini-fault trees;
4. Creation of ordered links;
5. Fault tree synthesis.

These stages are not entirely separate. It is, for example, necessary to create the events which make up the plant event network before defining loops.

A simple example is used to demonstrate the application of the hybrid method. The example selected is the gas-liquid separation section of the basic ammonia let down plant as described in chapter 5; this is shown in Fig. 8.5. Many of the steps described here may be carried out by a computer program, such as the OFTS program described in chapter 9.

8.3.1: Problem Definition

The first step is to define the problem, in terms of major and minor process units, lines, nodes, components, connections, and operating pressures. For the gas-liquid separation section the following are defined:

major process units: C1 (type: gas-liquid separator, P > 1 atmosphere)

minor process units:
V1 (valve, status open, P > 1 atmosphere)
V2 (valve, status open, P > 1 atmosphere)
V3 (valve, status closed, P > 1 atmosphere)

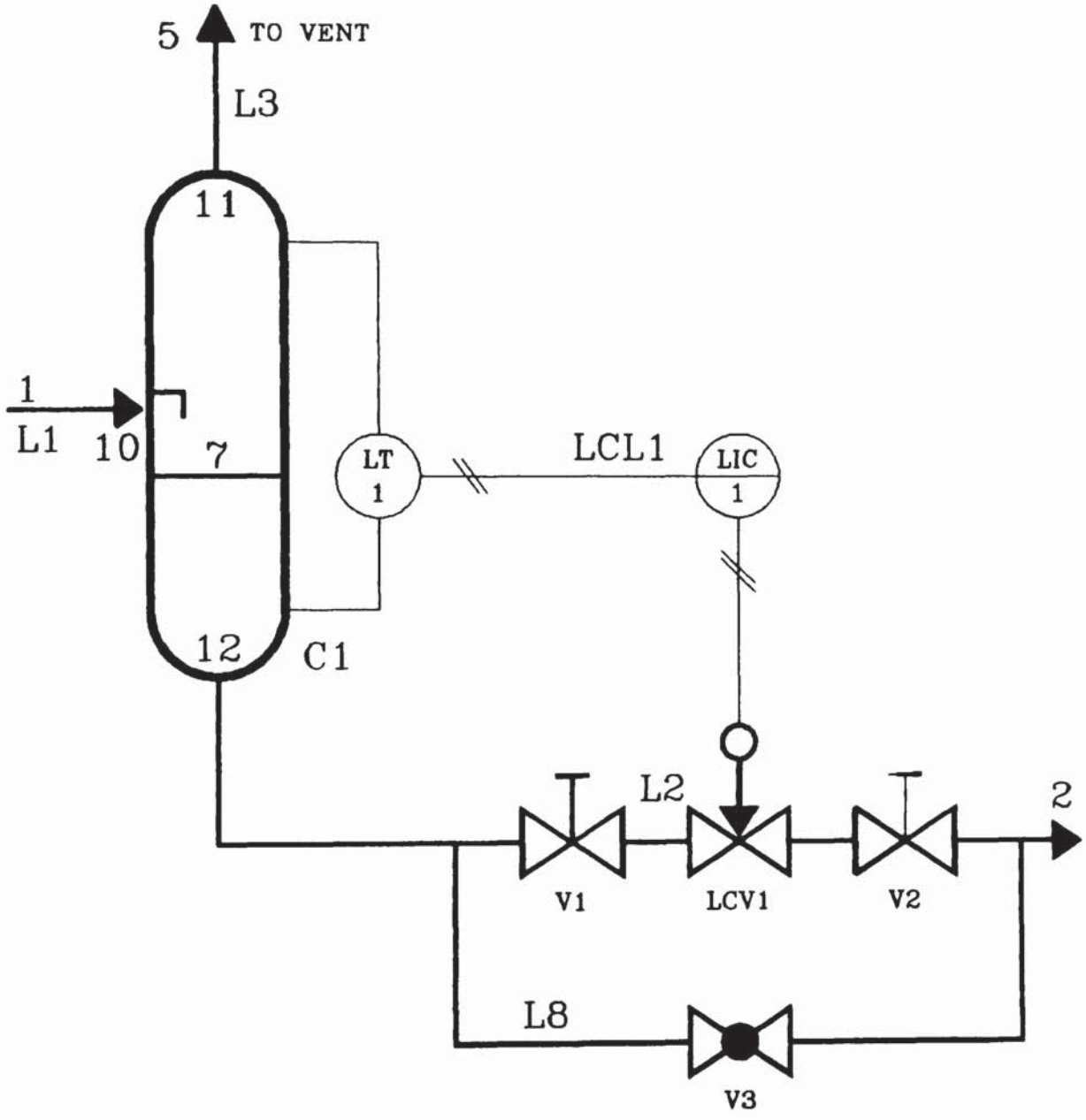
lines:
L1 (input line, open, P > 1 atmosphere)
L2 (output line, open, P > 1 atmosphere)
L3 (output line, open, P > 1 atmosphere)
L8 (bypass line, closed by V3, P > 1 atmosphere)

nodes:
1, 2, 5 (external nodes)
7 (part of gas-liquid separator, "internal" node)
10 (temporary, part of gas-liquid separator, "feed" node)
11 (temporary, part of gas-liquid separator, "tops" node)
12 (temporary, part of gas-liquid separator, "bottoms" node)

components:
ammonia gas (gas, exists at nodes 1, 7, 5)
synthesis gas (gas, exists at nodes 1, 7, 5)
ammonia liquid (liquid, exists at nodes 1, 7, 2)
synthesis gas in solution (liquid, exists at nodes 1, 7, 2)

connections:
flow from node 1 to node 10 via line L1
flow from node 12 to node 2 via line L2

Fig. 8.5: The Gas-liquid Separation Area of the Basic Ammonia Let Down Plant



flow from node 11 to node 5 via line L3
no flow from node 12 to node 2 via line L8

8.3.2: Generation of the Basic Plant Event Network

From this information, the basic plant event network corresponding to process deviations and failure modes of process units may be constructed. The first step is to generate a set of "events". Events consist of unit failures and undesirable deviations of any process variable. Those variables used for this problem are: Q (flow); P (pressure); T (temperature); L (level); X (composition). Deviations of variables are defined for phases and components as well as for bulk values of the variables. For example: "1 Q LO" represents low bulk flow at node 1; "1 Q LIQUID LO" represents low liquid phase flow at node 1; "1 X LIQUID SYN-GAS-SOLN HI" represents high concentration of synthesis gas in solution at node 1. Deviatory flow events that may be defined at node 1 of the gas-liquid separation area are:

bulk flow deviations: Q NO, Q LO, Q HI, Q REV
liquid phase flow deviations: Q LIQUID NO, Q LIQUID LO, Q LIQUID HI, Q LIQUID REV
gas phase flow deviations: Q GAS NO, Q GAS LO, Q GAS HI, Q GAS REV.

All deviatory events must be created in this way, these form the basic building blocks of the plant event network. Events are not created at temporary nodes.

Events representing failure modes of all the units and lines in the section are created next. Such failure modes are unit-specific. Failure mode events created for the unit C1 are: EXT-FIRE (external fire); LEAK-TO-ATMOS (leak to the atmosphere, only created if the operating pressure is greater than 1 atmosphere); BURST (total rupture).

A library model of input-output and failure mode equations for the gas-liquid separation unit, C1, is used to build links in the plant event network, but first these equations must be modified to include line and minor process unit failure modes. This step is that of model expansion, as

described in section 6.5. As an example of model expansion consider a general input-output equation for a gas-liquid separation unit:

INTERNAL L LIQUID NO = BURST + BOTTOMS Q LIQUID HI
+ FEED Q LIQUID NO ...(8.2)

The node INTERNAL is equivalent to node 7, FEED is equivalent to node 10, and BOTTOMS is equivalent to node 12. The temporary nodes, 10 and 12, are replaced by nodes 1 and 2 respectively by applying the right-hand-side-node-replacement operator, adding failure modes for lines L1 and L2. The resulting equation for no internal liquid level is:

7 L LIQUID NO = C1 BURST
+ 2 Q LIQUID HI + L2 LEAK-TO-ATMOS
+ 1 Q LIQUID NO + L1 BLOCKED ...(8.3)

Links which are fundamental to the plant event network should also be created. An example of such a link is shown by equation 8.4.

2 Q LIQUID NO = 2 Q NO ...(8.4)

Event boundary conditions must next be added. Boundaries are created by the use of a look-up table of complementary deviations, shown in Table. 9.2. For example "2 Q LO" is a boundary of "2 Q HI" because LO and HI are complementary deviations.

This completes the basic plant event network, although this will be modified later. This basic plant event network represents the causal failure logic of the processing system, ignoring the effects of control loops and protective systems.

8.3.3: Definition of Loops

Having created a plant event network to represent plant process failure chains it is necessary to consider the effects of control loops and protective systems. A precursor to this is loop definition.

For each loop in the plant the following information is required: loop name, class, type, controlled variable, controlling variable, measured variables, controlling action, measured action, activation order, saturating order, and the units which make up the loop. This information is required for each low or high deviation of the controlled variable. For the loop LCL1 in the gas-liquid separation section, the following information is created for both LO and HI deviations of the controlled variable:

class: control loop;
type: direct action, ie. the controlled variable is also the measured variable;
controlled variable: 7 L LIQUID;
controlling variable: 2 Q LIQUID;
measured variable: 7 L LIQUID;
controlling action: negative, ie. an increase in the controlling variable tends to decrease the controlled variable;
measured action: positive (by default because the loop type is direct);
activation order: 0;
saturating order: 1;
control units: LT1 (a level transducer);
LIC1 (a level indicating controller);
LCV1 (a level control valve);

With this information it is possible to model loop actions and failure modes.

The unit failure modes which cause certain loop failure modes must be calculated and linked into the plant event network. For example, for the control loop LCL1, the causes of the control loop failure modes FAILED-HI, FAILED-LO and STUCK, are given by:

LCL1 FAILED-HI = LT1 FAILED-LO + LIC1 FAILED-HI + LCV1 FAIL-CLOSED ...(8.5)

LCL1 FAILED-LO = LT1 FAILED-HI + LIC1 FAILED-LO + LCV1 FAIL-OPEN ...(8.6)

LCL1 STUCK = LT1 STUCK + LIC1 STUCK + LCV1 STUCK ...(8.7)

The effects of control loop failure on the controlled variable should be modelled, and incorporated into the plant event network. Activation failure of a loop produces an input to deviations of the controlling variable, thus LCL1 FAILED-HI is a possible cause of 2 Q LIQUID LO; LCL1 FAILED-LO is a possible cause of 2 Q LIQUID HI. The effects of loop STUCK are not considered until the final fault tree synthesis stage.

Next, the effects of control loop actions on the controlling variable should be incorporated into the plant event network. In the above example, correct control loop activity is modelled by creating inputs to deviations of the controlled variable thus:

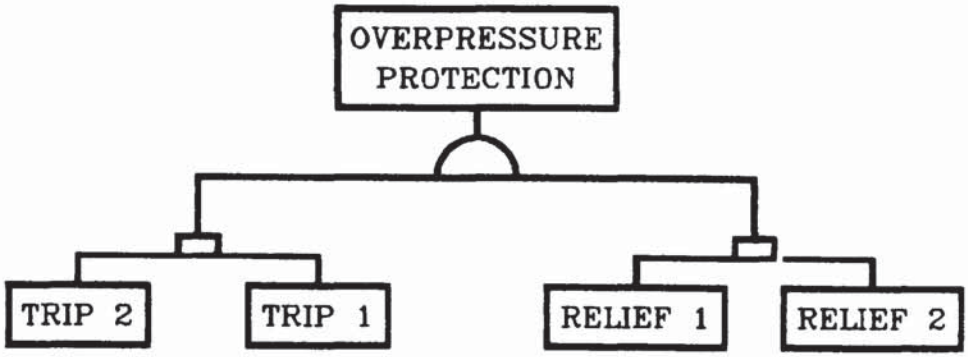
2 Q LIQUID NO = 7 L LIQUID LO order 1 ...(8.8)
2 Q LIQUID LO = 7 L LIQUID LO order 0 ...(8.9)
2 Q LIQUID HI = 7 L LIQUID HI order 0 ...(8.10)

Boundary conditions corresponding to loop failure modes must be created and associated with the gates produced by these equations. For equations 8.8 and 8.9 the boundary conditions are LCL1 STUCK and LCL1 FAILED-HI; for equation 8.10 the boundary conditions are LCL1 STUCK and LCL1 FAILED-LO.

The next stage in the modelling process is to define the protection logic for controlled events. Protection logic, which describes the combinations of correctly functioning loops that must be achieved in order to protect the controlled variable, is defined as described in section 7.2.2. In the example under consideration, there is only one loop protecting the controlled variable, so this loop forms the only input to the protection mini-trees for first-order deviations of the controlled variable 7 L LIQUID.

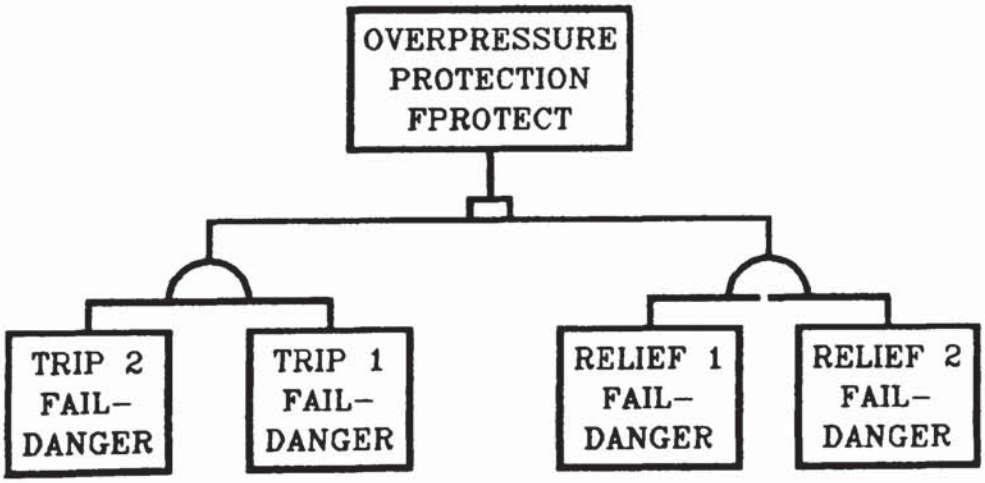
The generation of more complex protection mini-trees is illustrated with reference to the plant shown in Fig. 7.1. The protection logic against overpressure is that one of the relief valves lifts and one of the trip loops operates correctly. The protection mini-tree created to model this is shown in Fig. 8.6. Intermediate "groups" are used to add structure to the tree.

Fig. 8.6: Protection Mini-tree for the Plant Uprating Scheme



This process is repeated for each order of each LO or HI deviation of each controlled variable, and for each order of each measured event protected against. Protection mini-trees are then converted into protection failure mini-fault trees by replacing protection OR, AND and OUT-OF gates with protection failure OR and AND gates which combine loop invariant failure modes to define the protection failure logic for the controlled event. For the gas-liquid separation area example, the single event LCL1 STUCK represents the protection failure logic for first-order deviations of the controlled variable. The protection failure mini-fault tree for the plant uprating scheme obtained from the protection mini-tree shown in Fig. 8.6 is shown by Fig. 8.7.

Fig. 8.7: Protection Failure Mini-fault Tree for the Plant Uprating Scheme



8.3.4: Entering the Causal and Limiting Orders of Event Chains

The next stage of the method is to use the plant event network to create local failure event chains for LO and HI deviations of controlled and measured variables. Failure event chains are single branches of fault trees, including any combinatory events, which terminate at primary or ordered events, as described in section 7.2.3. The highest order of the top event of the failure event chain that the base event will cause, assuming that the plant's control systems are functioning correctly, should be entered. This is the causal order. Similarly, the limiting order, which represents the highest order of the top event that may result if the plant's control systems suffer invariant failure, of the base event should be recorded. If the base event is an ordered event, that is it is a deviation of a controlled or measured variable, then causal and limiting orders should be defined for each order of the base event.

In the gas-liquid separation area example, an event chain for the controlled event 7 L LIQUID LO is represented by equation 8.11:

7 Q LIQUID LO = 2 Q LIQUID HI = LCL1 FAILED-LO ...(8.11)

This chain has a causal order of 1, and a limiting order of 1.

8.3.5: Defining NO and FULL Deviations of Controlled Events

The deviations NO and FULL are considered, in protection terms, to be equivalent to the highest possible order of the corresponding LO and HI events respectively. A single input, corresponding to the highest ordered LO or HI deviation, to NO or HI deviations of controlled variables is created, all other inputs are removed. Thus, in the example under analysis, the only input links allowed for the deviations NO and FULL of the controlled variable are those of equations 8.12 and 8.13.

7 Q LIQUID NO = 7 Q LIQUID LO order 1 ...(8.12)
7 Q LIQUID FULL = 7 Q LIQUID HI order 1 ...(8.13)

8.3.6: Fault Tree Synthesis

The plant event network, causal and limiting orders of event chains, and protection failure mini-fault trees produced in the manner described above provide the necessary information for fault tree synthesis.

First, a top event should be selected. If this is not an ordered event, then fault tree propagation should be based on the plant event network until all base events are primaries or ordered events.

During fault tree generation, an above-event register is maintained for each event in the tree. This register consists of direct and indirect above-events as described in section 8.2.6. The indirect above-event register is used for boundary checking. The boundary conditions associated with each above-event are checked against the tree event to which the register applies, if the event matches one of these boundaries, it is declared invalid. Invalid events are propagated to identify other invalid events: if they form a combinatory input to an AND gate then each of the inputs to this gate are also invalid; any non-primary event in the tree which has no valid input events is invalid; any event in the tree descended from an invalid event is invalid. When the fault tree is complete, invalid events are propagated in this way and then removed.

Two types of input to controlled events are generated. Event chains which have a causal order equal to the order of the controlled event form direct inputs. For example, the event chain described by equation 8.11 forms an input to the first-order deviation of the controlled event 7 L LIQUID LO. This is represented by equation 8.14:

7 L LIQUID LO order 1 = 2 Q LIQUID HI = LCL1 FAILED-LO ...(8.14)

A second type of input represents loop protection failure. A combinatory input is created, linking the protection failure mini-fault tree for the order of the controlled event under consideration and inputs which cause the next lower order of the controlled event. This input is represented by equation 8.15:

CONTROLLED EVENT order n = (order n FAIL-TO-PROTECT mini-fault tree)
* (CONTROLLED EVENT order n-1)

...(8.15)

For the event 7 L LIQUID LO order 1 this would be:

7 L LIQUID LO order 1 = LCL1 STUCK * 7 L LIQUID LO order 0

...(8.16)

Inputs to the lower ordered event would then be created in the same way, until the zeroth order when direct inputs may be created.

During propagation of controlled events by these two methods it is necessary to check that no input event chain exceeds its limiting order. This is achieved by checking the direct above-event register. If an event which appears in a base event's above-event register is of higher order than the base event's limiting order for that deviation, then the event chain corresponding to the base event should be declared invalid, and removed from the fault tree.

Where an unprotected order of a controlled event appears in the fault tree, then a direct input from the next lower order may be created. This may arise where a controlled variable is also measured by an indirect loop if extra orders are created to describe the inputs to the indirect loops.

When a measured event appears in a fault tree, its direct above-event register is checked to see if the controlled event of an order which is being protected against the particular order of the measured event appears above this event in the fault tree. If this is the case, then the protection failure mini-fault tree is combined by an AND gate with the measured event.

During fault tree synthesis, the direct above-event registers of events are checked to see if the same event appears higher up in the fault tree. If it does, then it is declared invalid to prevent loops from occurring.

Once all invalid events have been removed from the fault tree, the tree is complete. The fault tree for the event 7 L LIQUID LO order 1 is shown in Fig. 8.8.

8.4: Discussion

This chapter describes the way in which the plant reduction and event ordering methods are combined into one hybrid method. The hybrid method provides a powerful approach to fault tree synthesis; it is capable of producing relatively small, well structured fault trees for a wide range of plants and plant control strategies.

This hybrid method has been implemented as an expert system in order to generate fault trees for the ammonia let down plant described in chapter 5, and problems which involve variations on this plant's control strategy. The program which achieves this is described in the next chapter.

Fig. 8.8: Fault Tree for Low Liquid Level in the Gas-liquid Separator

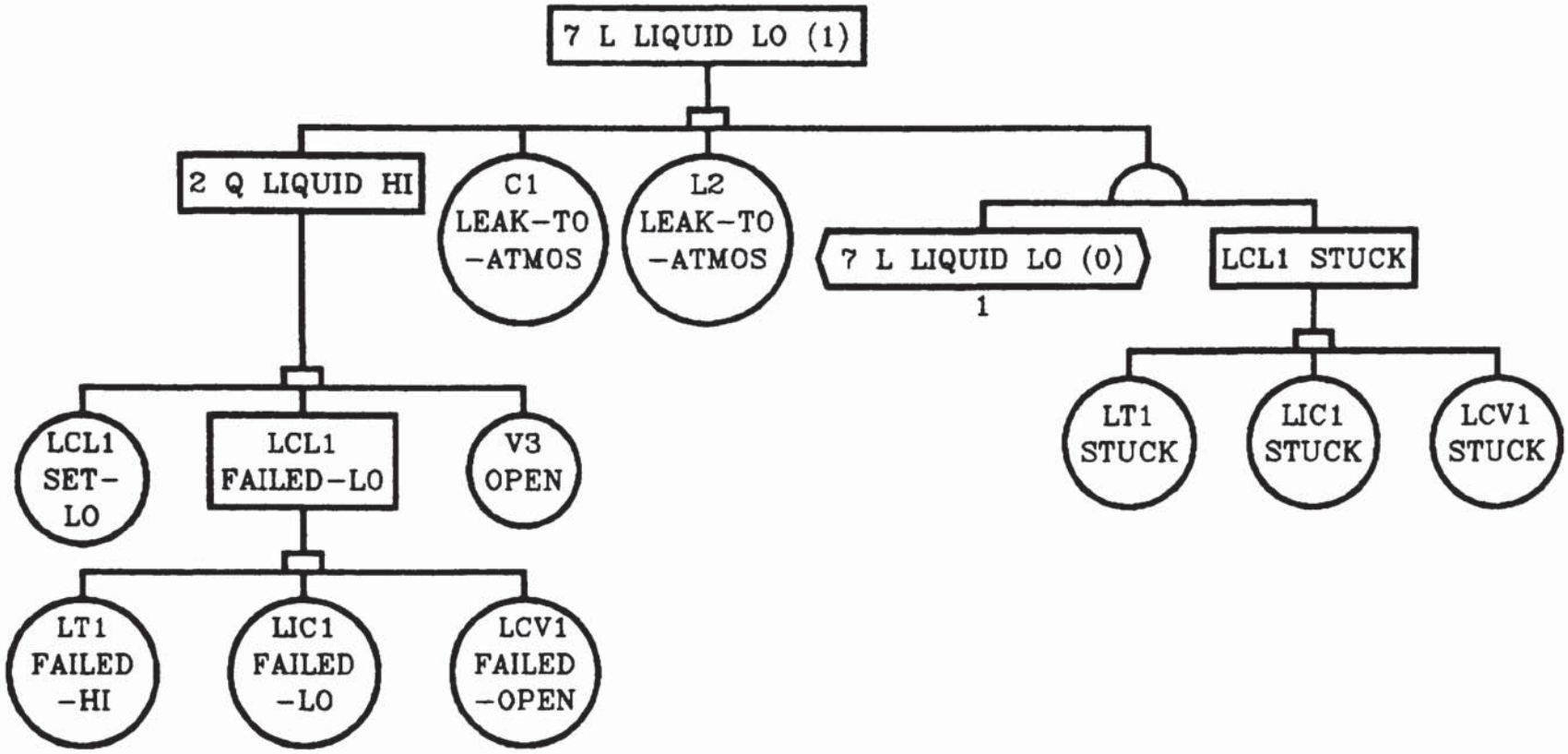
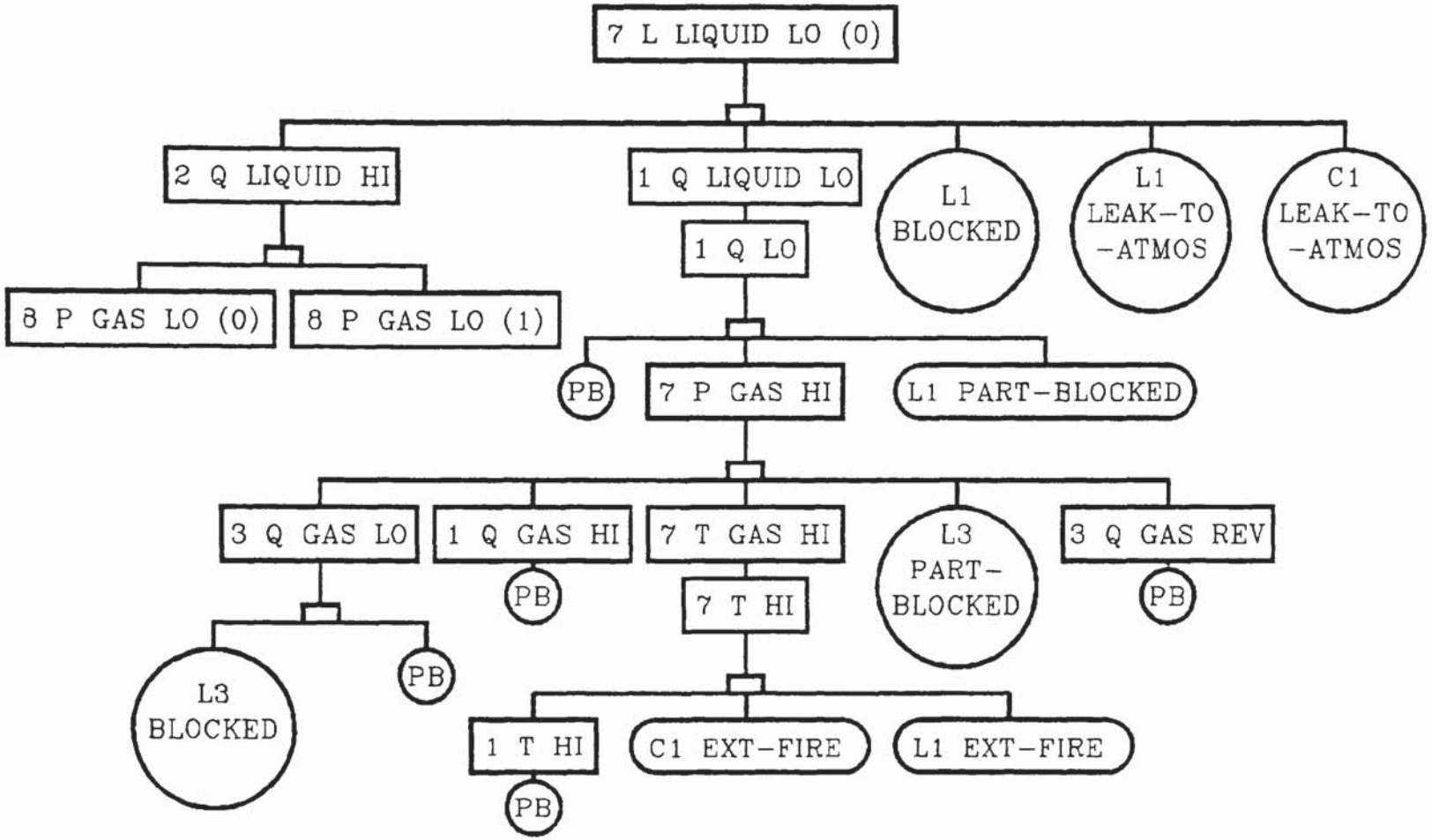


Fig. 8.8a: Continuation 1 of Fault Tree from Fig. 8.8



PB: "Primary" deviation from across the plant boundary

CHAPTER NINE

9. The OFTS Program

The Ordered Fault Tree Synthesis (OFTS) program incorporates the methods of problem reduction and event ordering described in chapters 6 and 7 into a single hybrid method as described in chapter 8. This chapter describes the implementation of this program as an expert system using OPS5. OPS5 is described in Appendix A; details on how to use the OFTS program are given in Appendix D.

9.1: Representing Objects Using OPS5

Objects are represented in OPS5 as elements. The class name of the element determines the object, information about the object is stored as values of certain attributes that are defined for each element class. Attributes may be viewed as data fields, the data which fill these fields are the values. The first step in expert system building is to define the element classes and their attributes.

A number of factors must be considered when choosing which element classes and attributes to use. OPS5 actually assigns a field for every attribute name used in the program to every element, no matter whether or not a particular attribute is used by a certain element. It is thus desirable, in terms of memory usage, to use as few different attribute names as possible. It is also desirable that attribute names be descriptive.

Another important factor is whether to use bi-directional or unidirectional representation of relationships. Consider the relationship A causes B. Unidirectional representation of this relationship would use a single element, A, with the attribute CAUSES which would take the value B. If this relationship were represented bi-directional, the element, B, would also be used to store this relationship, by use of an attribute CAUSED-BY, which would contain the value A. Bi-directional storage of relationship takes more memory, but in some circumstances is more efficient in terms of searching and rule matching.

A third consideration is the generality of elements. It is sometimes better to use a fairly general element class and to specify it by using attributes, such as TYPE or CLASS. This approach allows class characteristics to be defined which may be applied to all elements of a class. The drawback is that more elements are used, thus increasing memory usage.

9.1.1: Representing Chemical Process Plant

Process units are represented by separate classes for major and minor units, to avoid the use of a type-attribute. This representation is used because there are no rules that could be applied to both major and minor units. Major process units are represented by the MAJOR-UNIT element which takes the following attributes:

NAME, the name of the unit;
ISA, the type of unit;
PRESSURE, in atmospheres.

The ISA attribute should match one of the unit types for which input-output and failure mode equations are stored as production rules. This allows the equations for this type of unit to be applied to the unit in question. The PRESSURE attribute is used to determine which of the failure modes LEAK-TO-ATMOS or LEAK-FROM-ATMOS are applicable. If the PRESSURE is greater than one atmosphere then LEAK-TO-ATMOS is a viable failure mode, else if the PRESSURE is less than one atmosphere then LEAK-FROM-ATMOS is allowed.

Minor process units are represented by the element MINOR-UNIT, for which the following attributes are defined:

NAME, the unit name;
ISA, the unit type;
LINE, the name of the line in which the unit is situated;
STATUS, which defines the operating status, if any, of the unit.

The ISA attribute allows failure modes for a particular type of unit to be applied to specific units. The STATUS attribute allows more than one operating mode to be defined for a type of unit, the STATUS defines which of these modes the unit should be in for the plant operating state under investigation. For example, two operating states are defined for gate valves: OPEN and CLOSED. The STATUS attribute allows any particular gate valve to be defined as OPEN or CLOSED.

Control units are represented by the MINOR-UNIT element, but an additional attribute, LOOP, is defined instead of LINE. No further attributes are required to define the difference between minor process units and control units. The LOOP attribute holds the name of the loop to which the unit belongs. If the unit forms part of several loops then an element is created for each loop.

Lines are represented by the LINE element, which takes the following attributes:

NAME;
IS, either OPEN or CLOSED;
PRESSURE, the operating pressure in atmospheres.

PRESSURE is used, as for major units, to determine the validity of the failure modes LEAK-TO-ATMOS and LEAK-FROM-ATMOS.

Nodes are represented by the NODE element, for which the following attributes are defined:

NUMBER, the number by which the node is referred;
IS, either TEMPORARY, INTERNAL or NIL;
FLOW-TO, the number of any node to which flow occurs from the node in question;
FLOW-FROM, the number of any node from which flow occurs to the node in question;
FLOW-VIA, the name of the line through which the flow defined by either the FLOW-TO or FLOW-FROM occurs;
NODE-OF, the name of a major process unit to which the node

belongs;
COMPONENT, the name of a component defined at the node under the plant operating conditions in question;
MPU-DETAILS, where the node is a NODE-OF a major process unit this attribute defines which characteristic node of the major unit this node represents.

FLOW-TO, FLOW-FROM and FLOW-VIA define the plant topology. Here, bi-directional representation of flow is used; this improves the efficiency of event generation rules, reducing the number of NODE elements required to match left-hand-side rule conditions. NODE-OF and MPU-DETAILS are necessary to link nodes in with process unit model libraries.

Control loops, trip loops and pressure relief systems are represented by the LOOP element, for which the following attribute fields are used:

NAME, the loop name;
CLASS, may be CONTROL, TRIP or RELIEF;
TYPE, may be DIRECT or INDIRECT;
CONT-ACTION, the controlling action, which may be POSITIVE or NEGATIVE.

Loops are all represented by the same element, with the CLASS attribute subdividing them into control or trip loops, or pressure relief systems. It is necessary to use this general loop element, as these three loop classes are handled in much the same way; many rules apply equally to each class. The TYPE attribute is used to sub-classify loops depending on whether they have separate measured and controlled variables, in which case they are termed INDIRECT, otherwise they are termed DIRECT.

