Rules for the computer-aided synthesis of fault trees

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfillment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: [https://dspace.lboro.ac.uk/2134/7011](https://dspace.lboro.ac.uk/2134/7011)

Publisher: © Jasbir Singh Mullhi

Please cite the published version.
This item is held in Loughborough University’s Institutional Repository (https://dspace.lboro.ac.uk/) and was harvested from the British Library’s EThOS service (http://www.ethos.bl.uk/). It is made available under the following Creative Commons Licence conditions.

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
RULES FOR THE COMPUTER-AIDED SYNTHESIS OF FAULT TREES

by

J.S. MULLHI

A Doctoral Thesis
Submitted in partial fulfilment of the requirements for the award of Ph.D. of the Loughborough University of Technology

December 1989

© by Jasbir Singh Mullhi 1989
DECLARATION OF ORIGINAL WORK

This is to certify that the work described in this thesis is my own work, that the original work is my own except as specified in acknowledgements, and that neither this thesis nor the original work contained within it has not been submitted to any other institution for a higher degree.
PREFACE

This thesis describes the development of a computer-aided fault tree synthesis package for application in the process industries. It builds on the previous research work carried out in the Plant Engineering Group at Loughborough University. The emphasis has been put on describing the underlying methodology as opposed to the actual computer programs.

The methodology described was developed by modelling a number of “real” systems, which had already been analysed using manual fault tree construction techniques by British Gas plc. Additionally a number of standard examples from the literature were utilised, as well as a large number of contrived examples to fully evaluate the package. The problems encountered and their solution are described.

The culmination of this project was the implementation of the computer package at the Midlands Research Station of British Gas plc. It is not intended that the package should replace the fault tree expert. It should rather be viewed as a tool to facilitate the work of the process engineer, particularly during the design phase. This should enable the evaluation of many more options, which would otherwise have been proved prohibitive by the effort required to manually synthesise the fault trees.
ACKNOWLEDGEMENTS

The author would like to express his appreciation to British Gas plc for funding this project and to Control Division, Midlands Research Station for providing technical support.

The author also acknowledges the invaluable guidance provided by Prof. F.P. Lees and Dr. M.L. Ang during the course of this project.
3.3 MODELLING
3.3.1 INTRODUCTION
3.3.2 UNIT PORTS
3.3.3 PROPAGATION EQUATIONS
3.3.4 EVENT STATEMENTS
3.3.5 DECISION TABLES
3.3.6 TERMINAL EVENT MODELLING
3.3.7 MINI-FAULT TREES

3.4 FAULT TREE SYNTHESIS

3.5 CONSISTENCY CHECKS
3.5.1 BOUNDARY CONDITIONS
3.5.2 NOT ALLOWED FAULTS
3.5.3 DUPLICATED EVENTS
3.5.4 PORT CHANGES
3.5.5 PARALLEL CONSISTENCY

3.6 ADVANCED FEATURES
3.6.1 SEQUENTIAL OPERATIONS
3.6.2 SECONDARY FAILURES

3.7 CONCLUSION

4.0 ENHANCEMENT OF THE MODEL GENERATION ALGORITHM

4.1 INTRODUCTION
4.2 DEFICIENCY IN MK1 MODEL GENERATION ALGORITHM
4.3 PROPOSED SOLUTION
4.4 APPLICATION TO TANK MODEL.
4.4.1 INTRODUCTION
4.4.2 REMOVAL OF EVENT STATEMENTS

4.5 APPLICATION TO HEAT EXCHANGER MODEL
4.6 INCLUSION OF BASIC FAULTS IN THE MODEL
4.6.1 INTRODUCTION
4.6.2 FAULT LIBRARY
4.6.3 DATA INPUT BY THE USER
4.6.4 THE ANALYSIS ALGORITHM
4.6.5 USER MODIFICATION OF THE AUTOMATICALLY GENERATED EVENT STATEMENTS

4.7 CONCLUSION

5.0 ELIMINATION OF DIVIDER HEADER COMBINATIONS

5.1 INTRODUCTION
5.2 FAULTFINDER MK1 SOLUTION
5.3 DEFICIENCY IN THE MK1 SOLUTION.
5.4 PROPOSED SOLUTION
5.5 APPLICATION TO AN EXAMPLE SYSTEM
5.6 TERMINOLOGY
5.6.1 LOOP
5.6.2 BALANCED LOOP
5.6.3 OVERALL LOOP

5.7 DETERMINATION OF LOOPS
5.8 USE OF A SINGLE DIVIDER/HEADER MODEL
5.9 NO FLOW LEGS
5.10 DECOMPOSITION AND INPUT TO MASTER PROGRAM
5.10.1 MORE THAN TWO FLOW LEGS
5.10.2 FLOW CAPACITIES
5.10.3 INPUT TO ASSIST THE DETECTION ALGORITHM
5.11 EFFECTS ON FAULT TREE SYNTHESIS
5.12 PARALLEL SYSTEMS
5.12.1 INTRODUCTION
5.12.2 APPLICATION TO AN EXAMPLE SYSTEM
5.12.3 PROPOSED SOLUTION
5.12.4 IMPLEMENTATION
5.12.5 SEPARATING THE LEAKAGE AND BLOCKAGE TYPE FAULTS.
5.13 CONCLUSION

6.0 RULES FOR THE DECOMPOSITION OF CONTROL LOOPS
6.1 INTRODUCTION
6.2 TYPES OF CONTROL LOOP
6.3 FAILURE MODES FOR THE CONTROL LOOPS
6.4 FILLING IN THE CONTROL LOOP TEMPLATES
6.5 DEFICIENCY IN THE MK1 SOLUTION
6.6 RULES FOR DECOMPOSING CONTROL LOOPS
6.6.1 MODIFIED REPRESENTATION OF LAPP-POWERS HEAT EXCHANGER SYSTEM
6.6.2 FEEDFORWARD MIXER COMPOSITION SYSTEM
6.6.3 LEVEL CONTROL SYSTEM
6.6.4 LIHOU HYDROCARBON/OXYGEN REACTOR
6.6.5 DISTILLATION COLUMN PROBLEM
6.6.6 COMPLEX CONTROL LOOPS
6.7 AUTOMATIC DETECTION OF CONTROL LOOPS
6.8 CONCLUSION

7.0 RULES FOR HANDLING TRIP SYSTEMS
7.1 INTRODUCTION
7.2 BASIC APPROACH
7.3 TYPES OF TRIP SYSTEMS
7.4 OPEN VALVE FEEDBACK TRIP SYSTEMS
7.4.1 INTRODUCTION
7.4.2 BASIC RULE
7.4.3 REFINEMENT OF THE BASIC RULE
7.5 OPEN VALVE FEEDFORWARD TRIP SYSTEMS
7.6 CLOSED VALVE TRIP (CVT) SYSTEMS
7.7 INCLUSION OF TL-FN-F
7.8 RATIONALISATION OF TL-FN-F BRANCHES
7.9 SEQUENTIAL OPERATIONS
7.10 COMBINED CONTROL AND TRIP LOOPS
7.11 CONCLUSION

8.0 MODELLING UTILITY FAILURES
8.1 INTRODUCTION
8.2 MODELLING
8.3 DEFICIENCY IN THE MK1 SOLUTION
8.4 PROPOSED SOLUTIONS
8.4.1 SOLUTION 1
8.4.2 SOLUTION 2
8.5 WORKED EXAMPLE
8.5.1 INTRODUCTION
8.5.2 IMPLEMENTED SOLUTION
8.6 CONCLUSION

9.0 FAULT TREE ANALYSIS
9.1 INTRODUCTION
9.2 MINIMAL CUTSETS
9.3 INTERFACE WITH THE ANALYSIS PROGRAMS
  9.3.1 FTAP FORMAT
  9.3.2 PREP FORMAT
  9.3.3 INTERFACE PROBLEMS
9.4 POST-CUTSET ANALYSIS ALGORITHMS
  9.4.1 MUTUALLY EXCLUSIVE EVENTS
  9.4.2 COMBINATORIAL EXPLOSION
9.5 CONCLUSION

10.0 WORKED EXAMPLES
  10.1 INTRODUCTION
  10.2 LAPP-POWERS SYSTEM
  10.3 PIPELINE MIXER SYSTEM
  10.4 BUTANE VAPORISER SYSTEM
  10.5 BURNER CONTROL SEQUENCE
  10.6 PUMPS IN PARALLEL REDUNDANCY SYSTEM
  10.7 PRESSURE REDUCTION INSTALLATION

11.0 CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

REFERENCES

APPENDIX A SUITE OF FAULTFINDER PROGRAMS
APPENDIX B LAPP-POWERS HEAT EXCHANGER SYSTEM
APPENDIX C COMPENDIUM OF FAULTS
1.0 INTRODUCTION

1.1 BACKGROUND

A fault tree is a graphical representation showing the logical relationship between a system hazard or undesired event, the top event, and its primary causes. It can be used as a means of quantifying the top event in terms of failures in the system components. Fault trees can be generated for any top event and any system. However, in this thesis the discussion is confined to their application to the process industries.

In order to illustrate the concept of a fault tree, consider the simple plant layout in Figure 1.1:

![Simple Tank System](image)

The system consists of a tank of water with two hand valves on the outlet, which are both closed. Assume that the causes of the event SOME-FLOW at location X are to be investigated.

By defining SOME-FLOW as the undesired event, a fault tree can be developed to find the primary causes of this top event. This fault tree is given in Figure 1.2.

The different types of symbol in the fault tree should be noted. Essentially these fall into two categories: logic symbols and event symbols. For a full description of these refer to Table 1.1.

It is seen that the fault tree given in Figure 1.2 is composed entirely of these symbols. Commonly no text is written inside the logic symbols; however text is required inside the event symbols to describe the state of the plant at the current level of development in the fault tree. The top event in Figure 1.2 is thus developed to four primary causes representing failures in the two hand valves, and one diamond event cause.
Figure 1.2
Fault Tree for Tank System of Figure 1.1

<table>
<thead>
<tr>
<th>SYMBOLS</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="AND Gate Symbol" /></td>
<td>AND Gate - output exists only if all the inputs exist</td>
</tr>
<tr>
<td><img src="image" alt="OR Gate Symbol" /></td>
<td>OR Gate - output exists if any of the inputs exists</td>
</tr>
<tr>
<td><img src="image" alt="Transmissive Event Symbol" /></td>
<td>Transmissive event requiring further development to find primary causes</td>
</tr>
<tr>
<td><img src="image" alt="Basic Event Symbol" /></td>
<td>Basic event representing primary causes</td>
</tr>
<tr>
<td><img src="image" alt="Transmissive Event Not Further Developed Symbol" /></td>
<td>Transmissive event is not or cannot be further developed, usually because it crosses the system boundary</td>
</tr>
</tbody>
</table>

Table 1.1
Fault Tree Symbols
If the tank were normally empty, then the causes of some level in the tank would lie further upstream, in the inlet to the tank. Since the nature of the upstream plant is unknown, the event SOME LEVEL IN TANK must be flagged as a diamond event. Alternatively, if it can be assumed that some level in the tank is a normal state, then the fault tree in Figure 1.2 would reduce to that given in Figure 1.3 below:

Fault tree synthesis is only a precursor to fault tree analysis. Fault trees can be analysed using two techniques: qualitative analysis and quantitative analysis. Qualitative analysis is the process by which the minimum cutsets of a fault tree are determined.

A cutset is the set of primary failures which can give rise to the top event; a minimum cutset would be one which does not contain another cutset within itself. By determining all the minimum cutsets for a fault tree, the complete set of failure modes for that top event are obtained. Returning to the fault tree in Figure 1.3, four cutsets are obtained:

1) Hand valve A fails open
   Hand valve B fails open

2) Hand valve A fails open
   Hand valve B leaks

3) Hand valve A leaks
   Hand valve B fails open

4) Hand valve A leaks
   Hand valve B leaks

Figure 1.3
Modified Fault Tree for Tank System of Figure 1.1
Since this is a very simple tree the cutsets listed above are in fact minimal. The next step in the process would be to carry out quantitative analysis. By knowing the failure rate data for the events in the cutsets, the frequency or probability of the top event in the tree can be ascertained. The availability of failure rate data determines the degree of resolution required when synthesising the fault tree. An event for which failure rate data is available can be regarded as a primary cause and need not be developed any further.

Failure rate data for hand valves should be available and so the probability of the top event in Figure 1.3 should be determinable. If the probability of the top event is found to be too high, then either the valve type can be changed or the plant re-designed. The fault tree can thus be used to assess the reliability and safety of a system and if necessary direct the attention of the design engineer to those features which are most critical.

1.2 ESSENTIAL REQUIREMENTS OF A COMPUTER BASED METHODOLOGY FOR FAULT TREE SYNTHESIS FOR PROCESS PLANTS

Even the simple example used above highlights some fundamental points which any computer-based methodology must address. These are summarised below:

a) a system flow-sheet can contain much information about the plant and its operation, some of which is only apparent to an expert familiar with the system. All of this information must be explicitly defined in a form suitable for representation in the programming language being used.

b) failure models must exist which can propagate deviations in process variables from one unit to another, as well as describe how primary failures in the models can give rise to these deviations.

c) the methodology must be able to handle multi-valued deviations in the process variables, e.g. the flow could be high, low, none, some or reverse.

d) there must be a facility for carrying out two-way fault propagation, since the causes of any given event can occur either upstream or downstream of the current location.

e) the methodology must be able to cope with complex sub-systems within the plant, e.g. control loops and trip systems.

f) the fault tree must be both correct and be presented in a structurally intelligible format. There is a tendency with automated methodologies to destroy the structure which an expert manually synthesising the fault tree would clearly impart.

1.3 FAULTFINDER APPROACH TO FAULT TREE SYNTHESIS

Essentially there are three basic steps in using the methodology described in this thesis to synthesise fault trees:

- system decomposition, the information in the plant flow-sheet is extracted and input to the methodology.

- modelling, failure models are created for the different plant items.
- synthesis, based on the information supplied in the first two stages, the fault tree is automatically generated, for the specified top event, by the methodology.

FAULTFINDER is the suite of computer programs used to execute the above tasks. This thesis will concentrate on describing the underlying FAULTFINDER methodology and how it meets the requirements listed above, rather than describing the computer programs. However, Appendix A gives a brief overview of the different programs.

1.4 STRUCTURE OF THIS THESIS

The FAULTFINDER methodology and programs have evolved to their current status as the result of on-going research by various workers at Loughborough. The rest of this thesis is thus split into two basic sections as described below:

1.4.1 PART A

Chapter 2 presents a literature survey in the field of computer-aided fault tree synthesis, and indicates where FAULTFINDER fits into this overall picture.

Chapter 3 presents an overview of the basic FAULTFINDER methodology as it existed prior to the commencement of this project. This stage of the development will be referred to as FAULTFINDER MK1.

1.4.2 PART B

This part will outline those developments which were made during the course of this project. This stage of the development will be referred to as FAULTFINDER MK2.

Chapter 4 outlines the refinements made to the model generation program, to facilitate the task of creating failure models.

Chapter 5 outlines the developments to the general methodology, which eliminated the need to identify one type of sub-system, divider-header combinations, during the decomposition stage.

Chapter 6 presents rules for the decomposition of control loops, a sub-system requiring special treatment in the methodology.

Chapter 7 presents rules for the decomposition of trip systems, a sub-system requiring special treatment in the methodology.

Chapter 8 describes the enhancements implemented to overcome the problems presented by common mode failures, with respect to utility supplies.

Chapter 9 outlines the interface of FAULTFINDER with two fault tree analysis packages PREP and FTAP.

Chapter 10 describes some test cases which were used to develop the current methodology.

The final chapter in this thesis presents the overall conclusions and the recommendations for further work.
1.5 CONVENTIONS USED FOR REPRESENTING FAULT TREES

This section describes the conventions used to represent the events and symbols in the fault trees contained in this thesis. The event names used to represent the plant state refer to one of the following two types of event:

- variable deviations in process parameters. The format and meaning of these is described in Chapter 3.

- faults in process items. The format of these is described in Chapter 3. Since a fault name within the FAULTFINDER methodology can comprise a maximum of eight characters, the full meaning of each fault name is given in Appendix C.

In general the fault tree symbolism used complies with that given in Table 1.1. However, for some of the larger fault trees, particularly those in Chapter 10, the symbols enclosing the events have been omitted and the logic gates have been replaced by text strings, e.g. AND/OR. This has been done for ease of presentation.

Furthermore in some branches, where the intermediate steps in the propagation chain are not significant to explain a particular point, the intermediate events are omitted. Such an occurrence is identified by using a dotted line to link the particular events.
PART A

CHAPTER 2  LITERATURE SURVEY
CHAPTER 3  OUTLINE OF METHODOLOGY
2.0 LITERATURE SURVEY

2.1 INTRODUCTION

This chapter presents a review of the literature relating to fault tree synthesis. Due to the widespread application of the fault tree technique and the resulting volume of published papers, this review is of necessity restricted to those areas which have had the greatest bearing on this project. The focus of attention has therefore been the attempt to develop formal methodologies for handling the systems in the process industries.

The following is an overview of the topics addressed:

- basic fault tree concepts.
- origins of fault tree synthesis and adoption of the technique by the process industries.
- development of formal methodologies to handle systems from the process industries.
- problems encountered in the application of the technique to the process industries and their solution.
- automation of these methodologies in the form of computer codes.

2.2 FAULT TREE CONCEPTS

A fault tree consists of a series of events and system states linked together by logic gates. Initially an undesired or top event must be defined. The causes of the top event are then developed and linked by the appropriate gates. Each of these causes is subsequently traced down to its basic cause events, which are normally faults within items of the plant.

The two basic logic gates are the AND gate and the OR gate. An AND gate is used when two or more events must occur together to cause the event under consideration. An OR gate is used when any one of a number of events is alone sufficient to cause the event under consideration.

Tables 2.1a and 2.1b summarise the conventional fault tree logic and event symbolism. In practice the AND and OR gates account for the vast majority of cases encountered. This is probably due to the inability of most of the fault tree analysis programs to handle the other types of gate. There was in fact much criticism of an EX-OR gate used by Lapp and Powers [Ref 2-1], by Yellman [Ref 2-2], Locks [Ref 2-3], Lambert [Ref 2-4] and Henley and Kumamoto [Ref 2-5], amongst others. It is indeed possible to reduce some of the gates in Table 2.1a, to their more basic equivalents comprising of AND and OR gates only.

Henley and Kumamoto [Ref 2-6] give a good overview of the different types of fault tree symbol.

2.3 PURPOSE OF FAULT TREE SYNTHESIS

It should be noted that fault tree synthesis is only an intermediate step, albeit a very important one, in the assessment of the reliability of usually complex systems. The motivation for synthesising fault trees is to enable fault tree analysis.

Before proceeding any further it is worthwhile clarifying some of the terminology as this is used somewhat interchangeably in the literature.
<table>
<thead>
<tr>
<th>SYMBOLS</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="AND Gate" /></td>
<td>AND Gate - output exists only if all the inputs exist</td>
</tr>
<tr>
<td><img src="image" alt="NOT Gate" /></td>
<td>NOT Gate - output exists only if some events exist whilst others do not.</td>
</tr>
<tr>
<td><img src="image" alt="OR Gate" /></td>
<td>OR Gate - output exists if any of the inputs exists</td>
</tr>
<tr>
<td><img src="image" alt="EOR Gate" /></td>
<td>EOR Gate - output exists if only one of the inputs exists</td>
</tr>
<tr>
<td><img src="image" alt="INHIBIT Gate" /></td>
<td>INHIBIT Gate - output exists only when the conditional input is satisfied</td>
</tr>
<tr>
<td><img src="image" alt="DELAY Gate" /></td>
<td>DELAY Gate - output exists only after the specified time has elapsed</td>
</tr>
<tr>
<td><img src="image" alt="MATRIX Gate" /></td>
<td>MATRIX Gate - output exists if one or more combinations of the inputs exists</td>
</tr>
</tbody>
</table>

Table 2.1a
Fault Tree Logic Symbols
<table>
<thead>
<tr>
<th>SYMBOLS</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmissive event requiring further development to find primary causes</td>
<td></td>
</tr>
<tr>
<td>Basic event representing primary causes</td>
<td></td>
</tr>
<tr>
<td>Transmissive event is not or cannot be further developed, usually because it crosses the system boundary</td>
<td></td>
</tr>
<tr>
<td>A significant undeveloped event requiring further development to complete the fault tree</td>
<td></td>
</tr>
</tbody>
</table>

Connecting/Transfer Symbols

A similarity transfer - the input is similar but not identical to the like identified input

An event that is normally expected to occur

Table 2.1b
Fault Tree Event Symbols
Essentially there are four main steps in the analysis of any system using the fault tree technique. These are:

- system definition
- fault tree construction
- qualitative evaluation
- quantitative evaluation

In the current context fault tree synthesis addresses the first two steps. The last two steps are in the domain of fault tree analysis and have not been considered in any great depth in this review.

A comprehensive list of the publications relating to the above four fields is given by Lee et al [Ref 2-7].

2.4 FAULT TREE ORIGINS

Fault tree synthesis/analysis was first conceived by Watson [Ref 2-8] in connection with the study of the Minuteman Missile Launch Control System in 1961. At its inception the technique was largely utilised by the aerospace, nuclear and electronics industries. The main reasons being:

- complexity of these systems defied any informal analysis techniques, and
- potential failure paths needed to be identified a priori due to the loss potential associated with system failure.

It was not until the early 1970s that the first published work started to appear on the use of this technique by the process industries. The following reasons were cited by Powers and Tompkins [Ref 2-9] in 1974, for the relatively late adoption of the technique by the process industries:

- the systems were not complex enough to warrant these approaches. Informal techniques were able to discover most of the hazardous failure modes.

- the systems are remarkably robust. Even when several unit failures have occurred the total process does not fail completely.

- the consequences of failure often mean that the plant is shut down and may be restarted.

- the inherent hazards in most of the chemical industries are due to the reactivity of the chemicals being processed. The prediction of the behaviour of these chemicals under a wide range of conditions is difficult. Hence, a great deal of uncertainty surrounds the definition of hazards.

- the difficulty in generating chemical process models, which can be easily solved and applied under a wide range of normal/abnormal operating conditions.

- the effort required to produce good quality fault trees. This could easily run into man-months or even man-years.
In [Ref 2-9] Powers and Tomkins also cite a number of reasons for the growing suitability of using a formal approach like fault tree synthesis for the process industries. These arose due to the increasing complexity of such systems and the potential hazards associated with them. The main reasons being:

- larger processing units and trend towards single train installations.
- more complete process integration to enable energy recovery and waste recycling.
- reduction of intermediate storage capacity.
- centralisation of control.
- growth of computer-based systems.
- multiplexing of equipment.
- location of plants closer to population centers.

2.5 SYSTEM DEFINITION

The starting point for analysing any chemical process system will normally either be a flow diagram or a P & I diagram. In assessing the causes of any particular hazard, it is not possible to consider the whole system in its entirety except for the most trivial cases. A common approach adopted has therefore been to consider the system as being composed of a number of inter-connected components. The task of fault tree synthesis is thus broken down to manageable proportions. By synthesising a sub-tree for each component it is possible to build the overall fault tree for the whole system by simply linking together the various sub-trees.

Many of the workers in this field have generally decomposed the systems at a level which usually gives a one-to-one correspondence between the units in the physical system and the components in the decomposed representation. The main divergence from this approach has been by Shafaghi et al [Ref 2-10], in which they describe a technique for decomposing the plant based on the control and protective loops contained within it.

In order to be able to build a fault tree from the system components it is necessary to model the behaviour of the components both under normal operation and also under fault conditions. If the decomposition and modelling is done in a context-independent manner, i.e. regardless of what the component is connected to, then it is possible to build up a library of failure models which can be used again and again in a wide variety of applications. The problems inherent in trying to create models of this nature are discussed by Brown and De Kleer [Ref 2-11].

In general it can be taken that the finer the level of decomposition of the system, the greater will be the applicability of the resultant failure models. However the price paid for this is, that much greater effort is required in building the models for the increased number of components.

It follows then that models created from a decomposition at the unit/component level will be fairly context-independent whereas those arising from structure modelling as used by Shafaghi et al, by decomposing the system around control/protective loops will almost certainly not be context-independent.
2.6 UNIT MODELLING

The behaviour of process units is described in terms of mass, energy and momentum balances. This representation could be achieved by writing the full unsteady state differential equations. Powers and Tompkins [Ref 2-9], however highlight three main disadvantages with using this approach:

- models are complex and contain very detailed information about the system.
- models are normally difficult to solve.
- models are usually constrained to a specific operating region and a specific mode of failure.

They go on to suggest that the biggest disadvantage is probably the last one, since if a system has been modelled to the extent of a dynamic simulation then the associated hazards are well on the way to being understood. The whole point of fault tree synthesis and fault tree analysis is to identify unforeseen failure modes.

The concept of information flow models is introduced where the coupling between variables is defined. This is illustrated in the following example of the heat exchanger model taken from [Ref 2-9]:

![Figure 2.1 Heat Exchanger Model of Powers and Tompkins](image)

<table>
<thead>
<tr>
<th>Independent Variables</th>
<th>Dependent Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1, P1, T1</td>
<td>m4, P4, T4</td>
</tr>
<tr>
<td>m2, P2, T2</td>
<td></td>
</tr>
<tr>
<td>m3, P3, T3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable Coupling Matrix for Heat Exchanger Model of Figure 2.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent Variables</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>m1</td>
</tr>
<tr>
<td>T2</td>
</tr>
<tr>
<td>U</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>P1</td>
</tr>
<tr>
<td>P2</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P4</td>
</tr>
</tbody>
</table>

Table 2.2
Almost all the work published is based upon using this concept of qualitative modelling. Despite its apparent simplicity it has proved an adequate vehicle for generating and analysing fault trees.

2.7 KNOWLEDGE REPRESENTATION FORMALISMS

2.7.1 INTRODUCTION

This section outlines the main formalisms used by various workers in developing the methodologies for the creation of the unit models. The following techniques are reviewed:

- directed graphs (digraphs)
- decision tables
- mini-fault trees (minitrees)
- reliability graph
- failure transfer functions

2.7.2 DIGRAPHS

In 1977, Lapp and Powers [Ref 2-1] illustrated the use of a digraph representation to model the system. A digraph is composed of a set of nodes which are connected by directed edges. The nodes on the graph represent the process variables and certain types of failure events. If one variable affects another then a directed edge is drawn from the independent to the dependent variable.

The significant aspects of digraphs will be illustrated by reference to the following model for a pneumatic control valve:

![Figure 2.2 Pneumatic Control Valve Model](image)
The parameters of prime interest are the application of pressure (P) at location 3 and its resultant effect on the flow rate (M) at location 2. The basic digraph for this would be:

![Figure 2.3
Simple Digraph for Control Valve Model of Figure 2.2](image)

In general it is necessary to indicate the gain on the edge. The gain would be -1 for a normal control valve, giving the following mapping between the two variables:

<table>
<thead>
<tr>
<th>P3</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>+10</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>+10</td>
<td>-10</td>
</tr>
</tbody>
</table>

Table 2.3
Mapping Between Control Valve Parameters (1)

The gain can thus be interpreted as a (partial) derivative: \( \frac{dM2}{dP3} = -1 \).

Had the valve been of the slam shut type then the gain between the two variables would have been -10. This would have given the following mapping between P3 and M2:

<table>
<thead>
<tr>
<th>P3</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>+1</td>
<td>-10</td>
</tr>
<tr>
<td>+10</td>
<td>-10</td>
</tr>
</tbody>
</table>

Table 2.4
Mapping Between Control Valve Parameters (2)

The following points are worthy of note:

- five deviation states are used to model disturbances: -10,-1,0,+1,+10.
- the disturbance in the dependent variable is obtained by multiplying the deviation in the independent variable by the gain.
- the maximum deviation is 10.

In general a common interpretation imposed on these deviation states is:

0 normal state for the variable.
-1, +1 moderate disturbances (low or high) which would be correctable by control loops.
-10, +10 large disturbances (very high and very low) which would overload control loops, but would be correctable by trip systems.

Edge dependent relationships can also be included as shown below:

![Figure 2.4](image)

Hence, both the usual relationships and the influence that failures would have, can be modelled within the same digraph.

Using the principles outlined above a digraph for the entire system can be developed by building digraphs for the components and then linking them together.

The application of the digraph technique has been illustrated by Lapp and Powers [Ref 2-1], Andrews and Morgan [Ref 2-12], Cummings, Lapp and Powers [Ref 2-13] and Allen [Ref 2-14].

2.7.3 MINITREES

The use of this technique has been demonstrated by a number of workers, most notably Kelly and Lees [Ref 2-15], Lees, Andow and Murphy [Ref 2-16] and Taylor [Ref 2-17].

The technique is based upon a modular decomposition of the plant into its constituent units. Minitrees are then used to represent the failure model for each unit. In general a minitree is required to describe the propagation of each deviation in each variable through the model.
Consider the following unit model for a non-return valve:

![Schematic Representation of Non-Return Valve](image)

**Figure 2.5**
Schematic Representation of Non-Return Valve

Using the concepts outlined in [Ref 2-15] and [Ref 2-16], the following two minitrees would describe the propagation of low and reverse flow through the unit:

![Minitrees for Non-Return Valve Model](image)

**Figure 2.6**
Minitrees for Non-Return Valve Model

Similarly minitrees would be required for any other deviation in flow e.g. high, none etc., and for the complete set of deviations for the other variables in the methodology.

In the above two minitrees, the output events are deviations in variables and the inputs either deviations in variables or basic event failures. Hence minitrees can describe both fault initiation and fault propagation.

The minitrees are generated from three primary input sources:

a) propagation equations, which describe the propagation of variable deviations through the model.

b) event statements, which describe fault initiation in a model.

c) decision tables, which allow the incorporation of AND logic into the model.

Further details of these can be found in Chapter 3 of this thesis.
The minitrees used by Taylor [Ref 2-17] to build the failure models are somewhat different. They have the following general structure:

Figure 2.7
General Format of Minitrees Used by Taylor

Hence considering the following system:

Figure 2.8
Schematic Representation of a Heat Exchanger System

This would yield the following minitree for a high temperature at location X:

Figure 2.9
Minitree for Heat Exchanger System of Figure 2.8
It can be seen from the above minitree, that in general it is necessary not only to model the failures, but also to AND the causes with any compensatory events.

The technique used for building the minitrees is based upon:

a) drawing equation bigraphs for the component which connect the equations and variables.

b) marking arrows on the equation bigraph to produce a cause-effect graph to describe the propagation of disturbances through the plant.

c) extracting signal flow graphs from the cause-effect graphs which show the influence of each variable on other variables.

The normal deviation states used are:

- normal, zero and negative.
- disturbed high/low (correctable by control loops).
- high/low (corrected by trip systems).
- very high/low (cannot be corrected).

**2.7.4 DECISION TABLES**

This technique has been described by Salem et al [Ref 2-18]. The decision table is used to describe each possible output state of a component in terms of its inputs and the internal operational or failed states. The following example is taken from the above paper to illustrate the application of this technique to a fuse component model:

<table>
<thead>
<tr>
<th>Row Number</th>
<th>Input State</th>
<th>Internal Mode</th>
<th>Output State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2.5
Decision Table for Fuse Component Model

where:

a) For Input/Output States:  
0 = no signal  
1 = normal  
2 = overload

b) For Internal Modes:  
0 = good  
1 = failed open  
2 = failed closed
It should be noted that any number of states can be modelled using this technique; in the above component model three states have been utilised.

An important extension of this technique is the introduction of “Don’t Care” logic into the model, in order to facilitate both the modelling and the fault tree construction phases. This is achieved on the following basis:

a) the rows which have identical outputs are searched. Hence, in the above model if Output 0 is first investigated then Rows 1-3, 5, 7 and 8 would be flagged.

b) each column in these rows is then analysed to see if they agree in all but one column. The following two sets would thus result: Set 1 (Rows: 1-3); and Set 2 (Rows: 2, 5 and 8).

c) in the column which disagrees, if it contains every possible state, then it can be replaced by a “Don’t Care” state.

Hence, in the above model Rows 1-3 would be replaced by a single row thus:

<table>
<thead>
<tr>
<th>Row Number</th>
<th>Input State</th>
<th>Internal Mode</th>
<th>Output State</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Row</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

where '-' represents “Don’t Care” state.

It should also be noted that in general this reduction step can only be applied to the input and internal columns. It cannot be used on the output state column.

2.7.5 RELIABILITY GRAPH

Camarda, Corsi and Trentadue [Ref 2-19] introduced the concept of using a reliability graph in which all the possible ways of correct system operation are shown. The advantage of this approach is that a reliability graph is much easier to construct than a failure model representation, since the number of ways in which a physical system can operate are much fewer than those in which it can fail. The main features of the reliability graph are:

a) the branches represent the components/events necessary to the success of the system. Each branch can only have one of two states: good or failed.

b) information flow can either be uni or bi-directional along the arrow.

c) there is only one input and one output node from the system

The success paths of the tree are represented by tie-sets. A tie-set is the combination of nodes between the terminal nodes which are traversed in the direction of the arrows along the branches. A tie-set is called minimal if each node is traversed once only.

Hence, for a notional system having the following reliability graph:
Having evaluated the minimal tie-sets, the next step is to evaluate the minimal cutsets from the minimal tie-sets. This is achieved by logic inversion.

The use of this technique is not really suitable for the domain of the process industry problems. It is only really adequate for electrical circuit type systems, where simple success/failure state modelling is adequate. Chemical process systems require more detailed failure models than simple two-state logic.

Another disadvantage is that it would be necessary to understand reliability theory before utilising this technique.

### 2.7.6 FAILURE TRANSFER FUNCTIONS

The use of this technique was illustrated by Fussell [Ref 2-20]. Failure transfer functions were used to describe the failure modes of the components making up the system; each failure mode being represented by a unique failure transfer function.

A failure transfer function was defined as comprising up to six parts as illustrated in Figure 2.11.
The six parts in the above figure have the following meaning:

**OUTPUT EVENT** - mode of failure.

**OUTPUT LOGIC GATE** - logic with which the transfer function for this output event is linked to other transfer functions having the same output event.

**INTERNAL EVENTS** - events requiring development within the transfer function.

**INTERNAL LOGIC GATES** - logic linking the development of the internal events.

**INPUT EVENTS** - these are either primary events or undeveloped events. They represent the limit of resolution of the transfer function within this component.

**DISCRIMINATOR** - determines which failure transfer functions may co-exist.

The use of failure transfer functions is illustrated in Figure 2.12, which contains the two basic failure modes for a fuse. It should be noted that since the two failure modes are mutually exclusive, the two transfer functions have different discriminators.

![Figure 2.12 Failure Transfer Functions for Fuse Model](image)

Failure transfer functions are thus analogous to minitrees and are generated by a conventional failure mode analysis.

### 2.8 PROBLEMS ENCOUNTERED AND THEIR SOLUTION

#### 2.8.1 INTRODUCTION

From the literature survey, three main categories of problem were identified. These can be summarised thus:
- the methodology must be able to support two-way fault propagation since the causes of any given hazard can lie either upstream or downstream of the current location.

- the most significant faults in process plants relate to control loops and trip systems since they are designed to counter the propagation of faults. Whilst a unit-based modelling technique approach makes the problem manageable and suitable for automation, it also invariably loses the implicit inter-relationships within the sub-systems which combine to form the control/trip systems.

- checks are needed during the fault tree construction phase to ensure the consistency of the overall tree.

The following sections will outline the approaches adopted by various workers to overcome these problems.

2.8.2 TWO-WAY FAULT PROPAGATION

Andow [Ref 2-21] and Andow and Lees [Ref 2-22] describe a convention for achieving two-way fault propagation through the use of propagation equations. The essential requirement is that two variables are required, one defined at the outlet of the unit and the other at the inlet. Taking the propagation of flow, as an example, through the following pipe unit:

```
1 -------> 2
```

Figure 2.13
Schematic Representation of Pipe Model

the following two equations would be used:

\[
Q_{2\text{OUT}} = f(P_{1\text{IN}} - P_{2\text{OUT}}) \\
P_{1\text{IN}} = f(Q_{1\text{IN}} - Q_{2\text{OUT}})
\]

It should be noted that two variables, flow (Q) and pressure (P) are used. Q being defined at the outlet will trace the upstream causes, whereas P defined at the inlet will trace the downstream causes.

If the event being developed is a high flow at the outlet of the pipe, then the sub-tree of Figure 2.14 would result from the above two propagation equations. Of the three events awaiting development in this tree:

- \( P_{2\text{OUT}} \) LO will propagate into the downstream unit
- \( Q_{1\text{IN}} \) HI will propagate into the upstream unit
- \( Q_{2\text{OUT}} \) LO is inconsistent and should be removed

The last event highlights an inherent problem in setting up a two-way propagation mechanism; left to its own devices it will loop around indefinitely. This problem was highlighted by Martin-Solis et al in [Ref 2-23]; a solution based on the use of boundary conditions was presented.
In the above paper Martin-Solis also highlights another problem in applying this approach. If basic events are included in the above tree then it would become:

Clearly PART-BLK in the pipe cannot cause Q2OUT HI; it will tend to cause Q2OUT LO. Hence the concept of Not Allowed Faults was introduced to overcome this problem.

Taylor [Ref 2-17] uses a similar technique to that described above for achieving two-way fault propagation. A pair of variables is used: SUPPR representing the supply pressure and tracing the upstream causes, and BACKPR representing a back pressure and tracing the downstream causes.
2.8.3 CONTROL LOOPS AND PROTECTIVE DEVICES

Much of the complexity surrounding fault tree synthesis is the result of sub-systems of this nature in the plant configuration. In order to overcome this problem, the following three basic approaches have been illustrated by various workers:

- the use of special operators when loops are encountered during the propagation path, e.g. by Lapp and Powers [Ref 2-1].

- to decompose the system based on the control/trip loops and to create models based on these sub-systems, e.g. by Shafaghi [Ref 2-10].

- to create failure models which contain not only information on failure modes, but also to AND these events with “No Compensation” events related to failures of the control loops and trip loops, e.g. by Taylor [Ref 2-17].

2.8.3.1 OPERATORS OF LAPP-POWERS

The Lapp-Powers [Ref 2-1] technique of fault tree synthesis is based upon the following principle:

When the causes of an event are developed, it is necessary to check that nothing else happens which will cancel the original effect.

In the methodology this is assumed to be true for all events other than those which are interconnected by feedback or feedforward loops in the digraph model. Consequently for those events which do not fall on loops of this nature, the causes can be linked by a simple OR gate; for events which fall on a loop the following two operators are applied dependent upon the type of loop.

![Negative Feedback Loop Operator of Lapp and Powers](image.png)
An important point about the above two operators relates to their application. In the case of the NFBL operator it is applied at each point along the loop. In the case of the NFFL operator it is applied just once when the point common to both sides of the loop is reached.

Lambert in [Ref 2-4] reviews the NFBL operator of Lapp-Powers and presents a 'simplified' alternative which need be applied just once when the loop is encountered. This operator is given in Figure 2.18, where:

- **n**: number of nodes on the system digraph for the negative feedback loop.
- **j**: node where disturbance enters loop and propagates downstream around the loop.
- **l**: node of original entry into the negative feedback loop.

### 2.8.3.2 WORK OF SHAFAGHI

Shafaghi et al in [Ref 2-10] describe an alternative system decomposition and modelling technique. It is based on the premise that fault trees resulting from the unit model representation yield poorly structured trees which appear opaque to the user. This situation tends to be most noticeable in the case of systems containing control loops and protective loops.

The technique described in the above paper is thus based on decomposition and modelling at the control loop and protective loop level. The following stages are involved:

- identification and definition of the control loops and the controlled variables in the system.
- the representation of the plant as a digraph of inter-connected control/protective loops.
Figure 2.18
Modified Operator of Lambert

- creation of failure 'models' for each loop containing the disturbance and fault information.

- synthesis of the fault tree is initiated by defining a deviation in one of the controlled variables.
A generic control loop operator of the following format is employed:
The use of this technique is illustrated using the Propane Pipeline Plant of Lawley [Ref 2-24].

The biggest disadvantage of the technique is the need to model the failures associated with a control loop each time that any given system is decomposed.

Kelly [Ref 2-25] and Kelly and Lees [Ref 2-15] describe a technique which combines the unit based technique with the sub-system level treatment required for control loops. The system decomposition and modelling is done at the unit level. However, the user is required to supply additional information to describe the functionality and structure of the sub-systems which make up the control loops and trip systems.

Use is then made of operators analogous to those described by Shafaghi to correctly structure the tree. Further details of these operators can be found in Chapter 6 of this thesis.

2.8.3.3 MODELLING TECHNIQUE OF TAYLOR

Taylor [Ref 2-17] treats loops by extending the cause and effect graph modelling concept from components to the system level. Hence negative loops (whether feedback or feedforward) are included by using special minitrees in those components which can potentially counter the disturbances. These minitrees have the general structure which has already been given in Figure 2.7. The most important feature is the presence of the No Compensation Branch.

2.8.4 FAULT TREE CONSISTENCY

As all automated methodologies are dependent upon the linking together of failure mode information for sub-sections of the system to form the overall tree, there is a tendency for this procedure to introduce inconsistencies. One cause of this has already been outlined in Section 2.8.2 with respect to two-way fault propagation.

Essentially two types of inconsistency may appear:

a) impossible events
b) certain events

Consistency of trees is ensured by analysing events against those already synthesised and removing those which are contradictory. A good discussion of this problem is presented by Salem et al in [Ref 2-18].

2.9 FAULT TREE CONSTRUCTION CODES

2.9.1 INTRODUCTION

This section outlines the significant features of a number of automated fault tree construction codes. The following packages are reviewed:

- DRAFT
- FTS
- CAT
- RIKKE
- FAULTFINDER


2.9.2 DRAFT

The application of this code is illustrated with respect to electrical systems in [Ref 2-20] by Fussell. The use of the technique is based upon:

a) modular decomposition of the system to identify the components.

b) creation of failure transfer functions for the components, as already described in Section 2.7.6.

c) determination of sub-systems (termed the “component coalition scheme”). In the context of electrical systems this means identifying the circuit paths.

d) the methodology differentiates between four categories of fault event:

i) first order events - fault events used as top events, which need to be developed manually before the automated methodology can take over.

ii) second order events - fault events which refer to sub-systems rather than individual components.

iii) third order events - events which cause a component to fail due to sub-system failure as opposed to component failure.

iv) fourth order events - fault events which cause component failure due to related component failure. These events are developed from the failure transfer functions.

Hence in addition to creating the failure transfer functions for the components, it is also necessary to specify the first, second and third order events.

e) the user has to specify the not allowed events of first and second order events.

2.9.3 FTS (FAULT TREE SYNTHESIS)

The application of this code is illustrated by Lapp and Powers in [Ref 2-1] using a heat exchanger system. The basic algorithm for this is as follows:

a) generate the digraph and find all the negative feedback and feedforward loops, as already described in Section 2.7.2.

b) select the node representing the top event.

c) determine local causes of this event by noting the inputs to the node of the digraph.

d) delete any local causes which violate consistency.

e) select the appropriate operator depending on whether negative feedback/ feedforward loop passes through the current node. If either one of these loops is involved then store the event for later consistency checks.
f) select a node corresponding to an undeveloped event and return to step c). If only primal events remain, stop.

2.9.4 CAT (COMPUTER AUTOMATED TREE)

The use of this technique has been illustrated by Salem et al in [Ref 2-18]. The basic algorithm for this is as follows:

a) the system is decomposed into components. Failure models for each component type are created using decision tables, as already described in Section 2.7.4. The system is represented as a chart of inter-connected components. This is achieved by defining nodes. A node is connected to one or more succeeding components. Hence, for each component it is necessary to define the component type, output node number and the input node number.

b) define the top event. This must be defined in terms of system states which can serve as starting points for the fault tree.

c) link this to output events of appropriate component model.

d) different rows of the decision table are linked by OR logic.

e) the events in the columns within any given row are linked by AND logic.

f) don't care events are left out of the tree.

g) internal mode events are flagged as primary events.

h) input events are developed further by backtracking to find causes in preceding components.

i) consistency checks are based upon defining system states and boundary conditions. A system state is a specific condition of the system either in terms of a node or an internal mode. Boundary conditions are system states which cannot be reset and persist as the tree is constructed.

All events at a particular point must be compatible with system states already defined. If a state has already been set then there are two possibilities:

- state is identical, in which case it is deleted as sure to occur.
- state is contradictory, in which case it is deleted as cannot occur.

Consistency of the tree is checked at three levels:

- during the construction of a gate, when checks are made against the pre-defined system states as described above.

- the above step may yield excess, redundant or contradictory gates or events. These are removed after completing each gate or group of gates, using the following criteria:

  i) eliminating single input gates
  ii) checking all primary inputs beneath just completed AND gates for consistency and redundancy


iii) checking all primary inputs beneath just completed OR gates for reduction of inputs to minimal cutsets.

- removal of "good" states from the tree by assuming they have a probability very close to 1, and the production of transfers within the tree.

2.9.5 RIKKE

The use of this package has been described by Taylor [Ref. 2-17]. The essential aspects of the methodology have been described in Sections 2.7.3 and 2.8.3.3. Since a very detailed modelling procedure is used, whereby the minitrees contain compensatory events as well as fault causes, the synthesis algorithm itself simply involves the combination of these minitrees by propagating faults from one unit to another.

The main features of the package are:

- system is specified by defining the components and their inter-connections. This is done by building a system flow-sheet using a graphical front-end.

- for each component type it is necessary to define the function and failure model.

- the top event is defined in terms of output events of a given component mini fault tree.

- fault tree construction proceeds by locating the mini fault trees for the input events and any non-normal condition events. The non-normal conditions are developed first, followed by the input events. Once the causes of an event within the component have been determined, then the causes are located by searching through the associated components. The system handles positive feedforward and negative feedforward/feedback loops using the following criteria:

  a) loops are detected by keeping track of which mini fault trees have been added to the tree, such that when new causes are sought for input events, a search is made to check if the input event is the output event of a mini fault tree already added to the tree.

  b) negative loops are handled by having special minitrees of the form "output disturbance arises if input arises and a component remains in non-compensating state".

  c) the causes of a non-compensating state are found by finding reasons why a compensation event should occur and then negating these causes.

2.9.6 FAULTFINDER

The use of this package is described by Kelly and Lees in [Ref 2-15]. The essential features of the methodology are reviewed in Chapter 3 of this thesis.

2.9.7 WORK OF NAPIER AND PALMER

The main thrust of their paper [Ref 2-26] is to consider the practical aspects of actually implementing a computer-based system, from the perspective of a design engineer. The following are cited as the minimum requirements:

- programs must be highly interactive, allowing for changes in the design based upon the analysis.
it should be possible to alter and improve upon the logic implicit in the synthesis.

- programs should be upwardly portable to derive the advantages from the more powerful generation of computers.

The methodology is based upon developing an overall tree from a library of sub-trees. The illustrative examples presented are based upon a modular decomposition of the plant at the unit level and the synthesis of sub-trees for these. It is not clear how the methodology would handle control loops or how consistency is ensured.

Much emphasis is placed upon the definition of an architecture suitable for enabling different fault tree construction techniques to be utilised to build a fault tree from the sub-trees.

The most significant points relate to the issues raised in terms of developing a package which can be integrated into the design phase and making it appealing to the design engineer.

**2.9.8 CAFTS (COMPUTER AIDED FAULT TREE SYNTHESIS)**

The use of this package is described by Poucet in [Ref 2-27]. One of the main thrusts of this paper is to consider the relative merits of a fully automatic and an interactive approach to fault tree synthesis. The technique described is an automated methodology, which allows the user to control the synthesis of the tree at each level of development, in an analogous manner to a computer-aided design package.

The following are the main steps in synthesising a fault tree:

a) split the system into sub-systems.

b) define the failure criteria for the top event at the output of a given sub-system.

c) enter the description of the current sub-system.

d) automatic generation of a macro fault tree for the sub-system output event, using transfer logic models which define the causes of an output event in terms of inputs and internal states.

e) interactive expansion of the macro fault tree for the sub-system by specifying the components which go to make up each sub-system and using the modular component models.

f) inputs to the system are further developed by repeating steps c) to e), and specifying the logic to link the events between the sub-systems.
3.0 OUTLINE OF METHODOLOGY

3.1 INTRODUCTION

This chapter describes the basic methodology by outlining the three main stages involved in the generation of fault trees using FAULTFINDER. The three main stages are:

- system decomposition
- creation of failure models
- fault tree synthesis

Each of these stages will be discussed more fully by reference to the Lapp-Powers Heat Exchanger System. This has already been used extensively in the literature by other workers in this field. A detailed description of this system can be found in Appendix B.

3.2 SYSTEM DECOMPOSITION

3.2.1 INTRODUCTION

During system decomposition it is necessary to translate all the relevant features of this system into a form suitable for input to the computer program. For any system the minimum amount of information required is the unit models to be used, the way these units are linked together, and the top event of interest.

3.2.2 THE DECOMPOSITION DIAGRAM

The decomposition diagram for this system is given in Appendix B, Figure B.2. The actual units specified will depend upon the decisions made at the decomposition stage. There is flexibility within the methodology to allow the user to choose the level of decomposition. Comparing the flow and decomposition diagrams for the current system, Figure B.1 and B.2 of Appendix B respectively, it can be seen that there is in general a one-to-one correspondence of units. The additional units in the decomposition diagram are:

- dummy heads/tails (Units 1, 6, 7 and 11). These are specific units used to impose an envelope around the system under study.

- pipe (Unit 4). It should be noted that the specification of connections between units does not in itself imply the existence of any physical links. The numbered streams in the decomposition diagram are the logical connections representing the flow of information from one unit to another unit. The arrow indicates the normal direction of flow. The single pipe unit has been included for demonstration purposes; it has not been universally modelled between each unit as this would produce a very large fault tree clouded with fairly trivial pipe-type faults.

- the setpoint units (Units 13 and 15) represent the source of the setpoint value to the controller (Unit 12) and the trip switch (unit 14), respectively. The electrical power supply (Unit 17) and the instrument air supply (Unit 16) represent the flow of utility to the pump (Unit 7), controller (Unit 12) and the trip switch (Unit 14).
The representation of common components, like utilities, as separate units is a convention in the methodology to model common-mode failures. Consider again the setpoint units. If it had been assumed that the setpoint value had been assigned by the same person then both the controller and the trip switch would have been linked to the same setpoint unit. Using this technique, errors in the setpoint value would have been traced to the same unit, thereby correctly finding the common-mode failure.

During the decomposition of this system it has been assumed that the two setpoint values were assigned independently. A more detailed discussion of this modelling convention is presented in Chapter 8, with respect to utility supply failures.

3.2.3 OTHER LEVELS OF DECOMPOSITION

There are other levels of decomposition which may be carried out. Consider the trip system in the decomposition diagram. The flow sensor (Unit 9) has been modelled as a single unit which emits a signal dependent upon the flow. One extreme would be to decompose the flow sensor as an orifice plate, the connecting pipework and valves, and a pressure transducer. The other extreme would be to decompose a number of items in the flow diagram as a single unit. For example the flow sensor (Unit 9) and trip switch (Unit 14) could be decomposed in this manner.

The actual level of decomposition will depend upon the nature and depth of the analysis being carried out. The essential criterion is that the decomposed diagram should not alter the functionality of the system as given in the flow diagram.

During this project an intermediate level of decomposition has been carried out which generally yields a one-to-one correspondence between units in the flow and decomposition diagrams. This is borne out by the current example.

This level of decomposition has also yielded models which are fairly context-independent and so can be re-used in a number of applications, without unnecessarily complicating the modelling process.

3.2.4 INPUT OF THE DECOMPOSITION DIAGRAM TO FAULTFINDER

Having prepared the decomposition diagram, it is necessary to input this information to the computer program. Section B.4 of Appendix B gives a complete listing of all the input required by FAULTFINDER to generate a fault tree for the current system.

For each unit the corresponding model library reference number is specified. If the model does not exist in the model library, then it must be created using the model generation program. The information on how the various units are linked together is specified by reference to each connection. Four items of data are specified for each connection: the upstream unit and port numbers, and the downstream unit and port numbers. Upstream and downstream refer to the normal direction of information flow.

For most systems this represents the minimum amount of information needed to synthesise the fault tree. However, the atomistic approach taken during the decomposition stage, whereby the flow diagram is translated into a series of inter-connected unit models, tends to destroy the relationship between components forming control and trip loops. The failure modes associated with these sub-systems cannot be modelled via the individual unit models. Consequently, additional information is needed to define the functionality of control and trip loops.
The additional information needed for control loops is:
- sensor unit number
- control valve unit number
- the other units in the control loop, e.g. controller
- the sensed variable
- the connections where the sensed variable is regulated
- whether there is a separate manipulated stream; if so, then the connections where the flow is manipulated
- whether the control loop is feedforward or feedback

The additional information needed for trip systems is:
- trip valve unit number
- other units in the trip system
- if the trip system is one where the trip valve opens to provide flow, the connections which will have flow when the valve opens.

Supplying this information alerts the synthesis algorithm to the presence of sub-systems which require special treatment. This information can then be used to re-structure the fault tree accordingly. Control loops and trip systems as used within the methodology are discussed more fully in Chapters 6 and 7, respectively.

(A discussion of the other type of sub-system identified in Appendix B is contained in Chapter 5 on Divider-Header Combinations).

### 3.3 MODELLING

#### 3.3.1 INTRODUCTION

In order to synthesise a fault tree, failure models must exist for all the units identified during the decomposition stage. The general approach has been to generate context-independent models which can be used in a variety of applications. Partly due to this objective, the top event of a fault tree is modelled separately from the models for the units. If this were not the case, then any given top event would have to be included in each unit model. This is clearly a cumbersome approach and has thus been rejected.