Controlled events are represented by the CONTAINED-EV element, which has the following attributes:

NUMBER, the number of the controlled event;
BY-LOOP, the name of any loop by which the event is controlled;
ORDER, the controlled order of the loop.

Separate elements are defined for each loop which controls this event. Each loop may have more than one controlled event, representing different deviations of the controlled variable.

Measured events are represented by the MEASURED-EV element, with the following attributes:

NUMBER, the number of the measured event;
BY-LOOP, the loop name for which the event is measured;
MEAS-ACTION, the measured action;
INITIATING-ORDER, the initiating order of the measured event for the loop;
SATURATING-ORDER, the order of the measured event which will saturate the loop;
CED-EVENT, the controlled event for which this measured event applies.

Measured event elements are required for each event measured by the loop.

Controlling events are represented by the CONTAINING-EV element, which takes the following attributes:

NUMBER, the event number;
BY-LOOP, the loop name;
CED-EVENT, the controlled event for which this controlling event applies.

More than one controlling event may be created per loop, corresponding to different deviations of the controlling variable, but only one controlling variable may be defined for each loop.

9.1.2: Representing the Plant Event Network

Events in the plant event network are represented in OPS5 using the EVENT element, this takes the following attributes:

NUMBER, the number by which the event is referred throughout the system;
NODE, the node at which the deviation occurs;
VAR, the variable which has deviated from its normal value;
PHASE, the phase to which the deviation applies;
COMPONENT, the component to which the deviation applies;
DEV, the deviation;
FROM, value is PLANT-BOUNDARY if the deviation enters from across the boundary of the plant section being analysed;
UNIT, the unit, major, minor, or control, in which the failure occurs;
FAILMODE, the mode of failure of the unit.

Gates are represented by the GATE element, which takes the attributes:

NUMBER, the number by which the gate is referred to by the system;
ISA, either OR or AND;
CAUSED-BY, the number of an input event to the gate;
CONSEQ-OF, the number of the output event of the gate;
IN-ORDER, the order of the input event;
OUT-ORDER, the order of the output event.

A separate GATE element is required for each input to the gate. It can be seen that this representation is unidirectional. Gates point to both input and output events, but there is no reverse representation from event to gate.

Boundary conditions are pointed to by the BOUNDARY element. This has three attributes:

GATE, the number of the gate with which the boundary is associated;
EVENT, the number of the event with which the boundary is associated;
BOUNDARY, the number of the boundary event.

This element defines boundaries associated with both events and gates.

9.1.3: Representing Protection Mini-trees

Protective groups and loops are used to create protection mini-trees as described in section 7.2.2. Groups are intermediate nodes in these mini-trees, represented by the GROUP element, and the following attributes:

NAME, the group name;
PROTECTED-EV, the number of the event being protected;
PROTECTED-ORDER, the order of the event being protected;
MEASURED-EV, the measured event being protected against;
MEASURED-ORDER, the order of the measured event being protected against.

The attributes PROTECTED-EV, PROTECTED-ORDER, MEASURED-EV, and MEASURED-ORDER need only be defined for the top group in the mini-tree. Groups may appear in the mini-trees for several events. Once a group's input logic has been defined for one mini-tree it remains unchanged in other mini-trees.

The other building blocks of protection mini-trees are logic gates, represented by the GROUP-LINK element, with the following attributes:

GROUP, the output group;
LOGIC, either OR, AND or OUT-OF;
N-LOOPS, the number of input loops or groups which must function correctly to provide protection;
M-LOOPS, the total number of input loops and groups;
INPUT-GROUP, the name of an input loop or group.

The N-LOOPS and M-LOOPS elements are only defined if the input logic is OUT-OF. This tree linking is, like that used in the plant event network, unidirectional.

9.1.4: Representing Causal Order Links

Routed links are represented by the LINK element, which uses the

Routed links are represented by the LINK element, which uses the following attributes:

GATE, refers to intermediate gates in the event branch, and points to the appropriate gate in the plant event net;
ISA, refers to the gate type, either AND or OR;
NUMBER, the link number;
EVENT, is the top event of the event branch;
CAUSE, is the base event of the branch;
IN-ORDER, refers to the order of the base event;
CAUSAL-ORDER, the causal order of the base event in terms of the top event;
LIMITING-ORDER, the limiting order of the base event in terms of the top event;
INTERMEDIATE, the number of an intermediate event in the branch;
MAX-ORDER, the highest allowed order of the top event.

Several elements will be required to represent links containing intermediate events. Intermediate events include any combinatory events associated with the branch. The MAX-ORDER element defines limiting values for the causal and limiting orders when they are being defined by the user.

9.1.5: Representing Fault Trees

Fault tree representation is achieved simply by one element, TREE, which points to events in the plant event network and describes the logic gates required for their combination. The following attributes are used:

IS, describes the type of tree event, either TOP, PRIMARY, INVALID or NIL;
NODE, a numbered node in the tree;
GATE, a number which identifies the gate to which the event forms an input;
ISA, refers to the gate, either OR or AND;
EVENT, the event number in the plant event network;
ORDER, the order of the event;

OUTPUT, the node directly above in the fault tree;
CONSEQ-NODE, refers to the node which represents the top event of the event branch which was used to create this node.

It can be seen that TREE elements create a unidirectional representation which does not require separate gate elements. Fault trees are represented in terms of numbered nodes; these point to events in the plant event net.

The above event register is stored by the ABOVE-EVENT element, with the following attributes:

FOR-NODE, the node for which the register applies;
IS, either DIRECT or INDIRECT;
EVENT, the number of the above event;
ORDER, the order of the above event;
NODE, the node of the above event.

Boundary conditions are implicitly represented: above events point to their boundaries in the plant event net. Boundary conditions associated with gates in the plant event net are handled differently, however. These are associated with specific nodes in the fault tree, these nodes being the input nodes to the particular gate for which the boundaries apply. These are represented by the TREE-BOUNDARY element, which takes the following attributes:

NODE, the tree node with which the boundary is associated;
EVENT, the boundary event;
ORDER, the boundary order.

9.2: Control Strategies for Large OPS5 Programs

When writing anything other than very small expert systems it is important to carefully define strategies for controlling program execution, even if there are no procedural elements to the system. There are several control strategies commonly used in expert system design. Methods to implement these are available in OPS5.

When a system can be separated into several distinct parts, a method which improves program efficiency and aids program development and debugging is **partitioning of the rule-base.** This is achieved by splitting the rule-base into distinct subsets, such that only the rules of any one subset may execute at any given time. Once the goals of a subset are achieved, control is passed to the next subset. Rule-base partitioning is accomplished in OPS5 programs by adding a control element to the condition part of rules. The control element indicates to which subset the rules belong. A rule may only be instantiated when its control element condition is matched by the control element in working memory; only one control element exists in working memory at any time. Partitioning in this manner improves program efficiency by reducing the number of instantiations that need be sorted at each recognize-act cycle. Program building and debugging is also made easier as programs may be composed as a number of relatively small sub-systems.

A technique often combined with rule-base partitioning is the use of **demons.** Demons are rules that execute when certain conditions are achieved, whatever rule subset is currently instantiated. Demons are usually used only in complex systems, for which they are useful as a means of error-checking the database. In OPS5, demons may be created by writing rules without conditional control elements. It is necessary to use the means ends analysis conflict resolution strategy. Normal rules include control elements as their first condition element; data produced in a rule subset will normally be of higher recency than control elements, so a demon which contains a data element as its first condition element will execute preferentially over normal subset rules. Demons are not widely used in the OFTS program, although there are some rules, implemented as demons, which check the validity of user-input information. Demons are also used in OFTS to allow the user to respond "SAVE" or "STOP" at input prompts. If "SAVE" is entered then a demon executes to save the current state of working memory and the conflict set. If "STOP" is entered then a demon which halts program execution will fire.

Goal-directed reasoning is a technique which classifies rules by the goals they are attempting to achieve. Goals may be defined as working memory elements, and some rules may contain condition elements that

memory elements, and some rules may contain condition elements that require the existence of a goal, or set of goals, for them to be matched. This ensures that these rules are instantiated only if these goals are required. Goal-directed reasoning is somewhat similar to partitioning of the rule-base, although it is more flexible because goals may be created only if and when they are required; rule-base partitioning usually requires that all subsets are instantiated at some point.

Procedural programming may be accomplished in several ways in OPS5. One method is to add as a condition of one rule a pattern may only be created by the previous rule in the required sequence. This conditional pattern is usually the existence of a particular working memory element, or the existence of a certain value of an attribute. A more flexible method of defining procedures is to use agendas. An agenda contains a list of tasks to be performed. When a task is completed another is selected from the agenda. In OPS5, agendas may be stored by vector attributes. The next value in a list may be pulled from the agenda by specific control rules, the value then being transferred to a control element.

The OFTS program uses two agendas and dual tasks to implement rule-base partitioning and, where required, procedural structures. The control element used for both of these functions is the TASK element, which has two attributes:

MAJOR, defines the active rule subset;
MINOR, defines the active rule, or set of rules, in a procedure.

All rules, except demons, contain TASK as their first condition element. The value of the MAJOR attribute defines to which subset the rule belongs. A list of subsets is produced in the STARTUP statement. This list is stored in an agenda, with the following attributes:

TYPE, either MAJOR or MINOR;
STACK, a vector attribute which contains a list of tasks.

The MAJOR agenda is used for rule-base partitioning, the MINOR agenda is

used for procedures. MINOR agendas are produced by rules in specific subsets at the start of a procedure.

Two control rules, PULL-MAJOR-TASK and PULL-MINOR-TASK, pull values from these agendas to update tasks. The rules for task pulls are:

1. Control rules will only execute when no other rules are instantiated. This ensures that tasks are complete before they are halted. This is achieved by using a very low recency element as the first condition element on the left-hand-side of control rules.
2. Minor task pulls execute preferentially over major task pulls. This ensures that a procedure is complete before a new subset is activated.

The means ends analysis conflict resolution strategy is used to enable this control strategy to be implemented.

An overview of the rule subsets used by the OFTS program is shown in Table 9.1.

9.3: Problem Definition

Initial problem definition prior to running the OFTS program requires the definition of NODE, LINE, MINOR-UNIT, MAJOR-UNIT and COMPONENT elements. This data effectively defines the plant and its connections, except for control systems. These elements may be created in a STARTUP statement, by a small OPS5 program, or by hand. Problems are currently defined by a small OPS5 program, PROBLEM, which creates most of these elements in a STARTUP statement, but uses rules to define components at nodes and prompt user input of MPU-DETAILS attributes for nodes. A listing of the PROBLEM program appears in Appendix G.

9.4: Event Generation

The event-generation rule subset, matched by the task EVENT-GENERATION, creates events that are to be used in the plant event network and fault trees. Two types of event are produced: deviatory

Table 9.1: Rule Subsets in the OFTS System

RULE SUBSET	FUNCTION
Control	Pulls tasks from agendas.
Event generation	Creates events to form the basis of the plant event network.
Event linkages	Creates gates which link events corresponding to the major process unit equations.
Properties	Creates gates corresponding to fundamental system properties.
Boundaries	Creates boundary conditions.
Loop specification	Creates control, trip and relief loops.
Minor units	Allows the user to define minor process units and control units. Creates failure modes and gates representing their consequences for minor process units.
Loop failure modes	Creates gates representing the unit causes of loop failure modes.
Protective combinations	Allows the user to create protection mini-trees.
No-full linkages	Creates gates between NO and FULL deviations and ordered LO and HI deviations respectively.
Controlling events	Creates additional causes of deviations of controlling variables
Event ordering	Generates mini-fault trees and allows the user to define the causal and limiting orders of individual event branches.
Group failure	Converts protection mini-trees into protection failure mini-fault trees.
Synthesis	Generates fault trees.

events of variables defined at nodes; failure modes of major and minor process units.

The exact deviatory events generated at a node depend on the type of node, whether it is internal to a major process unit, or is a line node, and the existence of flow, phases and components at that node. If a component will normally exist at a node, for example, the deviations "X NO", "X LO" and "X HI" are created at that node for that component. If the component does not normally exist at that node, then the deviation "X SOME" is created for the component. This process is carried out for the variables Q, P, L, T, and X; for the required phases, in this case LIQUID and GAS; and for the required components, these being all the components that exist within a problem.

Unit failure mode rules are unit-specific. Units are represented by their unit type, details of the failure modes that apply to specific types of unit are stored as rules; these are applied to the specific units defined by the problem. Line failure mode rules are used to generate failure modes for specific lines used in the problem. Each unit and line requires the input of its internal pressure by the user. A pair of rules, INQUIRE-UNIT-PRESSURE and INQUIRE-LINE-PRESSURE, prompt the user to enter this information for each line and unit in the problem. The pressure for each line or unit is compared to atmospheric pressure to determine whether "LEAK-TO-ATMOS" or "LEAK-FROM-ATMOS" is created as a failure mode.

A rule, INQUIRE-ABOUT-PLANT-BOUNDARIES, asks the user for the numbers of nodes that exist at the boundary of the plant area covered by the problem. Any events that may arise from across the plant boundary are marked as ^FROM PLANT-BOUNDARY by the rule DEFINE-PLANT-BOUNDARY-EVENTS.

9.5: Linking Events: the use of Process Unit Models

Process unit models are used to link events by AND and OR logic gates. These models are represented by rules in the event-linkages rule subset. Rules are unit-specific, they are true for a certain unit type and

are applied to specific major units that correspond to the appropriate type of unit. Rules for gas-liquid separators and flash vessels, corresponding to the units in the ammonia let-down system have been generated. The process unit input-output and failure mode equations upon which these rules are based are shown in Tables C.1 and C.2. Line failure modes, as shown in Tables 6.1 and 6.2, are included in rules which represent these models; "node replacement" is implicit within these rules.

A typical rule in this subset is GAS-LIQUID-SEPARATOR-TOPS-Q-LIQUID-SOME, shown in Fig. 9.1. A temporary node, representing the "TOPS" node of a gas-liquid separator is replaced by the node to which it is connected by the line bound to <LINE>. The event <C1>, representing "FULL" internal liquid level of the unit, is linked as a cause of "SOME" liquid flow at the replaced "TOPS" node. No other causes, either deviatory, line or unit failure mode events are identified. A gate counter is used to produce a unique number for the "OR" gate produced.

Fig. 9.1: The Rule GAS-LIQUID-SEPARATOR-TOPS-Q-LIQUID-SOME

```
(P GAS-LIQUID-SEPARATOR-TOPS-Q-LIQUID-SOME
   (TASK ^MAJOR EVENT-LINKAGES)
   (NODE ^NUMBER <TOPS> ^IS TEMPORARY ^NODE-OF <UNIT>
^MPU-DETAILS TOPS ^FLOW-TO <TARGET> ^FLOW-VIA <LINE>)
   (MAJOR-UNIT ^NAME <UNIT> ^ISA GAS-LIQUID-SEPARATOR)
   (EVENT ^NUMBER <CONSEQ> ^NODE <TARGET> ^VAR Q ^PHASE LIQUID
^COMPONENT NIL ^DEV SOME)
   {(GATE-COUNTER ^COUNT <GATE>)<GC>}
   (NODE ^NUMBER <INTERNAL> ^NODE-OF <UNIT> ^MPU-DETAILS INTERNAL)
   (EVENT ^NUMBER <C1> ^NODE <INTERNAL> ^VAR L ^PHASE LIQUID ^DEV
FULL)
  -(GATE ^ISA OR ^CONSEQ-OF <CONSEQ> ^CAUSED-BY <C1>)
-->
   (MODIFY <GC> ^COUNT (COMPUTE <GATE> + 1))
   (MAKE GATE ^NUMBER <GATE> ^ISA OR ^CONSEQ-OF <CONSEQ>
^CAUSED-BY <C1>))
```

A rule in this subset, RENUMBER-OR-GATE, ensures that all "OR"

gate inputs to an event have the same gate number, unless a boundary is associated with a gate produced, in which case that gate retains its unique number.

An example of a rule which includes line failure modes is GAS-LIQUID-SEPARATOR-TOPS-Q-GAS-NO-HIGH-PRESSURE-LINE, shown in Fig. 9.2. This rule identifies three causes of "TOPS Q GAS HIGH": "INTERNAL P GAS LO", tops line "BLOCKED" and tops line "LEAK-TO-ATMOS". It is checked that this is a high pressure line, that is the line's internal pressure is greater than atmospheric, by the very fact that the line failure mode "LEAK-TO-ATMOS" exists; if this were a low pressure line, then this failure mode would not exist, "LEAK-FROM-ATMOS" would exist instead. A separate rule for a low pressure line, does not generate the failure mode "LEAK-TO-ATMOS". The temporary node, "TOPS", is again replaced by the node to which it is connected by a line.

There are some rules in the subset which simultaneously replace several temporary nodes. These rules are those that generate the causes of internal deviations, in terms of deviations at input, output and internal nodes. An example of this is the rule GAS-LIQUID-SEPARATOR-INTERNAL-P-GAS-HI-LOW-PRESSURE-LINE, which identifies eight causes of "INTERNAL P GAS HI" for gas-liquid separation units.

Other rules that comprise the event-linkages subset are those that add general property links. An example of these is PROERTIES-PHASE-DEV-TO-BULK. This rule creates bulk input deviations corresponding to equivalent phase output deviations. This is valid only for the variables flow, Q, and temperature, T.

9.6: Creating Boundary Conditions

Complementary boundary conditions, as shown in Table 9.2, are added during the event-linkages phase by ADD-COMPLEMENTARY-BOUNDARIES rules. These rules represent the complementary relationships, and generate boundary conditions for most deviatory events.

Fig. 9.2: The Rule

GAS-LIQUID-SEPARATOR-TOPS-Q-GAS-NO-HIGH-PRESSURE-LINE

```
(P GAS-LIQUID-SEPARATOR-TOPS-Q-GAS-NO-HIGH-PRESSURE-LINE
   (TASK ^MAJOR EVENT-LINKAGES)
   (NODE ^NUMBER <TOPS> ^IS TEMPORARY ^NODE-OF <UNIT>
^MPU-DETAILS TOPS ^FLOW-TO <TARGET> ^FLOW-VIA <LINE>)
   (MAJOR-UNIT ^NAME <UNIT> ^ISA GAS-LIQUID-SEPARATOR)
   (EVENT ^NUMBER <CONSEQ> ^NODE <TARGET> ^VAR Q ^PHASE GAS
^COMPONENT NIL ^DEV NO)
   {(GATE-COUNTER ^COUNT <GATE>)<GC>}
   (NODE ^NUMBER <INTERNAL> ^NODE-OF <UNIT> ^MPU-DETAILS INTERNAL)
   (EVENT ^NUMBER <C1> ^NODE <INTERNAL> ^VAR P ^PHASE GAS ^DEV LO)
   (EVENT ^NUMBER <C2> ^UNIT <LINE> ^FAILMODE BLOCKED)
   (EVENT ^NUMBER <C3> ^UNIT <LINE> ^FAILMODE LEAK-TO-ATMOS)
  -(GATE ^ISA OR ^CONSEQ-OF <CONSEQ> ^CAUSED-BY <C1>)
-->
   (MODIFY <GC> ^COUNT (COMPUTE <GATE> + 1))
   (MAKE GATE ^NUMBER <GATE> ^ISA OR ^CONSEQ-OF <CONSEQ>
^CAUSED-BY <C1>)
   (MAKE GATE ^NUMBER <GATE> ^ISA OR ^CONSEQ-OF <CONSEQ>
^CAUSED-BY <C2>)
   (MAKE GATE ^NUMBER <GATE> ^ISA OR ^CONSEQ-OF <CONSEQ>
^CAUSED-BY <C3>))
```

Additional boundary conditions may be associated with gates from process unit model equations. Where boundaries are associated with a gate, the gate is not renumbered if there is more than one input "OR" gate to the same event.

Additional boundary conditions may be identified. The OFTS system generates line BLOCKED and PART-BLOCKED as boundaries to HI flow deviations at nodes at either end of these lines.

The final rules in the boundaries subset test whether events that are defined at the plant boundary are valid deviations. If an event is not a valid deviation, as defined by the user, then it, and any gates associated with it, are removed from working memory.

Fig. 9.3: The Rule PROPERTIES-PHASE-DEV-TO-BULK

```
(P PROPERTIES-PHASE-DEV-TO-BULK
   (TASK ^MAJOR EVENT-LINKAGES)
   (EVENT ^NUMBER <EV> ^NODE <N> ^VAR { <V> << Q T >> } ^PHASE { <P>
<> NIL } ^COMPONENT NIL ^DEV <D>)
  -(GATE ^ISA OR ^CONSEQ-OF <EV>)
   (EVENT ^NUMBER <CAUSE> ^NODE <N> ^VAR <V> ^PHASE NIL
^COMPONENT NIL ^DEV <D>)
   {(GATE-COUNTER ^COUNT <COUNT>)<GC>}
-->
   (MODIFY <GC> ^COUNT (COMPUTE <COUNT> + 1))
   (MAKE GATE ^NUMBER (COMPUTE <COUNT> + 1) ^ISA OR ^CONSEQ-OF
<EV> ^CAUSED-BY <CAUSE>))
```

Table 9.2: Complementary Deviations

Deviation	Complementary Deviations
NO	HI, FULL
LO	HI, FULL
HI	NO, LO, REV
FULL	NO, LO
SOME	--
REV	HI

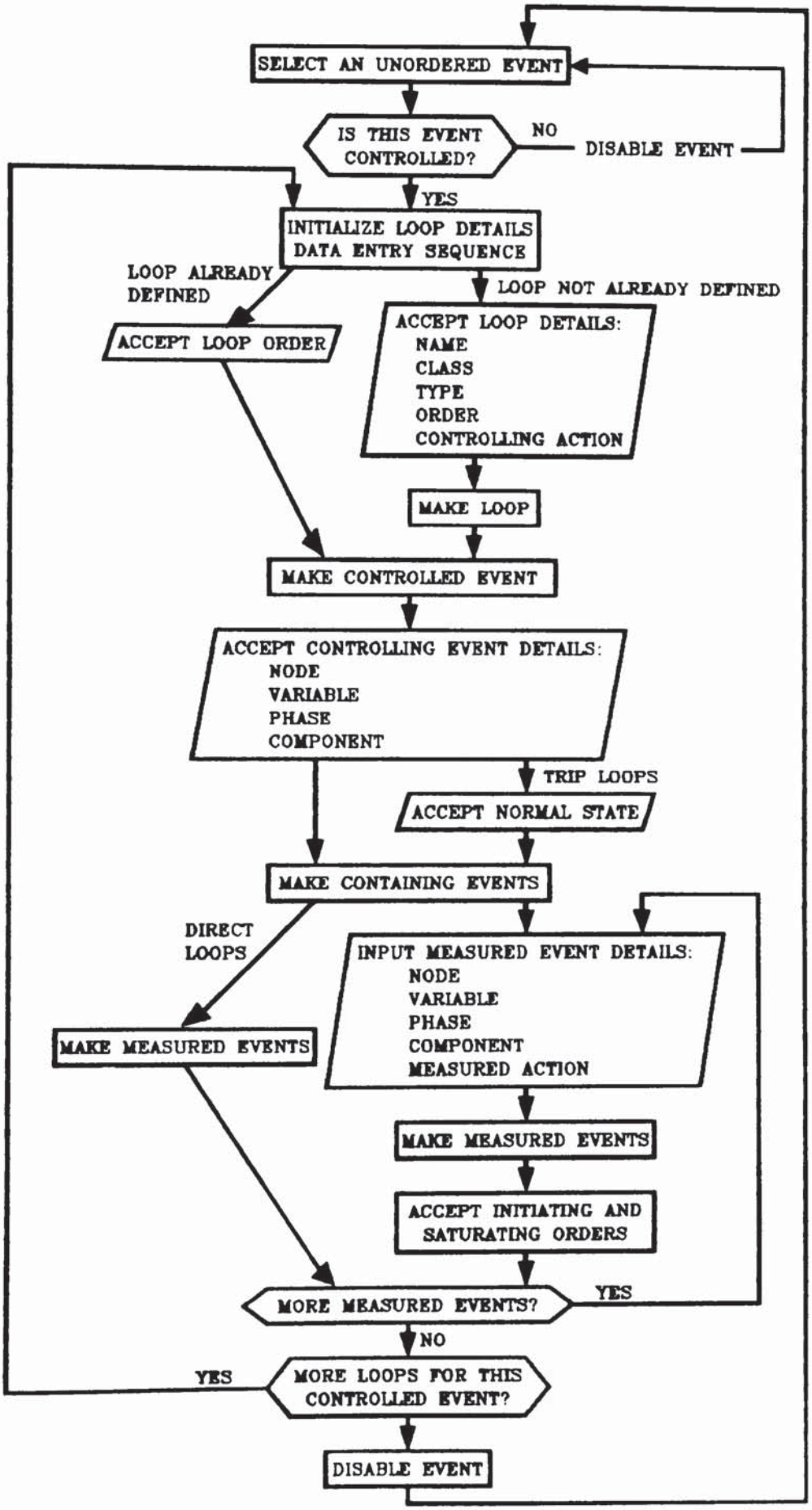
9.7: Definition of Loops

Loops are defined by the user, who is prompted for information by rules in the loop specification subset. This subset is organised procedurally, a flowsheet which represents this is shown in Fig. 9.4.

Two procedural sequences are used to obtain information from the user. One sequence asks for details about loops, controlling events and measured events. The second sequence accepts information about measured variables. The subset is arranged such that first an event is selected; only those events that are deviations at nodes which have previously been defined by the user as controlled nodes are chosen. If this event is controlled by one or more loops then the loop details data entry sequence will execute. This sequence allows the user to define all the loops that control this event. The information defined for each loop is: loop name, class, type, order, controlled action, and controlling variable details. Controlling events are generated from user input of the node, variable, phase and component of the controlling variable. Some details may be defined by default, where this is the case rules do this without need for user input. The LOOPSPEC-GENERATE-CONTAINING-EV-... rules select the deviations of the controlling variable used to create the controlling events; for a control loop, for example, the deviations LO and HI are used to generate controlling events.

A second procedural sequence prompts the user for measured events. This sequence only executes if the loop being defined is ^TYPE INDIRECT, direct loops have the same measured event and controlled event. This sequence accepts the following measured variable details: node, variable, phase, component and measured action. Measured events are generated for certain deviations of this variable, depending on the loop class, measured action, and the deviation of the controlled variable for which the loop is being defined. After the measured event has been created, the initiating and saturating orders should be entered. At the end of this sequence, the user is asked if there are any more measured variables for this loop. If the response is in the affirmative, then program control will loop back to the start of the sequence.

Fig. 9.4: Flowsheet of the Loop Specification Rule Subset



Once the measured events have been entered, the user is asked if there are any more loops controlling the selected event. If there are, then the program loops back to the start of the loop details input sequence, otherwise the selected event is deactivated, and another event is selected. Previously created loops are allowed to control a newly selected event; in this case, some of the inputs may be by-passed.

Control through this subset is maintained by agendas, which are created at event selection and at the start of the two data entry sequences.

9.8: Definition of Minor Units

Minor process units and control units are entered by the user in response to rules in the minor units subset. This subset, like the loop specification subset, uses procedural data entry sequences to prompt and accept user input. The flowsheet for this subset is shown in Fig. 9.5.

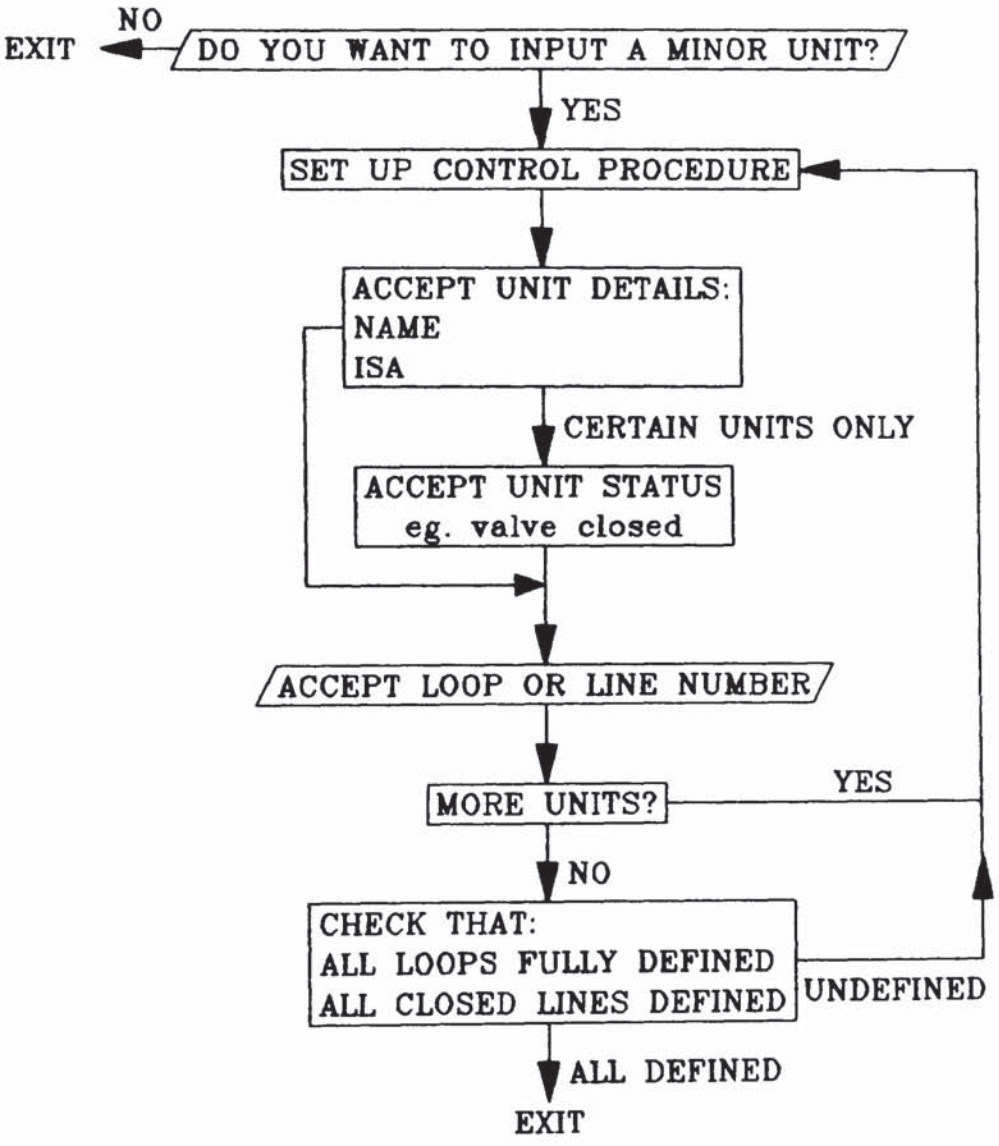
For each minor unit the following details should be entered: unit name; unit "isa" type; unit status, for certain unit types only; numbers of lines or loops that the unit forms part of.

A help rule, MINOR-UNITS-HELP-UNIT-TYPES, is available to help the user to enter the unit type. This rule prints a list of all the currently defined unit types. The help rule executes if the user responds "HELP" to the unit type prompt.

Only certain types of unit require their status to be defined. Examples defined by rules in the OFTS system at present are: gate valves, which may be "OPEN" or "CLOSED"; solenoid valves, which may be "VENT-TO-OPEN" or "VENT-TO-CLOSE". The status as entered defines the normal operating status of the unit in question for the particular problem under evaluation.

When the user ends minor units input, the system checks that all the units necessary to define loops, lines and bypasses have been entered. If further input is required, the system warns the user about the nature

Fig. 9.5: Flowsheet of the Minor Units Rule Subset



of the omission and returns control to the unit input entry sequence. Examples of units which are necessary to fully define certain loops, lines and bypasses in the context of the test problems to be evaluated are:

1. Control loops must have a transducer, a controller and a final control element.
2. Trip loops must have a transducer or switch, a solenoid valve and a final control element.
3. Closed lines or closed bypass lines must contain a closed valve.
4. Pressure relief systems must contain a pressure relief valve.

When all minor units have been defined, events corresponding to failure modes for each unit are created, and these are linked as causes to other events in the plant event network. Only process unit failure modes and links are created in this subset; control unit failure mode links are generated by the loop failure modes subset, described in the next section. The causal links created for gate valves in lines are shown in Table 6.3; those for bypass lines are shown in Table 9.3.

Table 9.3: Failure Modes and Their Consequences for Gate Valves in Bypass Lines

Bypass Is	Bypassed Line Is	Valve Status	Failure Mode	Output Event
Closed	--	Closed	OPEN	Q HI*
Open	Open	Open	CLOSED	Q LO
Open	Closed	Open	CLOSED	Q NO
Open	--	Open	PART-CLOSED	Q LO

* If greater than one closed valve isolates a bypass line, then the OPEN failure mode for each valve is combined under a single AND gate

Only failure mode rules for open or closed gate valves in lines or bypass lines are included in the minor units subset at present; this should be extended to include other minor process unit failure modes as other plants are analysed.