The purpose of a unit model is to express both how faults propagate through the unit and also how they initiate within it. The principles of unit modelling will be highlighted by reference to the heat exchanger model used in the Lapp-Powers system. A schematic representation of this model and the information needed to specify the failure model is given in Section B.5 of Appendix B.

#### 3.2.2 UNIT PORTS

Firstly it is necessary to identify the ports on the unit. A port is the entity via which information flows into and out of the model. The heat exchanger model thus has four ports. Ports 1 and 2 refer to the flow of nitric acid, and ports 3 and 4 refer to the flow of coolant. The ports can be numbered in any order, but any given unit is allowed a maximum of nine ports (numbered 1 to 9). It is also necessary to classify the type of port. The methodology recognises the five basic types of port given in Table 3.1 below:
**3.3.3 PROPAGATION EQUATIONS**

**3.3.3.1 INTRODUCTION**

Propagation equations describe the propagation of disturbances through the unit. A disturbance is represented as a variable deviation of the form \( X_{NPTYPE \ DEV} \) where:

- \( X \) is a variable from the list of variables
- \( N \) is the port number (1-9)
- \( PTYPE \) is the port type from the list of port types
- \( DEV \) is the deviation in the variable from the list of deviations

A propagation equation takes the general form:

\[
X = F(Y, -Z)
\]

and would be interpreted as:

- \( X \) deviates high if \( Y \) deviates high or \( Z \) deviates low
- \( X \) deviates low if \( Y \) deviates low or \( Z \) deviates high

Hence, the propagation equations are a convenient way of representing a lot of information and can either be arrived at heuristically or can be derived from the differential or algebraic equations applicable to the model. They are ideal for fault tree work, because the information of interest is the discrete deviation of variables from some normal value.

The different port types have already been considered under Section 3.3.2 above. The complete set of variables and deviations recognised by FAULTFINDER is given in Tables 3.2 and 3.3 below:

<table>
<thead>
<tr>
<th>PORT TYPE</th>
<th>IDENTIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inlet</td>
<td>IN</td>
</tr>
<tr>
<td>Outlet</td>
<td>OUT</td>
</tr>
<tr>
<td>Vessel</td>
<td>VES</td>
</tr>
<tr>
<td>Utility</td>
<td>UTIL</td>
</tr>
<tr>
<td>Signal</td>
<td>SIG</td>
</tr>
</tbody>
</table>

Table 3.1
List of Port Types

Hence ports 1 and 3 are inlets, and ports 2 and 4 the outlets.

A vessel port is used to model the internal variables of a model, e.g. level. A utility port is reserved for modelling utility supply failures, e.g. power supply. A signal port is used to model failures in the signal flow, e.g. the signal to the control valve affecting its stem position.
The use of each of these variables is described below in Section 3.3.3.2. Section 3.3.3.3 then discusses the valid combination of deviations relating to each variable.

### 3.3.3.2 FAULTFINDER VARIABLES

#### 3.3.3.2.1 Basic Conventions

The basic modelling conventions will be described with respect to the flow variables.

In order to achieve two-way fault propagation and ensure that the propagation chain does not break down, certain rules have to be followed in the setting up of propagation equations. Consider the equations for the flow of nitric acid through the heat exchanger:
There are a number of points to note about these two equations:

- Flow $Q$, is modelled in terms of pressure gradient $G$, and vice versa. The absolute pressure $P$, is not used to model flow. This greatly simplifies the modelling process as there is a direct correspondence between the deviations in the $G$ and $Q$ variables, e.g. high pressure gradient yields high flow etc.

- Two variables are needed to achieve two-way fault propagation, one being defined at the inlet port and the other at the outlet. Figure 3.1 gives the partial fault tree for the high flow of nitric acid through the heat exchanger. The $G_{2}\text{OUT HI}$ branch would propagate to the downstream units and the $Q_{1}\text{IN HI}$ branch to the upstream units. The boundary condition checks prevent the propagation chain from looping around. This is the reason why $Q_{2}\text{OUT HI}$ would not appear as a cause of $G_{1}\text{IN HI}$.

- The left hand side of the propagation equation is termed the output. The modelling convention developed stipulates that $Q$ should be output at the outlet ports and $G$ at the inlet ports, i.e. the propagation equations are in terms of $Q_{2}\text{OUT}$ and $G_{1}\text{IN}$. 

![Partial Fault Tree for High Flow at Outlet of Heat Exchanger](image-url)
3.3.3.2 Other Variables

Pressure (P) and Relief (R)

Although flow Q is modelled in terms of pressure gradient G, an absolute pressure variable P is available for modelling deviations in this variable. Again there is a need for two-way fault propagation, since the deviation can either arise upstream or downstream of a given point. Since a single variable can only trace faults in one direction, it is necessary to have a second variable to meet this requirement. The variable chosen for this is relief R. The equations for these two variables are:

\[
P_{\text{OUT}} = F(P_{\text{IN}}) \\
R_{\text{IN}} = F(R_{\text{OUT}})
\]

It will be noted that unlike G and Q it has not proved possible to derive equations to directly relate the P and R variables. However, the principle is the same. P is defined at the outlet and so can find the upstream causes, whereas R is defined at the inlet and so can trace the downstream causes. One way that this principle has been used is illustrated in Figure 3.2. This top event model could be used for developing the causes of high pressure at the outlet of the heat exchanger in relation to the nitric acid stream. The P branch in this figure would use the P_{\text{OUT}} equation to find the upstream causes, whereas the R branches would find the downstream causes by using the equation for R_{\text{IN}}.

![Figure 3.2](image)

This top event model indicates that a high pressure can occur due to any of the following reasons:

- high pressure upstream (P_{\text{OUT}} HI)
- reduced relief downstream (R_{\text{IN}} LO, e.g. partial blockage)
- no relief sink downstream (R_{\text{IN}} NONE, e.g. complete blockage)
- relief reversal, i.e. back pressure (R_{\text{IN}} REV)
Temperature (T/U) and Composition (X/Y)

Temperature deviations can arise either due to flow in the normal direction and an upstream source causing the deviation, or reverse flow transmitting the deviation from a downstream point. Again in order to meet the two-way propagation requirement, two variables are needed.

\[
\begin{align*}
T2OUT &= F(T1IN) \\
U1IN &= F(U2OUT)
\end{align*}
\]

In this case T2OUT is defined at the outlet to trace the upstream causes. The other variable, U1IN, has been defined at the inlet to trace the downstream causes. An example of the use of these variables is given in the top event model for HIGHTEMP in Figure 3.5.

Composition is modelled in exactly the same way as temperature, and in the absence of any mass transfer within the unit the propagation equations would simply be:

\[
\begin{align*}
X2OUT &= F(X1IN) \\
Y1IN &= F(Y2OUT)
\end{align*}
\]

In this instance X2OUT would be used to find the upstream causes, and Y1IN the downstream causes.

Only in units where mass transfer is taking place, is it necessary to assign components to the composition variables. Thus for a binary system propagation equations would have to be written for XA2OUT, XB2OUT, YA1IN, YB1IN.

Level (L), Signal (S) and Setpoint (W)

Level is only applicable to vessel-type units and represents accumulation within the unit. Consequently, it would only be defined at a vessel port. The use of propagation equations in conjunction with vessel-type ports is elaborated in Chapter 4.

The signal variable is used exclusively to model the flow of signals, electrical or pneumatic, in control and trip systems. It is defined at a signal port. It is much simpler to model than the process variables as the methodology does not require two-way propagation of this variable. The existence of a loop ensures that the complete path can be traced by simply tracing faults in one direction only.

The setpoint variable is used exclusively to model the setpoint value to controllers and trip switches. It is defined at a signal port and is analogous to the signal variable.

The use of the last two variables is illustrated by reference to the reverse-acting controller model of Figure 3.3.

The propagation equation for signal flow through this unit is given below:

\[
S2SIG = F(-S1SIG, W3SIG)
\]

This says that the output signal will be:
3.3.3.3 DEVIATION OF VARIABLES

The methodology makes use of nine deviation states as already given in Table 3.3. It should be noted that not all the deviations apply to all the variables. The deviations corresponding to any given variable are presented in Table 3.4:

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>HI</th>
<th>LO</th>
<th>NONE</th>
<th>REV</th>
<th>SOME</th>
<th>NCHA</th>
<th>SHAC</th>
<th>NOR</th>
<th>NOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>G</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>T</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>U</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>P</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>R</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>S</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>L</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>W</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.4
Deviations Corresponding to Each Variable

It should be noted that a dash in the above table indicates that the deviation state does not apply to that particular variable, whereas a Y indicates a valid combination.

The approach taken has been to keep the number of deviation states to a minimum to facilitate the modelling. Most of these are self-explanatory. The meaning of the less obvious are summarised below:

a) Low, if the input signal is high or if the setpoint is low
b) High, if the input signal is low or if the setpoint is high.
$SNCHA$ is used to indicate an invariant signal, when a change is expected. It is used to model both trip loop functional failure and control loop latent failure.

$SHAC$ is used only to model trip loops and is a logical state variable. It is used to trace the events which should cause the trip to activate.

$P\ REV$ models a reversal of pressure. A pressure source is effectively turned into a relief sink.

$R\ REV$ models a reversal of relief. A relief sink is turned into a pressure source.

The $NOR$ and $NOP$ deviations arise because $P$ and $R$ are used as a pair of variables and that one variable is only capable of tracing faults in one direction at a time.

$PNOR$ models a pressure deviation and indicates that the pressure is so high that there can be no relief at this point.

$RNOP$ models a relief deviation and indicates that the potential for relief is so high that there can be no back pressure at this point.

### 3.3.4 EVENT STATEMENTS

#### 3.3.4.1 INTRODUCTION

The propagation equations of a model describe how variable deviations propagate through the unit. However, variable deviations must initiate somewhere. Event statements are a means of defining initiating faults and the variable deviations arising from these.

The general format of an event statement is:

**Identifier** FAULT: Variable Deviation List

The **Identifier** indicates the type of fault being modelled and can be one of the following:

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Basic Fault</td>
</tr>
<tr>
<td>O</td>
<td>Operator (In)action</td>
</tr>
<tr>
<td>I</td>
<td>Intermediate Event</td>
</tr>
<tr>
<td>S</td>
<td>Status Flag</td>
</tr>
<tr>
<td>V</td>
<td>Variable Deviation</td>
</tr>
</tbody>
</table>

Table 3.5  
Fault Identifiers

The **FAULT** is the actual fault name. A selection of fault names used by **FAULTFINDER** is given in Table 3.6. **FAULTFINDER** keeps a complete list of all the faults used in all the models. If a new fault name is utilised during the creation of a new
model, then the fault list should be updated by the user. Fault names are restricted to eight characters.

<table>
<thead>
<tr>
<th>FAULT NAME</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>LK-LP-EN</td>
<td>leak to a low pressure environment</td>
</tr>
<tr>
<td>HV-F-SH</td>
<td>hand valve fails shut</td>
</tr>
<tr>
<td>COMP-BLK</td>
<td>complete blockage</td>
</tr>
<tr>
<td>SEN-STK</td>
<td>sensor stuck</td>
</tr>
<tr>
<td>CV-F-HI</td>
<td>control valve fails in high aperture</td>
</tr>
</tbody>
</table>

Table 3.6
Selection of Fault Names Used in FAULTFINDER

The colon separates the causes on the left from the output event(s) on the right.

The right hand side will usually be a list of variable deviation events, as a single cause will normally affect a number of variables.

3.3.4.2 EVENT STATEMENTS IN THE HEAT EXCHANGER MODEL

The complete list of event statements as used in the heat exchanger example is presented in Section B.5 of Appendix B. This is quite a lot of information for the user to have to specify. The next chapter in this thesis discusses ways of reducing this input.

Each of the event statements will now be discussed.

a) F  EXT-HEAT: T2OUT HI, T4OUT HI, U1IN HI

If there is an external heat source then the shell-side fluid - the coolant - will be heated, giving rise to T4OUT HI. This will result in the hot stream being insufficiently cooled, giving rise to T2OUT HI. If the flow of the hot stream were reversed then, the reverse temperature of this stream would also be high causing U1IN HI.

The event statement for an external cold source is analogous to the above, but with the deviations being LO rather than HI.

b) F  FOULING: T2OUT HI, T4OUT LO, U1IN HI, U3IN LO

It has been assumed that fouling increases the resistance to heat transfer but does not affect the flow. The increased resistance to heat transfer means that the hot stream will not be sufficiently cooled, resulting in the cold stream being heated to a lesser degree. This applies to both the normal and reverse flow directions.

c) V  G1IN LO:D(DUMMY)
V  G1IN NONE:D(DUMMY)
V  G1IN REV:D(DUMMY)
I  D(DUMMY):T4 OUT LO, U3IN LO
In order to explain the need for these event statements it is necessary to look at the propagation equation for T4OUT and to consider the way the model generation program analyses this equation. This equation is:

\[ T4OUT = F(G1IN, -G3IN, T1IN, T3IN) \]

Using the rules specified earlier, one cause of T4OUT LO will be G1IN LO. However, it can be seen that T4OUT LO will also result if G1IN is NONE or REV. The latter will be the case if it is assumed that the reverse flow is less than the normal flow.

Similar reasoning applies to the U3IN LO event.

Another point to note is the use of the intermediate event D(DUMMY). There are three main reasons for the use of intermediate events in the methodology. Firstly, it is used for generic faults which may be caused by several initial faults but which give rise to the same output event. Here all the flow faults giving rise to the temperature deviations are grouped together to enhance the structure of the fault tree. Secondly, intermediate events are used to ensure that any given mini-fault tree in a model consists of only one level of development. Thirdly, they can be used to incorporate AND logic into the model (see the section on Decision Tables below).

In a similar manner the event statements relating to E(DUMMY) model the deviations in the temperature of the hot stream as a result of deviations in the coolant stream that cannot be obtained from the propagation equations.

d) The event statements relating to the faults LK-LP-EN, INT-LK, PART-BLK and COMP-BLK model the effects of leakages and blockages. For this model it has been assumed that LK-LP-EN would affect the flow of the shell-side fluid only; that the tubes are at a higher pressure than the shell, such that an INT-LK would cause flow from the tubes to the shell only; and that the faults COMP-BLK and PART-BLK apply only to the tubes.

i) consider initially the event statement for LK-LP-EN. This will tend to increase the upstream flow but decrease the downstream flow. Flow is modelled using a pair of variables: G defined at the inlet and Q at the outlet. Since the deviations in G and Q are equivalent, the effect of a leak on the inlet pressure gradient can be obtained by simply considering the effect on the inlet flow.

Hence LK-LP-EN will cause:

G3IN HI, G3IN SOME, i.e. increase the inlet pressure gradient for flow, or create one where none existed before.

Q4OUT LO, Q4OUT NONE, Q4OUT REV, i.e. reduce, terminate or even reverse the outlet flow.

R3IN HI, R3IN SOME, R3IN NOP, i.e. increase the inlet relief or create a relief source where none existed before or even increase the inlet relief so as to eliminate any possibility of a back pressure.

P4OUT LO, P4OUT NONE, P4OUT REV, i.e. reduce or even reverse the pressure at the outlet.
Additionally, a shell-side leak will reduce the amount of coolant flowing through the exchanger, resulting in an effect on the temperatures. All the outlet temperatures will thus be higher.

ii) the assumption made above about the relative pressures of the shell and tubes, means that the effect of an internal leak will be to increase the inlet flow of the hot stream, but to reduce its outlet flow. Thus the influence on G, Q, P and R for the hot stream is analogous to that for LK-LP-EN given above for the shell side fluid. The effect of this on the coolant stream will be to decrease the flow into the exchanger but to increase the flow out of it.

Thus considering the coolant stream only, the effect of an INT-LK will be:

G3IN LO, G3IN NONE, G3IN REV, i.e. reduce or reverse the inlet pressure gradient.

Q4OUT HI, Q4OUT SOME, i.e. increase or create an outlet flow where none existed before.

R3IN LO, R3IN NONE, R3IN REV, i.e. reduce or reverse the relief into the heat exchanger.

P4OUT HI, P4OUT SOME, P4OUT NOR, i.e. increase or create a pressure at the outlet of the exchanger. The pressure could be increased so much that there may be no means of relief into the unit.

In addition to the flow deviations, the internal leakage will cause temperature and composition deviations. Since less of the hot stream is flowing through the tubes, it has been assumed that the outlet temperature of this will be lower, and since the hot stream is mixing with the cold stream, that the outlet temperature of the coolant will be higher than normal.

The deviations X4OUT HI and Y3IN HI indicate the presence of impurity in the coolant stream. It is assumed that the hot stream is a single component, thus there is no need to use component indices with the X or Y variables.

iii) a partial blockage will have the following effects:

G1IN LO, Q2OUT LO, i.e. partially reduce the inlet pressure gradient and the outlet flow.

R1IN LO, P2OUT LO, i.e. reduce the relief into and the pressure out of the unit.

T2OUT LO, T4OUT LO, U3IN LO, i.e. reduce the respective temperatures since less hot fluid is passing through the system.

iv) a complete blockage is analogous to a partial blockage and causes:

G1IN NONE, Q2OUT NONE, i.e. eliminates the pressure gradient into the unit and the flow out of it.

R1IN NONE, R1IN NOP, i.e. prevents relief into the unit and prevents a back pressure from propagating through the unit.
P2OUT NONE, P2OUT NOR, i.e. prevents pressure at the outlet and prevents the unit from providing a relief from downstream.

T4OUT LO, U3IN LO, i.e. reduces the temperature of the coolant since no cooling is taking place.

e) an assumption made is that the amount of reverse flow is small compared to the normal flow. Thus under reverse flow conditions U1IN LO and U3IN HI for the hot and cold streams, respectively, would be normal events.

The following event statement encapsulates this information:

S NORMAL: U1IN LO, U3IN HI

Of the five types of event statement highlighted, none appears in this model for operator (in)action. An example of such an event statement from a hand valve model which is normally open is given below:

O HV-D-SH: Q2OUT NONE, P2OUT NONE

3.3.5 DECISION TABLES

The general format of a decision table as used by the methodology is:

I Cause I Cause T Output Event List

where:

I is the cause identifier and could be any one of F, O, I, V and S as defined for the event statements. The Causes are the combination of events necessary to give rise to the output events. The Output Event List is the effects and can either be variable deviations or intermediate events. The T separates the causes from the effects.

Any given effect can only occur if all the causes occur simultaneously. Hence, decision tables are used by the methodology for introducing AND logic into the model.

No decision tables were used in the heat exchanger model. Their application can be highlighted by reference to the following hand valve model which is normally closed:

![Schematic Representation of Hand Valve Model](image-url)
In order to get some flow at the outlet, there must be a pressure gradient for flow at the inlet and the valve must be open to allow flow. This scenario can be modelled by a combination of two event statements and a decision table:

\[
\begin{align*}
F \quad & \text{HV-F-OP: A(DUMMY)} \\
O \quad & \text{HV-D-OP: A(DUMMY)} \\
V \quad & \text{G1IN SOME I A(DUMMY) T Q2OUT SOME}
\end{align*}
\]

The opening of the valve either spontaneously or by operator action is included as causes of A(DUMMY). This event is then combined with an inlet pressure gradient deviation, via the decision table, to model the outlet flow deviation. This example illustrates how AND/OR logic can be combined in a model; the use of the intermediate event ensures that there is only one level of development at a time. The interpretation of decision tables as intermediate events is illustrated in section 3.3.7 below (see Figure 3.5).

### 3.3.6 TERMINAL EVENT MODELLING

For reasons already given, the top event of the fault tree is modelled separately from the unit models. The top event of interest will normally be a textual string, e.g. HIGHTEMP for the current example. The primary purpose of the top event model is to develop this into variable deviation events which can in turn be traced to the units in the plant.

Assume that the top event of interest is a high temperature at the outlet of the pipe (Unit 4). This can be modelled by using the following two decision tables:

\[
\begin{align*}
V \quad & \text{T2OUT HI V G1IN SOME V Q2OUT SOME T HIGHTEMP} \\
V \quad & \text{U1IN HI V Q2OUT REV T HIGHTEMP}
\end{align*}
\]

It is important that the port numbering in the top event model should match that of the unit where the top event is being specified. Since port 1 in the pipe model is an inlet and port 2 an outlet, this top event model will work correctly. In actual fact it would be adequate for any model where there was a similar correspondence of ports.

### 3.3.7 MINI-FAULT TREES

Thus far the sections on propagation equations, event statements and decision tables have described three paradigms that the user can use to input information relating to fault propagation and fault initiation. This section describes how the model generation algorithms translate this information into a suitable format for use by the synthesis algorithm.

The basic procedure used is to generate mini-fault trees (minitrees) for all the output events in the model. The minitrees represent how the minitree top event (minitop) is caused within the model.

The minitree arising from the above top event model for HIGHTEMP is presented in Figure 3.5.

The events DTROW1 and DTROW2 are just intermediate events and identify the decision table row representing the failure mode.
3.4 FAULT TREE SYNTHESIS

Once the unit and top event models have been created and the data on the decomposed system been specified, the synthesis process is entirely automatic. Fault tree synthesis commences with the top event and proceeds until all the branches terminate in basic or diamond event causes.

Consider again the top event model for the current system given in Figure 3.5. This minitree contains five transmissive events awaiting development: T2OUT HI, G1IN SOME, Q2OUT SOME, U1IN HI and Q2OUT REV. The synthesis package will now search for minitrees corresponding to all these events in the pipe model. Taking T2OUT HI as an illustrative example, the package will find the mini-fault tree for high temperature at the outlet of the pipe and add it to the already developed tree. One cause of this new event will be high temperature at the inlet of the pipe. This is clearly an event requiring further development. The program will now determine which unit is upstream of the pipe and will search for the minitree for T1IN HI in this new unit.

At the very simplest level, fault tree synthesis proceeds in this manner, whereby the minitrees are extracted from the respective models, and are then connected together to form an overall tree. Even with this fairly trivial approach, checks are needed to ensure the consistency of the resultant fault trees. These checks are essentially of two types and ensure series and parallel consistency of the tree.

Series consistency refers to the consistency of events within any given branch; parallel consistency refers to the consistency between events in different branches.

Consistency checks are carried out by noting the state of the plant at any given level of development. Boundary conditions are set up defining the events which would violate this state. Any inconsistent events are then deleted from the tree. Consistency checks are
considered in more detail in the next section.

However, as mentioned during the account of the decomposition stage, the methodology requires that certain sub-systems be identified so that they may be accorded special treatment during the synthesis stage.

In the Lapp-Powers System the event T2OUT HI, in Figure 3.5, can only propagate if the control loop and the trip system fail to take corrective action. Fault tree synthesis is therefore complicated by having to identify those events which these sub-systems can protect, and where necessary to incorporate the failures of these systems into the fault tree.

A detailed discussion of these sub-systems is delayed for later chapters relating to trip systems, control loops and divider-header combinations. Chapter 10 gives a detailed account of each step of the synthesis algorithm for generating a fault tree for the Lapp-Powers heat exchanger system.

3.5 CONSISTENCY CHECKS

This section outlines five basic consistency checks carried out by the fault tree synthesis algorithm.

3.5.1 BOUNDARY CONDITIONS

Boundary conditions are automatically imposed by the variable deviations in the fault tree, and define the plant states which are inconsistent with the current plant state. Considering the model for the pipe unit:

```
1 ----> 2
```

Figure 3.6
Schematic Representation of Pipe Model

The fault tree for Q2OUT HI in the above model is given in Figure 3.7. It shows the joining together of two minitrees to form an overall tree. It should be noted that the first occurrence of Q2OUT HI imposes boundary conditions relating to the plant state. For the current branch the following checks would be imposed:

- at the current location no other deviations in the flow variables can occur
- the occurrence of the same deviation for the flow variable will add nothing to the fault tree.

Hence, the second occurrence of Q2OUT HI would be deleted.
3.5.2 NOT ALLOWED FAULTS

Since LK-HP-EN is noted as a cause of Q2OUT HI, it is obvious that LK-LP-EN cannot cause this event (although it is a valid cause of G1IN HI). Hence, LK-LP-EN would be deleted as a not allowed fault of Q2OUT HI.

3.5.3 DUPLICATED EVENTS

If instead of Q2OUT HI, the fault tree was being developed for Q2OUT LO for the pipe model, then it would have the form given in Figure 3.8.

In this case the second occurrence of PART-BLK is a duplicated event and is removed from the tree, since it adds nothing new to the fault tree.
3.5.4 PORT CHANGES

Consider the header model of Figure 3.9. Suppose the event $G \text{1IN HI}$ is being developed. The fault tree for this would be of the format given in Figure 3.10.

When port 2 is reached, the path can either propagate out of the unit or it can reverse back into port 3. This is due to the two-way propagation facility. However, the reversal through port 3 will be inconsistent with the event $G \text{1IN HI}$.

This type of problem is overcome by counting the number of inlet/outlet ports through which the propagation path has passed. The number allowed is limited to two. Any other occurrences are deleted from the tree. Hence, $G \text{3 HI}$ will be deleted since this event occurs at a third port in this unit.

3.5.5 PARALLEL CONSISTENCY

All the checks described thus far have been series consistency checks; they are carried out with respect to the branch already synthesised. Parallel consistency checks are needed to ensure that all the branches under an AND gate are consistent. They can only be carried out once the whole tree has been synthesised.
Figure 3.9
Schematic Representation of Header Model

Figure 3.10
Fault Tree for High Flow in Leg 1
3.6 ADVANCED FEATURES

This section describes two advanced features of the methodology for handling sequential operations and secondary failures.

3.6.1 SEQUENTIAL OPERATIONS

The FAULTFINDER methodology has the ability to handle sequential operations. This is in connection with situations where the unit models comprising the plant need to be changed at the start of each step in the sequence. Under such circumstances the plant configuration at each step in the sequence needs to be defined. Examples of configurational changes are:

- valve changing from open to closed
- pump being switched on/off

Since the failure models for an open/closed valve are different, it is necessary to specify which failure model should be used at each step in the sequence. Additionally, the user can specify a different top event model for each step in the sequence.

For each step in the sequence, FAULTFINDER synthesises a separate sub-tree. The overall structure of the tree is as follows:

![General Structure of Sequential Operations Fault Tree](image)
The above fault tree is shown for three steps only; a maximum of twenty steps are allowed. The top event by default is termed SEQ-ABRT (Sequence Aborts) for sequential operations. There are essentially two branches to this. SEQ-F-AT (Sequence Fails At) contains the sub-tree for that step; SEQ-F-AF (Sequence Fails After) traces down to the next step in the sequence.

3.6.2 SECONDARY FAILURES

It has already been mentioned that the FAULTFINDER modelling has been carried out in a context-independent manner. In keeping with this philosophy it is possible to create secondary failure models which can be applied to sections of the plant in specific applications. Examples of these scenarios for the Lapp-Powers system could be:

- high temperature of coolant/process stream could be due to an exothermic reaction caused by the mixing of water and nitric acid.

- a leakage in the coolant stream could be caused by the nitric acid leaking into it and causing corrosion.