9.9: Loop Failure Modes

The loop failure modes subset consists of three rule types:

1. Rules that generate loop failure modes;
2. Rules that generate control unit failure modes;
3. Rules that link loop failure modes to their causal control unit failure modes.

All these rule types are instantiated by the same control element; there is no procedural aspect to this subset. The loop failure modes and their causes as represented by these rules are shown in Tables 6.4 and 6.5.

9.10: Definition of Protection Logic

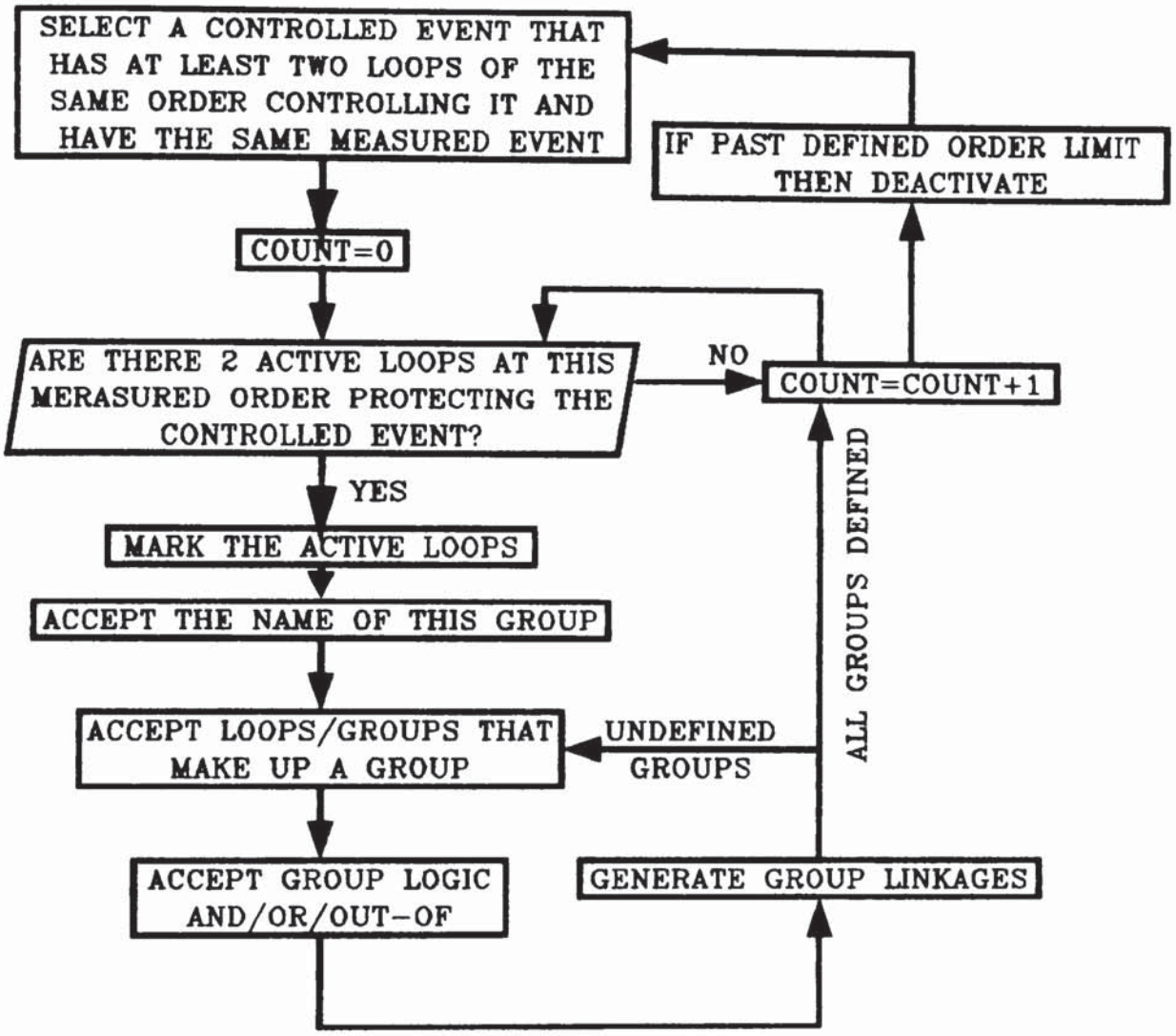
The logic that describes the protection of events by control and trip loops and pressure relief systems is defined by the user. The protective combinations rule subset facilitates input of this data, and generates protection mini-trees.

This subset does require a fairly rigid control structure; this is shown by Fig. 9.6. The system selects an event of a certain order, which is controlled by two or more activated, but not saturated, loops at a certain selected order of the measured event. The rules which achieve this selection, COMBINS-SELECT-CONTAINED-EVENT and COMBINS-SELECT-MEASURED-EVENT-ORDER, are general rules which may be applied to either direct or indirect loops. All of the loops that protect the selected order of the controlled event against the selected measured event are printed to aid the user, and made into base-level groups.

The user must enter the name of the top group which represents the mini-tree. If the entered group name matches that of a previously

created group, the system checks that this is intended, and if so then creates this group as the top group. The input mini-tree for a group is constant, whatever event is being protected against. It is therefore unnecessary to explore previously defined groups any further.

Fig. 9.6: Flowsheet of the Protective Combinations Rule Subset



For each undefined group in the mini-tree, the input groups and their protection logic must be entered. This process is repeated successively until all of the base groups have either previously been entered, or represent individual loops. A counting procedure is used to define the M-LOOPS attribute of GROUP-LINK when the input logic is OUT-OF, prior to user input of N-LOOPS.

When the mini-tree is complete, new measured or controlled events or orders are selected. This process continues until all protection for the plant which involves combinations of loops has been defined.

9.11: Deviations of Controlling Variables

The controlling event rule subset generates causes, over and above those created previously, of deviations of the controlling variable for loops where the controlling variable is different to the controlled variable.

Two cases where additional causes of controlling events may exist are identified:

1. Loop failure modes which affect the controlling variable; FAILED-HI and FAILED-LO for control loops, and FAIL-SAFE for trip loops.
2. Deviations of the controlled variable, together with correct action by the loop concerned.

These two failure mechanisms are represented by rules which create the appropriate links in the plant event network. Instead of defining an event, CORRECT-ACTION, which corresponds to loop normal condition, or using a NOT gate, boundary conditions corresponding to certain loop failure modes are associated with gates generated by rules representing case 2 above. Failure modes corresponding to case 1 are shown in Table 9.4; controlled variable deviations and boundary conditions representing case 2 are shown in Table 9.5.

9.12: NO and FULL Links

Causal links of controlled NO and FULL deviatory events created from process unit models are replaced by an input link from the highest order of the respective LO or HI deviation of the same variable by the no-full linkages subset. Inputs to NO and FULL deviations of variables which are not controlled are not affected.

Table 9.4: Loop Failure Modes and their Effect on Controlling Variables

Loop Class	Controlled Deviation	Controlled Action	Controlling Deviation Caused	Loop Failure Mode
Control	--	Positive	NO, LO	FAILED-LO, SET-LO
Control	--	Positive	HI	FAILED-HI, SET-HI
Control	--	Negative	NO, LO	FAILED-HI, SET-HI
Control	--	Negative	HI	FAILED-LO, SET-LO
Trip	LO	Positive	HI, SOME	FAIL-SAFE
Trip	LO	Negative	NO	FAIL-SAFE
Trip	HI, SOME	Positive	NO	FAIL-SAFE
Trip	HI, SOME	Negative	HI, SOME	FAIL-SAFE

Table 9.5: The Effects of Loop Correct Responses on Controlling Variables

Loop Class	Measured Deviation	Controlling Action	Controlled Action	Controlling Deviation Caused	Boundary Conditions
Control	LO	Positive	Positive	NO, LO	STUCK, FAILED-HI
Control	LO	Positive	Negative	NO, LO	STUCK, FAILED-LO
Control	HI	Positive	Positive	HI	STUCK, FAILED-LO
Control	HI	Positive	Negative	HI	STUCK, FAILED-HI
Control	LO	Negative	Positive	HI	STUCK, FAILED-HI
Control	LO	Negative	Negative	HI	STUCK, FAILED-LO
Control	HI	Negative	Positive	NO, LO	STUCK, FAILED-LO
Control	HI	Negative	Negative	NO, LO	STUCK, FAILED-HI
Trip		Positive	--	NO	FAIL-DANGER
Trip		Positive	--	HI, SOME	FAIL-DANGER
Trip		Negative	--	HI, SOME	FAIL-DANGER
Trip		Negative	--	NO	FAIL-DANGER

9.13: Causal and Limiting Ordering of Failure Chains

Failure chains for controlled and measured events are created, and user-input of causal and limiting orders for these failure chains accepted by the causal ordering rule subset.

This subset has only a loosely applied control structure. Firstly, a top event for ordering is selected, this being either controlled or measured by a loop or loops. Two rules, ORDERING-PROPAGATE-LINK and ORDERING-PROPAGATE-TO-CONJUNCTION, generate failure chains. Failure chains are represented by LINK elements.

A rule with low specificity, ORDERING-BACKTRACK, backtracks when a chain is searched which does not yield any further base-events for ordering above those already ordered. The low specificity of this rule ensures that any searching, or "propagation", rules matched by the same propagating event will execute preferentially if they are instantiated. Backtracking is also used to prevent invalid cyclic reasoning, such as represented by equations 9.1 and 9.2.

7 L LIQUID LO = 2 Q LIQUID HI = 7 L LIQUID HI ...(9.1)

7 L LIQUID HI = 2 Q LIQUID LO = L2 LEAK-TO-ATMOS ...(9.2)

Rules search for invalid cyclic reasoning in links, and backtrack where it is found to occur.

Demons are used to check failure chains for violation of boundary conditions; where violations are encountered the search backtracks. A demon is also used to mark "primary" combinatory events. In this case a primary event is defined as one for which there is no need to examine its input events, rather than one for which there are no input events. Under this definition, loop failure modes are regarded as primaries, rather than the failure modes of units which make up the loops.

Rules of higher specificity than link propagation rules are used to identify base-events and end propagation of failure chains. Where base-events are themselves controlled or measured events, LINK elements

are created for each possible order of the base-event. These are assigned causal and limiting orders separately.

For each failure chain generated, the events that make up the chain are printed, and the user asked for causal and limiting orders. The user may define the causal order as "INVALID", meaning that the failure chain will not produce the output event, or that its effect is considered to be negligible.

Low specificity rules remove backtracks and invalid links and deactivate selected events.

9.14: Generation of Group Protection Failure Mini-Fault Trees

The group failure subset creates group protection failure mini-fault trees from group protection mini-trees. AND gates in protection mini-trees produce OR gates in mini-fault trees; protection OR gates are equivalent to failure AND gates; OUT-OF gates result in sets of AND gates.

9.15: Fault Tree Synthesis

The synthesis subset creates fault trees from ordered links, the plant event network, and group protection failure mini-fault trees. After the first stage, which accepts details of the top event from the user, no rigid control structure is employed.

Two rules, SYNTH-PROPAGATE-FROM-LINK-TOP and SYNTH-PROPAGATE-LINK, generate inputs that result from ordered failure chains. Demons are used to disable failure chains which:

1. Contain events which violate boundary conditions of higher events in the fault tree; or
2. Contain events which would produce repetition of events in the fault tree in such a manner as to create loops in the tree structure; or
3. Have have a limiting order which is violated by higher events in the fault tree.

Demons are also used to maintain the register of above-events and to make boundary conditions which are associated specifically with certain nodes in the tree. Such boundary conditions arise from boundary conditions associated with gates in the plant event net.

Rules create failure chains that arise from direct protection failure by creating combinatory inputs, controlled event order n-1 and order n protection failure, to a controlled event of order n. Protection failure is represented either by the top event of a group protection failure mini-fault tree, or by the failure of an individual loop if that loop alone protects the nth order of the controlled event. If a lower order of the same controlled event forms an input to the same node in the fault tree as the order n event, then the direct protection failure rule will not execute for the order n event. This simplifies the fault trees produced by preventing repetition of event chains. A rule, SYNTH-PROPAGATE-FAILUREMODE-TO-FAILUREMODE, copies the mini-fault trees for group protection failure into fault trees. This same rule generates non-deviatory failure modes that are not explicitly represented by ordered links.

Indirect protection failure is incorporated by searching the direct above-event register of nodes corresponding to measured events. If one of the above events is protected against the measured event, then the measured event is combined by an AND gate with the appropriate protection failure mini-fault tree or loop failure mode.

The following consistency checks are applied after fault tree generation:

1. Events are checked against boundary conditions of above-events.
2. Deviatory events not arising from across the plant boundary must have at least one valid input.
3. An AND gate from the plant event network must have all of its input events forming valid inputs to the resulting AND gate in the fault tree.
4. Inputs to an invalid event are also invalid.
5. Events combined by an AND gate with an invalid event are also

invalid.

These checks are applied recursively until all invalid events are identified. These are then removed by a low specificity rule. Use of a low specificity rule here allows all of the invalid events to be identified prior to their removal.

9.16: Discussion

The OFTS program as described in this chapter has been implemented in OPS5 and tested using the basic ammonia let down plant described in chapter 5, and variations thereon. The results from testing this program are presented and discussed in the next chapter. It should be pointed out that the OFTS program as it stands is not a full implementation of the fault tree synthesis method described in this thesis. A number of aspects, such as manual control and cascade control, are yet to be modelled, and only those unit models required for the ammonia let down plant have been created.

During the course of this thesis, a number of areas which require future work have been identified, these are discussed in chapter 11.

CHAPTER TEN

10. TESTING THE FAULT TREE SYNTHESIS PACKAGE

10.1 Introduction

This chapter describes the testing of the OFTS fault tree synthesis program. Fault trees are generated for the basic ammonia let down plant described in chapter 5, and for a modified version of this plant. Two fault events are considered: overpressure of the flash vessel, C2; loss of liquid level in C2. Overpressure of C2 is important because it may lead to vessel rupture; loss of level in C2 is important because it would lead to gas breakthrough into the ammonia liquid exit line, creating problems further on in the process.

It should be noted that in order to accurately generate fault trees for a plant area, some knowledge of the context of that section of plant is necessary. It is, for example, important to identify which input events from across plant boundaries may arise. For the ammonia let down plant under consideration, Lihou [89] considered that only the input deviations listed in Table 10.1 were likely to occur at node 1. All other deviations at node 1 are defined as invalid for this analysis.

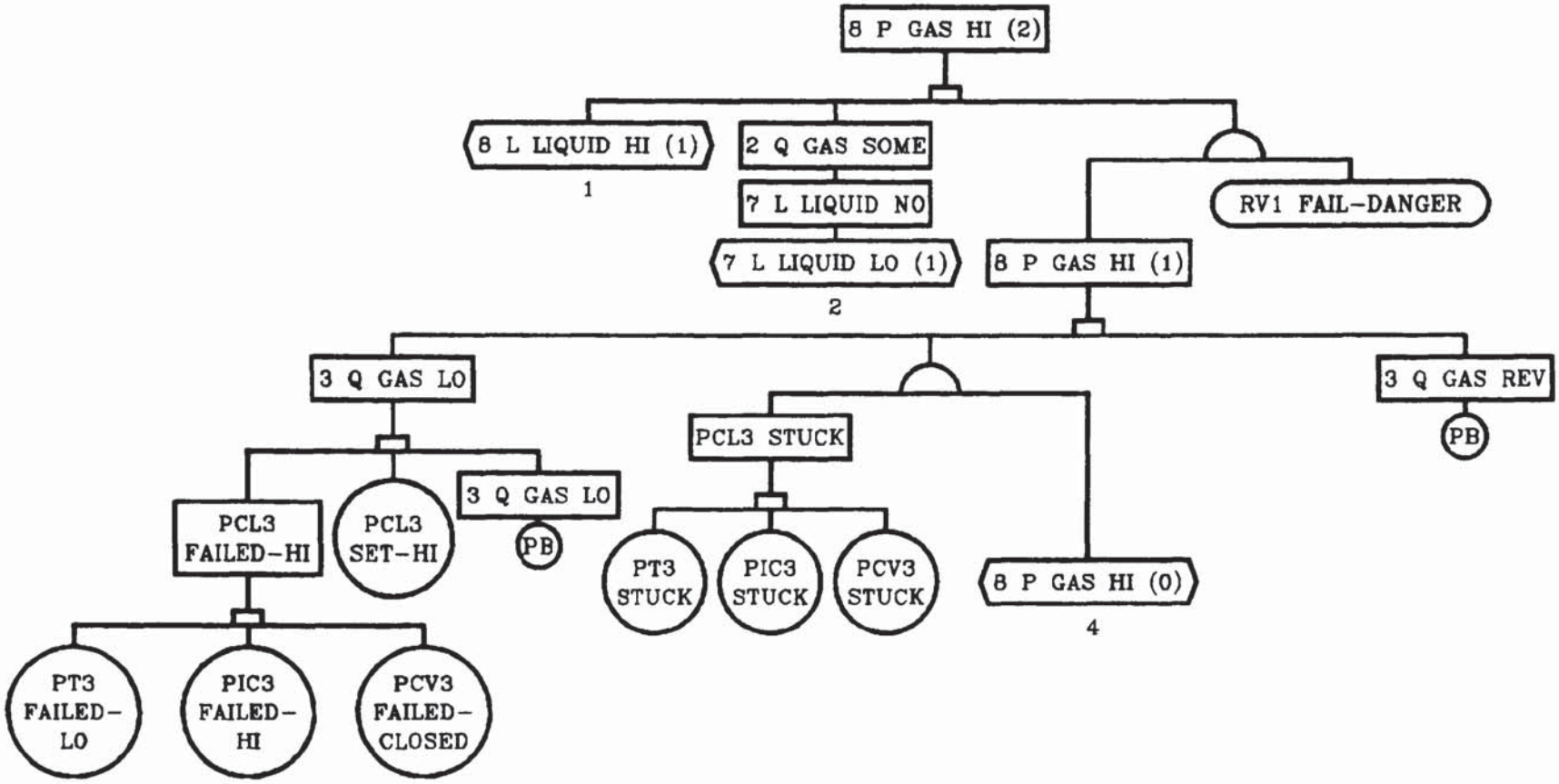
10.2: Results Obtained for the Basic Ammonia Let Down Problem

The basic ammonia let down problem is described fully in chapter 5, a flowsheet for the plant appears in Fig. 5.1. Three control loops and a pressure relief system are defined as described in Table 10.2. The event chains identified by the event ordering rules, and the causal and limiting orders assigned to them, are shown in Tables E.1 to E.6. When assigning limiting order values, the maximum permissible limiting orders were defined for most event chains, so that they will appear in the output fault trees. In fact, some of these event chains could be assigned lower limiting orders. This would simplify the fault trees produced. The fault tree generated for overpressure of vessel C2 is shown by Fig. 10.1; that generated for loss of liquid level in vessel C2 is shown in Fig. 10.2.

Table 10.1: Valid Deviations at Node 1 for the Ammonia Let Down Plant

DEVIATION	MEANING
1 Q NO	No flow entering in line 1
1 Q LO	Less flow entering in line 1
1 Q LIQUID LO	Less flow of liquid entering in line 1
1 Q GAS LO	Less flow of gas entering in line 1
1 Q GAS HI	More flow of gas entering in line 1
1 T LO	Low temperature of mxture entering in line 1
1 T HI	High temperature of mixture entering in line 1
1 X GAS AMMONIA-GAS HI	High concentration of ammonia gas entering in line 1
1 X GAS SYNTHESIS-GAS LO	Low concentration of synthesis gas entering in line 1

Fig. 10.1: Fault Tree for Overpressure of C2 for the Basic Ammonia Let Down Plant



PB: Deviation from across the plant boundary

Fig. 10.1a: Continuation 1 of Fault Tree from Fig. 10.1

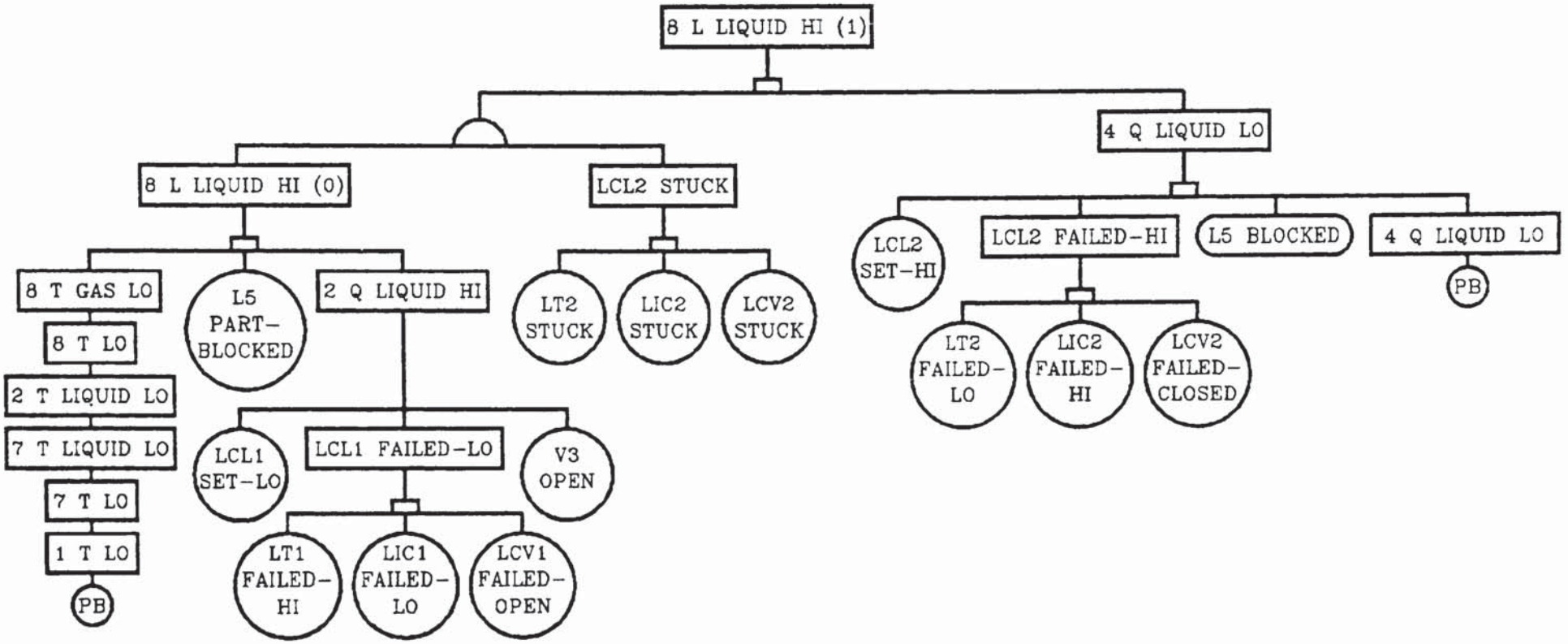


Fig. 10.1b: Continuation 2 of Fault Tree from Fig. 10.1

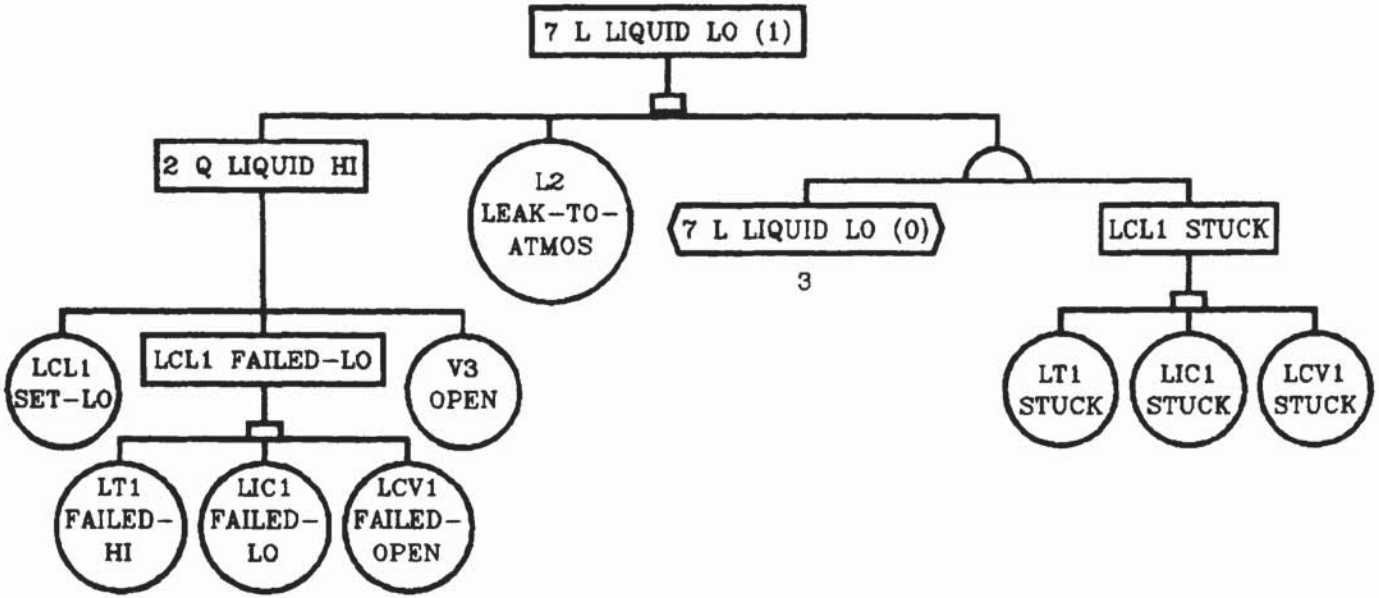


Fig. 10.1c: Continuation 3 of Fault Tree from Fig. 10.1

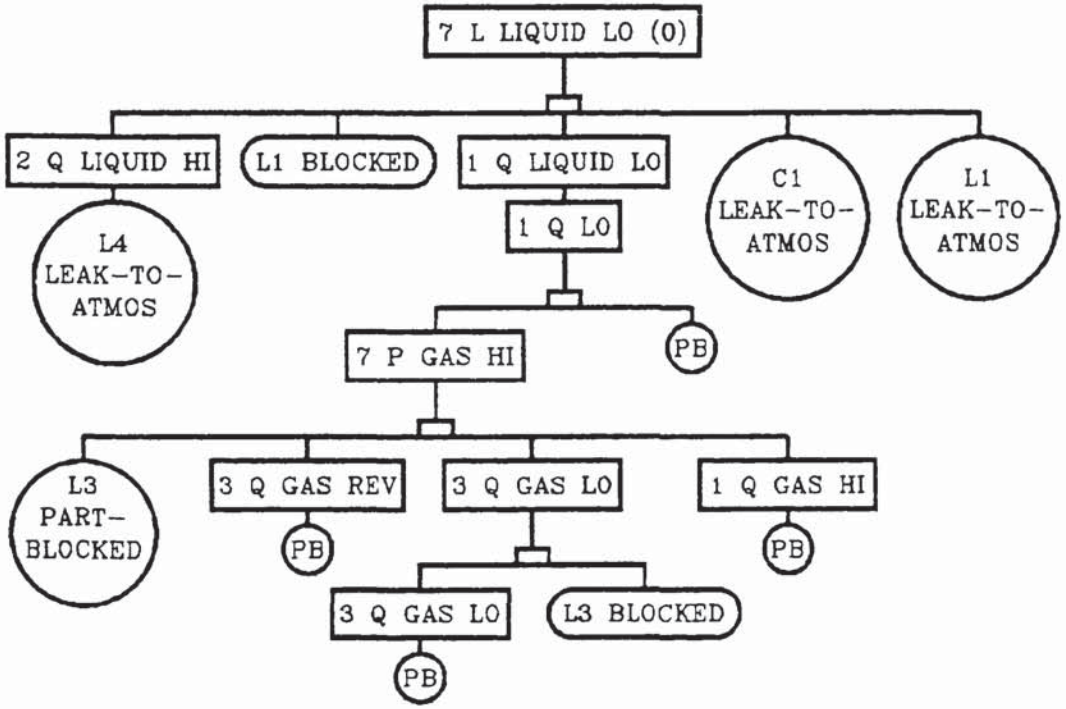


Fig. 10.1d: Continuation 4 of Fault Tree from Fig. 10.1

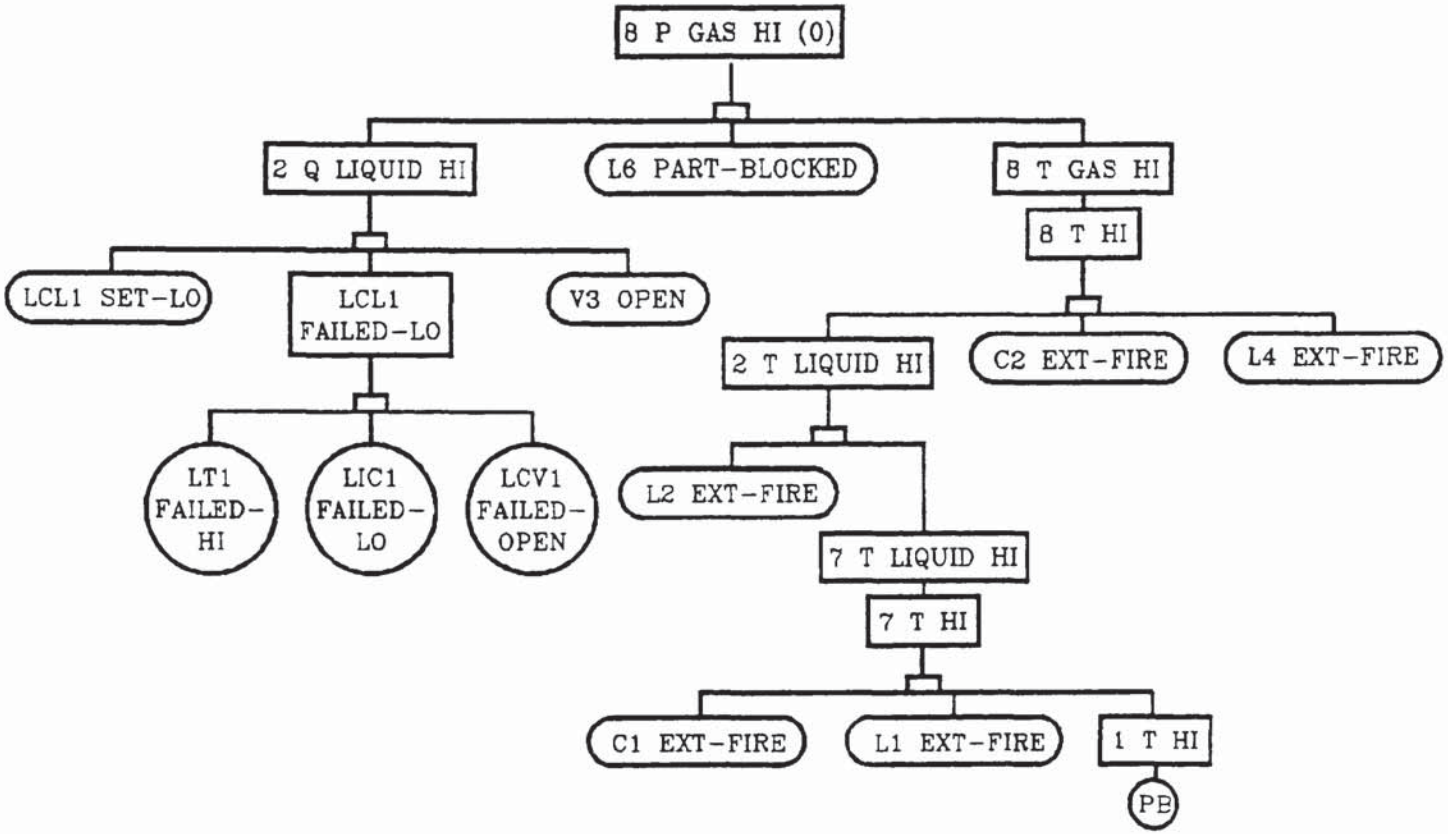


Fig. 10.2: Fault Tree for Loss of Liquid Level in C2 for the Basic Ammonia Let Down Plant