Failures relating to these scenarios can be incorporated into the tree by specifying which units or streams are susceptible to the given secondary failures. This information would be specified at the decomposition stage through the MASTER program.

3.7 CONCLUSION

This chapter has outlined the main features of the FAULTFINDER MK1 methodology. Subsequent chapters will describe the enhancements made to this during the course of this project, culminating in FAULTFINDER MK2.
| CHAPTER 4 | ENHANCEMENT OF THE MODEL GENERATION ALGORITHM |
| CHAPTER 5 | ELIMINATION OF DIVIDER-HEADER COMBINATIONS |
| CHAPTER 6 | RULES FOR THE DECOMPOSITION OF CONTROL LOOPS |
| CHAPTER 7 | RULES FOR HANDLING TRIP SYSTEMS |
| CHAPTER 8 | MODELLING UTILITY FAILURES |
| CHAPTER 9 | FAULT TREE ANALYSIS |
| CHAPTER 10 | WORKED EXAMPLES |
| CHAPTER 11 | CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK |
4.0 ENHANCEMENT OF THE MODEL GENERATION ALGORITHM

4.1 INTRODUCTION

As mentioned in the previous chapter, three basic modes of data input are available to the user for the creation of the unit models. These are:

- propagation equations
- event statements
- decision tables

Each mode of input will be illustrated by reference to the following simple tank model:

4.2 DEFICIENCY IN MK1 MODEL GENERATION ALGORITHM

It has already been outlined in Chapter 3 how the propagation equations are analysed to generate the minitrees for use by the fault tree synthesis algorithm. Considering the development of the minitree for L3VES HI, the above propagation equation would be used to include the following two causes: G1IN HI and Q2OUT LO. It can be seen that only the direct or inverse cause-effect relationships are matched. Considering now the cause in the outlet stream, Q2OUT LO in greater detail, it is clear that other deviations of this variable can also give rise to L3VES HI. In fact two deviations, Q2OUT REV and Q2OUT NONE, could also give rise to this event.

The only way of incorporating this information into the MK1 models was to make use...
of event statements. Hence, event statements of the form given below would be required:

V Q2OUT NONE: L3VES HI  
V Q2OUT REV: L3VES HI

This is an undesirable feature since:

a) it is possible to derive this form of relationship between the variables by developing a more 'intelligent' algorithm for the analysis of the propagation equations.

b) resorting to the use of event statements complicates the modelling process. It leads to the creation of larger models which are more difficult to comprehend, especially for a novice user.

4.3 PROPOSED SOLUTION

Considering again the above example, it will be seen that NONE and REV are really more extreme deviations than LO. Hence, if the less extreme LO deviation can cause the minitop event L3VES HI, then the more extreme deviations should automatically be included as causes.

Since the FAULTFINDER methodology has been developed using only the minimum possible set of variables and deviations, it is possible to define a 'scale of severity' for each variable and its associated deviations. The valid deviations associated with each variable have already been given in Chapter 3. Table 4.1 below, gives the 'scale of severity' developed from this.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Scale of the Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+3</td>
</tr>
<tr>
<td>G</td>
<td>HI</td>
</tr>
<tr>
<td>Q</td>
<td>HI</td>
</tr>
<tr>
<td>T</td>
<td>HI</td>
</tr>
<tr>
<td>X</td>
<td>HI</td>
</tr>
<tr>
<td>Y</td>
<td>HI</td>
</tr>
<tr>
<td>S</td>
<td>HI</td>
</tr>
<tr>
<td>L</td>
<td>HI</td>
</tr>
<tr>
<td>W</td>
<td>HI</td>
</tr>
<tr>
<td>PF</td>
<td>NOR</td>
</tr>
<tr>
<td>R</td>
<td>NOP</td>
</tr>
<tr>
<td>U</td>
<td>HI</td>
</tr>
<tr>
<td>PV</td>
<td>REV</td>
</tr>
</tbody>
</table>

Table 4.1  
Scale of Severity for Deviations
A number of points should be noted in interpreting this table:

a) 0 represents some notional normal value for a variable. The +1 to +3 deviation levels represent increasingly higher than normal levels for this deviation; the converse being true for the -1 to -3 deviation levels.

b) the deviations for the different variables are ranked on this scale between -3 and +3. A blank slot indicates that the variable does not have a corresponding deviation.

c) the SOME deviation has been omitted. Propagation equations are only written for flow systems. SOME is generally used to represent a deviation where NONE is the norm, i.e. in no-flow situations.

d) the SHAC and NCHA deviations for the S variable have not been included in this classification, since these represent logical states in the plant and are not true deviations in the variable itself.

e) two pressure variables have been included in the above table. PV represents the pressure in a vessel, whereas PF represents the pressure in flowlines. This difference arises because:

- in flow situations the normal correspondence of deviations for PF is:

  \[ \text{LO} > \text{NONE} > \text{REV} \]

- in a no flow situation, like the gas space above a liquid level, the correspondence is illustrated by the following example propagation equation:

  \[ \text{QIN} = F(-\text{PVES}) \]

  This means that:

  \begin{align*}
  \text{QIN LO, if PVES HI} \\
  \text{QIN NONE, if PVES NOR \quad \text{i.e. no relief into the tank}} \\
  \text{QIN REV, if PVES REV \quad \text{i.e. REV is more extreme than NOR.}}
  \end{align*}

  The application of Table 4.1 will be illustrated by considering two models from the MK1 methodology which had a large number of 'superfluous' event statements.

4.4 APPLICATION TO TANK MODEL.

4.4.1 INTRODUCTION

The schematic representation of this model is given in Figure 4.2. This diagram represents a tank model with two inputs, two outputs, a kickback input and a level port. It is assumed to contain liquid below its boiling point.

The MK1 methodology contained the list of propagation equations and event statements, given in Tables 4.2 and 4.3, respectively. Some event statements used to model the propagation of the reverse temperature and composition variables have been omitted since they are not pertinent to the current discussion.
Figure 4.2
Schematic Representation of Contrived Tank Model from MK1 Model Library

<table>
<thead>
<tr>
<th>Number</th>
<th>Propagation Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$G1_{IN} = F(Q1_{IN}, -L6_{VES})$</td>
</tr>
<tr>
<td>2</td>
<td>$R1_{IN} = F(-L6_{VES})$</td>
</tr>
<tr>
<td>3</td>
<td>$G2_{IN} = F(Q2_{IN}, -L6_{VES})$</td>
</tr>
<tr>
<td>4</td>
<td>$R2_{IN} = F(-L6_{VES})$</td>
</tr>
<tr>
<td>5</td>
<td>$G3_{IN} = F(Q3_{IN})$</td>
</tr>
<tr>
<td>6</td>
<td>$Q4_{OUT} = F(L6_{VES}, G4_{OUT})$</td>
</tr>
<tr>
<td>7</td>
<td>$T4_{OUT} = F(T6_{VES})$</td>
</tr>
<tr>
<td>8</td>
<td>$X4_{OUT} = F(X6_{VES})$</td>
</tr>
<tr>
<td>9</td>
<td>$P4_{OUT} = F(L6_{VES})$</td>
</tr>
<tr>
<td>10</td>
<td>$Q5_{OUT} = F(L6_{VES}, G5_{OUT})$</td>
</tr>
<tr>
<td>11</td>
<td>$T5_{OUT} = F(T6_{VES})$</td>
</tr>
<tr>
<td>12</td>
<td>$X5_{OUT} = F(X6_{VES})$</td>
</tr>
<tr>
<td>13</td>
<td>$P5_{OUT} = F(L6_{VES})$</td>
</tr>
<tr>
<td>14</td>
<td>$L6_{VES} = F(G1_{IN}, G2_{IN}, G3_{IN}, -Q4_{OUT}, -Q5_{OUT})$</td>
</tr>
<tr>
<td>15</td>
<td>$T6_{VES} = F(T1_{IN}, T2_{IN}, T3_{IN})$</td>
</tr>
<tr>
<td>16</td>
<td>$X6_{VES} = F(X1_{IN}, X2_{IN}, X3_{IN})$</td>
</tr>
</tbody>
</table>

Table 4.2
List of Propagation Equations for Tank Model of Figure 4.2

---

Schematic Representation of Contrived Tank Model from MK1 Model Library

---

$G1_{IN} = F(Q1_{IN}, -L6_{VES})$

$R1_{IN} = F(-L6_{VES})$

$G2_{IN} = F(Q2_{IN}, -L6_{VES})$

$R2_{IN} = F(-L6_{VES})$

$G3_{IN} = F(Q3_{IN})$

$Q4_{OUT} = F(L6_{VES}, G4_{OUT})$

$T4_{OUT} = F(T6_{VES})$

$X4_{OUT} = F(X6_{VES})$

$P4_{OUT} = F(L6_{VES})$

$Q5_{OUT} = F(L6_{VES}, G5_{OUT})$

$T5_{OUT} = F(T6_{VES})$

$X5_{OUT} = F(X6_{VES})$

$P5_{OUT} = F(L6_{VES})$

$L6_{VES} = F(G1_{IN}, G2_{IN}, G3_{IN}, -Q4_{OUT}, -Q5_{OUT})$

$T6_{VES} = F(T1_{IN}, T2_{IN}, T3_{IN})$

$X6_{VES} = F(X1_{IN}, X2_{IN}, X3_{IN})$
<table>
<thead>
<tr>
<th>Number</th>
<th>Event Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F LK-LP-EN: L6VES LO, L6VES NONE</td>
</tr>
<tr>
<td>2</td>
<td>F EXT-COLD: T6VES LO</td>
</tr>
<tr>
<td>3</td>
<td>F EXT-HEAT: T6VES HI</td>
</tr>
<tr>
<td>4</td>
<td>V G1IN NONE: A(DUMMY)</td>
</tr>
<tr>
<td>5</td>
<td>VG2IN NONE: A(DUMMY)</td>
</tr>
<tr>
<td>6</td>
<td>V G1IN REV: A(DUMMY), L6VES NONE</td>
</tr>
<tr>
<td>7</td>
<td>V G2IN REV: A(DUMMY), L6VES NONE</td>
</tr>
<tr>
<td>8</td>
<td>V Q4OUT NONE: B(DUMMY)</td>
</tr>
<tr>
<td>9</td>
<td>V Q5OUT NONE: B(DUMMY)</td>
</tr>
<tr>
<td>10</td>
<td>V Q4OUT REV: B(DUMMY)</td>
</tr>
<tr>
<td>11</td>
<td>V Q5OUT REV: B(DUMMY)</td>
</tr>
<tr>
<td>12</td>
<td>V G3IN LO: C(DUMMY)</td>
</tr>
<tr>
<td>13</td>
<td>V G3IN NONE: C(DUMMY)</td>
</tr>
<tr>
<td>14</td>
<td>V G3IN REV: C(DUMMY)</td>
</tr>
<tr>
<td>15</td>
<td>I C(DUMMY): A(DUMMY)</td>
</tr>
<tr>
<td>16</td>
<td>V G3IN HI: B(DUMMY)</td>
</tr>
<tr>
<td>17</td>
<td>I A(DUMMY): L6VES LO</td>
</tr>
<tr>
<td>18</td>
<td>I B(DUMMY): L6VES HI</td>
</tr>
<tr>
<td>19</td>
<td>V L6VES HI: R1IN NONE, R2IN NONE, G1IN NONE, G2IN NONE, P4OUT NOR, P5OUT NOR</td>
</tr>
<tr>
<td>20</td>
<td>V L6VES LO: P4OUT REV, P5OUT REV</td>
</tr>
<tr>
<td>21</td>
<td>V L6VES NONE: P4OUT REV, P5OUT REV</td>
</tr>
</tbody>
</table>

Table 4.3
List of Event Statements for Tank Model of Figure 4.2

This is rather a long list of event statements. It would be advantageous if the user could be shielded from having to provide all this information.

4.4.2 REMOVAL OF EVENT STATEMENTS

The aim of this section is to show how event statements numbered (4-21) in Table 4.3 above, can either be generated entirely automatically by the model generation program, or how it can prompt the user for additional information to incorporate the relevant failure modes into the model.

The FAULTFINDER convention to model the continuity of flow in a model is to express Q in terms of G, and vice versa. This information is encapsulated in the form of propagation equations with one cause of the output event being defined at an inlet port and the other at an outlet port. Rule 1 below highlights this concept.

The analysis of event statements 4-21 for this model indicates that these occur between two different variables where there is non-continuous flow. All the aforementioned event statements involve an inlet/outlet variable and a vessel variable, e.g. G1IN and L6VES, P4OUT and L6VES. It is clear that the event statements in this model are needed to model the relationship between the flow streams at the inlet and the outlet,
and the accumulation within the vessel.

The automatic derivation of these event statements is based on an analysis of the propagation equations in the model. In order to achieve this a number of rules have been formulated. These rules are outlined below.

**Rule 1 - Determine the ports having a continuity of flow.**

This would be achieved by examining all the propagation equations defining the $G$ variable. A continuity of flow would be detected if $G$ was defined in terms of $Q$ at two different ports. In this case there is no continuity of flow through the tank model of Figure 4.2.

This point is made clearer by comparing the tank with a pipe-type model depicted below:

![Figure 4.3](image)

**Figure 4.3**

Schematic Representation of a Pipe-Type Unit

The pipe-type model would have a propagation equation of the form:

$$G_{1IN} = F(Q_{1IN}, Q_{2OUT})$$

signifying a continuity of flow between ports 1 and 2.

The tank model on the other hand has a propagation equation of the form:

$$G_{1IN} = F(Q_{1IN}, -L6VES)$$

indicating a non-continuous flow between this inlet port and any other outlet ports on the model.

The following block diagram shows the relationships between the different ports in the tank model:

![Figure 4.4](image)

**Figure 4.4**

Block Diagram Highlighting the Information Flow Between the Ports of the Tank Model
Rule 2 - Identify the equations which express a relationship between two different variables, defined at ports where there is no continuity of flow.

For this purpose the G and Q variables are considered to be synonymous.

This rule would thus select the following propagation equations for further analysis:

<table>
<thead>
<tr>
<th>Number</th>
<th>Propagation Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$G_{1IN} = F(Q_{1IN}, -L6VES)$</td>
</tr>
<tr>
<td>2</td>
<td>$R_{1IN} = F(-L6VES)$</td>
</tr>
<tr>
<td>3</td>
<td>$G_{2IN} = F(Q_{2IN}, -L6VES)$</td>
</tr>
<tr>
<td>4</td>
<td>$R_{2IN} = F(-L6VES)$</td>
</tr>
<tr>
<td>5</td>
<td>$Q_{4OUT} = F(L6VES, G_{4OUT})$</td>
</tr>
<tr>
<td>6</td>
<td>$P_{4OUT} = F(L6VES)$</td>
</tr>
<tr>
<td>7</td>
<td>$Q_{5OUT} = F(L6VES, G_{5OUT})$</td>
</tr>
<tr>
<td>8</td>
<td>$P_{5OUT} = F(L6VES)$</td>
</tr>
<tr>
<td>9</td>
<td>$L6VES = F(G_{1IN}, G_{2IN}, G_{3IN}, -Q_{4OUT}, -Q_{5OUT})$</td>
</tr>
</tbody>
</table>

Table 4.4
Propagation Equations for Further Analysis

As an illustrative example, consider equation (1) from Table 4.4. This has been selected since:

a) there is non-continuous flow between ports 1 and 6
b) only the G and Q variables are synonymous, G and L are therefore two different variables.

Similar reasoning can be applied to the other propagation equations. On the other hand, for example, propagation equation (15) from the initial list is not considered further, since it contains only the temperature variable, T.

Rule 3 - Using the 'scale of severity' match the full range of cause-effect relationships.

The normal analysis procedure would match the direct/inverse cause-effect relationships. By using the 'scale of severity' the program can now also include the more extreme deviations if they exist. Referring to the first propagation equation above, L6VES NONE as well as L6VES LO would be included as a cause of G1IN HI. The inclusion of the more extreme deviations is, however, qualified under some situations; these qualifications are given below.

Qualification 1 of Rule 3

If the right-hand-side (RHS) variable is defined at a vessel port and the left-hand-side (LHS) variable is directly proportional to this, then only the direct cause-effect relationships should be matched. The importance of this is illustrated by considering propagation equation (6) from Table 4.4 above. The deviations in the pressure and level variables are closely coupled, since the manifestation of pressure at the outlet is dependent on a level of fluid being present in the tank.

Note: the number given in () in Table 4.4 relates to the corresponding entry in Table 4.2.
Nevertheless, there is one additional proviso to this statement that only the direct cause-effect relationships should be matched. This is that if the LHS variable has a deviation which is more extreme than the most extreme deviation in the RHS variable, then the user is prompted for more information.

It can be seen from Table 4.1, that there is no deviation in the level variable to correspond to the REV deviation in the pressure variable. Under such circumstances the user is asked if this more extreme deviation in the pressure variable can be caused by the less extreme deviations in the level variable. This is illustrated by considering the event P4OUT REV. This represents the existence of reverse pressure, i.e. relief into the tank. The user is first asked if L6VES NONE can give rise to this event. If the answer to this is no, then the process is aborted; otherwise the user is next asked if L6VES LO can also give rise to this event.

Using this technique both L6VES LO and L6VES NONE would be included as causes of P4OUT REV.

Qualification 2 of Rule 3

This situation is analogous to the previous case and covers the situation where there is an inverse relationship between the LHS and the RHS variables, and the corresponding deviation in the RHS variable does not exist. This situation is best illustrated by considering propagation equation (9), from Table 4.4. Since there is an inverse relationship between L6VES and Q4OUT, when the minitree for L6VES LO is developed, the cause Q4OUT HI is included. This illustrates that a -1 deviation LO, is matched with the +1 deviation HI. When the minitree for L6VES NONE is developed, there is no corresponding deviation for the Q variable. There is no deviation for the flow variable more extreme than HI. In this case the user is asked if the less extreme deviation HI can give rise to L6VES NONE.

The application of the above rules either automatically generates the information encapsulated in the aforementioned event statements or issues suitable prompts to assist the user in configuring the event statements interactively.

4.5 APPLICATION TO HEAT EXCHANGER MODEL

The schematic representation of a heat exchanger is given in Figure 4.5.

This represents a shell and tube heat exchanger. Ports 1 and 2 represent the tubes and carry the fluid to be cooled; ports 3 and 4 represent the shell-side and carry the fluid to be heated. The creation of the model assumes that the tubes are at a higher pressure than the shell and that there are no phase changes in the heat exchanger.

The propagation equations and event statements for the MK1 model for this unit are given in Tables 4.5 and 4.6, respectively.
**Figure 4.5**
Schematic Representation of a Heat Exchanger Model

<table>
<thead>
<tr>
<th>Number</th>
<th>Propagation Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$G_{IN}=F(Q_{IN},Q_{OUT})$</td>
</tr>
<tr>
<td>2</td>
<td>$R_{IN}=F(R_{OUT})$</td>
</tr>
<tr>
<td>3</td>
<td>$U_{IN}=F(U_{OUT},G_{3IN})$</td>
</tr>
<tr>
<td>4</td>
<td>$Y_{IN}=F(Y_{OUT})$</td>
</tr>
<tr>
<td>5</td>
<td>$Q_{OUT}=F(G_{IN},G_{2OUT})$</td>
</tr>
<tr>
<td>6</td>
<td>$T_{2OUT}=F(G_{IN},G_{3IN},T_{IN},T_{3IN})$</td>
</tr>
<tr>
<td>7</td>
<td>$X_{2OUT}=F(X_{IN})$</td>
</tr>
<tr>
<td>8</td>
<td>$P_{2OUT}=F(P_{IN})$</td>
</tr>
<tr>
<td>9</td>
<td>$G_{3IN}=F(Q_{3IN},Q_{OUT})$</td>
</tr>
<tr>
<td>10</td>
<td>$R_{3IN}=F(R_{OUT})$</td>
</tr>
<tr>
<td>11</td>
<td>$U_{3IN}=F(U_{4OUT},G_{1IN})$</td>
</tr>
<tr>
<td>12</td>
<td>$Y_{3IN}=F(Y_{OUT})$</td>
</tr>
<tr>
<td>13</td>
<td>$Q_{4OUT}=F(G_{3IN},G_{4OUT})$</td>
</tr>
<tr>
<td>14</td>
<td>$T_{4OUT}=F(G_{IN},G_{3IN},T_{IN},T_{3IN})$</td>
</tr>
<tr>
<td>15</td>
<td>$X_{4OUT}=F(X_{3IN})$</td>
</tr>
<tr>
<td>16</td>
<td>$P_{4OUT}=F(P_{3IN})$</td>
</tr>
</tbody>
</table>

**Table 4.5**
List of Propagation Equations for Heat Exchanger Model of Figure 4.5
<table>
<thead>
<tr>
<th>Number</th>
<th>Event Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F EXT-HEAT: T2OUT HI, T4OUT HI, U1IN HI</td>
</tr>
<tr>
<td>2</td>
<td>F EXT-COLD: T2OUT LO, T4OUT LO, U3IN LO</td>
</tr>
<tr>
<td>3</td>
<td>F FOULING: T2OUT HI, T4OUT LO, U1IN HI, U3IN LO</td>
</tr>
<tr>
<td>4</td>
<td>V G1IN LO: D(DUMMY)</td>
</tr>
<tr>
<td>5</td>
<td>V G3IN LO: E(DUMMY)</td>
</tr>
<tr>
<td>6</td>
<td>V G1IN NONE: D(DUMMY)</td>
</tr>
<tr>
<td>7</td>
<td>V G3IN NONE: E(DUMMY)</td>
</tr>
<tr>
<td>8</td>
<td>V G1IN REV: D(DUMMY)</td>
</tr>
<tr>
<td>9</td>
<td>V G3IN REV: E(DUMMY)</td>
</tr>
<tr>
<td>10</td>
<td>ID(DUMMY): T4OUT LO, U3IN LO</td>
</tr>
<tr>
<td>11</td>
<td>IE(DUMMY): T2OUT HI, U1IN HI</td>
</tr>
<tr>
<td>12</td>
<td>F LK-LP-EN: G3IN HI, G3IN SOME, Q4OUT LO, Q4OUT NONE, Q4OUT REV, R3IN HI, R3IN SOME, R3IN NOP, P4OUT NONE, P4OUT REV, T2OUT HI, T4OUT HI, U1IN HI</td>
</tr>
<tr>
<td>13</td>
<td>F INT-LK: G1IN HI, G1IN SOME, Q2OUT LO, Q2OUT NONE, Q2OUT REV, R1IN HI, R1IN SOME, R1IN NOP, P2OUT LO, P2OUT NONE, P2OUT REV, G3IN LO, G3IN NONE, G3IN REV, Q4OUT HI, Q4OUT SOME, R3IN LO, R3IN NONE, R3IN REV, P4OUT HI, P4OUT SOME, P4OUT NOR, X4OUT HI, Y3IN HI, T2OUT LO, T4OUT HI</td>
</tr>
<tr>
<td>14</td>
<td>F PART-BLK: G1IN LO, Q2OUT LO, R1IN LO, P2OUT LO, T2OUT LO, T4OUT LO, U3IN LO</td>
</tr>
<tr>
<td>15</td>
<td>F COMP-BLK: G1IN NONE, Q2OUT NONE, R1IN NONE, R1IN NOP, P2OUT NONE, P2OUT NOR, T4OUT LO, U3IN LO</td>
</tr>
<tr>
<td>16</td>
<td>S NORMAL: U1IN LO, U3IN HI</td>
</tr>
</tbody>
</table>

Table 4.6
List of Event Statements for Heat Exchanger Model of Figure 4.5

The application of Rule 1, yields that there is a continuity of flow between ports 1 and 2, and ports 3 and 4.

Rule 2 thus selects the following equations for further analysis:

\[
\begin{align*}
U1IN &= F(U2OUT, -G3IN) \\
T2OUT &= F(G1IN, -G3IN, T1IN, T3IN) \\
U3IN &= F(U4OUT, G1IN) \\
T4OUT &= F(G1IN, -G3IN, T1IN, T3IN)
\end{align*}
\]

Note the analogy between these equations and those selected for further analysis in the tank model. In the tank model the variables of interest were the internal vessel variables arising due to the relative flows into and out of the tank. In the current example the variable of interest is the temperature, which again is the result of non-continuous flow between the hot and cold streams. The two streams interact by exchanging heat; this manifests itself by affecting the relative temperatures of the two streams.

Using Rule 3 the information encapsulated in event statements 4-11 will be automatically inferred by the model generation program.
4.6 INCLUSION OF BASIC FAULTS IN THE MODEL

4.6.1 INTRODUCTION

Whilst the list of possible faults is almost infinite, the purpose of this section is to take some common faults and to write some general purpose event statements for them. This information can then be extracted and used in a number of applications when the event statements for any particular model can be developed.

The six pipe-type faults: LK-LP-EN, LK-HP-EN, PART-BLK, COMP-BLK, EXT-HEAT and EXT-COLD will be considered in relation to the two models used above. The approach adopted to handle these will now be outlined.

4.6.2 FAULT LIBRARY

The purpose of the event statements is to express the relationship between the basic faults and variable deviations, for example:

\[ \text{F EXT-HEAT: T2OUT HI} \]

It should be noted that the events appearing on the RHS of the event statement are the output events of the propagation equations, i.e. the variable deviation is written in terms of T2OUT and not T1IN.

Since the FAULTFINDER methodology comprises a fixed number of variables and corresponding deviations, it is possible to define the full set of deviations that can be caused by a given fault. For the six faults mentioned above, the fault library given in Tables 4.7 - 4.12 was developed.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Port</th>
<th>Deviation List</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>IN</td>
<td>NONE</td>
</tr>
<tr>
<td>Q</td>
<td>OUT</td>
<td>NONE</td>
</tr>
<tr>
<td>T</td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>SIG</td>
<td>NONE</td>
</tr>
<tr>
<td>L</td>
<td>VES</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>OUT</td>
<td>NONE, NOR</td>
</tr>
<tr>
<td>R</td>
<td>IN</td>
<td>NONE, NOP</td>
</tr>
<tr>
<td>U</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>VES</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.7
Effects of Fault COMP-BLK
### Table 4.8
Effects of Fault PART-BLK

<table>
<thead>
<tr>
<th>Variable</th>
<th>Port</th>
<th>Deviation</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>IN</td>
<td>LO</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>OUT</td>
<td>LO</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>SIG</td>
<td>LO</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>VES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>SIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>OUT</td>
<td>LO</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>IN</td>
<td>LO</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>VES</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 4.9
Effects of Fault LK-HP-EN

<table>
<thead>
<tr>
<th>Variable</th>
<th>Port</th>
<th>Deviation</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>IN</td>
<td>LO</td>
<td>NONE</td>
</tr>
<tr>
<td>Q</td>
<td>OUT</td>
<td>LO</td>
<td>NONE</td>
</tr>
<tr>
<td>T</td>
<td>OUT</td>
<td>HI</td>
<td>SOME</td>
</tr>
<tr>
<td>X</td>
<td>OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>SIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>VES</td>
<td>HI</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>SIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>OUT</td>
<td>NOR</td>
<td>HI</td>
</tr>
<tr>
<td>R</td>
<td>IN</td>
<td>NONE</td>
<td>LO</td>
</tr>
<tr>
<td>U</td>
<td>IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>VES</td>
<td>NOR</td>
<td>REV</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Port</th>
<th>Deviation List</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>IN</td>
<td>HI SOME</td>
</tr>
<tr>
<td>Q</td>
<td>OUT</td>
<td>LO NONE</td>
</tr>
<tr>
<td>T</td>
<td>OUT</td>
<td>Rev</td>
</tr>
<tr>
<td>X</td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>VES</td>
<td>LO NONE</td>
</tr>
<tr>
<td>W</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>OUT</td>
<td>LO NONE Rev</td>
</tr>
<tr>
<td>R</td>
<td>IN</td>
<td>HI SOME NOP</td>
</tr>
<tr>
<td>U</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>VES</td>
<td>LO NONE</td>
</tr>
</tbody>
</table>

Table 4.10
Effects of Fault LK-LP-EN

<table>
<thead>
<tr>
<th>Variable</th>
<th>Port</th>
<th>Deviation List</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>OUT</td>
<td>LO</td>
</tr>
<tr>
<td>X</td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>VES</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>SIG</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>OUT</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>IN</td>
<td>LO</td>
</tr>
<tr>
<td>P</td>
<td>VES</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.11
Effects of Fault EXT-COLD
The information in the fault library will be illustrated by reference to the fault COMP-BLK contained in Table 4.7. The first point of note is that a fault of this nature can only affect the ports where there is flow. Hence, in general only the variables defined at port types IN and OUT can be affected. Additionally, this fault can only affect the flow and pressure variables at these ports. The resultant deviations for these variables are thus included in the library. [The effect on the signal variable has also been included as it is just a special instance of a flow variable and a blockage can affect the flow of both electrical and pneumatic signals.]

Entries for the other faults are developed in a similar manner by analysing the likely effect on each of the variables. It should be noted that only the direct effects of a fault have been considered. More exotic scenarios such as:

EXT-HEAT ==> Phase Change of Fluid ==> Composition Deviations,

have been ignored. Only the direct effects such as:

EXT-HEAT ==> Temperature Deviations,

have been included.

**4.6.3 DATA INPUT BY THE USER**

When running the model generation program the user is required to enter two items of information:

- the name of the fault to be considered, and
- the port numbers affected by this fault
4.6.4 THE ANALYSIS ALGORITHM

For each fault type to be considered, the LHS of each propagation equation is examined. If the variable and the port type of the LHS variable is also included under the specified fault in the library, then information on the valid deviations of this variable are extracted.

For example, considering the tank model, the user could specify that the fault LK-LP-EN is to be considered and that this fault affects port number 6. The program will consider all the equations defined in terms of output variables at port number 6. The equations for the following three variables will be earmarked for further consideration: L6VES, T6VES and X6VES.

Further, the program will identify port 6 as a VES port. When the information under the fault LK-LP-EN is examined, the only event which matches the LHS of the above three equations is LVES. So for the current fault the two events L6VES LO and L6VES NONE are included as the effects of LK-LP-EN. This corresponds to the first event statement in the model.

The event statements generated using this procedure for the two models considered earlier are given in Tables 4.13 and 4.14.

<table>
<thead>
<tr>
<th>Number</th>
<th>Event Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F LK-LP-EN: L6VES LO, L6VES NONE</td>
</tr>
<tr>
<td>2</td>
<td>F EXT-HEAT: T6VES HI</td>
</tr>
<tr>
<td>3</td>
<td>F EXT-COLD: T6VES LO</td>
</tr>
</tbody>
</table>

Table 4.13
Event Statements for Tank Model

<table>
<thead>
<tr>
<th>Number</th>
<th>Event Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F LK-LP-EN: G1IN HI, G1IN SOME, R1IN HI, R1IN SOME, R1IN NOP, Q2OUT LO, Q2OUT NONE, Q2OUT REV, P2OUT LO, P2OUT NONE, P2OUT REV, G3IN HL, G3IN SOME, R3IN HI, R3IN SOME, R3IN NOP, Q4OUT LO, Q4OUT NONE, Q4OUT REV, P4OUT LO, P4OUT NONE, P4OUT REV</td>
</tr>
<tr>
<td>2</td>
<td>F LK-HP-EN: G3IN LO, G3IN NONE, G3IN REV, R3IN LO, R3IN NONE, R3IN REV, Q4OUT HI, Q4OUT SOME, P4OUT NOR, R3IN REV, Q4OUT HI, Q4OUT SOME, P4OUT NOR, P4OUT HI, P4OUT SOME</td>
</tr>
<tr>
<td>3</td>
<td>F PART-BLK: G1IN LO, R1IN LO, Q2OUT LO, P2OUT LO</td>
</tr>
<tr>
<td>4</td>
<td>F COMP-BLK: G1IN NONE, R1IN NONE, R1IN NOP, Q2OUT NONE, P2OUT NONE, P2OUT NOR</td>
</tr>
<tr>
<td>6</td>
<td>F EXT-HEAT: T2OUT HI, T4OUT HI</td>
</tr>
<tr>
<td>7</td>
<td>F EXT-COLD: T2OUT LO, T4OUT LO</td>
</tr>
</tbody>
</table>

Table 4.14
Event Statements for Heat Exchanger
It will be noted that the event statements for the tank model are the same. However, those for the heat exchanger are slightly different, in that they do not contain all the output effects. It should also be noted that the fault INT-LK was actually modelled as two faults: LK-LP-EN affecting ports 1 and 2, and LK-HP-EN affecting ports 3 and 4.

The differences between the two sets of event statements for the heat exchanger model will now be highlighted:

a) The event statements for PART-BLK and COMP-BLK do not contain the effect on T4OUT and U3IN. In actual fact including this information in the event statements is unnecessary. The normal analysis procedure for the propagation equations will include G1IN LO as a cause of T4OUT LO. Since PART-BLK is included as a cause of G1IN LO, including it as a cause of T4OUT LO will lead to a duplication of information. The FAULTFINDER methodology will actually delete the second occurrence of the duplicated event. These two scenarios are clarified by the minitrees given in Figure 4.6.

![Minitrees Highlighting Duplicated Event](image)

b) In the original model LK-LP-EN, representing a leak of cooling medium to the environment, is included as a cause of T2OUT HI, T4OUT HI and U1IN HI. Including these relationships requires a more intelligent algorithm. The current technique outlined only considers the direct effects on the stream upon which the fault manifests itself, e.g. a leakage affects only the flow variables.

The three variable deviations given above arise due to the influence of heat transfer between the two streams. Using T2OUT HI as an example, the basis of a more thorough analysis algorithm could be as follows:

i) look at the propagation equation for T2OUT. This is

\[ T2OUT = F(G1IN, -G3IN, T1IN, T3IN) \]
ii) look at all the output events caused by LK-LP-EN from the fault library. Two effects of this fault are to cause G3IN HI and Q4OUT LO.

iii) looking again at the propagation equation it will be noted that T2OUT is inversely proportional to G3IN. Hence if LK-LP-EN gives rise to G3IN HI then this will in turn give rise to T2OUT LO. This is in fact contrary to the information included in the manually created event statement, which states that LK-LP-EN should cause T2OUT HI.

iv) there is a need to carry out conflict resolution. It has already been noted that there is a continuity of flow between ports 3 and 4. A leakage in this stream will indeed cause a high flow at the inlet (G3IN HI); but it will also cause a low flow at the outlet (Q4OUT LO). Some sort of rule is needed to ascertain which effect is the stronger. In the current context it has been assumed that the net effect of this is to give rise to a higher temperature for the process stream (T2OUT HI).

c) The other anomaly concerns the event statements used to include the effects of INT-LK, as a result of the hot fluid in the tubes leaking into the cold fluid in the shell. The automatically generated event statements obtained by modelling INT-LK as a combination of LK-LP-EN and LK-HP-EN do not include the events X4OUT HI, Y3IN HI, T2OUT LO and T4OUT HI. These events arise due to the mixing of the process and coolant streams and there is no means available to deduce this information automatically. It has not proved possible to generate generic rules to cater for this situation and so it would be up to the user to add this information manually.

d) The basic fault FOULING has not been considered; this is too specific to the current application.

4.6.5 USER MODIFICATION OF THE AUTOMATICALLY GENERATED EVENT STATEMENTS

As indicated above for the heat exchanger model, the automatically generated event statements may not contain all the required information. Additionally as indicated for the fault INT-LK, this has been simulated by considering the combined effects of LK-LP-EN and LK-HP-EN. A facility has therefore been included for the user to be able to modify these event statements, once they have been derived by the model generation program.

4.7 CONCLUSION

This chapter has indicated how the model generation program has been enhanced to limit the amount of information required from the user. The propagation equation analysis algorithm has been enhanced to be able to include the complete set of cause-effect relationships.

Additionally the foundations have been laid for generating the event statements containing the information relating to basic failures. A fault library has initially been developed for a set of six faults. This has been demonstrated to significantly reduce the information required from the user. It has also been shown how some more complex faults can be broken down and simulated using these basic faults.
5.0 ELIMINATION OF DIVIDER HEADER COMBINATIONS

5.1 INTRODUCTION

Consider the section of plant below:

Figure 5.1
Simple Section of Plant

The start of the fault tree for HIGH FLOW in stream 3 would be of the form:

Figure 5.2
Start of Fault Tree for High Flow in Plant Section of Figure 5.1.

The G2 HI branch would thus find the upstream causes and the G3 HI branch the downstream causes. Although each model contains information for two-way fault propagation, the normal boundary conditions ensure that the path proceeds in the required direction.

Now consider the following section of plant:
The start of the fault tree for HIGH FLOW in stream 3 would be of the form:

Note: the dotted line in the fault tree indicates missing intermediate steps in the propagation chain.

When the propagation path reaches the divider, the causes of Q2 HI can occur in either the inlet leg or the other outlet leg(s). The causes in the inlet leg are found by the G1 HI branch, whereas the causes in the other outlet leg are found by the G4 LO/NONE/REV branches.
But now consider the following section of plant:

Figure 5.5
Plant Section Incorporating a Divider and Header in Combination.

The fault tree for HIGH FLOW in stream 4 would be of the form:

Figure 5.6
Fault Tree for High Flow in Plant Section of Figure 5.5.
This tree is incorrect in that G5 LO or G2 LO can never give rise to Q4 HI. The problem arises because fault tree synthesis is done vertically, so consistency checks can only be carried out with respect to the path already traced. It is clear from the above figure that owing to the two-way propagation facility, a loop in the information flow has been created, which can only be handled by additional consistency checks after the initial tree has been synthesised or by introducing additional boundary conditions during the actual synthesis stage.

5.2 FAULTFINDER MK1 SOLUTION

Kelly [Ref 5-1] partially solved the problem by treating this situation as a special sub-system in a manner comparable to that for control loops and trip loops. Such sub-systems were classified as 'Divider-Header Combinations' (DHCs). It is thus necessary for the user to identify these sub-systems during the decomposition stage and to supply information to the MASTER program to convey their structure and functionality. This situation is further complicated by the methodology differentiating between three types of DHCs:

a) a bypass system where both legs have flow, e.g. to regulate the temperature of a stream by adjusting the flow of the process stream through the heat exchanger. For simplicity the control valve set-up to regulate the flow between the two legs has been omitted from Figure 5.7.

![Figure 5.7](image)

**Figure 5.7**  
Bypass With Flow Divider-Header Combination.

b) a bypass system where one leg normally has flow but the other one does not, e.g. to enable on-line maintenance of a control valve. For simplicity the isolation valves around the control valve have been omitted from Figure 5.8:
c) a parallel system to handle redundancy, e.g. a pump bank with spare capacity (a maximum of five legs being allowed by this arrangement):

Furthermore, the divider and header models used for each of these combinations were different and so it was also necessary for the user to select the correct models to suit the particular application.

5.3 DEFICIENCY IN THE MK1 SOLUTION.

Whilst the approach outlined by Kelly produced correct fault trees for the three types of DHC outlined above, the solution was restricted to these cases and could not for example cater for overlapping DHCs or where each leg on a divider did not match up to
a corresponding leg on a header. (An example of an overlapping DHC is given later in this chapter - see Figure 5.11).

Apart from these problems a more fundamental criticism can be levelled at classifying DHCs as a special sub-system on the same lines as control loops and trip loops.

Control loops and trip loops are a significant feature in that the failure of these components introduces certain generic failure modes into the tree. Since they are an obvious and integral part of the plant, the user can reasonably be expected to identify, and then describe their structure and functionality in terms of the input required by the MASTER program.

On the other hand, DHCs are a somewhat abstract concept, which has been introduced to solve various problems encountered by the synthesis algorithm. The user is being asked to determine the functionality of structures whose presence may not be all that obvious in the first place, particularly if the DHCs overlap or are separated by a large number of intervening units. In order to highlight this point consider the Lawley Propane Pipeline System [Ref 5-2].

5.3.1 LAWLEY PROPANE PIPELINE SYSTEM

Figure 5.10 gives the flow diagram for this system. DHCs have been uniquely identified using the following convention:

Each DHC has been assigned a number 1,2 ..N. For DHC_1, the divider has been marked 1 and the corresponding header 1'. This is repeated for the other DHCs in the system.

The following problems were encountered in identifying the various DHCs for the above system:

- all the dividers have to be identified and then the legs on the divider have to be traced to find any corresponding headers. For a large system some DHCs could easily be over-looked.

- two of the DHCs overlap. It is not clear how these should be represented.

- in the three-way valve DHC, it is necessary to treat the three-way valve as a divider in its own right.

- there are a number of dividers and headers present which do not match up to form DHCs. These tend to confuse the issue.

Despite recognising all the DHCs in the above system, the fault tree would still be incorrect owing to the overlapping DHCs. However, the work by Kelly was important in identifying the problem and presenting a strategy which went some way towards providing a solution.
5.4 PROPOSED SOLUTION

Looking again at the fault tree in Figure 5.6, it is necessary to delete the two branches (G5 LO under Q2 HI and G2 LO under Q5 HI). This is necessary since a loop exists in the information flow structure and these branches are tracing faults in the direction opposite to that required for the event Q5 HI. The solution outlined in this thesis relies to a large extent on detecting these loops in the information flow structure and then setting up appropriate boundary conditions at the synthesis stage.

5.5 APPLICATION TO AN EXAMPLE SYSTEM

The principles behind the proposed solution will be described with reference to Figure 5.11. Although this is a rather contrived example, it nevertheless addresses many of the issues. Using the MK1 methodology it would not be clear how this configuration should be decomposed to specify the different DHCs. The three possible DHCs are given in Table 5.1 below:

<table>
<thead>
<tr>
<th>DHC</th>
<th>DIVIDER</th>
<th>UNIT</th>
<th>HEADER</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1
Possible DHCs for Plant Section of Figure 5.11

Even if the rules on decomposition could be clarified, the methodology is inadequate for this application since the third combination is produced as a result of the first two overlapping.

5.6 TERMINOLOGY

Before proceeding any further, the terminology used will be described. Essentially there are three main terms of interest:

- loop
- balanced loop
- overall loop

Each of these is described below.

5.6.1 LOOP

A loop defines a path in the information flow structure, which enables the propagation chain to double back on to itself.

Hence, an important aspect of the MK2 methodology is to detect loops in the information flow structure and to take appropriate action at the synthesis stage. Thus for the configuration in Figure 5.11 the following loops would be detected:

a) 3.3.4 - 5.5.6 - 7.12.11
b) 5.5.6 - 7.7.8 - 9.14.13
c) 3.3.4 - 5.13.14 - 9.8.7 - 7.12.11
Plant Section Comprising Multiple Overlapping Divider-Header Combinations

Figure 5.11
For each group of three numbers:
- the first number is the node number of the given divider or header.
- the second number is the edge number linking to the current node.
- the third number is the edge number linking to the next node.

The synthesis package could then utilise this information to introduce the relevant boundary conditions. However, in addition to identifying loops, it is also necessary to ascertain whether each loop is 'balanced' or not. The concept of a balanced loop is discussed below.

5.6.2 BALANCED LOOP

A loop is balanced if in the direction of fault propagation, the loop has a single exit point.

The significance of this point is illustrated with reference to the plant section of Figure 5.11. Supposing the fault tree for LOW FLOW in stream 9 is being developed. Then neglecting reverse flow and assuming that the tree could be prevented from looping back onto itself, the resultant tree would be of the following format:

![Fault Tree for Low Flow in Plant Section of Figure 5.11.](image-url)

Figure 5.12
Fault Tree for Low Flow in Plant Section of Figure 5.11.
Loop 5.5.6 - 7.7.8 - 9.14.13 is an unbalanced loop since it has an unbalanced leg via header unit 7. Hence, Q4 NONE at the 'inlet' of the loop can be a valid cause of Q9 LO, since the LO at the 'outlet' can be met by the unbalanced leg.

However, loop 3.3.4 - 5.13.14 - 9.8.7 - 7.12.11 is a balanced loop since it contains no unbalanced legs. Consequently Q2 NONE is not a valid cause since for Q9 LO to occur, there must be some flow at this point.

In actual fact the balance of a loop is also dictated by the direction of fault propagation. If the event of interest were G4 LO then loop 5.5.6 - 7.7.8 - 9.14.13 would be considered balanced since there is only one outlet from the loop.

This data on balanced loops is represented in terms of streams which represent complete continuity of flow between the inlet and outlet of any loop. There would thus be four entries for the above system:

a) 9 - 2 indicating that the loop is balanced in both directions.
b) 2 - 9
c) 4 - 9 indicating that the loop is balanced in one direction only.
d) 7 - 2

Where continuity of flow exists the synthesis package would introduce boundary conditions to ensure that only the loop start deviation is propagated to the loop outlet.

5.6.3 OVERALL LOOP

It is important to note the difference between loops and overall loop. Consider the plant section shown below:

Figure 5.13
Parallel System Illustrating Difference Between Loops and Overall Loop.
The program will note three unique loops for this section system:

a) 2.2.3 - 4.6.5  
b) 2.2.3 - 4.8.7  
c) 2.5.6 - 4.8.7

In terms of fault tree synthesis it is necessary to ascertain if there is a special relationship between connections 4 and 1. Since all three loops start and terminate at the same nodes, they need to be considered in unison. It is clear that the three loops in the above figure combine to form an overall loop.

5.7 DETERMINATION OF LOOPS

The user could be asked to specify the loops and to determine the balanced loops, as part of the input to the MASTER program. However, a more satisfactory approach would be to generate this data automatically from the information already provided on the way the units are inter-connected. This information is stored in an array and contains four items of information for each connection, as follows:

upstream unit number  
upstream port number  
downstream unit number  
downstream port number

This array for the configuration of Figure 5.11 is given below:

<table>
<thead>
<tr>
<th>STREAM NUMBER</th>
<th>UPSTREAM</th>
<th>DOWNSTREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNIT</td>
<td>PORT</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3*</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3*</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.2  
Configuration Data for Plant Section of Figure 5.11.

The steps involved in determining the loops will now be outlined.
5.7.1 STEP 1

Inspect the array and pick out the dividers and headers. The basic rule is that a divider has two/more outlets and only one inlet, whereas a header has two/more inlets but only one outlet.

The relevant entries in Table 5.2, for divider unit 3 have been highlighted with an (*). Unit 3 is the downstream unit to one connection only, hence this unit has only one inlet. It is the upstream unit to two connections, i.e. it has two outlets. Hence, it satisfies the above rule. Similar logic can be applied to identify the other divider, unit 5 and the two headers, units 7 and 9.

5.7.2 STEP 2

Condense the above array such that the only nodes are the dividers and headers and the edges the connections between these units.

The purpose of this step is to reduce the search space for the subsequent analysis. Essentially it disregards ‘flow-through’ units, having a single inlet and outlet, which contribute nothing to the detection of loops. The array given in Table 5.2 would thus be reduced to:

<table>
<thead>
<tr>
<th>STREAM NUMBER</th>
<th>UPSTREAM</th>
<th>DOWNSRREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNIT</td>
<td>PORT</td>
</tr>
<tr>
<td>1 (3+4)</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2 (5+6)</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>3 (1+8)</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>4 (3+9)</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5 (11+12)</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5.3
Condensed Configuration Data for Plant Section of Figure 5.11.

5.7.3 STEP 3

The purpose of this step is to find all the loops in the configuration. The basic criterion used is that a loop exists if the path traces over the same node twice.

To illustrate this point, consider starting at divider unit 3 in Figure 5.11. The possible flow legs are the two outlets or the inlet (streams 3,11 and 2). Each of these streams will then be considered to explore the different flow paths. Only one path can be traced at a time, the others being added to a stack for later processing. Any particular search path is terminated if it is not connected to a divider or header unit. Hence stream 2 would not be considered any further.

When stream 3 is considered, streams 5 and 13 from the next divider will be added to the stack. To continue the search entries are removed from the stack on a last-in-first out basis. Assuming stream 5 to be the last entry to have been added to the stack, the next step will note that this stream is in turn connected to a header, unit 7. The other legs, streams 12 and 7 on this header unit will in turn be added to the stack. A continuation of this process will eventually trace stream 12 back to divider unit 3. Since the path has encountered unit 3 twice, there must be a loop in the information flow structure for this

Note: the number given in () in Table 5.3 relates to the corresponding entry in Table 5.2.
system. Hence, the loop 3.3.4 - 5.5.6 - 7.12.11 will be noted. The other loops will be similarly traced.

5.7.4 STEP 4

Step 3 invariably generates duplicate loops since the same loop could start at any node in the given path. This list is next reduced to the unique loops only. This is achieved by:

a) re-arranging the information on a loop such that it always starts at a divider.

b) ensuring that the first and last connections are outlet legs of the above divider

c) sorting the entries by order of the number of nodes in each loop, thus ensuring that any sub-loops appear before any outer loops.

Two loops are deemed to be the same if each entry for each node along the path is the same.

In step 3 the scenario was developed along stream 3 for the divider unit 3 and the loop 3.3.4 - 5.5.6 - 7.12.11 was detected. The search mechanism implemented is deterministic and exhaustive. Consequently, when the entry for stream 11 for this unit is developed the loop 3.11.12 - 7.6.5 - 5.4.3 is also noted. This is clearly the same loop as the one identified above.

It is clearly undesirable to store and then process more information than is absolutely necessary. The objective of noting loops is to prevent the fault propagation path from doubling back on itself. Since the identification of either loop is sufficient to achieve this, it is necessary to remove any duplicated information.

5.7.5 STEP 5

It is necessary for the synthesis package to know the notional start and end of the loop. This may at first sight appear somewhat absurd, since a loop is normally regarded as a continuous entity with an undefinable start and end. However, for the current application it is necessary to define the primary divider and header which give rise to the loop. These two units can then be regarded as the two ends of the loop. In order to achieve this the following criteria are used:

a) a loop is assumed to start at the first divider in the path as determined by step 4.

b) the end of the loop will be assumed to occur at the last header in the path, where the path passes from one header inlet to another inlet on the same header.

These principles will be clarified by reference to the loop 3.3.4 - 5.13.14 - 9.8.7 - 7.12.11. This loop passes through two dividers and two headers. Unit 3 would be regarded as one end of the loop because, in the definition of this loop the path passes from one outlet leg, stream 3 to the other outlet leg stream 11, c.f. the other divider where the path passes from one inlet leg, stream 4 to an outlet leg, stream 13. This criterion is thus used to specify unit 3 as the primary divider.

Using similar logic unit 9 would be regarded as the other end of the loop since for this unit the path passes from one inlet leg, stream 14 to another inlet leg, stream 8, c.f.
header unit 7 where the path passes from an outlet leg, stream 7 to an inlet leg, stream 12.

Having marked the primary divider as the loop start, then the primary header can be considered as the unit where the loop doubles back on itself.

5.7.6 STEP 6

For each loop starting and terminating at the same divider and header it is necessary to determine whether the overall loop is balanced or not. This is necessary to determine whether there is a continuity of flow between the loop inlet and outlet and so set up additional boundary conditions. This concept has already been discussed in Section 5.6.

5.8 USE OF A SINGLE DIVIDER/HEADER MODEL

One limitation of the MK1 methodology already outlined was its dependence on three different types of divider-header combination. Since each of these makes use of different divider and header models, it was necessary for the user to select the correct models from the library at the decomposition stage. A significant advantage can be gained by having a single divider and header pair in the model library.

The need for the different models in the MK1 methodology will be illustrated by reference to the following header model and consideration of the propagation of the flow variable:

![Schematic Representation of Header Model](image)

Figure 5.14

Schematic Representation of Header Model

Figure 5.15 shows the minitrees for high flow through this unit; Figure 5.16 gives the corresponding minitrees for low flow through this unit. The differences can be summarised as follows:

a) the only meaningful deviations along the no flow leg are SOME and REV. There is normally no flow, SOME flow thus represents a deviation in one direction and REV a deviation in the opposite direction.
Figure 5.15
Minitrees for High Flow for Headers for the Three Types of DHC
Figure 5.16
Minitrees for Low Flow in Bypass with Flow Header

a) Minitrees for Low Flow in Bypass with Flow Header

b) Minitrees for Low Flow in Bypass without Flow Header

Note: Port 3 is assumed to have no flow

a) Minitrees for Low Flow in Parallel System Header
b) if there is normally only one flow path, then there must be a direct correspondence of the flow deviations. This is illustrated by the minitree for Q2 LO for the bypass with no-flow system. The only deviation of G1 is LO since there is only one flow leg.

c) the parallel system divider/header models had been set up to cater for a maximum of five legs. In order to handle the greater number of legs, without imposing a much greater memory load on the computer, an incomplete modelling of these units was carried out. The need for this is illustrated by reference to the five-legged header model of Figure 5.17.

![Figure 5.17](image1.png)

Schematic Representation of Five-legged Header Model

The complete minitree for low flow at the outlet would be of the form:

![Figure 5.18](image2.png)

Minitree for Low Flow for Five-legged Header of Figure 5.17

Hence, there would have been 15 branches to this minitop event, with similarly large minitrees for other minitop events. Two simplifications were thus made:

a) events in one inlet port cannot affect the events in another inlet port.
b) in the propagation of an event from an inlet to the outlet, only the direct deviations were matched.

These two points are borne out by the minitrees given in Figure 5.15 and 5.16.

It can be concluded from the above examples that there should be no differences between the models of a type 1 and type 3 DHC. The differences between a type 1 and a type 2 DHC arise because only a subset of the deviations is meaningful.

It was thus decided for the MK2 methodology that the two-legged divider/header models of Type 1 DHC (bypass with flow) be used universally, with the following qualifications:

a) the synthesis package should carry out some checks for the no-flow legs as described below (see Section 5.9).

b) systems with more than two legs should be modelled using a special decomposition technique described later (see Section 5.10.1).