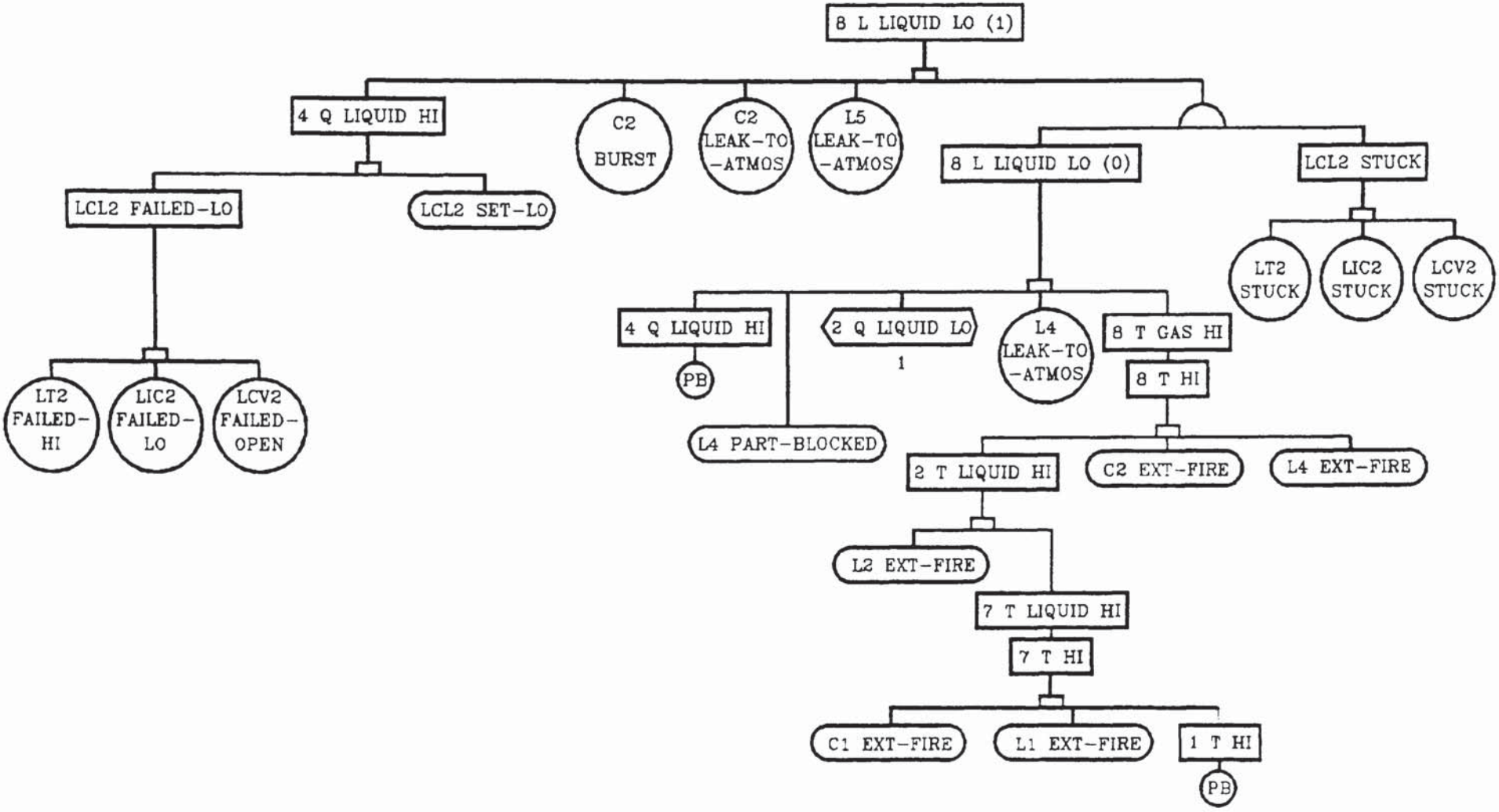


Fig. 10.2a: Continuation 1 of Fault Tree from Fig. 10.2

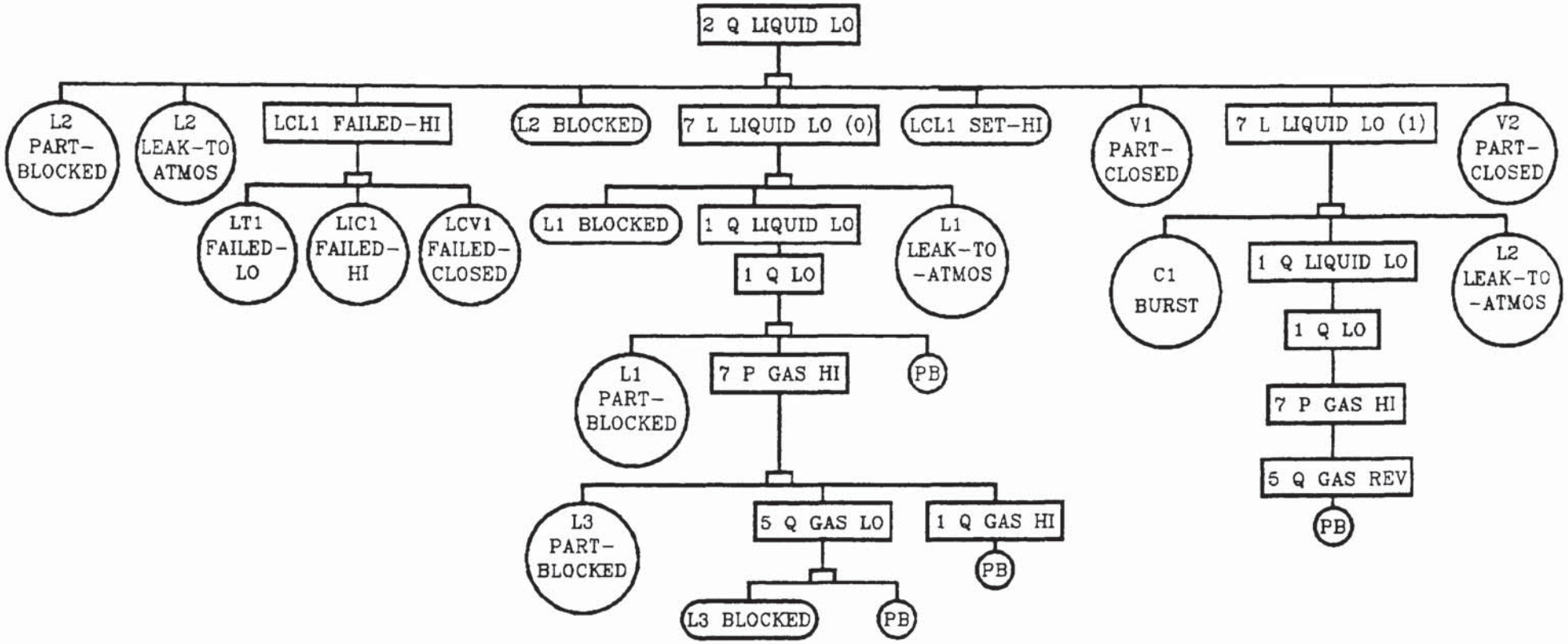


Table 10.2: Loops Defined for the Basic Ammonia Let Down Plant

LOOP NAME	LOOP CLASS	LOOP TYPE	CONTROLLED ACTION	CONTROLLED EVENTS	CONTROLLING EVENTS	LOOP ORDER
LCL1	CONTROL	DIRECT	NEGATIVE	7 L LIQUID LO/HI	2 Q LIQUID NO/LO/HI	1
LCL2	CONTROL	DIRECT	NEGATIVE	6 L LIQUID LO/HI	4 Q LIQUID NO/LO/HI	1
PCL3	CONTROL	DIRECT	NEGATIVE	8 P GAS LO/HI	3 Q GAS NO/LO/HI	1
RV1	RELIEF	DIRECT	NEGATIVE	8 P GAS HI	VENT Q GAS SOME	2

From an analysis of these fault trees, three principal hazards for this plant area may be identified:

1. Gas breakthrough to C2 due to loss of liquid seal in C1. It was assumed in this analysis that the relief valve RV1 would be unable to cope with the volumetric flow from line L2/L4 at ten times the pressure in C2 if gas breakthrough were to occur. For this reason, a causal order of 2 was assigned to the event chain including 2 Q GAS SOME; this chain thus forms a direct input to 8 P GAS HI (order 2).

2. Entrainment of liquid ammonia into line L3 and the relief vent line, restricting their capacity. It was assumed that if liquid entrainment in these lines should occur, then they would be unable to prevent overpressure of the vessel. For this reason, a causal order of 2 was assigned to 8 L LIQUID HI (order 1), resulting in a direct input to the top event 8 P GAS HI (order 2). A number of direct inputs to 8 L LIQUID HI (1) were identified: these input events could thus directly cause the top event, overpressure of C2.

3. A number of uncombined inputs to 8 L LIQUID LO (order 1) were identified: each of these could directly result in loss of liquid level in C2.

In order to reduce these hazards, it is recommended that the following modifications should be made to the plant:

1. The relief valve RV1 should be resized such that it would be able to cope with gas breakthrough in line L2/L4 or with liquid entrainment in the vent lines.
2. Additional low level trips should be installed to help prevent loss of the liquid seal in C1.
3. An additional high pressure venting system, operated by a trip system, should be installed to reduce the probability of control loop failure causing overpressure of C2.
4. Additional low level trip systems should be installed to reduce the likelihood of liquid seal loss in C2.

A modified ammonia let down plant which incorporates these recommendations is described in the next section.

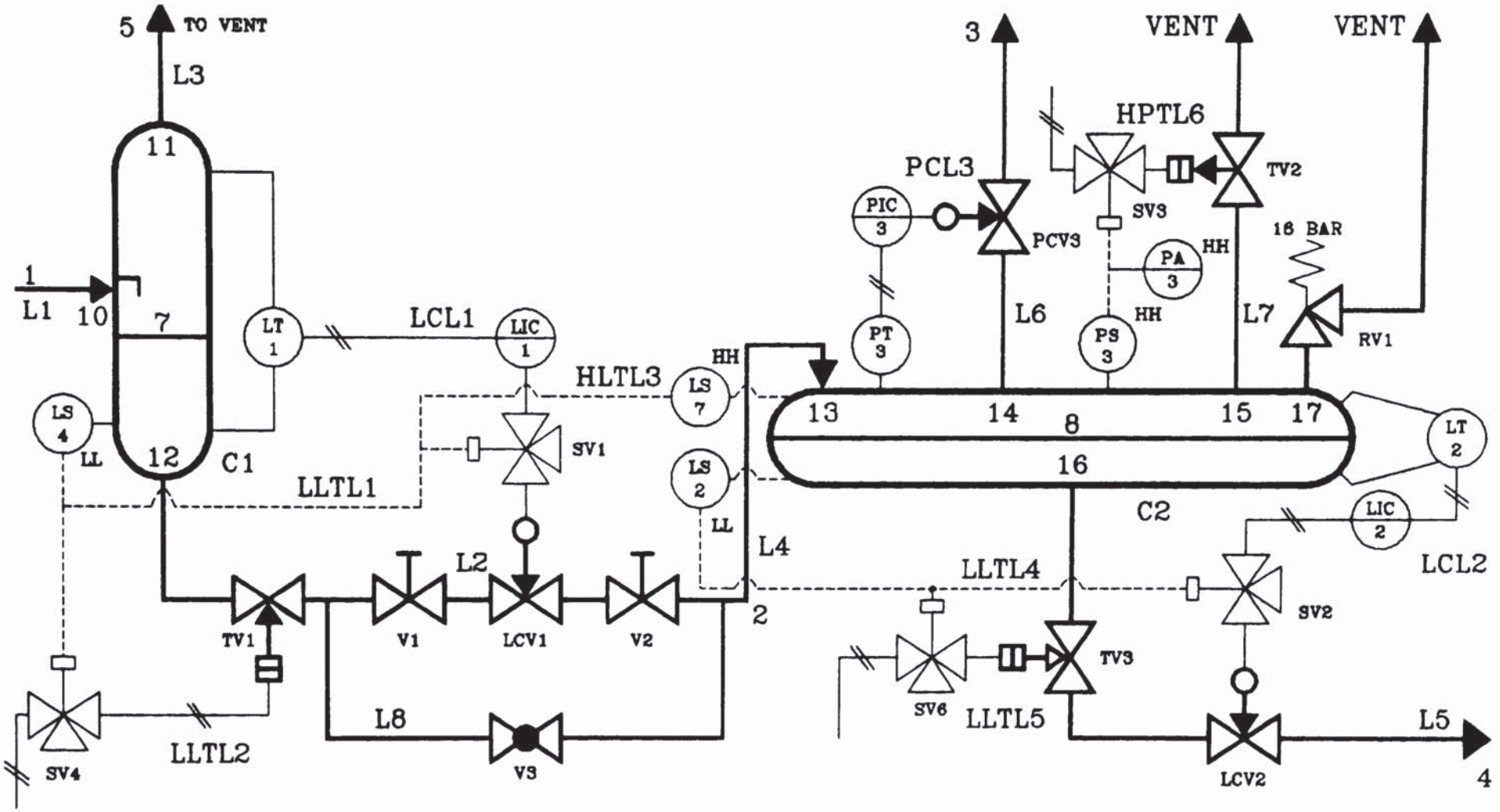
10.3 The Modified Ammonia Let Down Plant

The plant described in this section has been modified from the basic ammonia let down plant described previously in order to improve plant safety. The modifications made conform to the recommendations made in the last section. The OFTS program is used to generate fault trees for this more complex problem.

A flowsheet of the modified plant is shown in Fig. 10.3. Six trip loops have been incorporated: LLTL1; LLTL2; HLTL3; LLTL4; LLTL5; HPTL6. The function of LLTL1 and LLTL2 is to protect against loss of liquid seal in C1. HLTL3 protects against high level in C2. LLTL4 and LLTL5 protect against loss of liquid seal in C2. HPTL6 protects against high pressure in C2. These loops are defined as in Table 10.3. Both pairs of low level trip loops operate such that only one must function correctly in order that protection of their respective controlled variables is achieved.

For analysis of this problem it is assumed that the relief valve RV1 has been resized such that it is able to cope with gas breakthrough in line L2/L4 or liquid entrainment in the vent lines. The causal and limiting orders of event chains used for this analysis are shown in Tables E.7 to E.12. The fault trees for overpressure of C2 and loss of liquid seal in C2

Fig. 10.3: Flowsheet of the Modified Ammonia Let Down Plant



are shown in Figs. 10. 4 and 10.5 respectively.

Table 10.3: Loops Defined for the Modified Ammonia Let Down Plant

LOOP NAME	LOOP CLASS	LOOP TYPE	CONTROLLED ACTION	CONTROLLED EVENTS	CONTROLLING EVENTS	LOOP ORDER
LCL1	CONTROL	DIRECT	NEGATIVE	7 L LIQUID LO/HI	2 Q LIQUID NO/LO/HI	1
LCL2	CONTROL	DIRECT	NEGATIVE	8 L LIQUID LO/HI	4 Q LIQUID NO/LO/HI	1
PCL3	CONTROL	DIRECT	NEGATIVE	8 P GAS LO/HI	3 Q GAS NO/LO/HI	1
RV1	RELIEF	DIRECT	NEGATIVE	8 P GAS HI	VENT Q GAS SOME	3
LLTL1	TRIP	DIRECT	NEGATIVE	7 L LIQUID LO	2 Q LIQUID NO	2
LLTL2	TRIP	DIRECT	NEGATIVE	7 L LIQUID LO	2 Q LIQUID NO	2
HLTL3	TRIP	DIRECT	POSITIVE	8 L LIQUID HI	2 Q LIQUID NO	2
LLTL4	TRIP	DIRECT	NEGATIVE	8 L LIQUID LO	4 Q LIQUID NO	2
LLTL5	TRIP	DIRECT	NEGATIVE	8 L LIQUID LO	4 Q LIQUID NO	2
HPTL6	TRIP	DIRECT	NEGATIVE	8 P GAS HI	VENT Q GAS SOME	2

10.4 Discussion of the Results Obtained

The fault trees of Figs. 10.4 and 10.5 clearly show the improvements made to the plant design. There are no longer any direct inputs to 8 P GAS HI (order 3); this is due to the resizing of the relief valve RV1. HPTL6 is less effective: the two principal failure modes, gas breakthrough into line L2 and liquid entrainment in the vent lines, are unaffected by this loop. The major purpose of HPTL6 is to reduce the likelihood of control loop failure leading to overpressure of C2, however. The fault tree shows that this function is achieved. The likelihood of loss of liquid seal in C2 has been considerably reduced by the trip loops LLTL4 and LLTL5. The fault tree shows that it is now necessary for both of these loops to fail in order for the liquid seal to be lost, unless the causal event is leakage from line L5. This event is not considered to be too important, however. The line is not under high pressure, and the leak would have to

Fig. 10.4: Fault Tree for Overpressure of C2 for the Modified Ammonia Let Down Plant

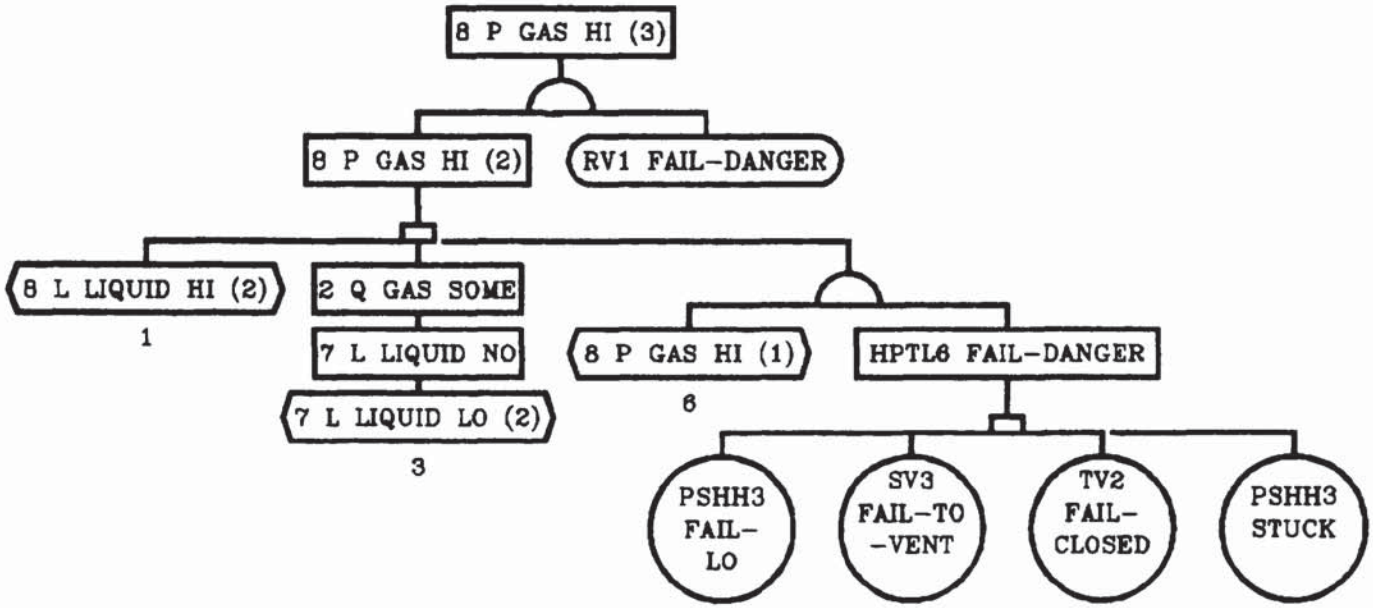


Fig. 10.4a: Continuation 1 of Fault Tree from Fig. 10.4

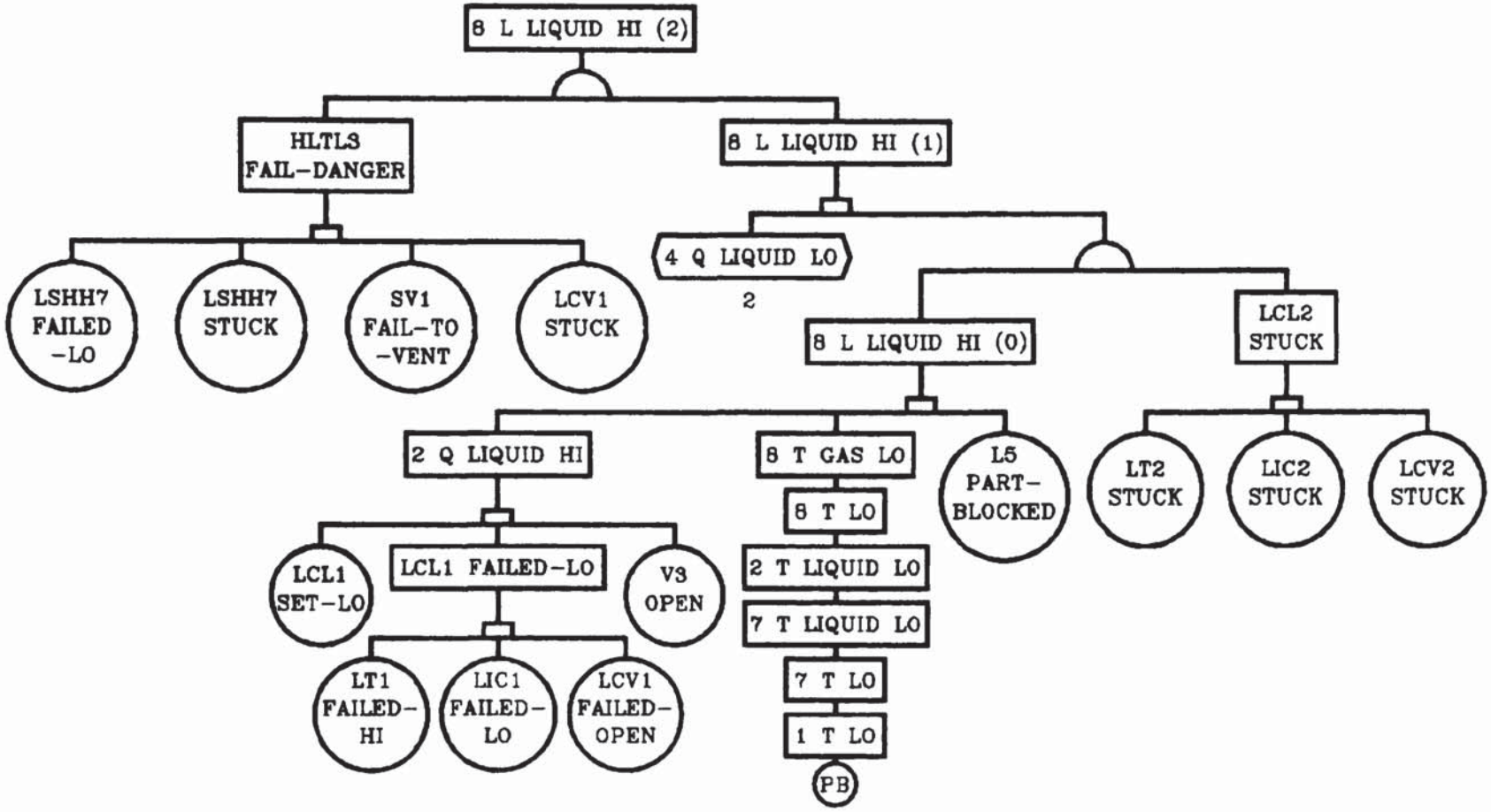


Fig. 10.4b: Continuation 2 of Fault Tree from Fig. 10.4

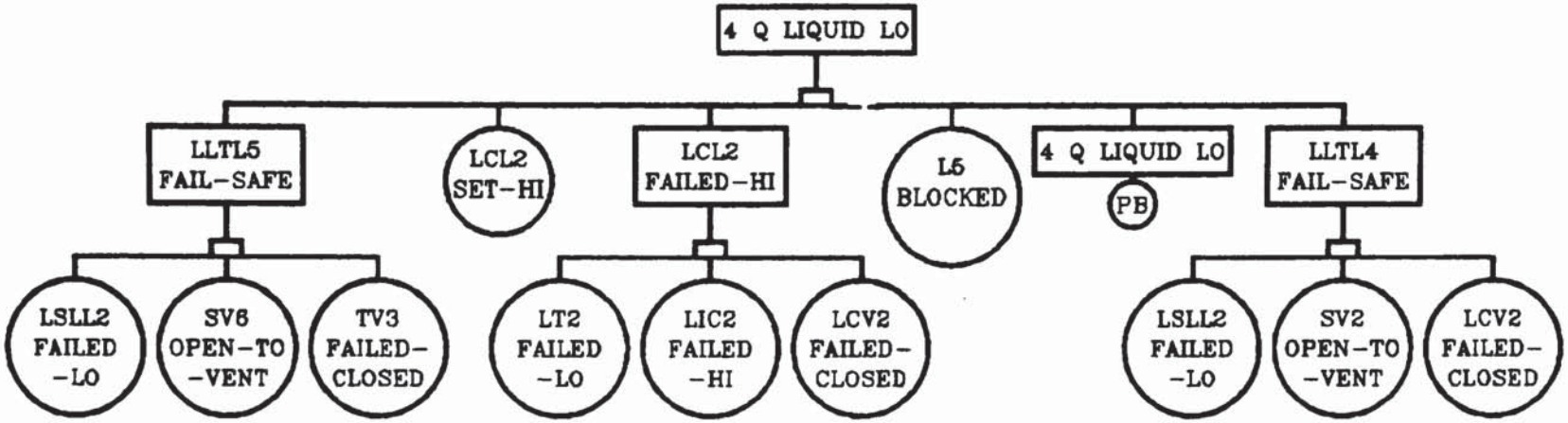


Fig. 10.4c: Continuation of Fault Tree from Fig. 10.4

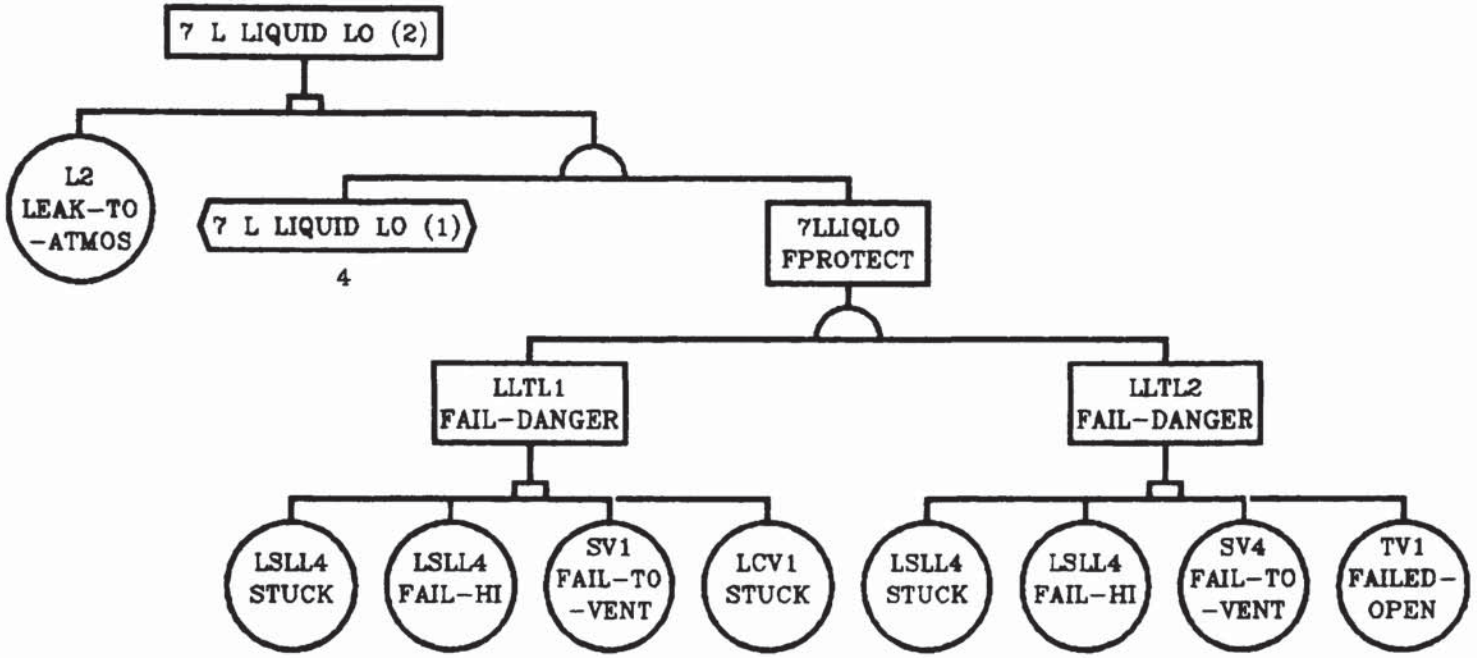


Fig. 10.4d: Continuation of Fault Tree from Fig. 10.4

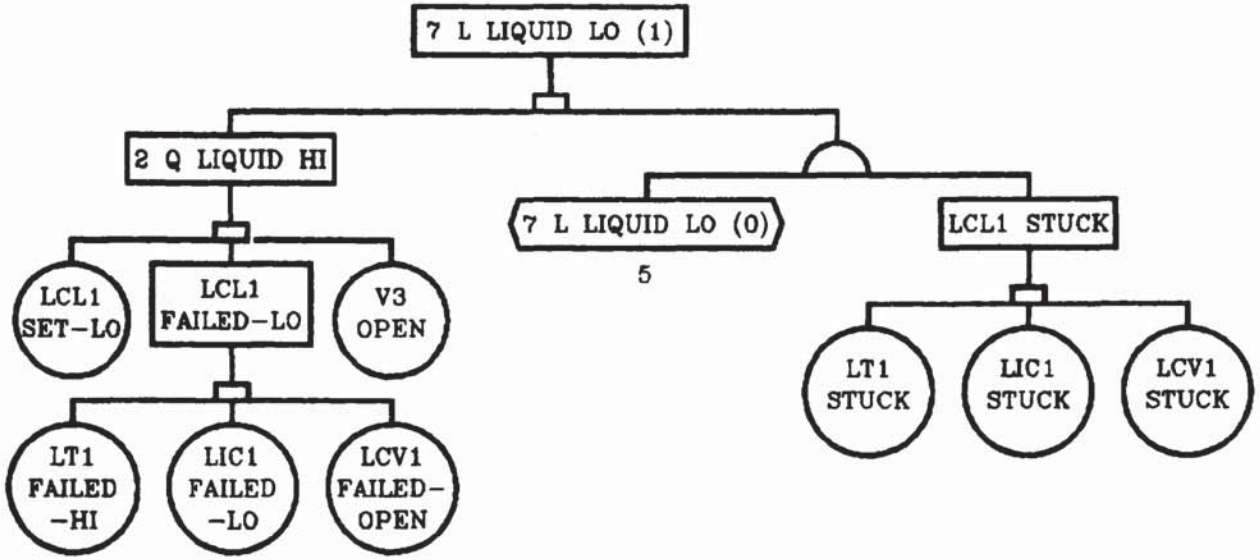


Fig. 10.4e: Continuation 5 of Fault Tree from Fig. 10.4

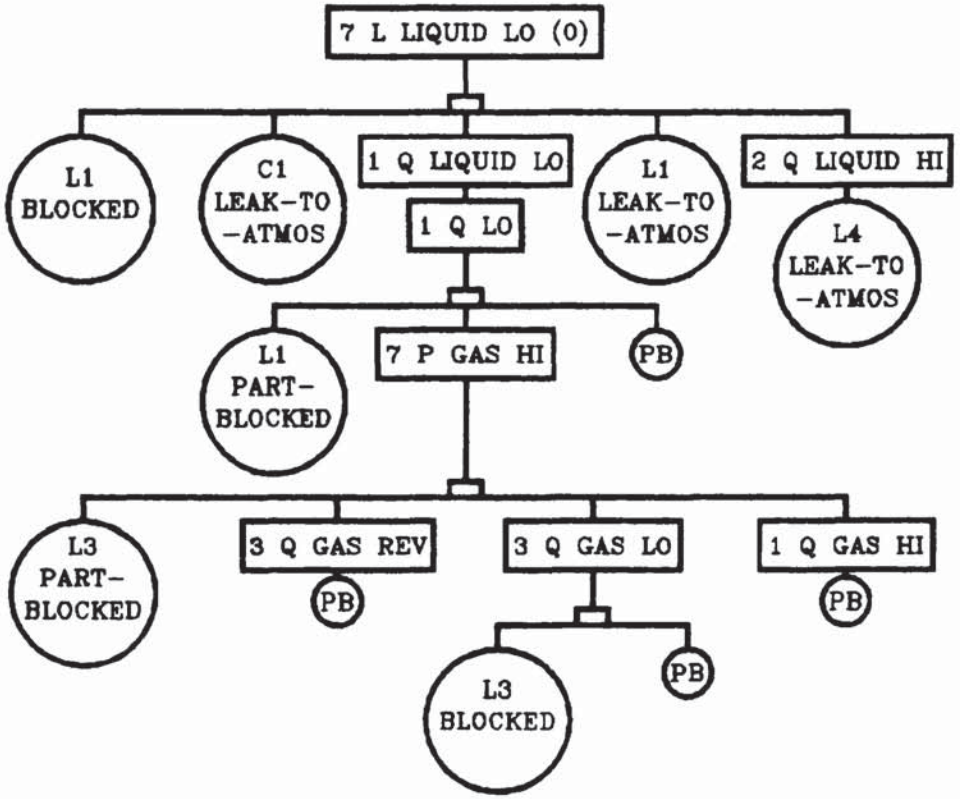


Fig. 10.4f: Continuation 6 of Fault Tree from Fig. 10.4

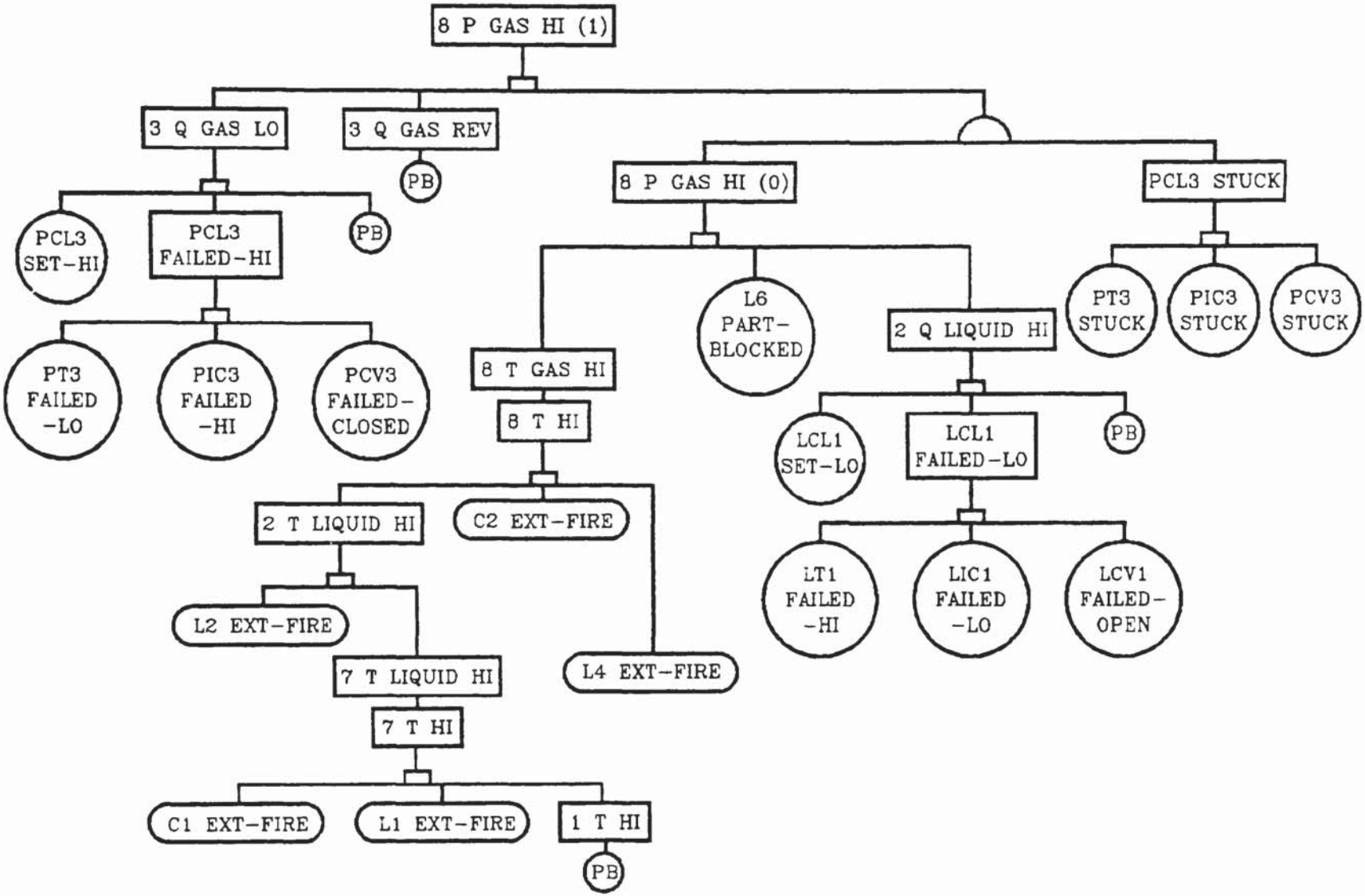


Fig. 10.5: Fault Tree for Loss of Liquid Level in C2 for the Modified Ammonia Let Down Plant