5.9 NO FLOW LEGS

Considering again the diagram of the header given in Figure 5.14, the minitree for Q2 LO is:

![Minitree for Bypass with Flow Header](image)

Now consider port 3 to be the no flow leg. If the propagation path passes down a no flow leg, then the only meaningful deviations of flow are REV and SOME. Hence, a rule is needed to delete causes from this leg if the deviation is neither REV nor SOME. G3 LO and G3 NONE would thus be deleted from the above minitree.
Now consider the minitree for Q2 HI:

![Minitree for High Flow in Bypass with Flow Header](image)

Figure 5.20
Minitree for High Flow in Bypass with Flow Header

In this case a rule is needed such that if a HI deviation is propagated to a no flow leg then it should be swapped for the SOME deviation. Hence, G3 HI would be replaced by G3 SOME.

Additionally if there is only one flow leg, then owing to the continuity of flow, a rule is needed to ensure that only direct deviations are traced. The minitree for Q2 LO is thus modified to give:

![Modified Minitree for Low Flow in Bypass without Flow Header](image)

Figure 5.21
Modified Minitree for Low Flow in Bypass without Flow Header

5.10 DECOMPOSITION AND INPUT TO MASTER PROGRAM

5.10.1 MORE THAN TWO FLOW LEGS

As already indicated for the five-legged header, due to the increased memory size requirements imposed by the divider/header models, the standard models in the library have been created with only two inlet/outlet legs as depicted below:
This may at first sight appear to be a severe restriction on the applicability of such models. Consider again the pump bank shown in Figure 5.9. This system has three legs and cannot directly be modelled using the above models. Two approaches were considered. The first would have involved the creation of a pair of divider and header models with an increased number of legs. This approach was rejected because creating models to cater for five legs was imposing a much greater requirement on the memory than any of the other models in the library. Furthermore, even if models with five legs were created then it would still not be an all-encompassing solution as systems with a greater number of legs could still arise.

The alternative solution adopted, has been to stick with the two-legged models but to carry out the decomposition on the following basis:

This approach is capable of handling an arbitrary number of legs.
5.10.2 FLOW CAPACITIES

Having specified the dividers and headers as part of the configuration, the MASTER program proceeds to detect any loops in the information flow structure. Whether or not any loops are found, the program next prompts for the flow capacity of the legs on the different divider and header units. There are two reasons why this information is necessary:

a) identify the no flow legs
b) cater for excess capacity in parallel systems

The standard divider and header models have been set up assuming that flow is the norm along both legs. The advantage of using a single divider and header pair of models has already been outlined. The action necessary to apply these models to the no flow situation has also been considered. Hence, during the decomposition stage it is necessary to identify any no flow legs on the dividers and headers so that the appropriate action may be taken by the synthesis program.

Additionally, when the divider and header models were created, an implicit assumption was made that both legs were needed to provide the required throughput. Thus for the header shown below:

![Figure 5.24](image)

**Figure 5.24**
Schematic Representation of Header Model

The minitree for low flow would be of the form:

![Figure 5.25](image)

**Figure 5.25**
Minitree for Low Flow for Header Model of Figure 5.24
i.e. if there is a blockage in either leg then low flow will occur at the outlet.

Now assume that each leg is capable of providing 100% of the capacity at the outlet. Under these circumstances an AND gate is required, since there must be blockage down both legs.

In order to cater for these different situations the user has the option of specifying three different flow capacities: 0, 50 or 100 indicating that the leg is capable of providing 0%, 50% or 100% of the required throughput.

One further restriction is that each leg in a parallel system must have the same flow capacity. This information is only used if the deviation being propagated is low flow or pressure at the outlet and is a means of simplifying the tree structure. Consider the pump bank system of Figure 5.9 again, where each leg has a 50% flow capacity. The minitree for low flow at the outlet would simplify to:

```
Low Flow at Outlet
  2/3
  / \   /
Low Flow in Leg 1 Low Flow in Leg 2 Low Flow in Leg 3
```

Having the same capacity allows an r/n gate to be used. If this were not the case, then each combination of legs would have to be evaluated. This could yield a much larger tree with much replication of branches and could quite easily lead to combinatorial explosion for a system with a large number of legs.

5.10.3 INPUT TO ASSIST THE DETECTION ALGORITHM

Consider the following schematic diagram of a heat exchanger model:
Having independent flow paths in the model complicates the process. Assume that the loop detection is progressing along the coolant stream and that the path is currently at port 3. The configuration will already have been specified such that this unit has two outlets, ports 2 and 4. Since no information is present to indicate that these are independent, the loop search algorithm does not know which path to follow. Under these circumstances the program will specify its current location and indicate the alternative flow paths present. The user is expected to select one of these alternatives so that the search mechanism may continue.

There is scope for automating this step by analysing the failure models in combination with the loop search algorithm. The failure models will have propagation equations for flow, which should indicate the independent nature of the flow paths:

\[
Q_{2\text{OUT}} = F(G_{1\text{IN}}, G_{2\text{OUT}})
\]

\[
Q_{4\text{OUT}} = F(G_{3\text{IN}}, G_{4\text{OUT}})
\]

This refinement has not been implemented in the methodology described in this thesis.

5.11 EFFECTS ON FAULT TREE SYNTHESIS

The identification of loops in the information flow structure and the specification of the flow capacities is sufficient for the synthesis program to set up the additional boundary conditions and so generate the correct fault trees.

The switch of deviations for the no flow legs has already been described in section 5.9 above. This section will thus concentrate on the boundary conditions necessary to lead the propagation path in the right direction. This will be illustrated by reference to simplified examples.

Consider the following parallel system:
Consider that the fault tree for HI flow at stream 1 is being developed. When the path passes to streams 2 and 4, the information on the loops prevents the stream 2 branch from passing down stream 5, and the stream 4 branch from passing down stream 3. Furthermore, if two streams have a continuity of flow, as streams 1 and 6, then a boundary condition is imposed that the same variables at these locations should share the same deviations. This last point will be made clearer by reference to the next example.

Consequently the general fault tree structure for the above example would be of the form:

Consequently the general fault tree structure for the above example would be of the form:
One other point to note is that once the propagation path reaches stream 6 then the fault tree structure will be replicated. Under these circumstances only one branch is fully developed, the other being flagged as a looped event as indicated by the asterisk. Loopied events are used in the methodology to prevent replication of the fault tree branches.

Now consider a by-pass without flow system:

![Schematic Representation of Bypass Without Flow System](image)

The fault tree for LO flow in stream 1 would be of the form:

![Fault Tree for Low Flow in Plant Section of Figure 5.30](image)
In this case it is not appropriate to prevent the Q7 LO branch from passing down stream 5. The reason for this is that low flow at the outlet of the header can occur due to leaks in the units in the no flow leg and fluid can pass down this leg either as a result of some flow passing down stream 5 or due to some fluid reversing down stream 2. The importance of this scenario is highlighted by the following two sections of the fault tree:

![Fault Tree Diagram]

*Figure 5.32*

Low Flow Branches for Plant Section of Figure 5.30 due to Faults in the No Flow Leg
By tracing the reverse flow branch it is seen that a leakage in unit 1 is ANDed with the hand valve failing open unit 2. However, the same flow branch indicates that a leakage in unit 1 alone is sufficient to give rise to the top event. This shows the need to be able to trace faults in both directions down a no flow leg in order to obtain the correct minimum cutsets. The normal boundary conditions are sufficient to prevent the propagation path from looping around indefinitely.

It should also be noted in this case that the additional boundary conditions would prevent the G5 REV branch from passing down stream 6, since there is continuity of flow between streams 1 and 6.

In the two examples considered above the given header unit was encountered from the outside. The next example illustrates the action taken when a unit is encountered from the inside. Figure 5.33 shows a common arrangement to regulate the temperature of a stream by controlling the flow of the process stream rather than the utility.

The start of the fault tree for the event T7 LO in this system is given in Figure 5.34 below:

![Fault Tree Diagram](image)

Figure 5.34
Start of Fault Tree for Plant Section of Figure 5.33

The same rules apply to the T9 LO and T5 LO branches as above; the former cannot have causes in stream 3 and the latter cannot have causes in stream 8.

However, one cause of T5 LO is G4 LO. This is a different variable and the boundary conditions set for the temperature variable do not apply to this event. Furthermore this variable will encounter the divider/header units from the inside.
Schematic Representation of a Heat Exchanger Bypass System

Figure 5.33

PIPE 2

DIVIDER 3

CONTROL VALVE 4

HEAT EXCHANGER 5

HEADER 6

PIPE 7

DUMMY TAIL 8

8

9

10

11

1

2

3

4

5

6

7
In this case boundary conditions are set to prevent stream 3 causes from propagating further than stream 9 and the stream 5 causes from propagating further than stream 8.

5.12 PARALLEL SYSTEMS

5.12.1 INTRODUCTION

Some sections of plant are provided to allow redundancy. For example, consider the two arrangements below:

Each pump is capable of providing 50% of the required throughput.

Each relief valve is capable of providing 100% of the required throughput.
In such circumstances the number of legs which have to fail, to give insufficient throughput is variable. One approach could be to create divider/header models to cater for different flow capacities. However, this would be contrary to the objective of having a single divider and header model in the library.

The alternative chosen is that the user should enter the flow capacities of each leg of the divider/header when entering the plant configuration. This allows the two models to be used in a context-independent manner.

Since these arrangements are a means of providing spare capacity, the only deviations which require special consideration are low flow, pressure and relief through the system.

5.12.2 APPLICATION TO AN EXAMPLE SYSTEM

The principle will be outlined by reference to low flow through the system given in Figure 5.35 above. This can occur either as a result of blockage-type or leakage-type faults. The faults in the different legs need to be classified according to one of these two categories. They require special treatment, since they have a different influence on the fault tree structure.

5.12.2.1 BLOCKAGE-TYPE FAULTS

A blockage type fault is one which gives rise to reduced flow due to an increased resistance, e.g. a hand valve failing partially shut. As there is spare capacity it is necessary to group these faults under an r/n gate, where

\[ r = n - \frac{100}{c} + 1 \]

where \( c \) is the capacity of one leg expressed as a %.

There is an assumption that the capacity of each leg is the same. This proviso has been made for two reasons:

a) for parallel systems equal capacities will usually be the norm.

b) otherwise the tree structure would be made unduly complex and large, with each combination of legs having to be evaluated separately.

5.12.2.2 LEAKAGE-TYPE FAULTS

This type of fault gives rise to low flow due to a loss of fluid from the system. It is assumed that a leakage in any one leg is sufficient to give rise to low flow since the loss of fluid from the system may be greater than the ability to compensate. They are thus fundamentally different from the above category of fault since an OR gate is called for.
5.12.3 PROPOSED SOLUTION

The solution adopted is a modification of that used by Kelly [Ref 5-1]. The modification is needed since:

a) the current methodology does not explicitly declare divider-header combinations; a loop-based approach is being used.

b) more than two-legged divider/headers are modelled as nested systems, as described in Section 5.10.

Furthermore, the MK1 methodology accorded this special treatment to low flow only. The current methodology has been extended to cater for low pressure and relief as well.

The basic principle used to distinguish between the two types of fault is outlined below, with reference to the following pipe unit:

![Figure 5.37 Pipe Unit](image)

The minitrees for low flow at the inlet and outlet are:

![Figure 5.38 Minitrees for Low Flow in Pipe Unit](image)

Hence, a blockage-type fault is one which gives rise to the low deviation in both directions of fault propagation, e.g. PART-BLK. A leakage-type fault is one which causes a low deviation only in the direction of fault propagation. Hence, if the upstream causes are being traced, then LK-LP-EN is a leakage-type fault.
5.12.4 IMPLEMENTATION

The generic form of the fault tree, for the plant section of Figure 5.35, is given below:

![Generic Fault Tree for Low Flow in Parallel Redundant System](Figure 5.39)
The domains of the internal and external branches are determined by there being a continuity of flow between streams 1 and 8. Hence the internal, leakage and blockage branches are prevented from tracing faults to either of these streams. Similarly, the external branch cannot trace faults back into the blockage and leakage branches. The criterion for deciding which faults to include in the leakage and blockage branches has already been described. The actual procedure for achieving this is outlined below.

5.12.5 SEPARATING THE LEAKAGE AND BLOCKAGE TYPE FAULTS.

Consider one leg of the above combination in greater detail:

![Diagram of a leg of the parallel system]

Consider the direction of fault propagation to be from stream 4 to stream 1. To simplify the explanation only the low flow deviation will be developed. The fault tree for this section of the plant will be of the following form:

![Fault tree for low flow in plant section]

Figure 5.41
Fault Tree for Low Flow in Plant Section of Figure 5.40
When the propagation path reaches G3 LO, the basic event causes of this are deleted, since they violate the prevailing boundary conditions. The two basic event causes of G3 LO, PART-BLK Unit 3 and LK-HP-EN Unit 3 are removed from the tree, because the former is a duplicated event whereas the latter is a not allowed fault of Q4 LO. A LK-HP-EN will tend to give rise to Q4 HI rather than Q4 LO. However, as mentioned earlier it is necessary to note the events which give rise to G3 LO as well as those that cause Q4 LO. Before the causes of G3 LO are deleted, a note of the events is made on a separate stack (G-stack) for later processing. This procedure would be similarly repeated for the other units, i.e. the pump and the hand valve in this case.

When the initial fault tree is synthesised, the tree is developed with the internal and external branches only. During fault tree rationalisation the contents of the internal branch are copied. The original branch is now labelled as the BLOCKAGE branch and the copied one the LEAKAGE branch. Use is now made of the events stored in the above G-stack. Events in the BLOCKAGE branch which also appear in the G-stack are left in the blockage branch. Any events which occur in the blockage branch but do not appear in the G-stack are deleted from this branch.

For the LEAKAGE branch the converse is true, i.e. any events which appear in both the G-stack and this branch are deleted from this branch whereas events appearing in this branch but not in the G-stack are retained in this branch.

The above criteria are clarified by reference to the start of the fault tree given in Figure 5.41:

![Fault Tree Diagram](image)

The above section of the tree would be copied. It has already been stated that the G-stack for this level of development will contain the events PART-BLK Unit 3 and LK-HP-EN Unit 3. The Blockage branch will thus retain the event PART-BLK since it appears in the G-stack. The event LK-LP-EN will be deleted from this branch as it does not appear in the G-stack.

Conversely, the PART-BLK event will be deleted from the Leakage branch since it does appear in the G-stack, but will retain the event LK-LP-EN as it does not appear in this stack.
5.13 CONCLUSION

This chapter has described the enhancement of the FAULTFINDER methodology such that the user no longer has to treat Divider-Header Combinations as a special sub-system during the system decomposition phase. Instead the package automatically identifies the dividers and headers in the configuration and then determines the loops in the information flow structure.

Additionally, the user is shielded from having to specify the type of divider/header model to use. A single pair of divider/header models can now cater for:

- bypass with flow systems.
- bypass without flow systems.
- parallel redundancy systems.

However, in order to achieve this the user has to specify the flow capacity of each leg of the dividers and headers in the configuration. The package automatically prompts for this information. A decomposition convention has also been outlined to enable the two-legged divider/header models to handle an arbitrary number of legs in the system.

Chapter 10 of this thesis presents a worked example based on the plant configuration of Figure 5.35.
6.0 RULES FOR THE DECOMPOSITION OF CONTROL LOOPS

6.1 INTRODUCTION

The methodology described thus far has involved the decomposition of the plant representation into its constituent units and the connections between these. The problems inherent in this technique have been highlighted by Kelly [Ref 6-1] and by Shafaghi [Ref 6-2]. Generally two types of problem are encountered:

- incorrect fault trees, since the combination of sensor, controller and control valve introduce failure modes into the tree which cannot be adequately modelled via an atomistic approach to the decomposition of individual components.

- opaque fault trees. Unlike a human expert, automated methodologies cannot readily perceive the structure that the overall tree should have. The failure modes associated with control loops are an important constituent of the fault tree, which an automated methodology can easily obscure by less important failures.

In order to overcome these problems, control loops are treated as a special sub-system during the decomposition stage, and the user is expected to supply additional information to describe the functionality of the control loop.

6.2 TYPES OF CONTROL LOOP

The methodology considers two basic types of control loop:

feedback
feedforward

The following figure shows a feedback control loop.

![Schematic Representation of a Feedback Control Loop](image_url)
The purpose of this control loop is to regulate the temperature of the tube-side medium downstream of the heat exchanger by manipulating the flow of coolant through the shell-side. The control loop is of the feedback type since the flow rate of coolant is adjusted only if there is a deviation in the temperature of the regulated stream.

The following figure shows a feedforward control loop.

![Figure 6.2: Schematic Representation of a Feedforward Loop](image)

Additional, the methodology needs to know whether the control loop has a separate manipulated stream. In essence all control loops act by adjusting the control valve aperture and so can be said to have a manipulating action. However, the distinction between a manipulated stream and a separate manipulated stream is of importance. Both control loops considered above have a separate manipulated stream, since the control valve is to be found in a separate process stream from that where the sensor is positioned. To highlight this distinction Figure 6.3 shows a system where there is no separate manipulated stream for the control loop.

![Figure 6.3: Schematic Representation of Control Loop without Separate Manipulated Stream](image)
6.3 FAILURE MODES FOR THE CONTROL LOOPS

The distinctions highlighted above are important since the function of the control loop ultimately determines its generic failure modes. For this purpose the fault tree synthesis algorithm has been programmed with three basic control loop operators which structure the fault tree to reflect the type of control loop under consideration.

For a feedforward or feedback control loop where a deviation in the regulated variable is being developed, the template to be applied is shown in Figure 6.4:

![Figure 6.4](image)

Control Loop Template for Regulated Variable Deviation

If the control loop has a separate manipulated stream and a deviation in the manipulated variable is being developed then the templates are slightly different for a feedback and feedforward control loop.

Figure 6.5 gives the template to be applied in the case of a feedback control loop.

Figure 6.6 gives the template to be applied in the case of a feedforward control loop.

Note: the meaning of the different branches in Figures 6.4, 6.5 and 6.6 is explained on page 6/5.
Figure 6.5
Manipulated Variable Deviation Template for a Feedback Control Loop

Figure 6.6
Manipulated Variable Deviation Template for a Feedforward Control Loop
The spontaneous and latent failure branches in the above figures represent the two basic modes of failure of a control loop due to a malfunction in any of its components. The spontaneous failure branch traces the causes which give rise to either an increased or decreased aperture in the control valve. These two states are of course mutually exclusive and only one will be meaningful in any given application. The latent failure branch traces the causes which render the control loop invariant. This state represents the condition where the control valve aperture is stuck in the plant steady-state position, such that any deviations in the process conditions cannot be compensated.

The distinction between the different templates will be highlighted by reference to a couple of examples. In the schematic representation of Figure 6.1, if the event being developed is a deviation in the temperature of the hot stream, then the template given in Figure 6.4 will be applied. The overload branch would contain faults like complete loss of coolant, since even the normal action of the control loop would be unable to compensate for this. The sensed variable deviation would trace the causes of the temperature deviation at the inlet to the sensor. Since the deviation in this stream is correctable by the normal action of the control loop, it is necessary to AND this branch with the latent failures in the control loop. It should be noted that there are no events which mislead this control loop.

If instead the event being developed was a deviation in the flow of the manipulated stream, then the template given in Figure 6.5 would be applied. The deviation of the sensed variable branch would trace the causes of a deviation in the temperature to the inlet of the sensor. This represents the normal action of the control loop, since in order to rectify a deviation in the sensed variable there must be a corresponding adjustment of the manipulated stream. However, if a deviation in the manipulated stream occurs, then this will eventually manifest itself on the sensed variable. The control loop should be able to correct for these failures. Hence this branch is ANDed with control loop latent failures.

The control loop in the schematic representation of Figure 6.2 is an example of a feedforward control loop. In this case, if the fault tree is being developed for a deviation in the manipulated stream then the template shown in Figure 6.6 would be applied. The basic difference between Figures 6.5 & 6.6 is that for the feedforward control loop the manipulated variable deviation is not ANDed with control loop latent failures. This is because unlike the heat exchanger example a deviation in the manipulated stream does not have an influence on the sensed variable. Since the control loop cannot detect deviations in the manipulated stream, then it is unable to correct for this type of failure.

6.4 FILLING IN THE CONTROL LOOP TEMPLATES

As mentioned earlier in Chapter 3, in order to correctly handle control loops in the methodology it is necessary to supply additional details.

In order for the methodology to select the appropriate control loop template it is necessary to specify:

- the sensed variable
- the regulated variable(s)
- the manipulated stream, if separate
- whether the control loop is feedforward or feedback

In order to define the scope of influence within the plant, it is necessary to specify:
- the streams where the variable(s) is(are) regulated
- the streams where the flow is manipulated, if there is a separate manipulated stream. It is assumed within the methodology that only the flow can be manipulated by altering the control valve aperture.

In order to define the components of the control loop spontaneous and latent failure branches it is necessary to specify:

- the control valve number
- the sensor unit number
- any intervening units between the sensor and the control valve, e.g. the controller.

Additionally, the failure models for the control loop components need to observe certain modelling conventions in order to work correctly with the above control loop templates in the synthesis algorithm.

6.5 DEFICIENCY IN THE MK1 METHODOLOGY

The basic templates described above and the conventions for creating the failure models for the control loop components had been developed by Kelly. However, two fundamental problems were encountered during the course of this project:

- there were no clear rules for decomposition, particularly in the case of complex control loops.
- the rules for fault tree synthesis required further development, particularly in the case of complex control loops and combined control and trip loops.

The rules for decomposition were developed by modelling a number of systems with a variety of control strategies. The rules developed will be illustrated by reference to a number of examples.

The additional rules for fault tree synthesis will also be illustrated by reference to specific examples modelled during the course of this project.

6.6 RULES FOR DECOMPOSING CONTROL LOOPS

The basic rules will be described with reference to a modified representation of the Lapp-Powers Heat Exchanger system. Guidelines for applying these rules will be given by applying the rules to a number of examples.

6.6.1 MODIFIED REPRESENTATION OF LAPP-POWERS HEAT EXCHANGER SYSTEM

The schematic representation of this system is given in Figure 6.7. It should be noted that the trip system which normally forms a part of this configuration has been omitted for simplicity.

6.6.1.1 RULE 1

- identify the control loops in the system.
This is straightforward for this system, but will not always be the case as later examples will show. The criterion to be used is that each sensor sending a corrective signal to a control valve constitutes a control loop. In this system there is a single sensor (Unit 4), acting via the controller (Unit 11) and sending a corrective signal to the control valve (Unit 9). Hence, this system has a single control loop.

6.6.1.2 RULE 2

- determine whether the control valve is in the same process stream as the sensor or whether it is in a separate stream.

The coolant and nitric acid streams are quite obviously two separate streams. Since the control valve is in a separate stream from the sensor, this control loop is said to have a separate manipulated stream.

6.6.1.3 RULE 3

- if the control loop has a separate manipulated stream, then determine the streams where the flow is manipulated.
The flow will be manipulated in all streams upstream and downstream of the control valve where there is a single, continuous flow path. Since the flow of coolant is a single flow path and there is no accumulation in any of the units, all the streams are specified as falling into this category.

Note the flow would be non-continuous if there were scope for accumulation within a unit, e.g. level in a tank, and it would not be a single stream if any unit had more than two inlets or outlets, e.g. dividers and headers, since these potentially offer alternative flow paths.

In general the tracing needs to be done both upstream and downstream of the control valve, until a unit of either of the above two types is located. All streams thus traced would be specified as having the flow manipulated in them.

Although not implemented as part of the methodology being described in this thesis, there is scope for automating this search for the manipulated stream. The algorithm would be based on finding divider/header-type and vessel-type units. A procedure for achieving the former has already been outlined in the previous chapter. Vessel-type units could easily be located by noting the port-type that the path passes through. In the case of vessel-type units, the path must pass from an IN/OUT port to a VES port.

6.6.1.4 RULE 4
- identify the variables regulated by the control loop.

If the control valve is in the same process stream as the sensor, then the sensed variable will also be the regulated variable. Any variables dependent on this variable would also be classified as being regulated by the control loop.

In this example, however, the control valve resides in a separate stream to that of the sensor. In such a control system there must be an interaction between the sensed and manipulated streams. By examining the units on either side of the control valve it is found that this interaction takes place in the heat exchanger. The failure model for this unit contains a propagation equation linking the temperature of the nitric acid stream to the flow of the coolant. Since the temperature is the only variable in the hot stream affected by a change in the flow of the coolant, this is noted as being the only variable regulated by the control loop.

Note that in general it is necessary to look both upstream and downstream of the control valve to find the place of interaction; the control valve could easily have been positioned downstream of the heat exchanger.

6.6.1.5 RULE 5
- define the streams where the variables identified in Rule 4 are regulated.

For a control loop which operates via the influence of a separate manipulated stream, these streams are those downstream of the point where the interaction takes place, i.e. downstream of the heat exchanger in this case. All streams downstream of the heat exchanger should be specified up to the next unit where this variable can be altered within the confines of that unit. Examples of this type of unit would be heat exchangers, reactors involving endo/exo-thermic reactions, etc.
For this example there is no unit downstream of the heat exchanger which has the potential to alter the temperature of this stream by heat transfer. Hence, all streams downstream of the heat exchanger are specified as having the temperature regulated by the control loop.

Again, although not implemented as part of the methodology being described in this thesis, there is scope to automate this search. For example, if there was a heat exchanger downstream of the section of plant of Figure 6.7, then the propagation equations for temperature in this unit could be used to ascertain the influence on the T variable.

Assuming that ports 1 and 2 represent the flow of the hot stream and ports 3 and 4, the flow of the coolant, then this model would have a propagation equation of the following form:

\[ T_{2\text{OUT}} = F(T_{1\text{IN}}, -G_{3\text{IN}}, G_{1\text{IN}}, T_{3\text{IN}}) \]

There is a continuity of flow between ports 1 and 2. Hence, since \( T_{2\text{OUT}} \) is dependent upon parameters from another stream, it can be concluded that this unit has the potential to influence the T variable.

6.6.1.6 RULE 6
- determine whether the control loop is feedforward or feedback.

A control loop is feedback if it acts on a deviation in its regulated variable, otherwise it is feedforward. For the current example the control loop is feedback since it can only act upon a deviation in the temperature at the outlet of the heat exchanger, the regulated variable.

6.6.1.7 RULE 7
- determine the units which make up the control loop.

The control valve and the sensor are obviously an integral part of any control loop. Additionally any units which lie between these two and which are responsible for transmitting a deviation in the sensed variable to the stem of the control valve, should be specified as forming part of the control loop. Hence, for the current example the controller and setpoint units are included in the definition of the control loop. The control loop latent and spontaneous failure branches are restricted to tracing faults in only these units.

6.6.1.8 SUMMARY OF INPUT TO FAULTFINDER

The following is a summary of the input required by FAULTFINDER to specify the control loop in the above system.

SENSED VARIABLE: T
VARIABLE SENSED IN UNIT: 4
CONTROL VALVE UNIT NUMBER: 9
OTHER UNITS IN CONTROL SYSTEM ARE: 11 12
VARIABLE T REGULATED IN CONNECTIONS: 3 4
FLOW MANIPULATED IN CONNECTIONS: 5 6 7 8 9
LOOP IS NOT OF THE FEEDFORWARD TYPE
6.6.2 FEEDFORWARD MIXER COMPOSITION SYSTEM

The schematic representation of this system is given in Figure 6.8. This system illustrates the difference between a feedforward and feedback system. The summary input required by FAULTFINDER is given below:

SENSED VARIABLE: Q
VARIABLE SENSED IN UNIT: 8
CONTROL VALVE UNIT NUMBER: 3
OTHER UNITS IN CONTROL SYSTEM ARE: 12 13
VARIABLE X REGULATED IN CONNECTIONS: 9 10
FLOW MANIPULATED IN CONNECTIONS: 1 2 3 4
LOOP IS OF THE FEEDFORWARD TYPE

In this and all subsequent examples only the significant new features arising from the application of the rules developed for the Lapp-Powers system considered above, will be highlighted.

Rule 3: gives streams 1-4 as having the flow manipulated in them; other connections further downstream are not included since the mixer violates the condition of a single stream.

Rule 4: yields that the interaction between the manipulated and sensed streams takes place in the mixer. The mixing of the two streams alters the composition, hence this is specified as being the regulated variable of the control loop. This is the only regulated variable as the mixing process does not affect any other variables.

Rule 5: indicates that both connections downstream of the mixer should be specified as having the composition regulated in them, since there is no scope for mass transfer to occur in the pipe (Unit 10).

Rule 6: indicates that the control loop is feedforward since its action is not dependent on a deviation in the regulated variable.

6.6.3 LEVEL CONTROL SYSTEM

This system has been included to indicate how the rules outlined above in the context of flow systems can be applied to tank-type units where the internal variables are also of interest. The schematic representation of this system is shown in Figure 6.9. The summary input required by FAULTFINDER is given below:

SENSED VARIABLE: L
VARIABLE SENSED IN UNIT: 7
CONTROL VALVE UNIT NUMBER: 5
OTHER UNITS IN CONTROL SYSTEM ARE: 8 9
VARIABLE L REGULATED IN CONNECTIONS: 6
FLOW MANIPULATED IN CONNECTIONS: 3 4 5
LOOP IS NOT OF THE FEEDFORWARD TYPE
Figure 6.8
Schematic Representation of a Feedforward Control System
Figure 6.9
Schematic Representation of Level Control System
Rule 2: yields that there is a separate manipulated stream. Owing to accumulation in the tank, the internal variables should be regarded as belonging to a separate 'stream', rather than to the inlet and outlet ports.

Rule 3: gives that streams 3,4 and 5 have the flow manipulated in them; streams 1 and 2 are not included since there can be accumulation in the tank.

Rule 4: yields that the interaction between the sensor stream and the manipulated stream takes place in the tank. Deviations in the manipulated stream affect the level, hence this is the variable regulated by the control loop.

Rule 5: gives that stream 6 is the one where the level is being regulated. In general for tank-type vessels the stream to which the sensor is connected will be the control loop regulated variable stream.

Rule 6: gives that the control loop is feedback since its action is dependent upon detecting a deviation in the regulated variable.

6.6.4 LIHOU HYDROCARBON/OXYGEN REACTOR

This system highlights an important principle in counting the number of control loops in the system, and further clarifies the method of determining the regulated variables and the streams where they are regulated. A simplified, flow diagram, along with the decomposition diagram for this system are given in Figures 6.10a and 6.10b, respectively.

The input required by FAULTFINDER to define the control loops is given below:

CONTROL LOOP: 1
SENSED VARIABLE: Q
VARIABLE SENSED IN UNIT: 5
CONTROL VALVE UNIT NUMBER: 4
OTHER UNITS IN CONTROL SYSTEM ARE: 18 19
VARIABLE Q REGULATED IN CONNECTIONS: 1 2 3 4 5 6
LOOP IS NOT OF THE FEEDFORWARD TYPE

CONTROL LOOP: 2
SENSED VARIABLE: Q
VARIABLE SENSED IN UNIT: 11
CONTROL VALVE UNIT NUMBER: 10
OTHER UNITS IN CONTROL SYSTEM ARE: 15 16 17
VARIABLE Q REGULATED IN CONNECTIONS: 7 8 9 10 11 12
LOOP IS NOT OF THE FEEDFORWARD TYPE

CONTROL LOOP: 3
SENSED VARIABLE: Q
VARIABLE SENSED IN UNIT: 11
CONTROL VALVE UNIT NUMBER: 4
OTHER UNITS IN CONTROL SYSTEM ARE: 15 18 19
VARIABLE X REGULATED IN CONNECTIONS: 13 22
VARIABLE T REGULATED IN CONNECTIONS: 13 22
FLOW MANIPULATED IN CONNECTIONS: 1 2 3 4 5 6
LOOP IS OF THE FEEDFORWARD TYPE
Rule 1: gives that there are three control loops, despite the fact that there are only two control valves and two sensors. The three control loops are:

a) Sensor Unit 5 providing a corrective signal to control valve Unit 4.
b) Sensor Unit 11 providing a corrective signal to control valve Unit 10.
c) Sensor Unit 11 providing a corrective signal to control valve Unit 4.

The other rules need to be applied to each control loop in turn.

6.6.4.1 CONTROL LOOP 1

The application of rules 2-5 indicates that there is no separate manipulated stream and that the regulated variable is flow, since the control valve is in line with the sensor. The flow is said to be regulated in streams 1-6. Other streams further downstream are not included since the reactor has more than one inlet.

Rule 6: gives that the control loop is feedback, since corrective action is dependent upon detecting a deviation in the regulated variable.

In the application of Rule 7 it should be noted that although Unit 15 is connected to Unit 18, it is not specified as part of this control loop. This is because this control loop acts purely on the signal received from sensor Unit 5.

6.6.4.2 CONTROL LOOP 2

The reasoning is identical to that for control loop 1.

6.6.4.3 CONTROL LOOP 3

Rule 2: gives that the control valve is clearly situated in a separate stream from that of the sensor. It can thus be deduced that there is a separate manipulated stream.

Rule 3: gives that the flow is manipulated in streams 1-6. Other streams are not included since the reactor has more than one inlet.

The application of Rule 4 indicates that the manipulated and sensed variable streams interact in Unit 13, the reactor. In essence this control loop is regulating the flow ratio to the reactor. Since a reaction is taking place, deviations in the flow ratio will manifest themselves via other simpler variables. These will be detected both within the reactor and at its outlet.

The control loop is therefore indirectly regulating the composition, since the ratio of the two inlet streams will influence the degree of reaction. However, since the reaction is exothermic, the degree of reaction will also influence the temperature. Hence, the control loop is said to be decreasing deviations in both these variables.

It is necessary to take this approach since the current treatment of control loops does not consider a complex variable like flow ratio in its own right for the diagnosis of control loop failures.
Rule 5: gives that the two variables are regulated in all the streams downstream of the reactor and also the streams connected to the vessel ports. Hence, streams 13 and 22 are said to have the temperature and composition regulated in them. Note the difference between this system and the level control system; level can only manifest itself at a vessel port, whereas temperature and composition will also manifest themselves at the inlet/outlet ports.

The application of Rule 6 yields that the control loop is feedforward, since it is acting upon a variable other than either of the regulated ones.

6.6.5 DISTILLATION COLUMN PROBLEM

This example has been included as it illustrates how the methodology handles the added complexities of combined heat and mass transfer. The flow diagram, along with the decomposition diagram for this system are given in Figure 6.11a and 6.11b, respectively.

The summary input required by FAULTFINDER is as follows:

CONTROL LOOP: 1
SENSED VARIABLE: T
VARIABLE SENSED IN UNIT: 24
CONTROL VALVE UNIT NUMBER: 8
OTHER UNITS IN CONTROL SYSTEM ARE: 25 26
VARIABLE X REGULATED IN CONNECTIONS: 3 4 5 6 7 8 9 10 11 25
VARIABLE T REGULATED IN CONNECTIONS: 3 25
FLOW MANIPULATED IN CONNECTIONS: 7 8
LOOP IS NOT OF THE FEEDFORWARD TYPE

CONTROL LOOP: 2
SENSED VARIABLE: L
VARIABLE SENSED IN UNIT: 27
CONTROL VALVE UNIT NUMBER: 9
OTHER UNITS IN CONTROL SYSTEM ARE: 28 29
VARIABLE L REGULATED IN CONNECTIONS: 29
FLOW MANIPULATED IN CONNECTIONS: 9 10 11
LOOP IS NOT OF THE FEEDFORWARD TYPE

CONTROL LOOP: 3
SENSED VARIABLE: T
VARIABLE SENSED IN UNIT: 30
CONTROL VALVE UNIT NUMBER: 22
OTHER UNITS IN CONTROL SYSTEM ARE: 31 32
VARIABLE X REGULATED IN CONNECTIONS: 33 12 14 15 16 17
VARIABLE T REGULATED IN CONNECTIONS: 33 12 14 15 16 17
FLOW MANIPULATED IN CONNECTIONS: 22 23 24
LOOP IS NOT OF THE FEEDFORWARD TYPE
CONTROL LOOP: 4  
SENSED VARIABLE: L  
VARIABLE SENSED IN UNIT: 33  
CONTROL VALVE UNIT NUMBER: 14  
OTHER UNITS IN CONTROL SYSTEM ARE: 34 35  
VARIABLE L REGULATED IN CONNECTIONS: 37  
FLOW MANIPULATED IN CONNECTIONS: 14 15 16 17  
LOOP IS NOT OF THE FEEDFORWARD TYPE

The application of Rule 1 yields the following four control loops:

a) Sensor Unit 24 providing a corrective signal to control valve Unit 8.
b) Sensor Unit 27 providing a corrective signal to control valve Unit 9.
c) Sensor Unit 30 providing a corrective signal to control valve Unit 22.
d) Sensor Unit 33 providing a corrective signal to control valve Unit 14.

6.6.5.1 CONTROL LOOP 1

Rules 2 and 3 yield that there is a separate manipulated stream and that the flow is manipulated in streams 7 and 8. Other streams are not included since upstream there is a divider and downstream there is a vessel.

Rule 4: gives that the sensed and manipulated streams interact in the distillation column, Unit 3. This control loop is effectively manipulating the reflux ratio; the equilibrium equations for the distillation column indicate that changes in the reflux ratio will affect the temperature and composition at the top of the column. Hence, these can be taken to be the control loop regulated variables.

Rule 5: gives that these two variables must be regulated in the streams associated with the vessel ports and any outlets at the top of the column. Streams 3 and 25 are thus said to have both the temperature and composition regulated in them.

Stream 3 leads to a condenser. Since heat transfer takes place in this unit, no other streams can be said to have the temperature regulated in them. However, if it assumed that total condensation takes place in the condenser then streams 4-11 are also said to have the composition regulated in them.

Rule 6: gives that the loop is feedback since it obviously acts upon a deviation in one of its regulated variables.

6.6.5.2 CONTROL LOOP 2

This control loop is analogous to the level control system already discussed. The only significant point of note is that the flow is only manipulated in streams 9, 10 and 11. Streams 5 and 6 are not included, since the flow along these can be influenced by the other outlet leg of the divider, Unit 7.
6.6.5.3 CONTROL LOOP 3

The main problem posed by this control loop is in determining the control loop regulated variables and the streams where those variables are regulated. Unlike all the previous examples there is no direct interaction between the manipulated stream and the sensed stream. In this case there is an indirect interaction, since the manipulation of the steam flow to the reboiler alters the boil-up rate to the distillation column.

Adjusting the boil-up rate regulates the temperature and composition in the bottom half of the distillation column. Hence, both these variables are said to be the control loop regulated variables.

Both the variables are regulated in streams 33, 12, 14, 15, 16 and 17. The inclusion of streams 14-17 may at first sight appear strange, since the reboiler is a source of heat and mass transfer. However, there is a significant difference between this unit and the condenser which influenced control loop 1. Unlike the flow of coolant to the condenser, the flow of steam to the reboiler is being directly manipulated by this control loop.

6.6.5.4 CONTROL LOOP 4

This is analogous to the level control system already described and poses no new problems.

6.6.6 COMPLEX CONTROL LOOPS

In all the examples considered thus far any variable in any given stream has been under the influence of only one control loop. By correctly decomposing the control loops, the synthesis algorithm is able to apply the appropriate control loop operator whenever it detects deviations in the variables which have been specified as being regulated or manipulated by the control loops.

The next two examples will highlight the situation where more than one control loop is influencing the same variable.

6.6.6.1 COMPOSITION CONTROL LOOP

A simplified version of this example has already been considered. In the current context a feedback control action has also been included. The schematic representation of this system is given in Figure 6.12.

The summary input required by FAULTFINDER is given below:

CONTROL LOOP: 1
SENSED VARIABLE: X
VARIABLE SENSED IN UNIT: 11
CONTROL VALVE UNIT NUMBER: 3
OTHER UNITS IN CONTROL SYSTEM ARE: 14 15 16
VARIABLE X REGULATED IN CONNECTIONS: 9 10 11 12
FLOW MANIPULATED IN CONNECTIONS: 1 2 3 4
LOOP IS NOT OF THE FEEDFORWARD TYPE
Figure 6.12
Schematic Representation of Combined Feedback and Feedforward Control Loops

STREAM A

1. DUMMY HEAD 1
2. PIPE 2
3. CONTROL VALVE 3
4. MIXER 5
5. PIPE 9

STREAM B

6. PIPE 7
7. FLOW SENSOR
8. PIPE 10
9. COMPONENT B SENSOR 11
10. PIPE 12
11. COMPONENT A SENSOR 10
12. PIPE 12
13. DUMMY TAIL 13
14. CONTROLLER 14
15. SETPOINT 15
16. INSTRUMENT AIR 16
17. PIPE 11

CONTROL LOOP: 2
SENSED VARIABLE: Q
VARIABLE SENSED IN UNIT: 8
CONTROL VALVE UNIT NUMBER: 3
OTHER UNITS IN CONTROL SYSTEM ARE: 14 15 16
VARIABLE X REGULATED IN CONNECTIONS: 9 10 11 12
FLOW MANIPULATED IN CONNECTIONS: 1 2 3 4
LOOP IS OF THE FEEDFORWARD TYPE

The application of Rule 1 yields two control loops:

a) Sensor Unit 11 providing a corrective signal to control valve Unit 3.
b) Sensor Unit 8 providing a corrective signal to control valve Unit 3.

6.6.6.1.1 CONTROL LOOP 1

Owing to the mixer Unit 5, Rules 2 and 3 indicate that there is a separate manipulated stream. The flow is said to be manipulated in streams 1-4.

The manipulated stream interacts with the sensed stream in the mixer, to influence the composition. Hence, this is specified as being the regulated variable in the streams downstream of the header.

The action of this control loop is obviously feedback since it depends upon detecting a deviation in the regulated variable in order to supply a corrective action.

6.6.6.1.2 CONTROL LOOP 2

The definition of this control loop is the same as that given in the earlier example.

6.6.6.1.3 PROBLEMS ENCOUNTERED BY THE SYNTHESIS ALGORITHM

Control loops of this nature present additional problems to the fault tree synthesis algorithm. Consider the situation where a fault tree is being developed for a composition deviation in any of the streams downstream of the header. Since this variable has been specified as being regulated by both the control loops, the synthesis algorithm is aware that the control loop operators should be applied, to include failures in these components into the fault tree.

However, the package cannot decide which operator should be applied first. Figure 6.13 illustrates how the order of application of the operators influences the structure of the fault trees. In this it has been assumed that the top event of interest is a high composition of component B and that the operator for control loop 1 is applied first.

The sequential application of operators assumes that one control loop is dominant over the other. This is illustrated in Figure 6.13, where control loop one is able to correct for failures in the components of control loop two but that the reverse is not true.

For this system the sequential application of the control loop operators is correct, since one control loop is feedforward whereas the other is feedback. However, this is not always the case as the next example will show.
Figure 6.13
General Form of Fault Tree for High Composition of the Regulated Stream
The synthesis algorithm can cope with this type of system if the following rule is observed: the master loop should be specified before the slave loop during the decomposition stage. This has been done in the above example as the master control loop has an index of 1, whereas the slave control loop has an index of 2.

Nevertheless, there are still a number of defects in this tree. They arise because the two control loops share a number of common components, and the fact that the control loop operator has been applied sequentially. Two examples of this type of defect are illustrated below:

- the CL-STK Loop 1 branch is anded with the CL-F-LA Loop 2 branch. The former will contain the failure CV-STK, whereas the latter will contain the failure CV-F-LA, for the control valve which is common to both control loops. Since control loop 1 is dominant and the operator for this is applied prior to that for control loop 2, what is required is a procedure which deletes any contradictory events in the slave loop template.

- both control loop templates contain branches for control loop overload, i.e. CL-O-LD Loop 1 and CL-O-LD Loop 2. It has already been intimated that the control loop overload branch will essentially contain failures related to the complete loss of the control loop manipulated stream, i.e. no flow. Since both the control loops are manipulating the same stream, there is likely to be much replication of faults in these branches. The only events which are different are those which relate to the control loop components and which are unique to each control loop, i.e. the sensors in this case. They can cause a complete loss of the manipulated stream by failing in a position which causes the control valve to completely shut. Hence, a procedure is necessary to rationalise the events appearing in the control loop overload branches to avoid this replication of faults and thereby enhance the transparency of the tree.

A detailed discussion of this system is delayed until Chapter 10, since some features to be introduced in later chapters have a bearing on this discussion.

6.6.6.2 PRESSURE REDUCTION INSTALLATION

This is a simplified version of a system to be discussed in Chapter 10. Figure 10.55 in Chapter 10 gives the flow diagram for this system. The system discussed here, essentially comprises four control loops. The decomposition diagram for this is given in Figure 6.14.

The purpose of the control loops is to ensure a correct pressure in Stream 8. PCL and PCH are reverse-acting controllers. The signal selector selects the lowest signal from the two controllers to feed to the control valve which is of the air-to-open type. PC and PCO are both direct acting controllers; the signal selector selects the highest signal from these to feed to the control valve which of the air-to-close type.

Using the rules developed earlier there is no problem in deriving the following information to input to the MASTER program:
CONTROL LOOP: 1
SENSED VARIABLE: P
VARIABLE SENSED IN UNIT: 7
CONTROL VALVE UNIT NUMBER: 5
OTHER UNITS IN CONTROL SYSTEM ARE: 11 12 13 6
VARIABLE P REGULATED IN CONNECTIONS: 5 6 7 8 9
VARIABLE Q REGULATED IN CONNECTIONS: 1 2 3 4 5 6 7 8 9
LOOP IS NOT OF THE FEEDFORWARD TYPE

CONTROL LOOP: 2
SENSED VARIABLE: P
VARIABLE SENSED IN UNIT: 6
CONTROL VALVE UNIT NUMBER: 5
OTHER UNITS IN CONTROL SYSTEM ARE: 11 12 13 7
VARIABLE P REGULATED IN CONNECTIONS: 5 6 7 8 9
VARIABLE Q REGULATED IN CONNECTIONS: 1 2 3 4 5 6 7 8 9
LOOP IS NOT OF THE FEEDFORWARD TYPE

CONTROL LOOP: 3
SENSED VARIABLE: P
VARIABLE SENSED IN UNIT: 8
CONTROL VALVE UNIT NUMBER: 3
OTHER UNITS IN CONTROL SYSTEM ARE: 14 15 16 4
VARIABLE P REGULATED IN CONNECTIONS: 3 4 5 6 7 8 9
VARIABLE Q REGULATED IN CONNECTIONS: 1 2 3 4 5 6 7 8 9
LOOP IS NOT OF THE FEEDFORWARD TYPE

CONTROL LOOP: 4
SENSED VARIABLE: P
VARIABLE SENSED IN UNIT: 4
CONTROL VALVE UNIT NUMBER: 3
OTHER UNITS IN CONTROL SYSTEM ARE: 14 15 16 8
VARIABLE P REGULATED IN CONNECTIONS: 3 4 5 6 7 8 9
VARIABLE Q REGULATED IN CONNECTIONS: 1 2 3 4 5 6 7 8 9
LOOP IS NOT OF THE FEEDFORWARD TYPE

Since there is a single continuous stream, it can be concluded that there is no separate manipulated stream. Adjusting the control valve aperture will influence the flow and pressure in the system. Therefore all the control loops are classified as feedback, since they are dependent upon detecting a deviation in the regulated variable before a corrective action can be applied.

However, in this system care is needed in defining which units go to make up each control loop. Since signals are compared in the selectors, the following classification of units to control loops is necessary:

a) Units 5, 6, 7, 11, 12, 13 are all specified as part of control loops 1 and 2.
b) Units 3, 4, 8, 14, 15, 16 are all specified as forming a part of control loops 3 and 4.
Altering the control valve aperture will affect both the pressure and the flow variables in this stream. The effect on flow will be the same both upstream and downstream of the control valve; however the effect on pressure will be contradictory. For example, reducing the control valve aperture will decrease the flow both upstream and downstream; however the effect on the pressure variable will be to increase the pressure upstream, but to decrease it downstream.

Hence, the flow variable has been specified as being regulated in all the upstream and downstream connections, whereas the pressure has been specified as being regulated in the connections downstream of the control valve. Consequently, if the event of interest is a deviation in the pressure in stream 9, then failure modes for all four control loops have to be incorporated into the fault tree since they are all capable of correcting the deviation.

The major problem imposed by this system relates to the fact that there are no master or slave loops; a failure in any loop is correctable by another loop. Hence, the sequential application of control loop operators is inappropriate in this case. What is required is a fundamental change in the methodology for applying the control loop operators.

The presentation of the solution to this problem is delayed until Chapter 10, as some features to be discussed in subsequent chapters have a bearing on this discussion. This system has been introduced here to highlight the generality of the rules for the decomposition of control loops.

### 6.7 AUTOMATIC DETECTION OF CONTROL LOOPS

Chapter 5 on Divider-Header Combinations has highlighted a technique for detecting loops in the information flow structure. By maintaining a list of control valves and sensors in the model library, it would be possible to automatically identify the control loops in the system based on the configuration data supplied to the MASTER program. This would be done by tracing the signal flow from the sensors to the control valves. A prototype of an algorithm to achieve this was developed which successfully identified the control loops in a number of test cases. Being able to do this for all systems would free the user from having to specify the following items of data:

- control valve unit number
- sensor unit number
- other units in the control loop

Having achieved this the next stage would be to program the rules given in this chapter, to identify the regulated and manipulated streams, based on information contained in the failure models. The principle of this technique is illustrated by reference to Figure 6.7 again.

The following steps could be followed to derive the necessary information:

a) by consulting the list of control valves and sensors, the algorithm would trace the path from sensor Unit 4 to control valve Unit 9 and thus be able to define the physical components making up the control loop.

b) by analysing the failure model for sensor Unit 4, it can be ascertained that the signal feeding to the control valve is dependent upon the temperature variable.
c) by tracing along the process stream containing the sensor in both directions, the control valve will not be found. Hence, it can be concluded that the control loop has a separate manipulated stream.

d) consequently by tracing along the stream containing the control valve it will be noted that the heat exchanger is common to both the sensed and manipulated streams. Hence, this is the point of interaction. By analysing the failure model it will be noted that the manipulated stream affects the temperature in the sensed stream. Hence, this will be noted as the control loop regulated variable.

The above were only initial ideas for an algorithm to automatically specify the control loops in the configuration. Insufficient time was available on this project to develop these any further. However, they are a basis for further work which should be undertaken to facilitate the use of FAULTFINDER.

6.8 CONCLUSION

This chapter has highlighted a number of rules to facilitate the decomposition of control loops. These rules should not only assist the user in the manual decomposition of control loops but will also enable the development of an automatic algorithm. The algorithm would be extensible to the case of trip systems which are the subject of the next chapter.

A detailed discussion of some aspects relating to the application of the control loop operators by the synthesis algorithm has been deferred, as some of the techniques used to arrive at a solution are to be described in later chapters.
7.0 RULES FOR HANDLING TRIP SYSTEMS

7.1 INTRODUCTION

Trip systems are an added layer of safety, generally in addition to control loops. They are called upon to act only as a last resort to either provide a flow path where none existed before, or to shut off the flow path where flow normally exists. Their action is dependent upon monitoring one or more process variables and then activating the trip valve if the value of the variable should exceed its threshold. In conjunction with control loops the trip system should only activate under circumstances where the control loop is unable to regulate the variable within the required limits.

Trip systems are an important feature of fault trees since they are designed to prevent the propagation of faults. They can fail in two basic modes:

Operational Failure - trip valve activates when there is no demand from the process

Functional Failure - trip valve fails to activate when there is a demand from the process

Trip loops, in common with the overall methodology, are modelled by decomposition at the unit level, i.e. sensor, trip switch and trip valve. However, in an analogous manner to the control loops this component-based approach cannot fully model the failure modes associated with trip systems. Hence, at the decomposition stage it is necessary to provide additional information which will treat the trip loop as a special sub-system and for which special treatment will be afforded by the fault tree synthesis algorithm.

7.2 BASIC APPROACH

As mentioned above, trip loops have two basic failure modes: trip loop operational failure (TL-OP-F) and trip loop functional failure (TL-FN-F). TL-OP-F requires no special treatment by the synthesis program and can be adequately modelled via the trip loop components. This is primarily because this failure mode simply represents an alternative failure path in the fault tree. This is illustrated by the following example:

![Diagram of a simple section of plant incorporating a trip valve](image)

Figure 7.1
Simple Section of Plant Incorporating a Trip Valve

If the trip valve is normally open and the event being developed is NO FLOW then the fault tree for this would be of the following form:
The TL-OP-F branch would trace the causes of spurious maloperation of trip loop components. It is linked to the rest of the tree by an OR gate. No special treatment is needed during fault tree synthesis since the trip system is not designed to protect against the event currently being propagated. This situation should be compared with TL-FN-F.

In the case of TL-FN-F, for a fault to propagate there has to be a deviation in the process variable AND a failure in the trip loop components preventing its activation. Diagnosing TL-FN-F does require special treatment in the methodology. The basic approach as outlined by Kelly [Ref 7-1], has been adopted and can be summarised thus:

**Step 1**

Synthesise the fault tree as normal, noting any events which propagate through the trip valves. These events will later be considered as possible candidates for ANDing with TL-FN-F.

**Step 2**

Synthesise a “SHOULD ACTIVATE (SHAC)” tree for each trip loop. This tree will propagate from the trip valve, through the trip switch and sensor, to the process stream. This tree will define the range of process conditions under which the trip system is designed to operate.

**Step 3**

Synthesise a sub-tree for the TL-FN-F branch. The causes of this will lie in the trip loop components and are failures which prevent the sub-system from activating, e.g. trip valve stuck.

**Step 4**

Compare the two trees synthesised under Steps 1 and 2. Any events which the system can detect and prevent are then ANDed with the TL-FN-F sub-tree.