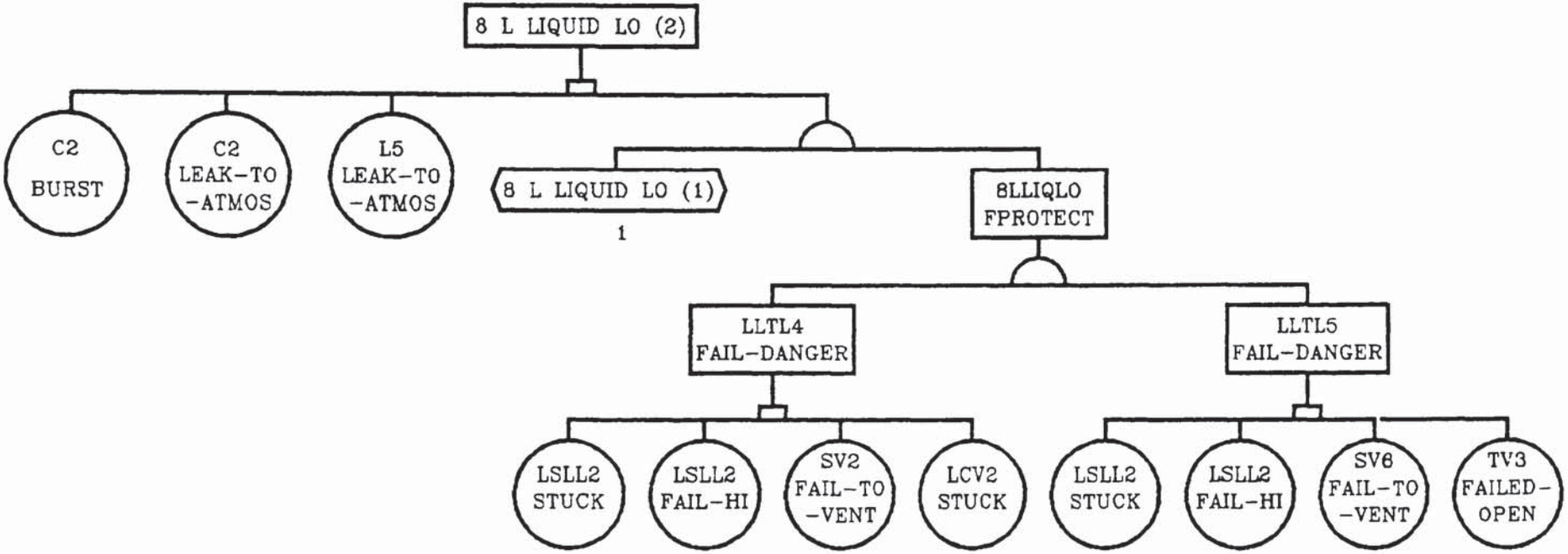


Fig. 10.5a: Continuation 1 of Fault Tree from Fig. 10.5

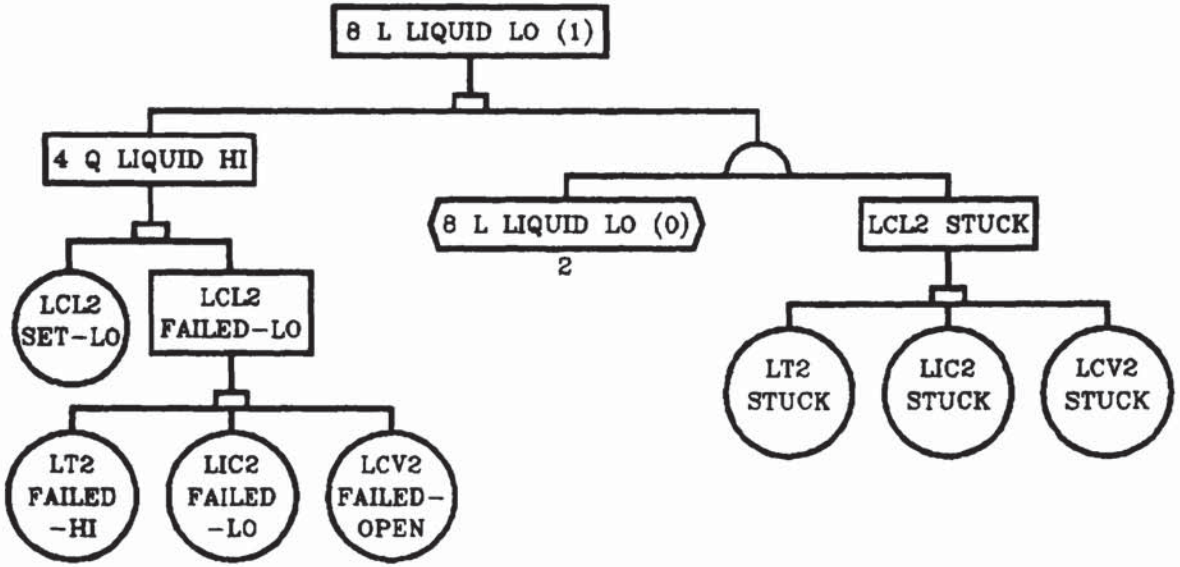


Fig. 10.5b: Continuation 2 of Fault Tree from Fig. 10.5

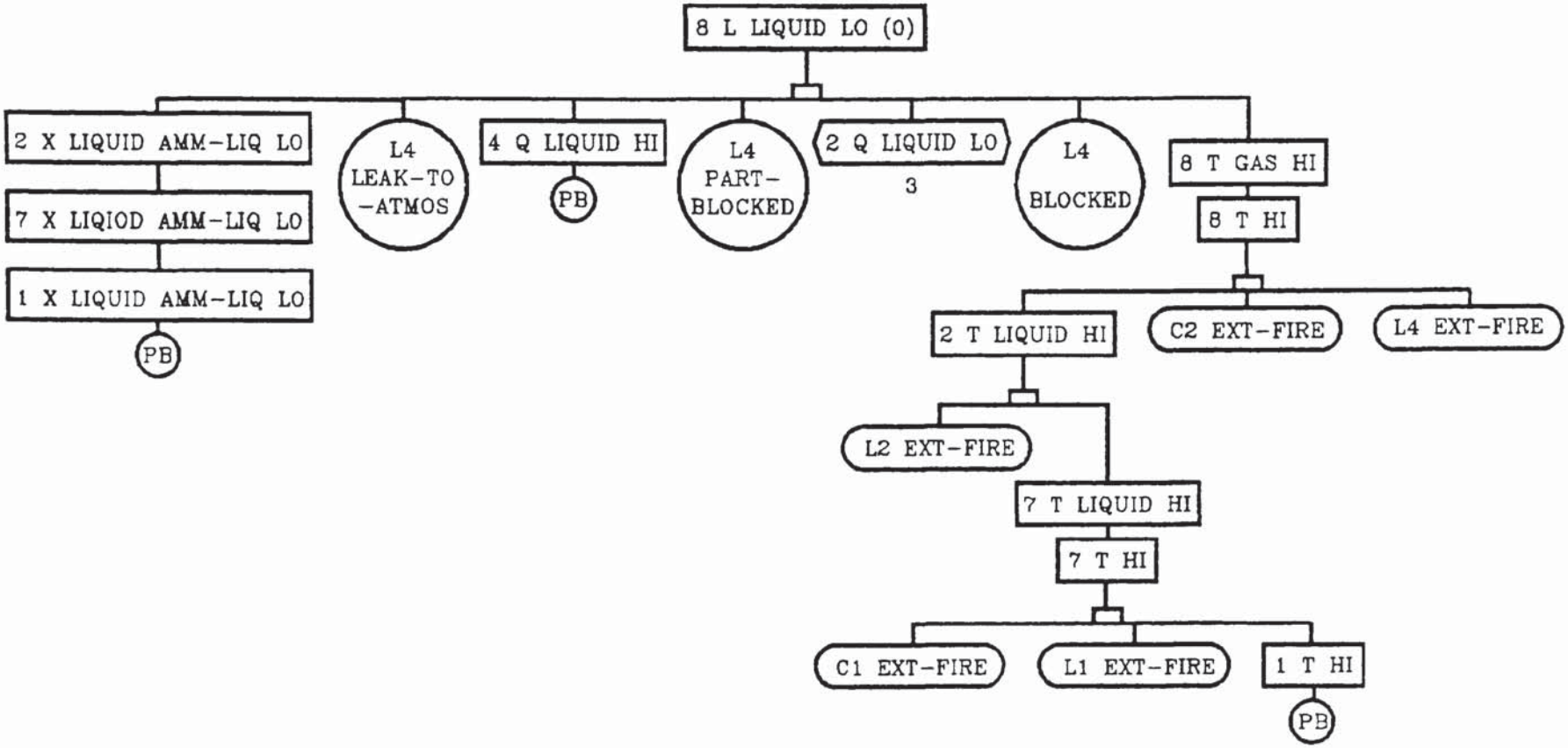


Fig. 10.5c: Continuation 3 of Fault Tree from Fig. 10.5

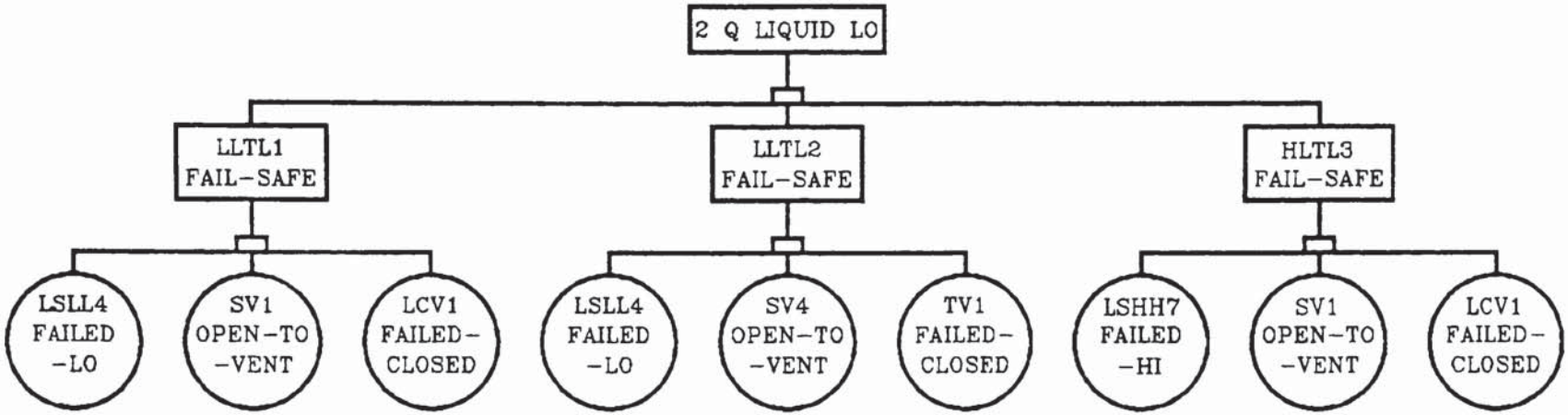


Fig. 10.5d: Continuation 3 of Fault Tree from Fig. 10.5

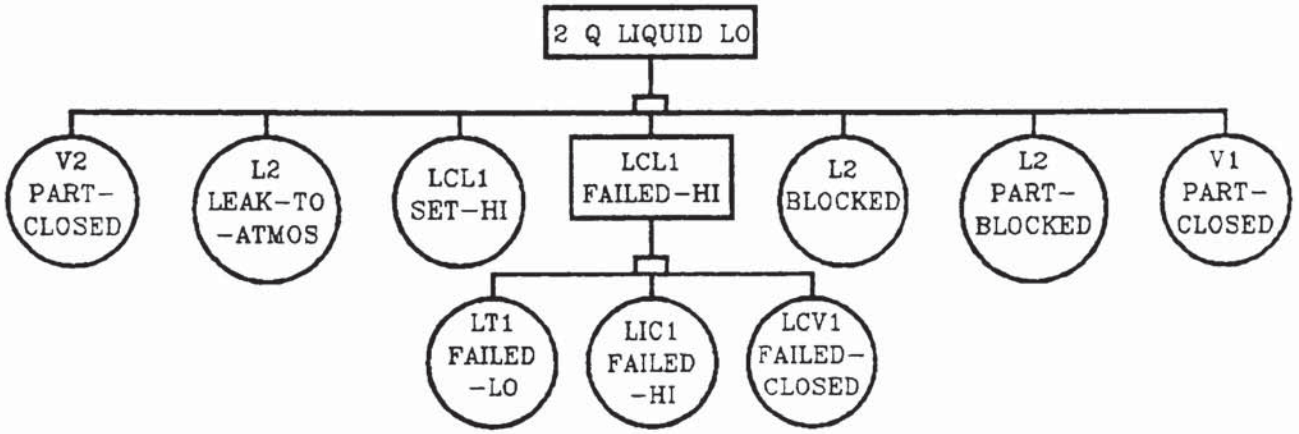
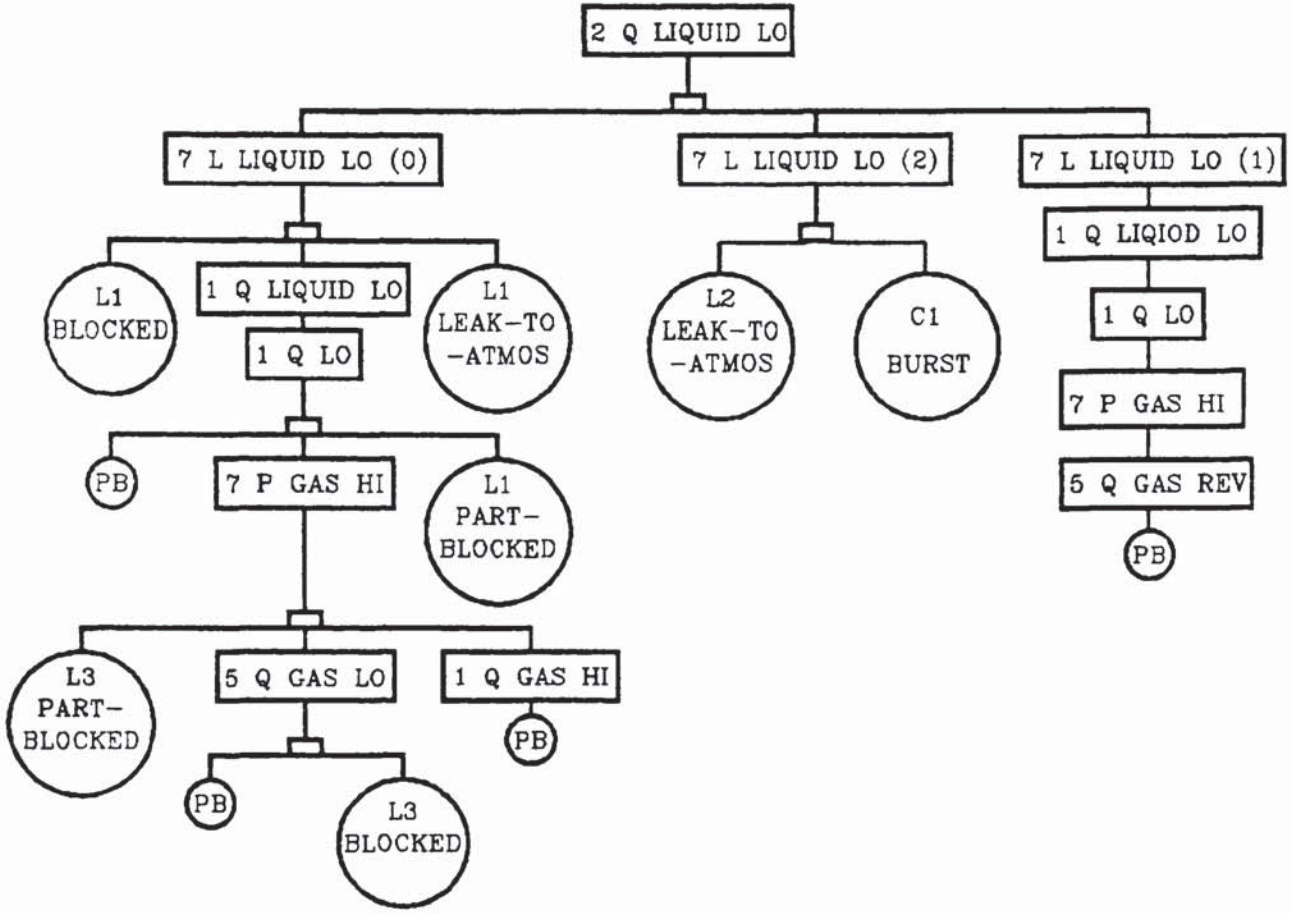


Fig. 10.5e: Continuation 3 of Fault Tree from Fig. 10.5



be considerable for liquid seal to be lost. In addition, if the trip valve TV3 is situated close to the exit from C2 then only leakage at this trip valve need be considered.

The fault trees produced compare very favourably with fault trees produced by other computer methods. Fig. H.1 shows part of a fault tree that was generated for overpressure of vessel C2 by a process unit modelling method similar to that used by Martin-Solis et al [36]. The part of the fault tree shown corresponds to the 2 Q GAS SOME input branch in the fault tree of Fig. 10.1. In this analysis combinatory events with LIC1 STUCK were not generated; taking this into account, the corresponding fault tree sizes are:

Process unit modelling: 61 events;
OFTS program: 16 events.

It can be seen that the OFTS-produced fault tree is almost four times smaller than that produced by process unit modelling.

Another improvement of the fault trees produced by the OFTS program is that they are considerably more readable than those produced by process unit modelling methods. This enhancement is due to the improved fault tree structure which results from the representation of control loops and protective systems as individual entities and the use of event ordering to define event magnitudes.

In his analysis of the ammonia let down plant, Lihou [89] ignored all failure event chains leading to overpressure of C2 except those associated with gas breakthrough in line L2/L4 and gas entrainment in the vent lines, and those involving loop failure. Many of the inputs even to these event failure chains were considered unimportant. The analysis described in this chapter which led to the fault trees of Figs. 10.1, 10.2, 10.4 and 10.5 includes nearly all of the identified event failure chains in the final fault trees. The fault trees produced could be considerably simplified if some of these event chains were considered unimportant and assigned lower limiting orders. Such event chains would not appear in the final fault trees. Such assignments of low limiting orders can only properly be

made when a full study is being undertaken and all data are available.

This analysis has shown that the OFTS program, implementing the plant reduction and event ordering methods, is capable of producing good fault trees for complex chemical process plant problems. The fault trees produced are small, well structured, comprehensive and error-free. No other computer-based fault tree synthesis method which uses standard process unit input-output equation libraries as a starting point is capable of producing fault trees of comparable quality.

CHAPTER ELEVEN

11. DISCUSSION, RECOMMENDATIONS FOR FUTURE WORK AND CONCLUSIONS

11.1: DISCUSSION

11.1.1: Introduction

Fault trees provide a good means for graphical representation of system failure logic. By analysing fault trees it is possible not only to estimate system failure probabilities, but also to identify specific system design flaws. Despite these considerable benefits, the use of fault tree analysis in the chemical process industry is not widespread. The major reason for this is that hand-generation of fault trees for large chemical process plants is an extremely time-consuming, and therefore expensive, task.

Attempts have been made to create algorithms for the automatic synthesis of fault trees for chemical process plant, and to implement such algorithms as computer-based methods. None of the methods produced to date, however, are suitable for extensive industrial use. There are two major of reasons for this:

1. The fault trees produced by these methods are voluminous, and generally lack any readily identifiable structure.
2. These methods make assumptions about process plant interactions which would severely limit their applicability to real-world problems.

The work presented in this thesis has shown that, by using a rule-based expert system to represent knowledge about process plant "building blocks", it is possible to overcome these problems. Two distinct methods are used to achieve this, these being:

1. Plant reduction. Rules about process lines, units and loops are used to reduce the number of nodes the system uses to define the problem. This reduction in nodes is reflected in equivalent reductions in the problem's state-space and in the fault trees

produced.

2. Definition of event magnitudes in terms of their controllability by control and trip loops and other protective systems. It is possible to define complex combinations of protective systems that must function correctly in order to maintain a variable within certain limits, and also to define magnitudes of events which are controllable or uncontrollable by certain loops, or combinations of loops. It is also possible to define in these terms the maximum magnitude of an output event that an input event of a certain magnitude may cause.

A hybrid system, OFTS, which applies these methods utilises models of major process units, in the form of input-output and failure mode equations, as a starting point, along with a description of the plant topology. The OFTS system has been implemented on a computer using the OPS5 production language.

During the development of the OFTS system, several points which require further discussion have arisen, these are discussed in the ensuing sections. The current program implementation is only a small fraction of what a full implementation would consist of. A number of areas which require further work have been identified; these are discussed in section 11.2.

11.1.2: The Distinction Between Major and Minor Process Units

In section 6.4, rules for the definition of process items as major process units or minor process units were stated. These rules are, however, really only guidelines. As experience of using the OFTS system, and of defining models for both major and minor process units, increases it is expected that more and more unit types will be regarded as minor, rather than major units. The advantage of such an approach would be a reduction in the size of the fault trees produced, but this benefit would only be achieved by increasing the analysis of plant items and their interactions, resulting in more complex production rules.

It is possible that the levels of representation could be increased further. The possibilities of a third level, functional area modelling, are

discussed in section 11.2.2. It is felt by the author that such an approach may be more beneficial than the definition of complex units as minor process units.

11.1.3: Low-Level and High-Level Process Operators

It was mentioned in section 6.5.2 that a closed bypass line may be represented either as a single entity, the "closed bypass", or as a combination of several entities: a line split, a closed line, and a junction. This is an example of a situation where a choice of modelling methods is available, the options being:

1. Low-level modelling, as a combination of fundamental entities; or
2. High-level modelling, where a highly-specified model is used.

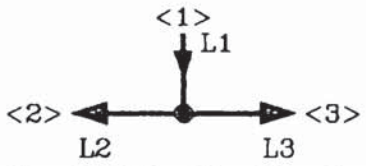
In this example, low-level modelling uses three fundamental models: line split, closed line, and line junction. High-level modelling uses one highly specific model: closed bypass line.

These two types of modelling are compared in Tables 11.1, 11.2 and 11.3 which show the models necessary to represent line L2 and a closed bypass line shown in Fig. 11.1. Fault trees produced using the two types of modelling are shown in Figs. 11.2 and 11.3. These fault trees are equivalent, but the one produced by low-level modelling is considerably larger.

There are advantages to both of these types of modelling. The use of comprehensive low-level modelling ensures that all possible plant configurations may be represented, but this type of modelling tends to result in larger, more complicated fault trees. In some situations, low-level modelling may produce inaccurate or spurious results. Process unit modelling as used in the methods developed by Fussell, Powers & Tompkins, Martin-Solis, etc, is modelling at a lower level than even the low-level models used in the OFTS method. This explains the poor quality of fault trees produced by these, and similar, methods.

High-level modelling results in smaller, more structured fault trees.

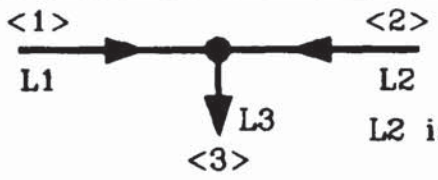
Table 11.1: Low-level Model for Closed Line Splits



L3 is closed

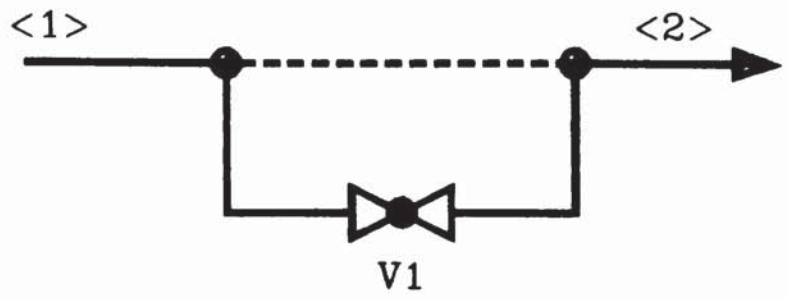
EVENT	CAUSE	BOUNDARY
<1> Q NO	<2> Q NO	<3> Q SOME
<1> Q LO	<2> Q LO	<2> Q SOME
<1> Q HI	<2> Q HI + <3> Q SOME	
<1> Q REV	<2> Q REV + <3> Q REV	
<2> Q NO	<1> Q NO	
<2> Q LO	<1> Q LO + <3> Q SOME	
<2> Q HI	<1> Q HI	
<2> Q REV	<1> Q REV	
<2> Q <PHASE> NO	<1> Q <PHASE> NO	
<2> Q <PHASE> LO	<1> Q <PHASE> LO	
<2> Q <PHASE> HI	<1> Q <PHASE> HI	
<2> Q <PHASE> SOME	<1> Q <PHASE> SOME + <3> Q <PHASE> REV	
<2> X <COMPONENT> NO	<1> X <COMPONENT> NO	
<2> X <COMPONENT> LO	<1> X <COMPONENT> LO	
<2> X <COMPONENT> HI	<1> X <COMPONENT> HI	
<2> X <COMPONENT> SOME	<1> X <COMPONENT> SOME + <3> Q <COMPONENT> REV	
<2> T (<PHASE>) LO	<1> T (<PHASE>) LO	
<2> T (<PHASE>) HI	<1> T (<PHASE>) HI	
<3> Q <PHASE> SOME	<3> Q SOME * <1> Q <PHASE> SOME	
<3> X <COMPONENT> SOME	<3> Q SOME * <1> X <COMPONENT> SOME	

Table 11.2: Low-level Model for Closed Line Junctions



EVENT	CAUSE
<3> Q NO	<1> Q NO
<3> Q LO	<1> Q LO
<3> Q HI	<1> Q HI + <2> Q SOME
<3> Q REV	<1> Q REV + <2> Q REV
<1> Q REV	<3> Q REV
<3> Q <PHASE> NO	<1> Q <PHASE> NO
<3> Q <PHASE> LO	<1> Q <PHASE> LO
<3> Q <PHASE> HI	<1> Q <PHASE> HI + <2> Q <PHASE> SOME
<3> Q <PHASE> SOME	<1> Q <PHASE> SOME + <2> Q <PHASE> SOME
<3> X <COMPONENT> NO	<1> X <COMPONENT> NO
<3> X <COMPONENT> LO	<1> X <COMPONENT> LO
<3> X <COMPONENT> HI	<1> X <COMPONENT> HI + <2> Q <COMPONENT> SOME
<3> X <COMPONENT> SOME	<1> X <COMPONENT> SOME + <2> Q <COMPONENT> SOME
<3> T (<PHASE>) LO	<1> T (<PHASE>) LO
<3> T (<PHASE>) HI	<1> T (<PHASE>) HI

Table 11.3: High-level Model for Closed Bypass Lines



EVENT	CAUSE
<2> <VAR> (<PHASE>) (<COMPONENT>) <DEV>	<1> <VAR> (<PHASE>) (<COMPONENT>) <DEV>
<1> Q (<PHASE>) NO	<2> Q (<PHASE>) NO
<1> Q (<PHASE>) LO	<2> Q (<PHASE>) LO
<1> Q (<PHASE>) HI	<2> Q (<PHASE>) HI + V1 PART-OPEN
<1> Q (<PHASE>) (<COMPONENT>) REV	<2> Q (<PHASE>) (<COMPONENT>) REV

NOTE: This operator only applies for situations where the pressure at <1> is greater than that at <2>.
NOTE2: (...) denotes optional value.
NOTE3: <...> denotes a variable binding .

Fig. 11.1: A Closed Bypass Line

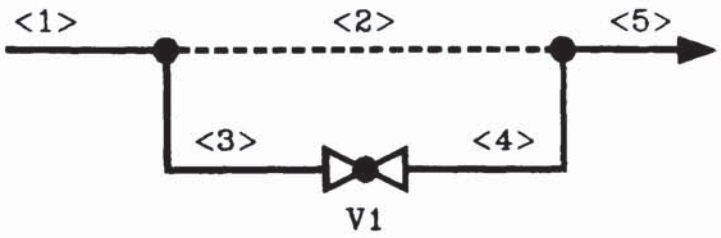


Fig. 11.2: A Fault Tree Produced Using Low-Level Modelling for a Closed Bypass Line

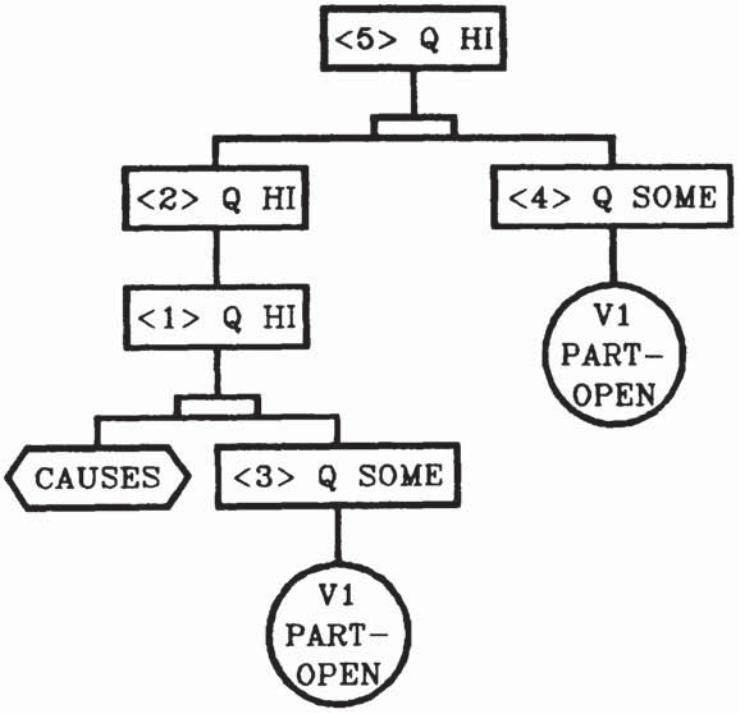
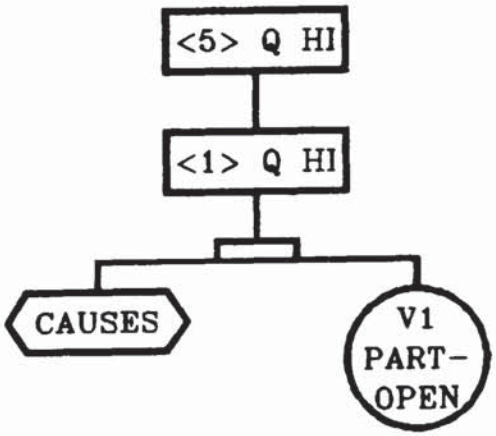


Fig. 11.3: A Fault Tree Produced Using High-level Modelling for a Closed Bypass Line



It would be impossible, however, to create a fully comprehensive range of high-level models because of the enormous number of different plant configurations that are possible. It is thus important that, even if a method uses high-level models for many plant sections, the use of lower level models is allowed should an unusual plant configuration occur. Such a method would be capable of producing good fault trees, whilst maintaining considerable representational flexibility.

11.1.4: Limitations of the Closed Bypass Operator

Careful analysis of the fault tree of Fig. 10.4 reveals one limitation of the closed bypass operator: no account is made of which minor units in line L2 are by-passed by the closed bypass and which are not. This is important when considering the failure mode V3 OPEN. The minimal cut set involving this failure mode should be: { V3 OPEN, LLTL2 FAIL-DANGER, RV1 FAIL-DANGER }. In fact the minimal cut set represented by the fault tree is { V3 OPEN, LLTL1 FAIL-DANGER, LLTL2 FAIL-DANGER, RV1 FAIL-DANGER }.

If the closed bypass operator were modified such that it takes account of which units are by-passed and which are not, then the correct result would be obtained: LLTL1 does not protect against V3 OPEN because it exists in parallel to the bypass line, whereas LLTL2 does protect against V3 OPEN because the final control element, TV1, is in series with the bypass line.

11.1.5: Limitations of Event Ordering

A limitation of the technique of assigning each loop an order for its controlled event and then allowing it to protect only that order of the controlled event becomes apparent if a third configuration of the ammonia let down problem is considered. In this configuration, the relief valve RV1 is not resized, but in combination with the trip loop venting system operated by HPTL6 it will handle gas breakthrough in line L2/L4 or liquid entrainment in the vent lines. RV1 and HPTL6 are each capable of handling all other causes of high pressure in vessel C2. This logic cannot be represented by the OFTS program as it stands: some input failure

event chains require that the protection logic is RV1 OR HPTL6, others require that the logic is RV1 AND HPTL6.

This problem is easily solved. If loops are allowed to protect more than one order of the controlled event, then order 2 protection logic may be defined as RV1 OR HPTL6; order 3 protection logic may be defined as RV1 AND HPTL6. The two major causal chains, gas breakthrough in line L2/L4 and liquid entrainment in the vent lines, would be assigned causal orders of 2; all other event chains would be assigned lower causal orders. Simplification of the fault tree at the synthesis stage to remove repetitions of events combined by AND gates would yield the correct fault tree.

11.1.6: Other Limitations of the OFTS Program

As it stands, there are a number of limitations to the hybrid method of fault tree synthesis and the OFTS program which implements this method. These limitations, listed here, are described more fully in section 11.2.

The OFTS program attempts only to model the plant units and control strategies which exist in the ammonia let down plant configurations described in chapters 5 and 10. No additional models of plant units have been created. In addition, cascade, ratio and manual control strategies are not modelled.

In the creation of input-output models for the major process units used in the ammonia let down problem, specific rather than generalized components have been used. No general code for component classification in the OFTS system has yet been developed.

Specific loop failure modes as described in section 6.6 have not yet been incorporated into the OFTS program. This mode of loop failure is thus overlooked.

11.1.7: Problems Encountered When Developing the OFTS Program

One of the biggest problems encountered during the development of

large expert system programs is that of testing that every rule in the rulebase executes when and how it should do. Some, indeed many, of the rules in a rulebase may not execute when a program is tested, so bugs in these rules may remain undetected. This problem is important in terms of quality assurance when developing an system which is to be applied in industry.

The approach taken during development of the OFTS program is to develop rules which aim to solve only particular test problems. These rules may successfully be tested. When expanding a rulebase which has been tested in this way, it is possible to maintain a reliable rulebase by applying the following rules:

1. Any new rules added to the rulebase must be tested by a suitable problem; or
2. Any new rules added to the rulebase must have exactly the same form as existing rules which have been tested.

For this reason, a few standard rule formats are used to represent knowledge in the OFTS program. By entering knowledge as rules which conform to these formats, it is possible for the user to ensure program reliability. Any rules added which do not conform to standard formats must be stringently tested. Major process unit models may be entered to conform with existing rule formats, but high-level process operators, for example, may not. Whenever new high-level process operators are added to the rulebase it is thus important that they are fully tested.

11.1.8: Significance of the OFTS Program

Fault trees for chemical process plants generated by existing automatic synthesis methods are poor in terms of size and structure, and the methods often applicable only to a narrow range of problems.

The methods described in this thesis result in small, structured fault trees, and are suitable for modelling a wide range of process plant configurations. This is demonstrated by developing and testing a program which implements these methods in order to generate fault trees for an

ammonia let down plant with two different control strategies and inherent assumptions about relief valve capacity. This tradeoff between fault tree size and program applicability is achieved without making any concessions in terms of fault tree accuracy.

This work demonstrates the ability of expert systems to model plant at several different levels of specificity. Such multi-levelled representation is capable of producing relatively simple fault trees whilst maintaining wide-scale applicability.

A new method of representing the interactions between process plant failure states is used. This method, the plant event network, represents deviatory states other than low or high, and is thus capable of modelling a wider range of plant failure modes. The plant event network uses an information flow approach, rather than mini-fault trees, in order to improve program efficiency.

Event ordering provides a means for the definition of magnitudes of variable deviations which is meaningful in terms of event controllability. Causal ordering allows the user to define the controllability of specific failure chains in terms of control loops, trip loops and protective systems. This, in conjunction with the definition of protection mini-trees, allows the user to accurately represent the plant control strategy and the propagation of failures through the plant.

11.2: RECOMMENDATIONS FOR FUTURE WORK

This thesis describes some basic methods that may be used to improve fault tree synthesis programs, and demonstrates their applicability by using a computer program which incorporates these methods to produce fault trees for a specific example. In order for a more robust computer program to be produced a number of topics need to be researched further; these are described in the ensuing sections. The major need, however, is for the application of the method to a wide range of test problems. Such a programme of testing would require that a large number of plant units, configurations and control strategies would be modelled, and many of the problems, described below, solved.

11.2.1: Major Process Unit Models

Before the OFTS program can be directly applied to a plant, input-output and failure mode equations for the plant's major process units are required. It is thus necessary that a library of models for such units is developed. There are several ways of achieving this:

1. Hand-writing of high-level rules such as those used to represent the gas-liquid separation and flash units in the OFTS system.
2. The use of a rule-building program which generates high-level rules for the representation of major process units from a list of input-output and failure mode equations entered by the user. Such rules may be added to a permanent library.
3. The use of low-level rules which are capable of modelling a wide range of process units.

Fig. 11.4 shows a rule which uses the OPS5 "BUILD" action to create a rule which could be added to the OFTS rulebase to represent an input-output equation for a major process unit. This rule requires only that the user define the input-output and failure mode equations for the unit to be modelled.

The possibility of using low-level rules to represent a large number of plant items arises because of the fact that many major process units have similar formats and have a large number of common input-output and failure mode equations; this is the case for the two major process units modelled by the OFTS program. A low-level modelling method would require that several types of rule are used to model units. Possible types of rule that may be used are:

1. Format rules, which model the format and connections of the unit;
2. Function rules, which model failures of the unit to achieve its functions;
3. Chemistry rules, which model the specific chemistry involved in the unit;
4. Unit specific rules, which model any failure modes specific to

certain types of unit.

Fig. 11.4: A Rule Which Could Form Part of a Rule-Building System for Producing Major Process Unit Models

```
(P BUILD-LHS-INPUT-RHS-INTERNAL
   {(EQUATION ^IN-NODE <IN-NODE> ^IN-VAR <IN-VAR> ^IN-PHASE
<IN-PHASE> ^IN-COMP <IN-COMP> ^IN-DEV <IN-DEV> ^OUT-NODE <OUT-NODE>
^OUT-VAR <OUT-VAR> ^OUT-PHASE <OUT-PHASE> ^OUT-COMP <OUT-COMP>
^OUT-DEV <OUT-DEV>)<EQN>}
   (TYPE ^NAME <UNIT-TYPE>)
   (NODE ^NAME <IN-NODE> ^IS INTERNAL)
   (NODE ^NAME <OUT-NODE> ^IS INPUT)
   {(RULE-COUNTER ^COUNT <COUNT>)<RC>}
-->
   (REMOVE <EQN>)
   (MODIFY <RC> ^COUNT (COMPUTE <COUNT> + 1))
   (BUILD \\ <UNIT-TYPE> \\ <COUNT>
       (TASK ^MAJOR EVENT-LINKAGES)
       (NODE ^NUMBER <INTERNAL> ^NODE-OF <UNIT> ^MPU-DETAILS \\
<IN-NODE>)
       (MAJOR-UNIT ^NAME <UNIT> ^ISA \\ <UNIT-TYPE>)
       (NODE ^IS TEMPORARY ^NODE-OF <UNIT> ^MPU-DETAILS <OUT-NODE>
^FLOW-FROM <FEED>)
       (EVENT ^NUMBER <CONSEQ> ^NODE <FEED> ^VAR \\ <OUT-VAR>
^PHASE \\ <OUT-PHASE> ^COMPONENT \\ <OUT-COMP> ^DEV \\ <OUT-DEV>)
       (EVENT ^NUMBER <CAUSE> ^NODE <INTERNAL> ^VAR \\ <IN-VAR>
^PHASE \\ <IN-PHASE> ^COMPONENT \\ <IN-COMP> ^DEV \\ <IN-DEV>)
      -(GATE ^ISA OR ^CONSEQ-OF <CONSEQ> ^CAUSED-BY <CAUSE>)
       {(GATE-COUNTER ^COUNT <COUNT>)<GC>}
    -->
       (MODIFY <GC> ^COUNT (COMPUTE <COUNT> + 1))
       (MAKE GATE ^ISA OR ^NUMBER <COUNT> ^CONSEQ-OF <CONSEQ>
^CAUSED-BY <CAUSE>)))
```

In order to use such low-level rules a system of chemical component classification is required. Such a system would allow the user to define

which of the specific components used in a particular problem fit certain component categories as described for each type of unit being modelled. In the ammonia let down problem it is necessary to define the relative volatilities of components entering the flash vessel. For some types of units this classification is insufficient, more complex data may be required.

11.2.2: Functional Area Modelling

One of the conclusions reached during this research is that it is possible to represent plant at a number of different levels. It may be possible that modelling can take place at higher levels than those used in this thesis. Two possible levels are identified:

1. Functional areas, which include a major process unit, pipelines, ancillary process units and control systems.
2. Plant areas, which include several major process units, ancillary pipework and units, and control systems.

Modelling on these levels allows the user to enter reduced fault trees to achieve better fault tree structure and size. Such models could only be used for common "standard" configurations, but rules could be written which are sophisticated enough to allow variations and certain additional features to be incorporated into standard models.

11.2.3: Additional Failure Modes of Control Loops and Protective Systems

Additional work on specific failure modes of loops is required to ensure that no event chains are overlooked. At present, specific loop failure modes are not modelled by the OFTS program.

11.2.4 Using Default Values for Causal and Limiting Orders

In order to improve representational efficiency for causal and limiting orders, a method of using default values could be used. If an event chain conforms to preset default values, then that chain need not be explicitly represented. The default values used would probably be a causal order of 0 and the maximum limiting order value allowed. This

representation would require a slightly more complex fault tree synthesis algorithm, but would considerably reduce computer memory usage.

11.2.5: Ordering of Combinatory Failure Event Chains

At present the OFTS program only allows single event chains to be ordered. Event chains may themselves contain combinatory input branches, but the program does not allow combinations of event chains to be ordered. It is possible that, taken in pairs, some combinations of event chains may have higher causal or limiting order values than taken singly.

Such combinations are probably quite rare. If it is assumed that the probabilities of event chains are all reasonably low, then combinations of event chains may usually be ignored. There are two situations, however, where it may be beneficial to model combinatory chains:

1. Where two chains have a common mode event;
2. Where one of the chains has a relatively high probability.

There may be a case for further research of this to see whether the effects of combinatory event chains should be modelled.

11.2.6: Modelling Manual Control and Operator Actions

The OFTS program does not model manual control loops, or operator responses to alarms. The normal way of modelling operator control is to represent the actions that the operator must perform in response to, say, an alarm and to generate failure modes as inputs to an OR gate. Such failure modes correspond to the failure of the operator to correctly perform the required actions. This is represented by equation 11.1:

```
MANUAL LOOP FAILURE= FAILURE OF ALARM
                   + FAILURE OF OPERATOR TO RESPOND TO ALARM
                   + FAILURE OF OPERATOR ACTION 1
                   + FAILURE OF OPERATOR ACTION 2
                   + ...
                   + FAILURE OF OPERATOR ACTION N  ...(11.1)
```

Standard failure probabilities for various operator failures may be used for quantitative fault tree evaluation. It would be relatively simple to model manual loops in this way for the OFTS program.

There are problems with this representation. No account is taken of sequential failure modes when the operator must carry out actions in a certain order. Also, no account is made of operator overload when many alarms occur concurrently. In such situations, operators may make several errors; this is in fact a possible common mode failure. Additionally, no account is taken of incorrect operator response whereby the operator not only fails to perform a necessary task, but actually performs the wrong task. There is a need for further work which attempts to model operator actions more accurately, and incorporates this modelling into a fault tree synthesis program such as OFTS.

11.2.7: Modelling Cascade and Ratio Control

At present, cascade and ratio control strategies are not modelled by the OFTS program. Incorporating these strategies into the program is, however, relatively simple. Cascaded control loops provide set points for other loops. It is possible to represent event chains involving cascaded loops by linking through deviations in the set point. For example, a control loop, "LOOP1", will give high output if its set point is higher than the normal design value. This is represented by equation 11.2:

CONTROLLED VARIABLE HI = LOOP1 SET-HI ...(11.2)

The effects of set point deviations on the controlling variable may be modelled similarly. If the set point is provided by a cascaded loop, "LOOP2", then causes of set point deviations may be identified:

LOOP1 SET-HI = LOOP2 FAILED-HI
+ LOOP2 SET-HI
+ LOOP 2 MEASURED VARIABLE DEVIATIONS ...(11.3)

The exact measured variable deviations that produce a high set point

output depend on the loop's configuration. Boundary conditions, representing loop failure modes, must be applied to the measured variable deviations input because a condition of this causal mode is that LOOP2 functions correctly.

Ratio control may be modelled similarly to any other indirect loops, defining two measured variables to represent the properties that must be kept in ratio. Correct loop action may be modelled such that a deviation of one variable may induce the same deviation of the other variable.

11.2.8: Taking Account of Process Dynamics

There are three areas where process dynamics may be important: control loop dynamics; common mode failures; speed of failure propagation.

Control loop dynamics are important when a loop responds too slowly to correct an input deviation. This is similar to an invariant failure of the loop. In most situations where a feedback control loop has a large time lag, feedback control is combined with feedforward control so that the loop can respond rapidly to certain deviations. However, usually only some of the input variables are measured by feedforward loops, the loop will still be susceptible to deviations of other input variables. Loop dynamics is not currently explicitly modelled in the OFTS system, but dynamics may be accounted for when causal orders are defined. If a loop responds only slowly to a certain event chain, then the chain should be defined as uncontrollable by that loop.

Repeated events in fault trees that would appear to result in common mode failures may not always do so if the process dynamics are unmatched. A single causal event may propagate through different event chains at different speeds, allowing one of the input deviations to an AND gate to be corrected before its combinatory deviation occurs. It may be possible to account for this by defining approximate event propagation times when defining causal and limiting orders. The current treatment, however, which assumes that common modes will occur concurrently, does produce "worst case" results, so any error is on the side of caution.

Event chain propagation speed is important when ranking the relative significance of plant failure modes. A failure event chain which propagates rapidly through the plant from causes to effects is less likely to be corrected than one which may take several times longer. Rapidly propagating event chains thus present a greater hazard. The technique of defining event chain propagation times described above may be useful for assessing relative speeds of failure propagation. The total time taken from cause to effect may be estimated by summing the individual event chain times.

11.2.9: Methods of Fault Tree Simplification

There are several techniques which may be used to simplify fault trees further. Expert systems may employ search methods to identify certain event combinations that may be simplified. Common mode failures, for example, may be identified and removed from their normal input position to form a single direct input to an AND gate. This process would enhance the visibility of failure logic, make quantitative analysis easier and reduce fault tree size, but it would disrupt the causal flow of the tree. This, and other tree simplification procedures that may be used to remove repeated events, would thus best be used after the first fault tree output stage, so that the user may identify the exact causal logic.

11.2.10: Identifying Common-Mode Failures

An important decision to make when generating fault trees is about how deeply to look at unit failure modes. The treatment used in the OFTS program is to define unit failure modes as primal events, but many of these could actually be analysed further. Further analysis will often not tell the user much about the plant and its failure modes, but it may be useful to consider the effects of wide-scale common mode failures on individual units. Utility failures and fire are examples of wide-scale common mode failures, because they may cause many units to fail simultaneously. It would be possible, for example, to classify units by their utility usage. This could be used to model the effects of utility failure by checking common mode failures that may result. This is an example of applying Bayes' Conditional Probability method, the probability

of a top event may be obtained by summing the probability assuming that common mode failures do not occur and the probabilities obtained for situations where common mode failures do occur.

11.3: OTHER USES OF THE OFTS METHOD

The fault tree synthesis method developed during this research is aimed primarily at hazard analysis and as an aid to hazard and operability studies. Fault trees are, however, also useful to real-time alarm analysis and fault diagnosis.

Fault trees produced by the OFTS program are suitable for use by alarm analysis and fault diagnosis programs, the major requirements of such programs being that fault trees are consistent, avoid cyclic logic, and maintain both local and global causality. Because the fault trees produced by this method are smaller than those produced by other computer-based methods, they would be a better starting point for analysis than those produced, for example, by the Lapp-Powers or Martin-Solis methodologies.

11.4: CONCLUSIONS

This research set out to:

1. Develop techniques for the reduction of chemical process plants in such a way as to simplify the fault trees produced;
2. Develop techniques for structuring fault trees such that they may easily be understood by human analysts;
3. Develop techniques for the representation of event magnitudes such that exact protective failure logics for a wide variety of plant configurations may be modelled;
4. Develop fault tree synthesis procedures which allow the user to make certain key decisions.

In addition, it was decided to investigate the use of expert systems as a means of implementing these techniques.

The main achievements of the work may be summarised as follows:

1. A functional method of plant reduction has been developed. This method represents process lines, control loops and protective systems as operators in order to reduce and structure the representation of chemical process plants. This method allows the inclusion of minor process units and process lines in expanded models of major process units. These expanded models may be derived automatically from standard process units models. The expanded unit models may then be used to reduce the number of nodes used to represent chemical process plant, and, in turn, to reduce the size of fault trees produced.
2. An intermediate representational stage, the plant event network, has been developed. This is an information flow network for the representation of process plant failure modes which incorporates deviations other than low or high. Included in the plant event network are boundary conditions, which may be associated with both events and gates. The plant event network may include both OR and AND gates.
3. A method of event ordering has been developed which allows the definition of the magnitude of deviations in a way which is meaningful in terms of event controllability by control loops and protective systems. In combination with this, protection mini-trees are used to define the exact failure protection logic for plant deviations.
4. A method, known as causal ordering, which allows the user to define the exact levels of controllability of failure event chains leading to deviations of controlled or measured variables has been produced. This method involves the generation of failure event chains from the plant event network.
5. A method of generating fault trees from the plant event network, ordered failure event chains and failure protection mini-trees has been developed.

The fault tree synthesis package has been implemented on a computer using the OPS5 expert system programming language. The package contains knowledge about the failure and operating modes of major and minor process units, process lines, control and trip loops, and

pressure relief systems.

The methods developed in this research have been applied to a test problem involving two major process units, three control loops, a pressure relief system, and up to six trip systems. The computer package has been used to generate fault trees for two top events in this problem. The fault trees obtained are a significant improvement on fault trees produced by other computer-based methods; they are small, well structured, accurate and complete.

The results obtained have shown that computer-aided fault tree synthesis for chemical process plants from standard process unit models is capable of producing clear, concise and accurate fault trees comparable to those produced by human analysts. Fault trees obtained in this way may be used as a significant aid to hazard and operability studies. In addition, they may be used to estimate plant hazard probabilities, for the evaluation of different plant configurations and control strategies, and may form the basis of on-line hazard aversion and alarm analysis programs.

LIST OF REFERENCES

1. Marshall, V.C.; "Major Chemical Hazards", Ellis Harwood Ltd., London, 1987, p.7.

2. Health and Safety Executive; "Advisory Commitee on Major Hazards - First Report", HMSO, London, 1976.

3. Powers G.J., Tompkins F.C.; "Fault Tree Synthesis for Chemical Processes", A.I.Ch.E. Journal Vol. 20 No.2, March 1974, pp 376-387.

4. Lees, F.P.; "Loss Prevention in the Process Industries", Vol. 1, Butterworths, London, 1980, p. 2.

5. Anon.; "The Flixborough Disaster - Report of the Court of Inquiry", HMSO, London, 1975.

6. Anon.; "The Technical Lessons of Flixborough", A Symposium in Dec. 1975, I.Chem.E, 1976.

7. Health and Safety Executive; "A Guide to the HSW Act", HMSO, London, 1980.

8. Health and Safety Executive; "Advisory Commitee on Major Hazards - Second Report", HMSO, London, 1979.

9. Health and Safety Executive; "Advisory Commitee on Major Hazards - Third Report", HMSO, London, 1980.

10. Powers, G.J. and Tompkins, F.C.; "Fault Tree Synthesis", NATO Conf. on Reliability, Liverpool, 1973.

11. IEEE Standards Committee, "IEEE Trial-Use Guide: General Principles for Reliability Analysis of Nuclear Power Generating Station Protection Systems", IEEE Std. 352, 1972.

12. Garner, N.R., Huetinck, J.A.; "Equipment Ageing Analysis: An Extension to Reliability", A.I.Ch.E Symp. Reliability of Operation in the Process Industries, San Juan, 1970.

13. Recht, J.L.; "Systems Safety Analysis: Failure Mode and Effect", National safety News, 1966.

14. Crosetti, P.A.; "Fault Tree Analysis with Probability Evaluation", Douglas United Nuclear Inc., Richland, Washington, 1971.

15. Lawley, H.G.; "Operability Studies and Hazard Analysis", Loss Prevention, CEP, A.I.Ch.E., Vol. 8, 1974, pp.105-116.

16. Kletz, T.A.; "HAZOP and HAZAN", I.Chem.E., London, 1983.

17. Roach, J., Lees, F.P.,; "Some Features of and Activities in Hazard and Operability (HAZOP) Studies"; The Chem. Engineer, I.Chem.E., Oct. 1981, pp.

456-462.

18. Recht, J.L.; "Systems Safety Analysis: An Introduction", National Safety News, Dec. 1965.

19. U.S. Nuclear Regulatory Commission; "Reactor Safety Study - An Assessment of Accident Risk in U.S. Commercial Nuclear Power Plants", WASH-1400 (NUREG - 75/10/014).

20. Lihou, D.A.; "Computer Aided Operability Studies for Loss Control", 3rd Int. Symp. in Loss Prevention and Safety Promotion in the Process Industries, Basle, Switzerland, Vol. 2, Sept. 1980, p.579.

21. Dow Chemical Co.; "Fire and Explosion Index Hazard Classification Guide - 5th Edition", Chemical Engineering Progress technical manual (A.I.Ch.E), 1981.

22. Lewis, D.J.; "The Mond Fire, Explosion and Toxicity Index: a Development of the Dow Index", A.I.Ch.E. Loss Prevention Symposium, Houston, April 1979.

23. Lewis, D.J.; "The Mond Fire, Explosion and Toxicity Index Applied to Plant Layout and Spacing", Loss Prevention No.13 (A.I.Ch.E.), p.20.

24. ibid reference 16, p.8.

25. Chemical Industries Safety and Health Council; "A Guide for Hazard and Operability Studies", Chemical Industries Association Ltd., London, 1977, p.7.

26. ibid. reference 25, p.6.

27. Katz, D.L.; "A View of Safety in the Petroleum Industry", National Academy of Eng. Symp.: Public Safety: A Growing Factor in Modern Design", Washington D.C., U.S.A., 1970.

28. Fussell, J.B; "Fault Tree Analysis - Concepts and Techniques", NATO Adv. Study, Inst. on Generic Techniques of System Reliability Assessment, Nordhoff Publishing Co., Liverpool, July 1973, pp. 133-162.

29. Spiegelman, A.; "Risk Evaluation of Chemical Plants", Loss Prevention, CEP, A.I.Ch.E., Vol. 3, 1969, pp. 1-10.

30. Browning, R.L.; "Use of Fault Tree To Check Safeguards", Loss Precention, CEP, A.I.Ch.E., Vol. 12, 1979, pp. 20-26.

31. Hill, F.J., Peterson, G.R.; "Introduction to Switching Theory and Logical Design", Wiley, New York, 1968.

32. Kletz, T.A.; "Plant Instruments: Which Ones Don't Work and Why", Loss Prevention, CEP, A.I.Ch.E., Vol. 13, 1980, pp. 68-71.

33. Browning, R.L.; "Use of a Fault Tree to Check Safeguards", Loss

Prevention, CEP, A.I.Ch.E., Vol. 12, 1979.
34. Fussell, J.B.; "A Formal Methodology for Fault Tree Construction", Nucl. Sci. Eng., Vol. 52, 1973, pp. 421-432.
35. Powers, G.J.; "Safety and Reliability Analysis", Couse Notes in Advanced Process Synthesis, Mass. Inst. Technol., Cambridge, 1973.
36. Martin-Solis, G.A., Andow, P.K., Lees, F.P.; "Fault Tree Synthesis for Design and Real-Time Applications", Trans. I.Chem.E., Vol. 60, 1982, pp. 14-25.
37. Salem, S.L., Apostolakis, G.E., Okrent, D; "A New Methodology For The Computer-Aided Construction of Fault Trees", Annals of Nuclear Energy, Vol. 4, 1977, pp. 417-433.
38. Salem, S.L., Apostolakis, G.E.; "The CAT Methodology for Fault Tree Construction", in Apostolakis, G.E., Garribba, S., Volta, G. (Eds.); "Synthesis and Analysis Methods for Safety and Reliability Studues", Plenum, 1978.
39. Salem, S.L., Wu, J.S., Apostolakis, G.E.; "Decision Table Development and Application to the Construction of Fault Trees", Nucl. Technol., Vol. 42, pp. 51-64, 1979.
40. Fussell, J.B., Burdick, G.R. (Eds.); "Nuclear Systems Reliability Engineering and Risk Assessment", SIAM, 1977.
41. Wu, J.S., Salem, S.L., Apostolakis, G.E.; "The Use of decision Tables in the Systematic Construction of Fault Trees", in reference 40.
42. Berenblut, B.J., Whitehouse, H.B.; The Chemical Engineer, I.Chem.E., Vol. 318,1977, p. 175.
43. Lapp, S.A., Powers, G.J.; "The Synthesis of Fault Trees", in reference 40.
44. Lapp, S.A., Powers, G.J.; "Computer-Aided Synthesis of Fault Trees", IEEE Trans. Reliability, Vol. R-26, 1977, pp. 2-13.
45. Lapp, S.A., Powers, G.J.; "Update of the Lapp-Powers Fault Tree Synthesis Algorithm", IEEE Trans. Reliability, Vol. R-28, 1979, pp. 12-14.
46. Allen, D.J., Madhava Rao, M.S.; "New Algorithms for the Synthesis and Analysis of Fault Trees"; Ind. Eng. Chem. Fundamentals, Vol. 19, 1980.
47. Shafaghi, A., Andow, P.K., Lees, F.P.; "Fault Tree Synthesis Based on Control Loop Structure", Chem. Eng. Res. Design., Vol. 62, 1984, pp. 101-110.
48. Lihou, D.A.; "Computer Aided Operability Study", Loss Prevention Bulletin No. 051, I.Chem.E, 1983.
49. Lihou, D.A.; "Aiding Process plant Operators in Fault Finding and

Corrective Action", in Human Detection and Diagnosis of System Failures, Rasmussen, J., Rouse, W.B. (Eds.); Plenum Press, 1980, pp. 501-521.

50. Lihou, D.A.; "Effective Use of Failure and Success Data", Lecture 8, Short Course on Hazard Evaluation Using Personal Computers, Lihou Loss Prevention Services Ltd./Dept. of Chem. Engg., University of Aston, May 1986.

51. Anyakora, S.N., Engel, G.F.M., Lees, F.P.; The Chem. Engr., I.Chem.E., No. 255, p. 396, 1971.

52. Lihou, D.A., Jones, M.C.; "CAFOS - The Computer Aid for Operability Studies", I.Chem.E. Symposium Series No. 97, I.Chem.E., 1986, pp. 249-260.

53. Fussell, J.B., Henry, E.B., Marshall, N.H.; "MOCUS - A Computer Program to Obtain Minimal Sets From Fault Trees", ANCR-1156, Aerojet Nuclear Co., Idaho Falls, Idaho, 1974.

54. Fussell, J.B., Vesely, W.E.; "A New Methodology For Obtaining Cut Sets From Fault Trees", Trans. Amer. Nucl. Soc., Vol. 15, 1972, p. 794.

55. Bengiamin, N.W., Brown, B.A., Schenck, K.F.; "An Efficient Algorithm for Reducing the Complexity of Computation in Fault Tree Analysis", IEEE Trans. on Nuclear Science, Vol. NS-23, Oct. 1976, pp. 1442-1446.

56. Caldarola, L., Wickenhouser, W.; "The Karlsruhe Computer Program for the Evaluation of the Availability and Reliability of Complex Repairable Systems", Nucl. Engg. & Design, Vol. 43, pp. 463-470.

57. Caldarola, L., Wickenhouser, W.; "Recent Advances in Fault Tree Methodology at Karlsruhe", ibid reference 40, pp. 518-542.

58. Rasmusson, D.M., Marshall, N.H.; "FATRAM - A Core Efficient Cut Set Algorithm", IEEE Trans. on Reliability, Vol. R-37, Oct. 1978, pp. 250-253.

59. Limnios, N., Ziani, R.; "An Algorithm for Reducing Cut Sets in Fault Tree Analysis", IEEE Trans. on Reliability, Vol. R-35, Dec. 1986, pp. 559-561.

60. Semanderes, S.N.; "ERLAFT - A Computer Program for the Efficient Logic Reduction Analysis of Fault Trees", IEEE Trans. on Nucl. Sci., Vol. NS-18, No. 1, Feb. 1971, pp. 481-487.

61. Wheeler, D.B., Hsuan, J.S., Duersch, R.R, Roe, G.M.; "Fault Tree Analysis Using Bit Manipulation", IEEE Trans. on Reliability, Vol. R-26, June 1977, pp. 95-99.

62. Bennetts, R.G.; "On the Analysis of Fault Trees", IEEE Trans. on Reliability, Vol. R-24, Aug. 1975, pp. 175-185.

63. Vesely, W.E.; "Analysis of Fault Trees By Kinetic Tree Theory", IN-1330, Idaho Nuclear Corp., Idaho Falls, Idaho, Oct. 1969.

64. Vesely, W.E.; "A Time-Dependent Methodology for Fault Tree Analysis", Nucl. Engg. and Design, Vol. 13, Aug. 1970, pp. 337-360.
65. Vesely, W.E., Narum, R.E.; "PREP and KITT Computer Code for the Automatic Evaluation of a Fault Tree", IN-1349, Idaho Nuclear Corp., Idaho Falls, Idaho, 1970.
66. Bennetts, R.G.; "Analysis of Reliability Block Diagrams by Boolean Technique", IEEE Trans. on Reliability, Vol. R-31, June 1982, pp. 159-166.
67. Jiongsheng, L.; "A New Approach for Fault Tree Analysis", Scientia Simica Series A, Vol. 25, Sept. 1982, pp. 983-992.
68. Ramadaan, S.Y.; "Reliability Analysis for Hazard and Operability Studies", PHD Thesis, University of Aston, Birmingham, UK, 1987.
69. Page, L.B., Perry, J.E.; "A Simple Approach to Fault Tree Probabilities", Computers & Chem. Engg., Vol. 10, No. 3, 1986, pp. 249-257.
70. Page, L.B., Perry, J.E.; "An Algorithm for Exact Fault Tree Probabilities Without Cut Sets", IEEE Trans. on Reliability, Vol. R-35, Dec. 1986, pp. 544-558.
71. Fisal, Z.B.; "Real Time Process Plant Fault Diagnosis", PHD Thesis, University of Aston, Birmingham, UK, 1989.
72. Ulerich, N.H., Powers, G.J.; "On-Line Hazard Aversion and Fault Diagnosis in Chemical Processes: The Digraph and Fault Tree Method", IEEE Trans. on Reliability, Vol. 37, No. 2, June 1988.
73. Rich, E.; "Artificial Intelligence", McGraw-Hill, Singapore, 1983, p. 5.
74. Newell, A., Simon, H.A.; "Computer Science as Empirical Inquiry: Symbols and Structures", Communications of the ACM, Vol. 19, No. 3, March 1976.
75. Simon, H.A.; "The Sciences of the Artificial, 2nd Ed.", MIT Press, Cambridge, Mass, 1981.
76. ibid reference 73, p. 58.
77. Pohl, I.; "Bi-directional Search", in Machine Intelligence Vol. 6, Meltzer, B,, Michie, D. (Eds.), American Elsevier, New York, 1971.
78. de Champeaux, D., Sint, L,; "An Improved Bi-directional Heuristic Search Algorithm", Journal ACM, Vol. 24, 1977.
79. Hewitt, C.; "PLANNER: A Language for Proving Theorems in Robots", Proc. IJCAI, Vol. 2, 1971.
80. Davis, Buchanan, B.G., Shortliffe, E.H.; "Production Rules as a Representation for a Knowledge-based Consultation Program", Artificial Intelligence, Vol. 8, No. 1, 1977.
81. McDermott, J.; "R1: A Rule-based Configurer of Computer Systems",

Artificial Intelligence, Vol. 19, Sept. 1982, pp. 39-88.

82. Shortliffe, E.H.; "Computer-based Medical Consultations: MYCIN", Elsevier, New York, 1976.

83. Davis, R.; "Experiments in Communication with a Knowledge-based Expert System", in Sime, M.E., Coombs, M.J. (Eds.), "Designing for Human - Computer Communication", Academic Press, London.

84. Swartout, W.R.; "Explaining and Justifying Expert Consulting Programs", Proc. IJCAI, Vol. 7, 1981.

85. McCarthy, J.; "Recursive Functions of Symbolic Expressions and their Computation by Machine", Communications of the ACM, Vol. 7, pp. 184-195, April 1960

86. van Melle, W., Scott, A.C., Bennett, J.S., Peairs, M.A.; "The EMYCIN Manual", Technical Report, Heuristic Programming Project, Stanford University, 1981.

87. Harmon, P., King, D.; "Artificial Intelligence in Business", Wiley Press, New York, 1985.

88. Warren, D.H.D., Pereira, L.M.; "Prolog - The Language and its Implementation Compared to Lisp", Proc. Symp. on Artificial Intelligence and Programming Languages, SIGPLAN Notices Vol. 12 No. 8, and SIGART Newsletter, Vol. 64, 1977.

89. Lihou, D.A.; "Cause: Consequence Equation From HAZOP", Lecture 10, Short Course on Hazard Evaluation Using Personal Computers, Lihou Loss Prevention Services Ltd./Dept. of Chem. Engg., University of Aston, May 1986.

90. Lawley, H.G., Kletz, T.A.; "High-Pressure-Trip Systems for Vessel Protection", Chemical Engg., I.Chem.E., May, 1975, pp. 81-88.

APPENDICES

APPENDIX A

A. The OPS5 Programming Language

A1: Language Structure

The OPS5 language consists of two key components: a data base called **working memory** and **production rules** that manipulate this data base. The language's run-time system uses a recognize-act cycle to process the contents of working memory and the productions rules.

OPS5 uses a run-time system to select and execute production rules, and to maintain working memory. A command interpreter allows the entry of high level commands which affect the monitoring of program execution, conflict resolution strategy, backtracking and trace functions. For a description of the OPS5 commands available see Table A.1.

A1.1: Working Memory

Working memory is a global data base consisting of **elements** used to describe a problem. Each element consists of a **class name**, a list of associated **attributes** and their **values**. The class name classifies the element according to what type of information the element contains. The attributes and values describe the element's characteristics. Elements with the same class name have the same attributes, but their values may be different.

An attribute consists of the attribute operator, ^, and an attribute name. An attribute's value will be "NIL" until it is specified otherwise. Values for attributes are represented by **atoms**. There are three basic types of atom:

1. Symbols, which are atoms which do not have a numeric value. These consist of strings of characters and numbers.
2. Integers, which may be signed.
3. Floating-point numbers, to a precision of approximately seven digits.

Fig. A.1: OPS5 Command Interpreter Commands

Command	Description
@	Opens a file containing OPS5 commands and executes them
ADDSTATE	Adds the contents of a file produced using SAVESTATE to the current state of working memory and the conflict set
AFTER	Specifies the number of recognize-act cycles that must execute before a specified catcher halts program execution
BACK	Restores working memory and the conflict set to a previous state
CALL	Calls an external routine
CLOSEFILE	Closes the open files asociated with the specified file association name and disassociates the name from the files
CS	Displays the contents of the conflict set
DEFAULT	Sets the terminal or a file as the default input source or output destination
DISABLE	Disables specified run-time system features
ENABLE	Enables specified run-time system features
EXCISE	Disables specified productions
EXIT	Exits the command interpreter, returning to the operating system
MAKE	Creates a working-memory element
MATCHES	Displays the time tags of working-memory elements that match condition elements in a specified production

Fig. A.1: OPS5 Command Interpreter Commands (cont'd)

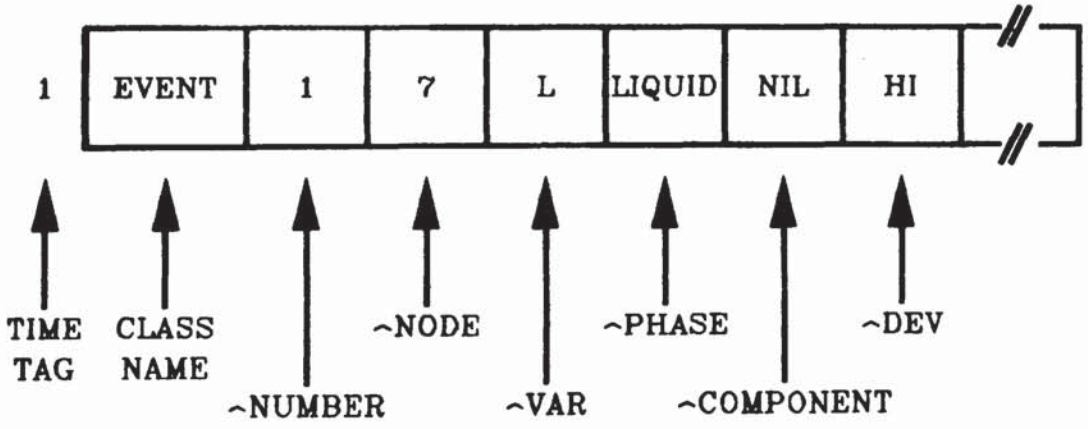
Command	Description
NEXT	Displays the instantiation the run-time system will select from the conflict set for the act phase of the next recognize-act cycle
OPENFILE	Opens a file and associates it with a file-identification name
PBREAK	Displays productions that have breakpoints set, sets breakpoints, or deletes breakpoints
PPWM	Displays working-memory elements that match a specified pattern
REMOVE	Deletes elements from working memory
REPORT	Generates a timimg report and a cause report that may be used for debugging and optimisation
RESTORESTATE	Clears then restores working memory and the conflict set to the state recorded in a file produced by SAVESTATE
RUN	Executes recognize-act cycles, the number of cycles to be executed may be specified
SAVESTATE	Copies the state of working memory and the conflict set to a file
STRATEGY	Sets or displays the conflict-resolution strategy
WATCH	Displays or sets the run-time system's trace level
WM	Displays the working-memory elements whose time tags are specified

Most attributes are **scalar attributes**, that is they take just one value. For each element class name, however, one **vector attribute** may be defined. The value of a vector attribute is a list of one or more atoms. The number of atoms that make up a vector attribute list may vary during program execution.

Attributes must be declared at the beginning of any program. The LITERALIZE declaration associates a class name with a list of attribute names, and tells the compiler to assign fields to the specified attribute names. The VECTOR-ATTRIBUTE declaration is used to define all vector attributes.

Each element is enclosed within parentheses and is assigned a unique time tag by the system. Time tags are consecutive integers that the system uses to evaluate recency during conflict resolution. A new time tag is assigned to each element added to working memory or modified by productions. Fig. A.1 shows how a working-memory element is represented by the system.

Fig. A.1: Internal representation of a Working-Memory Element



A1.2: Productions

Productions are the condition-action, or "IF...THEN...", statements of an OPS5 program. If data in working memory matches the conditions of a production, the run-time system can execute the production's actions. Each production consists of:

1. A production name;
2. A left-hand side (LHS);
3. A right-hand side (RHS);

The LHS and RHS are separated by the "-->" operator.

The left-hand side of a production contains one or more condition elements, a set of patterns that working memory must match. The run-time system compares the atoms in working memory elements against the corresponding patterns in condition elements. Condition elements are specified in a similar manner to working memory elements; combinations of class names, attributes and their values are specified. Negative conditions, represented by the "-" operator before the condition element, may be used to mean "no such element exists". A production is said to be satisfied, and may thus execute, when all of the positive conditions, and none of the negative conditions, are matched by elements in working memory.

Each component of a condition element, that is the class name, and attribute names where values are specified, are considered a term of the condition element. The values specified in the terms of a condition element may be atoms, variables, predicates, conjunctions, disjunctions, function calls, or a combination of these.

Atoms specified as a term imply that the working memory element's attribute value should match the conditional atom. The atom may be a symbol, integer or floating point; for a condition and a working memory atom to match they must be of the same type and the same value. Thus the integer 1 will not match the floating point 1.0. More than one value of a vector attribute list may be specified, if desired.

A variable is a symbol enclosed in angle brackets, "<" and ">", which may be matched by any atom in working memory. Variables refer to unknowns in a working memory element. The first time a variable is encountered in a production, it is bound to the atom in the working memory element matching the condition element. All subsequent occurrences of that variable in the production, either in the LHS or the RHS, represent

the same atom.

Predicates are operators that precede values, either constant atoms or variables, in condition element terms. Predicates test the atoms in working memory and produce a match if the atoms meet specific conditions. The predicates used in OPS5 are shown in Table. A.2.

Table A.2: OPS5 Predicates

Predicate	Meaning
=	equal to
<>	not equal to
<=>	same type as
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

A conjunction is a pattern of one or more conditional tests all of which must be true of an atom in a working memory element. A conjunction is equivalent to a logical AND. Conjunctions are specified by including the list of conditional tests within braces, "{" and "}". Tests that may make up conjunctions are: constants, variables, predicates, disjunctions. A conjunction is useful for binding a variable to an atom that satisfies one or more conditional tests. For example, suppose that the variable <NUMBER> is to be matched only by an integer between 5 and 10 inclusive, the following test is used:

```
^NUMBER { <NUMBER> >= 5 <= 10 }
```

where the integer is the value of the ^NUMBER attribute of an element.

A disjunction is a pattern containing a list of constant atoms. For the pattern to be matched, the working memory atom must match one of the atoms in the list. This test is equivalent to the logical OR. Disjunctions are specified by enclosing the list of atoms between double angle brackets, "<<" and ">>". The following test matches the integers 5, 6, 7, 8, 9, or 10:

```
^NUMBER << 5 6 7 8 9 10 >>
```

A term value may be obtained from a function call, either to the COMPUTE function or to an external function. Function calls perform an operation on any arguments specified along with the function call in brackets, and return once the function is complete. If an integer is bound to the variable <NUMBER>, then the following test checks that the value of the NUMBER attribute is the bound integer plus one:

```
^NUMBER (COMPUTE <NUMBER> + 1)
```

The COMPUTE function performs the necessary addition and returns the result as a condition. COMPUTE may also be used as a right-hand side action.

It is possible to bind working memory elements to variables, known as **element variables**, such that bound elements may be acted on by right-hand side actions. An element variable is specified by enclosing the variable name and a positive condition element in braces, for example:

```
{(COUNTER ^NUMBER <COUNT>)<COUNTER>}
```

The element variable <COUNTER> is bound to the particular element that satisfies this condition element.

The right-hand side of a production consists of one or more actions. Actions may perform the following operations:

1. Modify working memory.
2. Save and restore the state of working memory and the conflict

set.

3. Stop program execution.
4. Bind variables.
5. Manipulate files.
6. Write output.
7. Add productions to the executing program.
8. Call external subroutines.
9. Control loops.

An action includes the action name and its arguments enclosed in parentheses, for example:

```
(MAKE EVENT ^NUMBER 101 ^UNIT C1 ^FAILMODE LEAK-TO-ATMOS)
```

A list of OPS5 actions and their descriptions is presented in Table A.3. Most OPS5 actions require at least one argument. Arguments may consist of atoms, variables bound to atoms, element variables, or function calls that evaluate to atoms.

Variables may be used to represent an argument value in an action if the variable is previously bound to an atom. The variable may be bound either in a condition element, or by a BIND action.

Element variables may be included in right-hand side expressions as arguments to functions and actions. Element variables are bound by condition elements during the match phase of the recognize-act cycle, or by the CBIND action. When an element variable is used as an argument to an action or function call, the variable refers to the working memory element to which it is bound. For example, the following rule which increments a counter modifies the element bound to <COUNTER>.

Fig. A.2: Use of Binding in an OPS5 Rule to Increment a Counter

```
(P EXAMPLE-RULE-INCREMENTS-A-COUNTER
   {(COUNTER ^NUMBER <COUNT>)<COUNTER>}
-->
   (MODIFY <COUNTER> ^NUMBER (COMPUTE <COUNT> + 1)))
```

Table A.3: OPS5 Actions

Action	Description
ADDSTATE	Adds the contents of a file produced by the SAVESTATE action or command to the current state of working memory and the conflict set
AFTER	Specifies the number of recognize-act cycles that must execute before a specified catcher executes
BIND	Binds a variable to an atom
BUILD	Adds a new production to an executing program
CALL	Calls an external routine
CBIND	Binds an element variable to a working-memory element
CLOSEFILE	Closes the open files associated with specified file identification names and disassociates the names from the files
DEFAULT	Sets the terminal or a file as the default input source or output destination
HALT	Stops the run-time system from executing recognize-act cycles at a particular point during a program's execution
MAKE	Creates a working-memory element
MODIFY	Changes one or more atoms of a bound working-memory element
OPENFILE	Opens a file and associates it with a file-identification name
REMOVE	Deletes a working-memory element
RESTORESTATE	Clears and restores working memory and the conflict set to the state recorded in a file
SAVESTATE	Copies the state of working memory and the conflict set to a file
WRITE	Sends output from the program to the default output destination

If, say, this rule were matched by an element of class name COUNTER with value 3 of the NUMBER attribute, then the element would be modified by this rule to contain the value 4 for the NUMBER attribute.

An argument value may be evaluated by a call to an OPS5 function or an external function. A list of OPS5 functions is reproduced in Table A.4.

In addition to production rules, a STARTUP statement may be defined to initialize the conflict-resolution strategy, trace functions, working memory and the conflict set. A STARTUP statement may contain a list of actions and commands. The actions allowed are the same as those allowed as right-hand side actions for production rules; the commands are those that may be entered to the OPS5 command interpreter, see Table A.1.

A2: The Recognize–Act Cycle

The OPS5 run-time system uses a recognize-act cycle as shown in Fig. A.3 to control program execution. The cycle consists of the following steps:

1. **Match.** The elements in working memory are matched with the condition elements in the left-hand sides of the program's productions. The productions whose conditions are satisfied are available for selection and are placed in the conflict set.
2. **Conflict resolution.** A conflict-resolution strategy is employed to select one production for execution from the conflict set. The program halts if the conflict set is empty.
3. **Act.** The actions on the right-hand side of the selected productions are executed. If the RHS contains the HALT action, then program execution will halt.

A2.1: The Match Phase

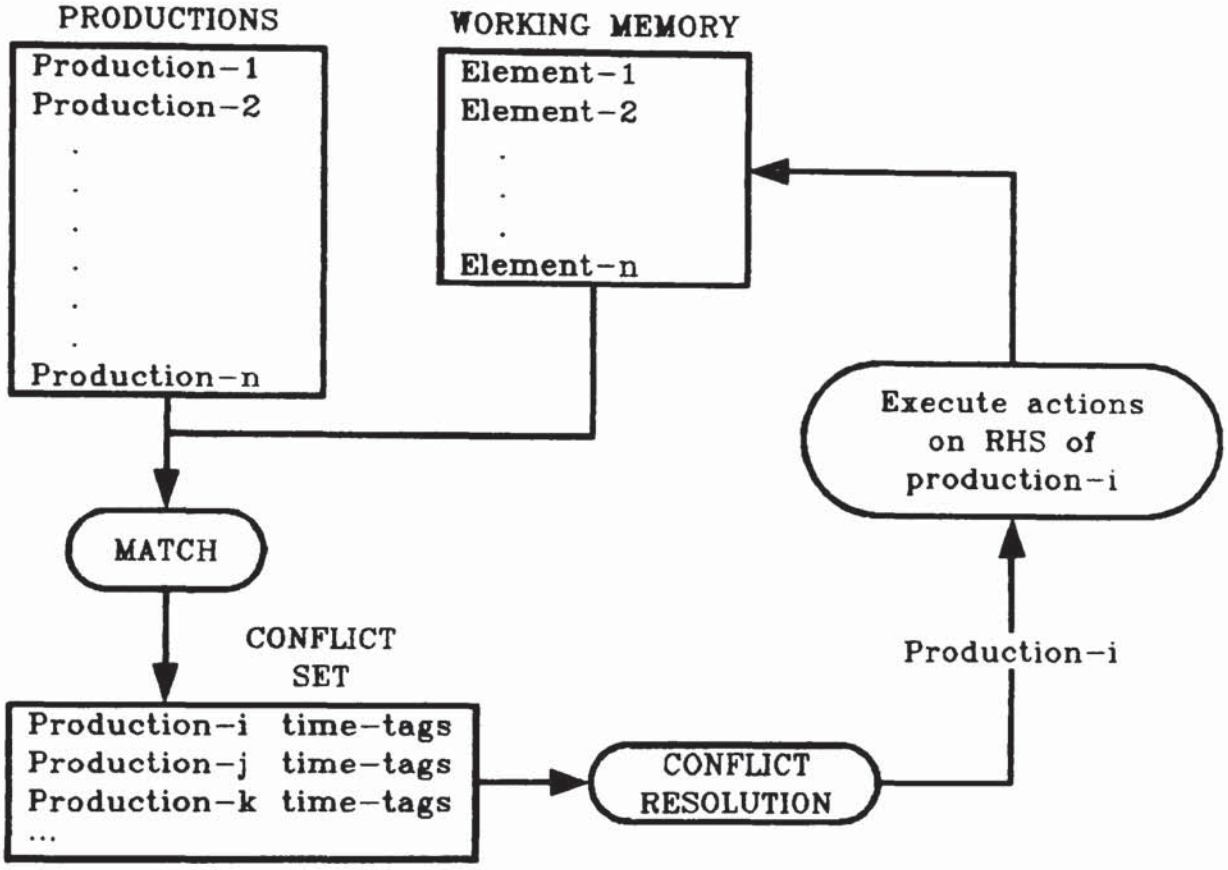
During the match phase of the recognize-act cycle, the run-time system compares the elements in working memory with the condition

Table A.4: OPS5 Functions

Function	Description
ACCEPT	Reads an atom or a list of atoms from a terminal or file.
ACCEPTLINE	Reads a line of input consisting of one or more atoms from the terminal or a file. A default value may be specified.
COMPUTE	Evaluates an arithmetic expression and returns the result.
CRLF	Causes the WRITE action to begin a new line of output.
GENATOM	Returns a unique system-generated atom.
LITVAL	Returns the integer that represents an attribute's field.
RJUST	Causes the WRITE action to right-justify output in a field of specified width.
SUBSTR	Copies a sequence of atoms from a working memory element to output produced by the WRITE action or to another working memory element created with the MAKE action or modified by the MODIFY action.
TABTO	Causes the WRITE action to start writing output in a specified column.

elements in each production's left-hand side. The conditions of a production are satisfied when every positive condition is matched by a working memory element, and when no working memory elements match negative condition elements.

Fig. A.3: The OPS5 Recognize-Act Cycle



As the left-hand sides of productions are satisfied, the run-time system creates a conflict set that contains records of the production name and the time tags of the working memory elements that match the condition elements. More than one set of working memory elements may satisfy a production's conditions. In this case, the production is included in the conflict set with each set of elements that match it. The production names and their satisfying elements are called **instantiations.**

A2.2: Conflict Resolution

During conflict resolution, the run-time system uses a strategy to select one of the instantiations from the conflict set. The selected

production is executed during the act phase.

There are two conflict-resolution strategies available in OPS5. These strategies are based on the following rules:

1. **Refraction.** An instantiation may be selected only once. This prevents the program from looping indefinitely on the same data;
2. **Recency.** Selects an instantiation that refers to the most recent data in working memory. This helps the system to create motion, and promotes depth-first rather than breadth-first searching. The system selects the instantiation with the highest time tags to ensure that the recency rule is applied;
3. **Specificity.** Selects an instantiation whose left-hand side is most specific. Specificity is determined by the number of conditional tests on a production's left-hand side.

The **Lexicographic-Sort** (LEX) conflict-resolution strategy used by OPS5 applies the refraction, recency and specificity rules. The LEX strategy is for programs that have no procedural element: they do not depend on the order in which rules are applied. The rules are applied as follows:

1. Apply refraction by removing from the conflict set the instantiations that the run-time system has selected for execution during a previous cycle.
2. Order the rest of the instantiations according to their recency and select the instantiation(s) with the highest level of recency.
3. If more than one instantiation is selected by stage 2, then order these instantiations according to their specificity and select the instantiation(s) with the highest level of specificity.
4. If more than one instantiation is selected by stage 3, then select one of these instantiations arbitrarily.

The **Means-Ends-Analysis** (MEA) conflict resolution strategy used by OPS5 contains an additional test over the LEX strategy. MEA is used for programs that deal with problems that may be divided into tasks; the first condition element is used to define the present goal or task. The rules

applied by MEA are as follows:

1. Apply refraction by removing from the conflict set the instantiations that the run-time system has selected for execution during a previous cycle.
2. Compare the first time tag of each instantiation still remaining in the conflict set and select the instantiation(s) with the most recent tag.
3. If more than one instantiation is selected by stage 2, then order the rest of the instantiations according to their overall recency and select the instantiation(s) with the highest level of recency.
4. If more than one instantiation is selected by stage 3, then order these instantiations according to their specificity and select the instantiation(s) with the highest level of specificity.
4. If more than one instantiation is selected by stage 4, then select one of these instantiations arbitrarily.

A2.3: The Act Phase

Once the run-time system has selected an instantiation, the recognize-act cycle enters the act phase. During this phase, variables are bound to values and the actions on the right-hand side of the selected production execute. Actions may modify working memory, write output to a file or the terminal, call an external subroutine, open or close files, or halt program execution. Once the act phase is complete, the match phase begins again.

The recognize-act cycle continues until either a HALT action is executed, or there are no instantiations in the conflict set.

APPENDIX B

B. VARIABLE NAMES AND UNIT FAILURE MODES USED IN THE OFTS SYSTEM

Table B.1: Variable Names Used in the OFTS Program

Variable Symbol	Variable Name
Q	Flow
P	Pressure
L	Level
T	Temperature
X	Composition

Table B.2: Component Names Used in the OFTS Program

Component Code	Component Name	Phase
AMM-LIQ	Ammonia Liquid	Liquid
AMM-GAS	Ammonia Gas	Gas
SYN-GAS	Synthesis Gas	Gas
SYN-GAS-SOLN	Synthesis Gas in solution	Liquid

Table B.3: Unit Failure Modes Used in the OPS5 Program

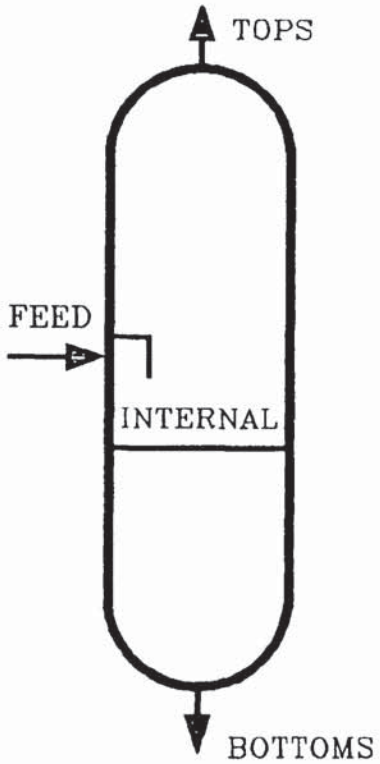
Unit	Failure Mode Code	Failure Mode Details
C1, C2, Lines	LEAK-TO-ATMOS	A leak from the unit or line to a lower pressure environment, such as the atmosphere.
C1, C2, Lines	EXT-FIRE	A fire external to the unit or line which causes heating of the contents.
Lines	BLOCKED	An uncontrollable blockage in the line or in a minor unit in the line.
Lines	PART-BLOCKED	A partial blockage, which produces controllable deviations.
Valves	OPEN	Valve open or partially open instead of closed.
Valves	CLOSED	Valve closed instead of open.
Loop or GROUP	FPROTECT	Failure of the loop or group to protect against deviation of the controlled variable.
Loop or GROUP	FAIL-SAFE	Loop or group activates spuriously.
Control Loop	FAILED-HI	Loop giving high output in terms of the controlled variable.
Control Loop	FAILED-LO	Loop giving low output in terms of the controlled variable.

APPENDIX C

C. INPUT-OUTPUT AND FAILURE MODE EQUATIONS USED FOR THE MAJOR PROCESS UNITS IN THE AMMONIA LET DOWN SYSTEM

C1: The Gas-liquid Separation Unit

Fig. C.1: A Vertical Gas-liquid Separator



Defined failure modes:

EXT-FIRE, representing an external fire;
LEAK-TO-ATMOS, representing a leak from a vessel or line to the atmosphere;
LEAK-FROM-ATMOS, representing a leak from the atmosphere to a vessel or line;
BURST, representing total rupture.

LEAK-TO-ATMOS is only valid for lines or vessels operating at greater than atmospheric pressure. LEAK-FROM-ATMOS is only valid for lines or vessels operating at less than atmospheric pressure.

Operating states:

atmospheric pressure or greater; or
less than atmospheric pressure.

In the following equations, the following conventions are used:

i represents any valid component;
i_{VC} represents a volatile component;
i_{NVC} represents a non-volatile component;
phase represents any valid phase.

Where one of these elements appears on both sides of an equation, the same value is bound to both sides.

Equations for Deviations of Variables at the Tops Node:

TOPS Q LIQUID SOME = INTERNAL L LIQUID FULL

TOPS Q LIQUID i SOME = INTERNAL L LIQUID FULL
(if component i normally exists at the internal node)

TOPS Q LIQUID i SOME = INTERNAL L LIQUID FULL
* INTERNAL X LIQUID i SOME
(if component i does not normally exist at the internal node)

TOPS X GAS i SOME = INTERNAL X GAS i SOME
(if the component i does not normally exist at the internal node)

TOPS Q GAS NO = INTERNAL P GAS LO

TOPS Q GAS LO = INTERNAL P GAS LO

TOPS Q GAS HI = INTERNAL P GAS HI

TOPS Q GAS REV = INTERNAL P GAS LO

TOPS T GAS LO = INTERNAL T GAS LO

TOPS T GAS HI = INTERNAL T GAS HI

TOPS X GAS i NO = INTERNAL X GAS i NO

TOPS X GAS i LO = INTERNAL X GAS i LO

TOPS X GAS i HI = INTERNAL X GAS i HI

Equations for Deviations of Variables at the Bottoms Node

BOTTOMS Q GAS SOME = INTERNAL L LIQUID NO

BOTTOMS Q GAS i SOME = INTERNAL L LIQUID NO
(if component i normally exists at internal node)

BOTTOMS Q GAS i SOME = INTERNAL L LIQUID NO
* INTERNAL X GAS i SOME
(if component i does not normally exist at the internal node)

BOTTOMS X LIQUID i SOME = INTERNAL X LIQUID i SOME

BOTTOMS Q LIQUID NO = INTERNAL L LIQUID NO

BOTTOMS Q LIQUID LO = INTERNAL L LIQUID LO

BOTTOMS Q LIQUID HI = INTERNAL L LIQUID HI

BOTTOMS Q REV = INTERNAL P LO

BOTTOMS T LIQUID LO = INTERNAL T LIQUID LO

BOTTOMS T LIQUID HI = INTERNAL T LIQUID HI

BOTTOMS X LIQUID i NO = INTERNAL X LIQUID i NO

BOTTOMS X LIQUID i LO = INTERNAL X LIQUID i LO

BOTTOMS X LIQUID i HI = INTERNAL X LIQUID i HI

Equations for Deviations of Variables at the Feed Node

FEED Q NO = INTERNAL P GAS HI

FEED Q LIQUID REV = INTERNAL L LIQUID FULL * INTERNAL P HI

FEED Q GAS REV = INTERNAL P GAS HI

FEED Q LO = INTERNAL P GAS HI

FEED Q HI = INTERNAL P GAS LO

Equations for Deviations of Variables at the Internal Node

INTERNAL L LIQUID NO = BURST + BOTTOMS Q LIQUID HI
+ FEED Q LIQUID NO

INTERNAL L LIQUID LO = LEAK-TO-ATMOS + BOTTOMS Q LIQUID HI
+ FEED Q LIQUID LO

INTERNAL L LIQUID HI = BOTTOMS Q LIQUID LO + FEED Q LIQUID HI

INTERNAL L LIQUID FULL = BOTTOMS Q LIQUID NO

INTERNAL P GAS LO = LEAK-TO-ATMOS + FEED Q GAS LO + TOPS Q GAS HI
+ FEED Q GAS REV + INTERNAL T GAS LO

INTERNAL P GAS HI = LEAK-FROM-ATMOS + FEED Q GAS HI
+ TOPS Q GAS LO + TOPS Q GAS REV
+ INTERNAL T GAS HI

INTERNAL T GAS LO = FEED T GAS LO

INTERNAL T GAS HI = FEED T GAS HI + EXT-FIRE

INTERNAL T LIQUID LO = FEED T LIQUID LO

INTERNAL T LIQUID HI = FEED T LIQUID HI

INTERNAL X phase i SOME = FEED X phase i SOME

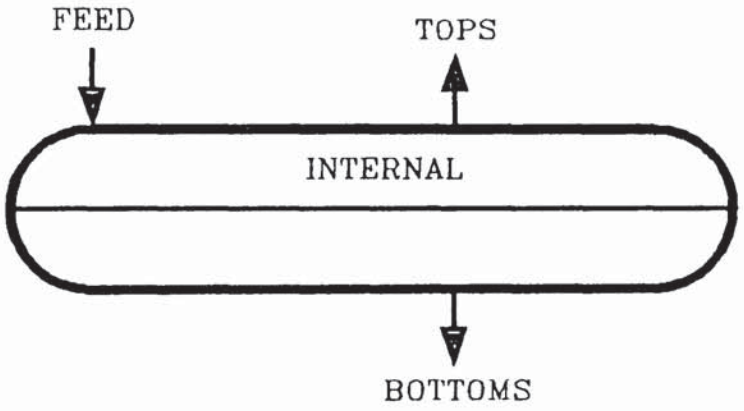
INTERNAL X phase i LO = FEED X phase i LO

INTERNAL X phase i HI = FEED X phase i HI

INTERNAL X phase i NO = FEED X phase i NO

C2: The Flash Vessel

Fig. C.2: A Horizontal Flash Vessel



Defined failure modes:

EXT-FIRE;
LEAK-TO-ATMOS;
LEAK-FROM-ATMOS;
BURST.

Operating states:

atmospheric pressure or greater;
less than atmospheric pressure

Equations for Deviations of Variables at the Tops Node

TOPS Q LIQUID SOME = INTERNAL L LIQUID FULL

TOPS Q LIQUID i SOME = INTERNAL L LIQUID FULL
(if component i normally exists at internal node)

TOPS Q LIQUID i SOME = INTERNAL L LIQUID FULL
* INTERNAL X LIQUID i SOME
(if component i does not normally exist at the internal node)

TOPS X GAS i SOME = INTERNAL X GAS i SOME

TOPS Q GAS NO = INTERNAL P GAS LO

TOPS Q GAS LO = INTERNAL P GAS LO

TOPS Q GAS HI = INTERNAL P GAS HI

TOPS Q GAS REV = INTERNAL P GAS LO

TOPS T GAS LO = INTERNAL T GAS LO

TOPS T GAS HI = INTERNAL T GAS HI

TOPS X GAS i NO = INTERNAL X GAS i NO

TOPS X GAS i LO = INTERNAL X GAS i LO

TOPS X GAS i HI = INTERNAL X GAS i HI

Equations for Deviations of Variables at the Bottoms Node

BOTTOMS Q GAS SOME = INTERNAL L LIQUID NO

BOTTOMS Q GAS i SOME = INTERNAL L LIQUID NO

(if component i normally exists at internal node)

BOTTOMS Q GAS i SOME = INTERNAL L LIQUID NO
* INTERNAL X GAS i SOME

(if component i does not normally exist at internal node)

BOTTOMS X LIQUID i SOME = INTERNAL X LIQUID i SOME

BOTTOMS Q LIQUID NO = INTERNAL L LIQUID NO

BOTTOMS Q LIQUID LO = INTERNAL L LIQUID LO

BOTTOMS Q LIQUID HI = INTERNAL L LIQUID HI

BOTTOMS Q LIQUID REV = INTERNAL L LIQUID LO * INTERNAL P GAS LO

BOTTOMS T LIQUID LO = INTERNAL T LIQUID LO

BOTTOMS T LIQUID HI = INTERNAL T LIQUID HI

BOTTOMS X LIQUID i NO = INTERNAL X LIQUID i NO

BOTTOMS X LIQUID i LO =INTERNAL X LIQUID i LO

BOTTOMS X LIQUID i HI =INTERNAL X LIQUID i HI

Equations for Deviations of Variables at the Feed Node

FEED Q NO = INTERNAL P GAS HI

FEED Q LIQUID REV = INTERNAL L LIQUID FULL

FEED Q GAS REV = INTERNAL P GAS HI

FEED Q LO = INTERNAL P GAS HI

FEED Q HI = INTERNAL P GAS LO

Equations for Deviations of Variables at the Internal Node

INTERNAL L LIQUID NO = BURST + BOTTOMS Q LIQUID HI
+ FEED Q LIQUID NO

INTERNAL L LIQUID LO = INTERNAL T GAS HI + BOTTOMS Q LIQUID HI
+ FEED Q LO + LEAK-TO-ATMOS + FEED X i_{NVC} LO

INTERNAL L LIQUID HI = BOTTOMS Q LIQUID LO + FEED Q HI
+ INTERNAL T GAS LO + FEED X i_{NVC} HI

INTERNAL L LIQUID FULL = BOTTOMS Q LIQUID NO

INTERNAL P GAS LO = LEAK-TO-ATMOS + FEED Q LO + TOPS Q GAS HI
+ FEED Q GAS REV + INTERNAL T GAS LO
+ FEED X ivc LO

INTERNAL P GAS HI = LEAK-FROM-ATMOS + FEED Q HI + TOPS Q GAS LO
+ TOPS Q GAS REV + INTERNAL T GAS HI
+ FEED X ivc HI

INTERNAL T phase LO = FEED T LO

INTERNAL T phase HI = FEED T HI + EXT-FIRE

INTERNAL X phase i NO = FEED X i NO

INTERNAL X phase i SOME = FEED X i SOME

INTERNAL X LIQUID ivc HI = INTERNAL P GAS HI + INTERNAL T LIQUID LO
+ FEED X ivc HI

INTERNAL X LIQUID i_{NVC} LO = INTERNAL P GAS HI
+ INTERNAL T LIQUID LO + FEED X i_{NVC} LO

INTERNAL X GAS i_{VC} LO = INTERNAL P GAS HI + INTERNAL T LIQUID LO
+ FEED X i_{VC} LO

INTERNAL X GAS i_{NVC} HI = INTERNAL P GAS H + INTERNAL T LIQUID LO
+ FEED x i_{NVC} HI

APPENDIX D

D. USING THE OFTS SYSTEM

D1: Problem Definition

Prior to running the OFTS program, the problem must be defined. Working-memory elements which describe major and minor process units, lines, nodes and components must be created. These elements should define the process connections and operating states. A simple way of creating this problem definition is to write a small program, such as the PROBLEM program listed in Appendix G. The problem program creates the working-memory elements which define the ammonia let down problem. Most of the elements to be created are added to working memory by a STARTUP statement. Production rules ask the user for further information, such as component existence at nodes and node type (MPU-DETAILS) information for nodes that are defined as nodes of a major process unit. An alternative route to problem definition would be to write a more general information prompting program which asks the user for all the information necessary for problem definition.

D2: Event Generation

The event generation rule subset creates event elements from the problem definition. Some additional information is required at this stage. The program asks the user

"Please enter the pressure (in atmospheres) of unit ..."

and similarly

"Please enter the pressure (in atmospheres) of line ...".

The user's answers are error checked, they must be entered in floating-point form, and then used to determine which of LEAK-TO-ATMOS and LEAK-FROM-ATMOS are valid events for each major process unit and line.

The event generation rule subset also asks the user about the existence of nodes which represent the problem's boundary, thus:

"Are there any nodes that exist at the plant boundary?
Enter the number of a node to be defined as a plant boundary.
Enter "N" if no more to be defined"

This prompt accepts the numbers of any nodes that are to be defined as plant boundaries. The input sequence is terminated by entering "N".

D3: Event Linking

The event linking rule subset is unit specific. No additional information is required by the subset which represents gas-liquid separation units, but the subset representing flash units does require information about the relative volatilities of the components involved. This information is obtained from the user by asking, for example

"What is the relative volatility of component syn-gas-soln in C2?
Either HIGH or LOW".

In this case the user should respond with "HIGH" as synthesis gas in solution is more volatile than ammonia liquid.

When control reaches the boundary condition generation part of event linking, a question asks whether each plant boundary event is valid or not, for example:

"Is event 1 Q LIQUID HI valid at the plant boundary? (Y/N Y is default)"

In response to this question the user may answer "Y" or press "RETURN" if the event is valid, otherwise answer "N". If the user answers "N" then all gates associated with the event, and the event element itself, are removed from working memory.

D4: Loop Specification

The loop specification rule subset sets up procedural loops which ask the user for information. The first question that is asked is

"Please enter numbers of any controlled nodes, "END" to finish".

This prompt allows the user to define the nodes at which controlled variables occur. This pre-selection of controlled nodes reduces the number of events that are matched by the event selection rule. This rule prints the details of events defined at controlled nodes, and asks the user if each particular event is a controlled event, thus:

"Selected event...
Node: ... Variable: ... Phase ... Component ... Deviation ...
IS THIS A CONTROLLED EVENT? (Y/N)"

If the user responds with "N" then another event is selected. If the user answers "Y" then the loop definition sequence is instantiated. This sequence prompts the user for information about the loops which control the selected events, thus:

"Enter loop name
for relief "loops" the name is simply the relief valve name"

"Enter loop class
Either CONTROL, TRIP or RELIEF"

"Enter loop type
Either DIRECT or INDIRECT"

A direct loop is one, such as a negative feedback control loop, for which the measured variable is the same as the controlled variable. If these variables are different, then the loop is termed indirect.

"Enter loop order in terms of the controlled event
The loop order is the order of the event which the loop acts to prevent"

"Enter the loop controlled action
This is the relationship between the controlling event and the controlled event
If an increase in the controlling event increases the controlled event then the action is POSITIVE
If an increase in the controlling event decreases the controlled event then the action is NEGATIVE"

The responses the the above questions define the LOOP element. Next, questions about the controlling variable are asked:

"Enter the controlling node"

"Enter the controlling variable"

"Enter the controlling component"

"Enter the controlling phase"

If the desired response to the component or phase prompts is "NIL", then just press "RETURN".

If the loop is an indirect loop, then similar prompts allow the user to define measured variables. In addition, the system asks the user for the initiating and saturating orders of the measured event, and for the measured action. As many measured variables as are required may be created for each loop.

Once the end of the data input sequence is reached, a rule asks

"Are there any more loops controlling this event? (Y/N)".

If the answer is "N" then the system loops back to event selection. If the answer is "Y" then the system loops back to the loop name prompt.

If a loop is entered which has previously been defined for a different event, then not all of the input prompts are repeated. Some of

the information, such as the loop class, type and controlled action, is obtained from the initial definition.

D5: Minor Process Units and Control Units Entry

The minor units rule subset accepts the names and details of any minor process units and control units that make up the plant. The initial question asked is

"Do you wish to define any minor units? (Y/N)"

If the answer is "Y" then the following minor units input sequence is instantiated:

"Enter unit name"
"Enter unit type (Type "HELP" for a current list of types)".

Responding to the above prompt with "HELP" instantiates a help rule which prints a list of the allowed unit types, and then re-instantiates the enter unit type prompting rule. After the unit type has been entered, if that type of unit requires an operating state to be defined then a further prompt asks for the unit status. For example, if a valve is entered, then the following prompt is produced:

"Enter valve status, either OPEN or CLOSED".

If the unit type indicates that the unit is a minor process unit then the next prompt will be:

"Enter the names of any line or bypass line that this unit forms part of "END" to end".

A similar prompt asks for the names of any loops that a control unit forms part of. As many lines or loops as are necessary may be identified in response to these questions. Typing "END" terminates input. At this point control returns to the initial subset prompt.

At the initial subset prompt if "N" is entered then the system checks whether all of the loops and closed lines in the problem have been fully defined. If further minor units are required, then control is returned to the initial prompt together with an error message such as

"Control loop LCL2 requires an actuator".

If the problem is sufficiently defined, then unit failure modes are created and linked to their consequences to complete the plant event network.

D6: Event Ordering

The event ordering subset prints event branches and requests that the user enters the causal and limiting orders of the event branch in terms of the top event. For example:

"CAUSAL AND LIMITING ORDERING SYSTEM
Link number: 17
The selected event number is 144
node: 8 var: P phase: GAS component: NIL dev: HI
The valid range of orders is 0 through 2
The causal event is number 120 which is linked by the following branch:
2 Q GAS SOME
7 L LIQUID NO
7 L LIQUID LO
The base event order is: 1
Enter the causal order, or INVALID"

This prompts the user to enter the causal order of the event branch printed. If the branch is considered either invalid or insignificant by the user, he may enter "INVALID" which removes the link. If the input is not "INVALID", then the next prompt asks for the limiting order of the branch:

"Enter the limiting order"

This process is repeated for every input event branch of every ordered event.

D7: Fault Tree Synthesis

The final stage of the OFTS system is fault tree synthesis. The only user input required is the top event details. The system prompts the user for the top event node, variable, phase, component and order. When a valid event has been entered, the system generates the fault tree automatically.

At present there is no graphical output program for the fault trees, but it is possible to output individual branches and to construct fault trees from these.

APPENDIX E

E. CAUSAL AND LIMITING ORDERS OF EVENT CHAINS AS USED IN TESTING THE OFTS SYSTEM

The following tables show the causal and limiting orders that were assigned to event chains when creating the fault trees of Figs. 10.1, 10.2, 10.4 and 10.5. Tables E.1 to E.6 show the causal and limiting orders used for the basic ammonia let down problem; Tables E.7 to E.12 show the causal and limiting orders used for the modified problem.

When these causal and limiting orders were assigned, most event chains were assigned the maximum limiting order value so that they would appear in the resulting fault trees. In a true analysis, some of these chains may be considered insignificant; these will thus be assigned a lower limiting order.

Table E.1: Causal and Limiting Orders for Event Chains Leading to 7 L LIQUID LO for the Basic Ammonia Let Down Problem

INPUT EVENT CHAIN						CAUSAL ORDER	LIMITING ORDER
L1 BLOCKED						0	1
C1 LEAK-TO-ATMOS						0	1
1 Q LIQUID LO	1 Q LO					0	1
1 Q LIQUID LO	1 Q LO	L1 PART-BLOCKED				0	1
1 Q LIQUID LO	1 Q LO	7 P GAS HI	L3 PART-BLOCKED			0	1
1 Q LIQUID LO	1 Q LO	7 P GAS HI	7 T GAS HI	7 T HI	1 T HI	0	1
1 Q LIQUID LO	1 Q LO	7 P GAS HI	7 T GAS HI	7 T HI	C1 EXT-FIRE	0	1
1 Q LIQUID LO	1 Q LO	7 P GAS HI	7 T GAS HI	7 T HI	L1 EXT-FIRE	0	1
1 Q LIQUID LO	1 Q LO	7 P GAS HI	5 Q GAS REV			0	1
1 Q LIQUID LO	1 Q LO	7 P GAS HI	5 Q GAS LO			0	1
1 Q LIQUID LO	1 Q LO	7 P GAS HI	5 Q GAS LO	L3 BLOCKED		0	1
1 Q LIQUID LO	1 Q LO	7 P GAS HI	1 Q GAS HI			0	1
2 Q LIQUID HI	LCL1 SET-LO					1	1
2 Q LIQUID HI	LCL1 FAILED-LO					1	1
2 Q LIQUID HI	V3 OPEN					1	1
2 Q LIQUID HI	L4 LEAK-TO-ATMOS					0	1
2 Q LIQUID HI	8 P GAS LO (0)					0	1
2 Q LIQUID HI	8 P GAS LO (1)					0	1
L2 LEAK-TO-ATMOS						1	1
L1 LEAK-TO-ATMOS						0	1
C1 BURST						1	1

Table E.2: Causal and Limiting Orders for Event Chains Leading to 7 L LIQUID HI for the Basic Ammonia Let Down Problem

INPUT EVENT CHAIN		CAUSAL ORDER	LIMITING ORDER
L2 PART-BLOCKED		0	1
2 Q LIQUID LO	LCL1 SET-HI	1	1
2 Q LIQUID LO	LCL1 FAILED-HI	1	1
2 Q LIQUID LO	V1 PART-CLOSED	0	1
2 Q LIQUID LO	V2 PART-CLOSED	0	1
2 Q LIQUID LO	L2 BLOCKED	1	1
2 Q LIQUID LO	L4 BLOCKED	1	1
2 Q LIQUID LO	L4 PART-BLOCKED	0	1
2 Q LIQUID LO	8 P GAS HI (0)	0	1
2 Q LIQUID LO	8 P GAS HI (1)	0	1
2 Q LIQUID LO	8 P GAS HI (2)	0	1

Table E.3: Causal and Limiting Orders for Event Chains Leading to 8 L LIQUID LO for the Basic Ammonia Let Down Problem

INPUT EVENT CHAIN						CAUSAL ORDER	LIMITING ORDER
C2 LEAK-TO-ATMOS						0	1
L4 LEAK-TO-ATMOS						0	1
L5 LEAK-TO-ATMOS						1	1
L4 PART-BLOCKED						0	1
4 Q LIQUID HI						0	1
4 Q LIQUID HI	LCL2 SET-LO					1	1
4 Q LIQUID HI	LCL2 FAILED-LO					1	1
2 Q LIQUID LO	LCL1 SET-HI					0	1
2 Q LIQUID LO	LCL1 FAILED-HI					0	1
2 Q LIQUID LO	7 L LIQUID LO (0)					0	1
2 Q LIQUID LO	7 L LIQUID LO (1)					0	1
2 Q LIQUID LO	V1 PART-CLOSED					0	1
2 Q LIQUID LO	V2 PART-CLOSED					0	1
2 Q LIQUID LO	L2 BLOCKED					0	1
2 Q LIQUID LO	L2 LEAK-TO-ATMOS					0	1
2 Q LIQUID LO	L2 PART-BLOCKED					0	1
C2 BURST						1	1
L4 BLOCKED						0	1
2 X LIQUID AMM-LIQ LO	7 X LIQUID AMM-LIQ LO	1 X LIQUID AMM-LIQ LO				0	1
8 T GAS HI	8 T HI	C2 EXT-FIRE				0	1
8 T GAS HI	8 T HI	L4 EXT-FIRE				0	1
8 T GAS HI	8 T HI	2 T LIQUID HI	L2 EXT-FIRE			0	1
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	1 T HI	0	1
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	C1 EXT-FIRE	0	1
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	L1 EXT-FIRE	0	1

Table E.4: Causal and Limiting Orders for Event Chains Leading to 8 L LIQUID HI for the Basic Ammonia Let Down Problem

INPUT EVENT CHAIN						CAUSAL ORDER	LIMITING ORDER
L5 PART-BLOCKED						0	1
4 Q LIQUID LO	LCL2 SET-HI					1	1
4 Q LIQUID LO	LCL2 FAILED-HI					1	1
4 Q LIQUID LO	L5 BLOCKED					1	1
4 Q LIQUID LO						1	1
2 Q LIQUID HI	LCL1 SET-LO					0	1
2 Q LIQUID HI	LCL1 FAILED-LO					0	1
2 Q LIQUID HI	7 L LIQUID HI (0)					0	1
2 Q LIQUID HI	7 L LIQUID HI (1)					0	1
2 Q LIQUID HI	V3 OPEN					0	1
8 T GAS LO	8 T LO	2 T LIQUID LO	7 T LIQUID LO	7 T LO	1 T LO	0	1

Table E.5: Causal and Limiting Orders for Event Chains Leading to 8 P GAS LO for the Basic Ammonia Let Down Problem

INPUT EVENT CHAIN		CAUSAL ORDER	LIMITING ORDER
C2 LEAK-TO-ATMOS		0	1
L6 LEAK-TO-ATMOS		1	1
L4 LEAK-TO-ATMOS		1	1
L4 PART-BLOCKED		0	1
8 T GAS LO	8 T LO 2 T LIQUID LO 7 T LIQUID LO 7 T LO 1 T LO	0	1
3 Q GAS HI		0	1
3 Q GAS HI	PCL3 SET-LO	1	1
3 Q GAS HI	PCL3 FAILED-LO	1	1
2 Q LIQUID LO	LCL1 SET-HI	0	1
2 Q LIQUID LO	LCL1 FAILED-HI	0	1
2 Q LIQUID LO	7 L LIQUID LO (0)	0	1
2 Q LIQUID LO	7 L LIQUID LO (1)	0	1
2 Q LIQUID LO	V1 PART-CLOSED	0	1
2 Q LIQUID LO	V2 PART-CLOSED	0	1
2 Q LIQUID LO	L2 BLOCKED	0	1
2 Q LIQUID LO	L4 BLOCKED	0	1
2 Q LIQUID LO	L2 LEAK-TO-ATMOS	0	1
2 Q LIQUID LO	L2 PART-BLOCKED	0	1
VENT Q GAS SOME	RV1 FAIL-SAFE	1	1

Table E.6: Causal and Limiting Orders for Event Chains Leading to 8 P GAS HI for the Basic Ammonia Let Down Problem

INPUT EVENT CHAIN						CAUSAL ORDER	LIMITING ORDER
8 L LIQUID HI (0)						0	2
8 L LIQUID HI (1)						2	2
L3 PART-BLOCKED						0	2
2 Q GAS SOME	7 L LIQUID NO	7 L LIQUID LO (1)				2	2
8 T GAS HI	8 T HI	2 T LIQUID HI	L2 EXT-FIRE			0	2
8 T GAS HI	8 T HI	L4 EXT-FIRE				0	2
8 T GAS HI	8 T HI	C2 EXT-FIRE				0	2
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	1 T HI	0	2
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	C1 EXT-FIRE	0	2
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	L1 EXT-FIRE	0	2
3 Q GAS REV						1	2
3 Q GAS LO						1	2
3 Q GAS LO	PCL3 SET-HI					1	2
3 Q GAS LO	PCL3 FAILED-HI					1	2
2 Q LIQUID HI	LCL1 SET-LO					0	2
2 Q LIQUID HI	LCL1 FAILED-LO					0	2
2 Q LIQUID HI	7 L LIQUID HI (0)					0	2
2 Q LIQUID HI	7 L LIQUID HI (1)					0	2
2 Q LIQUID HI	V3 OPEN					0	2

Table E.7: Causal and Limiting Orders for Event Chains Leading to 7 L LIQUID LO for the Modified Ammonia Let Down Plant

INPUT EVENT CHAIN						CAUSAL ORDER	LIMITING ORDER
L1 BLOCKED						0	2
C1 LEAK-TO-ATMOS						0	2
1 Q LIQUID LO	1 Q LO					0	2
1 Q LIQUID LO	1 Q LO	L1 PART-BLOCKED				0	2
1 Q LIQUID LO	1 Q LO	7 P GAS HI	L3 PART BLOCKED			0	2
1 Q LIQUID LO	1 Q LO	7 P GAS HI	7 T GAS HI	7 T HI	1 T HI	0	0
1 Q LIQUID LO	1 Q LO	7 P GAS HI	7 T GAS HI	7 T HI	C1 EXT-FIRE	0	0
1 Q LIQUID LO	1 Q LO	7 P GAS HI	7 T GAS HI	7 T HI	L1 EXT-FIRE	0	0
1 Q LIQUID LO	1 Q LO	7 P GAS HI	5 Q GAS REV			1	2
1 Q LIQUID LO	1 Q LO	7 P GAS HI	5 Q GAS LO			0	2
1 Q LIQUID LO	1 Q LO	7 P GAS HI	5 Q GAS LO	L3 BLOCKED		0	2
1 Q LIQUID LO	1 Q LO	7 P GAS HI	1 Q GAS HI			0	2
2 Q LIQUID HI	LCL1 SET-LO					1	2
2 Q LIQUID HI	LCL1 FAILED-LO					1	2
2 Q LIQUID HI	V3 OPEN					1	2
2 Q LIQUID HI	L4 LEAK-TO-ATMOS					0	2
2 Q LIQUID HI	8 P GAS LO (0)					0	2
2 Q LIQUID HI	8 P GAS LO (1)					0	2
L2 LEAK-TO-ATMOS						2	2
L1 LEAK-TO-ATMOS						0	2
C1 BURST						2	2

Table E.8: Causal and Limiting Orders for Rvent Chains Leading to 7 L LIQUID HI for the Modified Ammonia Let Down Problem

INPUT EVENT CHAIN		CAUSAL ORDER	LIMITING ORDER
L2 PART-BLOCKED		0	1
2 Q LIQUID LO	LCL1 SET-HI	1	1
2 Q LIQUID LO	LCL1 FAILED-HI	1	1
2 Q LIQUID LO	V1 PART-CLOSED	0	1
2 Q LIQUID LO	V2 PART-CLOSED	0	1
2 Q LIQUID LO	L2 BLOCKED	1	1
2 Q LIQUID LO	L4 BLOCKED	1	1
2 Q LIQUID LO	L4 PART-BLOCKED	0	1
2 Q LIQUID LO	8 P GAS HI (0)	0	0
2 Q LIQUID LO	8 P GAS HI (1)	0	0
2 Q LIQUID LO	8 P GAS HI (2)	0	0
2 Q LIQUID LO	8 P GAS HI (3)	0	0
2 Q LIQUID LO	LLTL1 FAIL-SAFE	1	1
2 Q LIQUID LO	LLTL2 FAIL-SAFE	1	1

Table E.9: Causal and Limiting Orders for Event Chains Leading to 8 L LIQUID LO for the Modified Ammonia Let Down Problem

INPUT EVENT CHAIN						CAUSAL ORDER	LIMITING ORDER
C2 LEAK-TO-ATMOS						0	2
L4 LEAK-TO-ATMOS						0	2
L5 LEAK-TO-ATMOS						1	2
L4 PART-BLOCKED						0	2
4 Q LIQUID HI						0	2
4 Q LIQUID HI	LCL2 SET-LO					1	2
4 Q LIQUID HI	LCL2 FAILED-LO					1	2
2 Q LIQUID LO	LCL1 SET-HI					0	2
2 Q LIQUID LO	LCL1 FAILED-HI					0	2
2 Q LIQUID LO	7 L LIQUID LO (0)					0	2
2 Q LIQUID LO	7 L LIQUID LO (1)					0	2
2 Q LIQUID LO	7 L LIQUID LO (2)					0	2
2 Q LIQUID LO	V1 PART-CLOSED					0	2
2 Q LIQUID LO	V2 PART-CLOSED					0	2
2 Q LIQUID LO	L2 BLOCKED					0	2
2 Q LIQUID LO	L2 LEAK-TO-ATMOS					0	2
2 Q LIQUID LO	L2 PART-BLOCKED					0	2
2 Q LIQUID LO	LLTL1 FAIL-SAFE					0	2
2 Q LIQUID LO	LLTL2 FAIL-SAFE					0	2
2 Q LIQUID LO	HLTL3 FAIL-SAFE					0	2
C2 BURST						2	2
2 X LIQUID AMM-LIQ LO	7 X LIQUID AMM-LIQ LO	1X LIQUID AMM-LIQ LO				0	2
L4 BLOCKED						0	2
8 T GAS HI	8 T HI	2 T LIQUID HI	L2 EXT-FIRE			0	2
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	1 T HI	0	2
8 T GAS HI	8 T HI	C2 EXT-FIRE				0	2
8 T GAS HI	8 T HI	L4 EXT-FIRE				0	2
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	C1 EXT-FIRE	0	2
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	L1 EXT-FIRE	0	2

Table E.10: Causal and Limiting Orders for Event Chains Leading to 8 L LIQUID HI for the Modified Ammonia Let Down Problem

INPUT EVENT CHAIN							CAUSAL ORDER	LIMITING ORDER
L5 PART-BLOCKED							0	2
4 Q LIQUID LO	LCL2 SET-HI						1	2
4 Q LIQUID LO	LCL2 FAILED-HI						1	2
4 Q LIQUID LO	L5 BLOCKED						1	2
4 Q LIQUID LO	LLTL4 FAIL-SAFE						1	2
4 Q LIQUID LO	LLTL5 FAIL-SAFE						1	2
4 Q LIQUID LO							1	2
2 Q LIQUID HI	LCL1 SET-LO						0	2
2 Q LIQUID HI	LCL1 FAILED-LO						0	2
2 Q LIQUID HI	7 L LIQUID HI (0)						0	2
2 Q LIQUID HI	7 L LIQUID HI (1)						0	2
2 Q LIQUID HI	V3 OPEN						0	2
8 T GAS LO	8 T LO	2 T LIQUID LO	7 T LIQUID LO	7 T LO	1 T LO		0	2

Table E.11: Causal and Limiting Orders for Event Chains Leading to 8 P GAS LO for the Modified Ammonia Let Down Problem

INPUT EVENT CHAIN			CAUSAL ORDER	LIMITING ORDER
C2 LEAK-TO-ATMOS			0	1
L6 LEAK-TO-ATMOS			1	1
L4 LEAK-TO-ATMOS			1	1
L4 PART-BLOCKED			0	1
8 T GAS LO	8 T LO	2 T LIQUID LO 7 T LIQUID LO 7 T LO 1 T LO	0	1
3 Q GAS HI			0	1
3 Q GAS HI	PCL3 SET-LO		1	1
3 Q GAS HI	PCL3 FAILED-LO		1	1
2 Q LIQUID LO	LCL1 SET-HI		0	1
2 Q LIQUID LO	LCL1 FAILED-HI		0	1
2 Q LIQUID LO	7 L LIQUID LO (0)		0	1
2 Q LIQUID LO	7 L LIQUID LO (1)		0	1
2 Q LIQUID LO	7 L LIQUID LO (2)		0	1
2 Q LIQUID LO	V1 PART-CLOSED		0	1
2 Q LIQUID LO	V2 PART-CLOSED		0	1
2 Q LIQUID LO	L2 BLOCKED		0	1
2 Q LIQUID LO	L4 BLOCKED		0	1
2 Q LIQUID LO	L2 LEAK-TO-ATMOS		0	1
2 Q LIQUID LO	L2 PART-BLOCKED		0	1
2 Q LIQUID LO	LLTL1 FAIL-SAFE		0	1
2 Q LIQUID LO	LLTL2 FAIL-SAFE		0	1
VENT Q GAS SOME	RV1 FAIL-SAFE		1	1
VENT Q GAS SOME	HPTL6 FAIL-SAFE		1	1

Table E.12: Causal and Limiting Orders for Event Chains Leading to 8 P GAS HI for the Modified Ammonia Let Down Problem

INPUT EVENT CHAIN						CAUSAL ORDER	LIMITING ORDER
8 L LIQUID HI (0)						0	0
8 L LIQUID HI (1)						0	0
8 L LIQUID HI (2)						2	3
L3 PART-BLOCKED						0	3
2 Q GAS SOME	7 L LIQUID NO	7 L LIQUID LO (2)				2	3
8 T GAS HI	8 T HI	2 T LIQUID HI	L2 EXT-FIRE			0	3
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	1 T HI	0	3
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	C1 EXT-FIRE	0	3
8 T GAS HI	8 T HI	2 T LIQUID HI	7 T LIQUID HI	7 T HI	L1 EXT-FIRE	0	3
8 T GAS HI	8 T HI	C2 EXT-FIRE				0	3
8 T GAS HI	8 T HI	L4 EXT-FIRE				0	3
3 Q GAS REV						1	3
3 Q GAS LO						1	3
3 Q GAS LO	PCL3 SET-HI					1	3
3 Q GAS LO	PCLE FAILED-HI					1	3
2 Q LIQUID HI	LCL1 SET-LO					0	3
2 Q LIQUID HI	LCL1 FAILED-LO					0	3
2 Q LIQUID HI	7 L LIQUID HI (0)					0	3
2 Q LIQUID HI	7 L LIQUID HI (1)					0	3
2 Q LIQUID HI	V3 OPEN					0	3

APPENDIX F

F. APPLICATION OF PROCESS UNIT MODELLING TO THE AMMONIA LET DOWN PLANT

During the course of this research, a fault tree synthesis prog which utilized a standard process unit modelling approach was develop The method used was similar to that of Martin-Solis et al [36]. A fa tree was produced for overpressure of the basic ammonia let down pl described in Chapter 5. Part of this fault tree is reproduced in Fig. F.

Fig. F.1: Part of a Fault Tree Produced by Process Unit Modelling

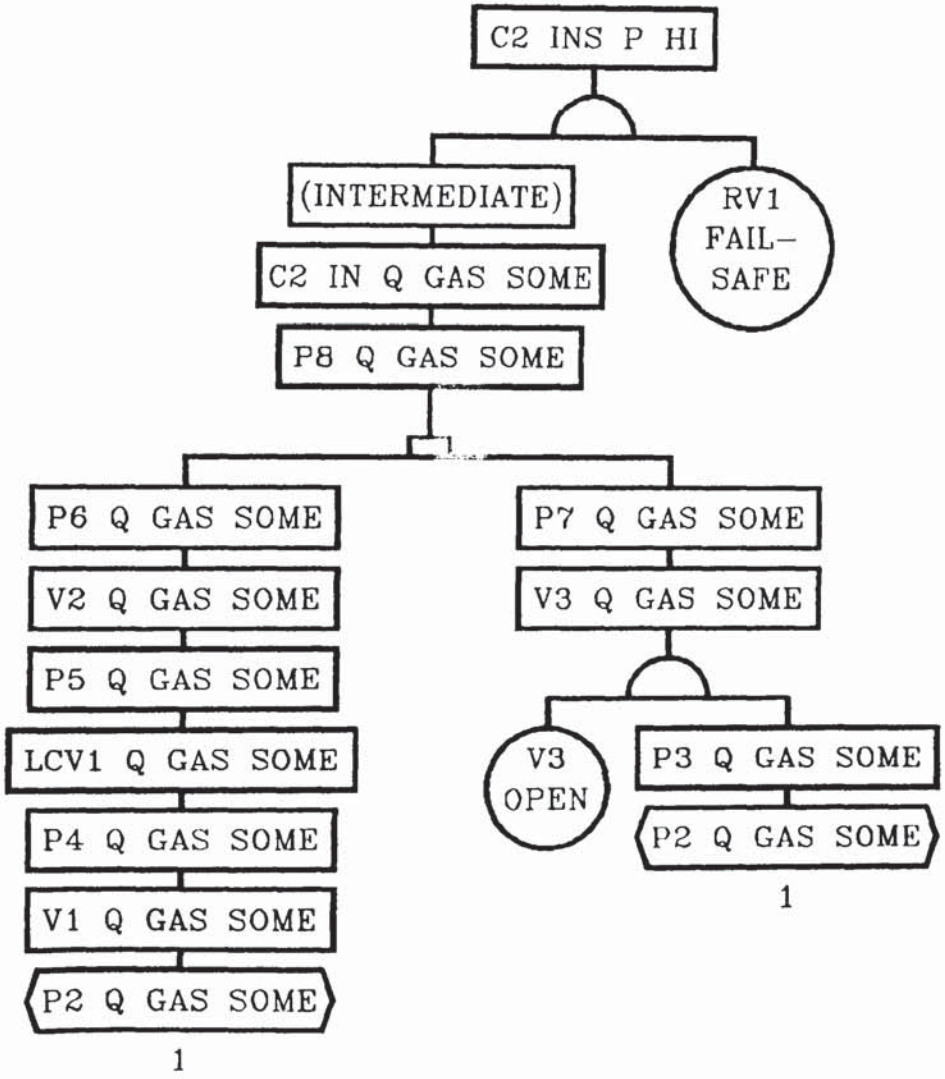


Fig. F.1a: Continuation 1 of Fault Tree from Fig. F.1