The biggest problem arises in deciding what events the trip system can prevent. At first sight it might appear that the trip system should be able to protect against all the events which cause the trip to activate. However, later sections will show that this step is
complicated by a number of factors. These will be highlighted and the steps taken to achieve a generic solution to the problem will be outlined.

### 7.3 TYPES OF TRIP SYSTEMS

The FAULTFINDER MK2 methodology differentiates between three types of trip system. They can be classified according to the following categorisation:

<table>
<thead>
<tr>
<th>Trip System</th>
<th>1) Closed Valve</th>
<th>2) Open Valve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a) Acts on Sensed Variable</td>
<td>b) Acts on Other Than Sensed Variable</td>
</tr>
</tbody>
</table>

The distinction between an open valve trip system (OVT) and a closed valve trip system (CVT) is quite straightforward. In an OVT the valve is normally open and closes on demand; for a CVT the converse is true. The distinction between the two classes of OVT is more subtle and essentially differentiates between feedback and feedforward OVT systems.

An example of a CVT is contained in the following schematic representation:

![Schematic Representation of a Closed Valve Trip System](image)

In this a kick-back leg is provided from the outlet of the pump back to the tank. Upon detecting a low flow, the trip valve is activated and opens to provide an alternative flow path, thus preventing the pump overheating.

Figure 7.4 contains an example of a Type 2a trip system.

Normally the control valve regulates the downstream pressure. If the deviation in this variable should exceed the capacity of the control system, then the trip system is called upon.

The trip system senses the pressure and the valve is shut upon the detection of the pressure reaching its threshold value. This prevents the propagation of high pressure further downstream. It is thus a feedback trip system.
An example of a Type 2b trip system is included in the following figure:

Upon sensing the loss of coolant in one stream, the trip system acts to cut off the flow of nitric acid in the other stream. It is thus a feedforward trip system.

The classification of the trip systems according to this categorisation plays a significant role in deciding what events any particular system can protect against. The criteria developed for handling these different systems are presented below.
7.4 OPEN VALVE FEEDBACK TRIP SYSTEMS

7.4.1 INTRODUCTION

Consider the following section of plant, adapted from Figure 7.4:

The hazardous top event of interest is a HIGH PRESSURE at the outlet of the pressure sensor. The fault tree for this event would be of the form given in Figure 7.7. This figure gives the status of the tree before the diagnosis and addition of TL-FN-F, and corresponds to the Step 1 tree as outlined in Section 7.2 above.
The SHOULD ACTIVATE tree for this system would start:

The rest of this tree would be the same as the tree given in Figure 7.7. Hence, it can be seen that all the events which cause the hazardous top event would also cause the trip to activate. However, even if the trip is activated, it cannot protect against all these events.

Consider the event LK-HP-EN in Pipe_2; even if the trip valve is activated it will not be able to shut off this source of high pressure. Although for this system this is a fairly trivial fault and arises because the pipe model was created in a context-independent manner, there are instances when this type of situation is highly significant. It illustrates an important principle in the diagnosis of the functional failure of trip systems and leads to the following rule.

7.4.2 BASIC RULE (RULE_1)

For a feedback trip system the only events it can protect against are the causes of the deviation which propagate through the trip valve. Hence, only the causes of the event marked with an * in Figure 7.7 will fall into the protection domain of the trip system.

7.4.3 REFINEMENT OF THE BASIC RULE

This rule is actually an over-simplification, since the direction of fault propagation and the variable concerned, are also important considerations. FAULTFINDER has the ability for two-way fault propagation; the trip valve can thus either be encountered via its outlet or inlet ports. In order to tackle the difficulties presented by this, the functionality of this type of trip system was investigated further and the following two conclusions reached.

Firstly, it was concluded that of the variables contained within the FAULTFINDER
methodology, a trip system of this nature could only directly counter deviations in the following four variables:

- flow
- pressure
- composition
- temperature

Secondly, since the activation of this type of trip system will totally cut off the flow, it is invariably going to act on a HIGH deviation in one of the above variables.

It should be noted that, whilst a feedback trip system can protect against deviations in vessel variables, e.g. level, the variable actually propagating across the trip valve will be the flow variable.

Using the above criteria, the basic rule can be refined to handle the complete set of circumstances which are likely to arise.

In order to do this, consider the simple schematic given below:

The first refinement of Rule_1 can be summarised thus:

**RULE_1.1**

For a feedback open valve trip system, if the trip valve is encountered via its OUTLET port during fault propagation, then the trip system can protect against the causes of the event which propagates through the trip valve, provided that these events would also cause the trip to activate.

Supposing now that the direction of fault propagation had been in the other direction as depicted in Figure 7.10, overleaf.
This situation is more complicated and so it is necessary to consider the deviation in each of the variables in turn.

a) \( \text{P HI} \)

The tree for this event would be of the following form:

This gives rise to the following refinement of Rule_1.

**RULE_1.2**

The trip system can only protect against the events downstream of the valve giving rise to reverse flow. Hence, the only branch that the trip system can protect against in this case, is the causes of R3 REV; this represents a build-up of pressure due to flow reversal through the valve, i.e. reverse relief.

The system cannot protect against the other events since:

- the causes of PI HI are all on the same side of the trip valve as the top event.
- the causes of R2 LO/NONE downstream of the valve will be blockage-type faults, i.e. giving rise to low or even no relief. The activation of the trip valve will probably aggravate the situation.

b) T HI/X HI

The trees for these two events would be similar and of the following form:

![Fault Tree for High Temperature](image1)

![Fault Tree for High Composition](image2)

In each case the event can either be caused by a deviation in the variable upstream or by a deviation in the variable downstream being transported by reverse flow. Based upon reasoning similar to that for R2 REV above, the following refinement of Rule_1 is needed.
RULE_1.3

In this case the trip system can only protect against the causes of G2 REV.

Generally in the methodology it has been assumed that for deviations in temperature and composition to propagate, the state SOME FLOW is taken as a normal event. Shutting the trip valve will certainly prevent flow, and this may suggest that the causes of T1 HI and X1 HI could be prevented from propagating to location 2 in Figure 7.11. However, if the deviation occurs due to a cause in close proximity of location 2, then it could manifest itself by other means, e.g. internal molecular motion of the fluid without any requirement for flow. Hence, the conservative approach has been taken that this trip system cannot protect against the causes of T1 HI or X1 HI.

c) Q HI

The situation of flow is more complicated than for the other three variables. The start of the tree for high flow at location 2 in Figure 7.10 would be of the following form:

![Fault Tree for High Flow](image)

The trip system will be able to protect against the causes of the event G2 HI which lie downstream of the trip valve. These will essentially be leakage-type faults giving rise to an increased pressure gradient for flow.

However, in this case the trip system can also protect against the causes of Q1 HI. This is because for a high flow to exist, there must both be a source and a sink. The branch Q1 HI will trace the causes of the source of high flow upstream; there is an inherent assumption that there will be a sink for flow further downstream.

In the methodology it is assumed that if high flow exists then there must be a flow path. This greatly simplifies the modelling as each deviation in the flow does not have to be ANDed with the existence of a corresponding flow path. This has proved a satisfactory approach in all other areas of the methodology. The diagnosis of functional failure in open valve trip systems is one instance where it would be necessary to have this AND gate. The presence of this AND gate in the pipe model would modify the tree given above in the way depicted by Figure 7.15.

In this instance the G2 SOME branch of Figure 7.15 would cross the trip valve. The causes of Q1 HI which would cause the trip system to activate would therefore be correctly diagnosed since they occur under the same AND gate as G2 SOME.
However, this approach was not favoured as it would have made the modelling unnecessarily complicated and cumbersome. Although this is a shortcoming in the methodology, it is only applicable to this fairly narrow field. The following refinement to Rule 1 was thus developed for circumstances where the trip valve is encountered via the inlet port.

RULE 1.4

Owing to the requirement of a flow path for these deviations to exist, it can be concluded that this type of trip system will be able to protect against both the causes of the G HI branch which lie further downstream of the trip valve, and also the causes of Q HI branch which are upstream of the trip valve.

The problem thus arises in being able to flag the Q1 HI branch since it will never propagate through the trip valve. This was solved by carrying out backtracking up the tree already synthesised, when the G2 HI branch was flagged as having encountered the trip valve via its inlet port. In order to do this the search procedure had to satisfy two goals:

- there must be two causes of a deviation in the flow, one at an inlet port and the other at an outlet port. This criterion will identify the location where the two-way fault propagation had started.

- both these deviations in the flow must be HIGH.

Applying these criteria to the fault tree of Figure 7.14 means that when the G2 HI branch is flagged, backtracking would be done to the previous level, i.e. Q2 HI, the top event in this case. It will be noted that this event has two causes and that:

- both are deviations in the flow
- one deviation is at an inlet port of the pipe and the other at the outlet port
- both deviations are high

Hence, Q1 HI is the required branch, which this trip system can protect against. This was an extremely trivial example as there were only two levels to the tree. A more complicated, though somewhat contrived example used to test this approach is presented below:

Supposing that the event of interest is HIGH FLOW at location 2. The tree for this event would be of the nature given in Figure 7.17.

In this case the event propagating through the trip valve, via the inlet port, is G9 HI. In accordance with the criteria outlined above, it is necessary to backtrack up the tree to be able to flag Q1 HI as being preventable by the trip system. One divider and one header model have been included in this configuration to show the generality of the backtracking algorithm. Although these two units give rise to the events G2 HI and G6 HI, the causes of which partially fulfil the criteria, it is only the causes of Q2 HI which are fully compatible. Hence, the event G1 HI will be suitably flagged as a potential candidate for ANDing with TL-FN-F.

### 7.5 OPEN VALVE FEEDFORWARD TRIP SYSTEMS

An example of such a trip system has already been presented in Figure 7.5. The trip valve is located in a flow stream physically separate from the sensed stream. In order for the trip to be effective there must be an interaction between the sensed and manipulated streams. In the current example this interaction occurs in the heat exchanger model in the form of heat transfer.

The problems presented in the feedback trip systems relating to the ability of the system to be able to compensate for a deviation in the same stream are not inherent in this case.
Hence, a much simpler diagnostic criteria can be applied. This is summarised in the following rule.

**RULE_2**

Any events which can activate the trip system and which also give rise to the top event should be ANDed with the functional failure of the appropriate trip system. Unlike the feedback systems, the domain of protection of the trip system is not restricted to events which propagate through the trip valve.
7.6 CLOSED VALVE TRIP (CVT) SYSTEMS

A modified version of the closed valve trip system given in Figure 7.3 is presented below:

The purpose of this type of trip system is to provide an alternative or new flow path where none existed before. The divider unit is therefore an integral part of this type of trip system. In order to function correctly leg 3 of the divider unit must provide adequate flow/relief upon demand. There is a special divider model in the model library which must be used. In this the appropriate deviations in legs 1 and 2 are ANDed with insufficient flow/relief down the no flow leg. Functional failure of the trip system can thus be caused either by a failure in any of the trip loop components, thereby preventing the activation of the trip valve or by there being LO/NONE/REV flow/relief down leg 3.

The following rule applies regarding diagnosis of those events that the trip system can protect against:

RULE_3

The causes of the deviations which are ANDed with failures down the no flow leg in the divider model (i.e. LO/NONE/REV flow/relief) should be compared with the SHOULD ACTIVATE (SHAC) tree to see if they should be ANDed with TL-FN-F.

7.7 INCLUSION OF TL-FN-F

The criteria for identifying the events which may require ANDing with trip loop functional failure have been described in the various rules above. The causes of these events are compared with events in the SHOULD ACTIVATE (SHAC) tree. Any events in the main tree which are common to the SHAC tree are then ANDed with trip loop functional failure.
This is illustrated by the following schematics:

Since event B is common to both the Main and SHAC trees, the event in the Main tree is ANDed with the event TL-FN-F. In order to accommodate this, an intermediate event denoted by X is incorporated into the tree.

7.8 RATIONALISATION OF TL-FN-F BRANCHES

A simple comparison of events between the SHAC and main trees, and the ANDing of common events in the main tree with TL-FN-F, leads to an opaque structure of the tree. Two rules are used to rationalise the location of these TL-FN-F branches.
RULE_4

If all the events under an OR gate are ANDed with TL-FN-F then the output of this gate should be ANDed with TL-FN-F instead.

This is illustrated below:

RULE_5

If any event under an AND gate is ANDed with TL-FN-F, then the output event should be ANDed with TL-FN-F instead.
It should be noted that both these actions are purely cosmetic and improve the transparency of the fault tree. They do not alter the logic in any way.

7.9 SEQUENTIAL OPERATIONS

The FAULTFINDER methodology has the ability to handle sequential operations. This is in connection with situations where the unit models comprising the plant can change. Under such circumstances the plant configuration at each step in the sequence needs to be defined. Examples of configurational changes are:
- valve changing from open to closed
- pump being switched on/off

Since the failure models for an open/closed valve are different, it is necessary to specify which model should be used at each step in the sequence. Additionally, the user can specify a different top event model for each step in the sequence.

This section outlines the solution to a shortcoming in the FAULTFINDER MK1 methodology which meant that it was unable to handle trip loop failures for situations where there were sequential operations.

For each step in the sequence, FAULTFINDER synthesises a separate sub-tree. The overall structure of the tree is as follows:

![Diagram of the sequential operations fault tree]

Figure 7.22
General Structure of Sequential Operations Fault Tree
The above fault tree is shown for three steps only; a maximum of twenty steps are allowed. The top event by default is termed SEQ-ABRT for sequential operations. There are essentially two branches to this. SEQ-F-AT contains the sub-tree for that step; SEQ-F-AF traces down to the next step in the sequence.

As has already been mentioned, the fault tree is initially synthesised without including the TL-FN-F branches. During the initial synthesis phase only the events which might require ANDing with TL-FN-F are noted. When this stage is complete, then the SHAC tree is synthesised to determine which events cause the trip system to activate.

There was a shortcoming in the MK1 methodology, in that the SHAC algorithm did not consider each step of the sequence in isolation and so ended up synthesising the incorrect SHAC trees; it is necessary to synthesise a SHAC tree for each step in the sequence reflecting the plant state represented by the failure models applicable to that step.

This algorithm was thus extended to treat events on a step-by-step basis.

7.10 COMBINED CONTROL AND TRIP LOOPS

A modified section of the plant relating to a combined control/trip loop given in Figure 7.4, is presented below:

A combined control/trip loop is one which has shared components between the two sub-systems. This would normally be the control/trip valve, but can include additional items, e.g. the vent valve in the above figure.

The problem arises because the control/trip valve is common to both the control and the trip system. For high pressure/flow to propagate through the system both the control and the trip loops have to fail.
Since the control loop is the primary protective system, the control loop algorithm imposes the tree structure shown in Figure 7.24:

![Diagram](image)

Figure 7.24
Application of the Control Loop Template

The program then recognises the secondary protective system and includes the functional failure of the trip system as shown in Figure 7.25:
Figure 7.25
Application of Control Loop Template and Trip Loop Functional Failure
The three failure states used for the control/trip valve are:

a) CV-STK, i.e. control valve fails in its normal position
b) CV-F-HA, i.e. control valve fails giving a high aperture
c) CV-F-LA, i.e. control valve fails giving a low aperture

The control loop algorithm needs all three states in the control valve model in order to correctly diagnose the control loop failure branches.

For the trip system to function the valve must close to prevent any flow. Therefore any of the above valve states is a valid cause of trip loop functional failure. However, the tree of Figure 7.25 is incorrect, since mutually exclusive failure states of the valve are being ANDed together. Consequently the program has been modified to check for common components between control and trip loops and to delete any events from the trip loop functional failure branch which are inconsistent with the control loop failure branches. The outcome of this action is highlighted in Figure 7.26.

Note that the trip loop functional failure branch is ANDed separately with each of the control loop failure branches and that the valve failure contained in it is consistent with that in the control loop failure branch.

7.11 CONCLUSION

This chapter has highlighted a number of rules to diagnose trip loop functional failure and to enhance the appearance of these branches in the final fault tree. In particular the shortcoming in the MK1 methodology with respect to combined control/trip loops and sequential operations has been addressed.

Chapter 10 of this thesis presents some worked examples including trip systems in the configuration.
Figure 7.26
Rationalisation of Control Loop and Trip Loop Failure Branches
8.0  MODELLING UTILITY FAILURES

8.1  INTRODUCTION

Faults in utility supplies are important because they are a potential source of common mode failures. Consider the section of plant given in Figure 8.1 below:

![Figure 8.1](image)

The activation of the trip switch is dependent upon the sensors detecting a deviation in the process stream. The reliability of the system is improved by duplicating the sensors; for the system to fail both the sensors must fail.

However, if both the units are connected to a common utility supply, then the failure of this is sufficient to fail the system. This is thus a common mode failure.

8.2  MODELLING

For the above system the utility supply failures could be modelled within the sensor unit models. However, this approach would be inadequate to alert the conventional fault tree analysis programs to the existence of the common mode failure and would result in the generation of incorrect cutsets.

If the utility supply failure were modelled within the sensor unit, then FAULTFINDER would incorporate a branch in the fault tree of the nature given in Figure 8.2, for the system given in Figure 8.1.

The Fault Tree Analysis program would include the following second order cutset for this sub-tree:

Utility Supply Failure Sensor 1 AND Utility Supply Failure Sensor 2

The only way to bring out the common mode nature of the fault is to be able to extend the fault propagation into the common mode unit. Hence, it is necessary to represent the configuration of Figure 8.1 as depicted in Figure 8.3.
This configuration would produce a fault tree of the format given in Figure 8.4. In this case the Fault Tree Analysis program would yield a single first order cutset:

Utility Supply Failure

in addition to the second order cutset: Sensor 1 Failure and Sensor 2 Failure.
8.3 DEFICIENCY IN THE MK1 SOLUTION

A basic limitation of the above technique was its inability to handle partial utility failures. In this context partial utility failure refers to failure of the utility to only some of the components in the configuration; it does not refer to the situation where insufficient utility is available to any particular unit, e.g. drop in pressure of the instrument air supply.

This situation is highlighted by the plant section given in Figure 8.5 below. The purpose of this plant section is to maintain a constant level in the tank by manipulating the flow into it.
In this instance the control valve, controller and sensor are dependent upon the same common air supply utility. Consider the case of a high flow of liquid into the tank. One cause of this would be spontaneous failure of the control loop resulting in higher than normal control valve aperture.

The control valve is of the air-to-open type. Using the models and synthesis algorithm of the MK1 methodology, a tree of the following form would be produced:

![Fault Tree Diagram](image)

Figure 8.6  
Fault Tree for Control Loop Spontaneous Failure of Plant Section of Figure 8.5

If the action of the control loop is studied in detail, it becomes apparent that there is a branch missing from the above tree.

The controller is reverse acting. It will emit a high signal to the control valve (causing an increase in its aperture) not only if it receives a low signal from the sensor, but also if it receives no signal. This effect could not be modelled using the MK1 methodology, as it would have introduced an inconsistency into the tree. The following figure shows the effect of including this event in the models:
As can be seen from the above figure, the sensor will not emit a signal either if it fails in a position indicative of no level in the tank, or if there is a loss of utility, since the sensor cannot function without a correct supply of utility.

The tree of Figure 8.7 is incorrect since the utility supply failure can never give rise to the top event; a utility supply failure would result in the control valve failing shut, resulting in no flow into the tank since it is of the air-to-open type.
8.4 PROPOSED SOLUTIONS

Two solutions were considered to overcome this problem. Each of these is described below.

8.4.1 SOLUTION 1

Consider again the tank level control system of Figure 8.5. It could be argued that since the control valve is dependent upon a supply of utility for normal functioning, the model should have this knowledge incorporated into it. Hence, the minitree for CL-F-HA in the control valve model should be of the following format:

![Diagram](Figure 8.8 Modified Minitree for CL-F-HA for Control Valve)

This should be compared to the normal minitree:

![Diagram](Figure 8.9 Normal Minitree for CL-F-HA for Control Valve)
The difference between the two minitrees is that in the former the requirement for some utility supply has been explicitly stated. This branch would eventually be traced to the Utility Supply unit itself. The standard FAULTFINDER methodology would suggest that the utility supply model should contain an event statement to the effect:

\[
\text{S NORMAL: UTL-SOME}
\]

i.e. the utility supply unit normally delivers some utility. Hence, in effect since a normal event should not be included in a failure tree, then the branch for some utility would be deleted from the final tree.

However, this could be circumvented by incorporating a pseudo-fault event into the utility supply unit thus:

\[
\text{F UTL-OK: UTL-SOME}
\]

Hence, the tree of Figure 8.7 given above would be changed to that given in Figure 8.10. It should be noted that the tree given below contains only those events pertinent to the current discussion.

---

**Figure 8.10**
Modified Form of Tree of Figure 8.7 Incorporating Some Utility State
This would thus prevent the generation of the first order cutset AIR SUPPLY UTL-FAIL, since it is ANDed with AIR SUPPLY UTL-OK. A post-cutset generation algorithm could then be written to reject any cutsets containing these mutually exclusive events for the same unit.

This approach was, however, rejected, since it unnecessarily complicated the modelling procedure. Additionally the inclusion of NORMAL events/states into a FAULT tree was considered to be both undesirable and unacceptable, in an attempt to overcome a limitation in the methodology. The alternative solution described below was chosen instead.

8.4.2 SOLUTION 2

This is based on the MASTER program being able to detect the utility supply units in the configuration. All the units which are linked to these utility units are then noted.

Hence, considering the plant representation of Figure 8.5, the MASTER program would note that there is one utility air supply unit and that it feeds the control valve, controller and sensor.

The algorithm for this is based on the following criteria:

- utility units are differentiated from the others because they have multiple outputs via utility ports. Additionally this is the only type of port associated with this unit.

- the units connected to these utility supply units are established by tracing back from the unit via its utility port until a utility supply unit is found.

This is based entirely on the information which is already present in the failure model, and that which has been supplied by the user, by virtue of the way that the various units are connected together, during the decomposition phase.

The fault tree synthesis algorithm can then utilise this information to generate logically consistent trees. This is best illustrated by reference to a worked example.

8.5 WORKED EXAMPLE

8.5.1 INTRODUCTION

The plant section of Figure 8.11a will be used; the decomposition diagram for this system is given in Figure 8.11b. The system consists of an effluent holding tank, which feeds into a river. There are two trip systems attached to this to prevent the discharge of contaminated effluent into the river. One trip system requires the manual intervention of the operator to shut the hand valve at the outlet of the tank. In this instance an alarm signal is generated upon detection of a high concentration in the tank.

The other trip system is automatic and would only operate if the operator fails to take corrective action. It should be noted that the trip valve in this is fed via a three-way solenoid (vent) valve. The trip valve is of the fail-safe mechanism because:

- it is of the air-to-open-type. If the air supply fails, then the trip valve will close.
Figure 8.11a
Flow Diagram for Effluent Discharge System
Figure 8.11b 
Decomposition Diagram for Effluent Discharge System
the solenoid is of the de-energise to vent type. Hence, if the power supply fails, then the vent valve will de-energise, causing the air to vent and resulting in the closure of the trip valve.

The hazardous top event of interest is the release of toxic waste into the river. The fault tree resulting from the MK 1 methodology is presented in Figure 8.12.

It should be noted that there are three main branches to this tree. These are:

Branch A: Failures related to Trip System 1 (automatic).
Branch B: Failures causing a high composition of pollutant.
Branch C: Failures related to Trip System 2 (manual).

It should be noted that these three branches are all ANDED together, since for the hazardous top event to occur there must be a high concentration of pollutant and both the trip systems must fail to act.

However, the detail of the fault tree is not quite correct. Consider Branch A in greater detail. This is incorrect, since failures of the sensor (Unit 5) are traced to the failure of the power supply (Unit 13). It has already been stated that since the solenoid valve is of the de-energise-to-vent type, a failure of the power supply will cause the trip valve to shut and thereby prevent the top event from materialising.

8.5.2 IMPLEMENTED SOLUTION

The solution implemented is really a fault tree rationalisation step, i.e. it is executed once the original fault tree has been synthesised. The fault tree produced by the synthesis algorithm of the MK 2 methodology is presented in Figure 8.13. A discussion of the resultant fault tree is presented below.

It should be noted that two main differences have occurred with respect to Branch A:

- the power supply failure associated with the sensor unit has been eliminated.
- the UTL-OK event has been incorporated into this branch.

The procedure for accomplishing this is outlined below.

The fault tree is searched from the top down, to locate those units which have been identified to be connected to power supply units. The first unit (in the propagation path) in Branch A which is connected to the power supply unit is the trip switch (Unit 14). It will be noted that although this unit is connected to the power supply unit, it does not have any faults associated with the power supply. Hence, it can be inferred that this unit expects a normal power supply and that no other units below this point in the fault tree may have faults associated with the power supply.

Although the power supply failure event has been deleted as a direct cause of the sensor failing, faults in the local power supply line to sensor are left in the tree. Whilst the actual faults are not strictly realistic, since the standard pipe model was used, they nevertheless illustrate the principal quite well.
TOXIC RELEASE

A

AND

B

AND

C

TL-FN-F
TRIP 1

TL-FN-F
TRIP 2

X6 HI

X1 HI

S18 NCHA

TV-FT-SH
UNIT 6

S12 NCHA

HV-FT-SH
UNIT 4

OP-FAIL
UNIT 12

SIG-CB
UNIT 15

S17 NCHA

SV-FT-OP
UNIT 15

S9 NCHA

S11 NCHA

AN-BROKE
UNIT 10

S15 NONE

W19 HI

S16 NCHA

SIG-CB
UNIT 14

TSW-DIS
UNIT 14

TSW-STK
UNIT 14

W10 HI

S14 NONE

SIG-CB
UNIT 9

TSW-DIS
UNIT 9

S8 NCHA

POW-LOSS
UNIT 13

POW-LOSS
UNIT 13

SEN-STK
UNIT 11

SEN-STK
UNIT 8

Figure 8.12
FAULTFINDER MK1 Fault Tree for System of Figure 8.11
Figure 8.13
FAULTFINDER MK2 Fault Tree for System of Figure 8.11
Additionally, the UTL-OK event has been included at the point where the trip switch was first encountered in this branch. This is necessary to ensure consistency with the events in the other trip loop functional failure branch. The need for this is exemplified by the following scenario.

Considering now Branch C, there is nothing to prevent the POW-LOSS Unit 13 event from being ANDed with the event TV-FT-SH Unit 6 in Branch A. However, it is not correct to AND POW-LOSS Unit 13 with any events below the point where the propagation path passes through the controller unit as this needs a normal power supply to function.

Although the tree of Figure 8.13 is logically consistent, the standard fault tree analysis packages will give rise to incorrect cutsets. In the absence of the UTL-OK event, cutsets of the following nature will be generated:

POW-LOSS Unit 13 AND TSW-STK Unit 14

The inclusion of the UTL-OK event will transform the above cutset into:

POW-LOSS Unit 13 AND TSW-STK Unit 14 AND UTL-OK Unit 13

Since this cutset contains two different failure modes for the same unit, then a post-cutset generation algorithm could be used to reject these erroneous cutsets.

Further discussion of this aspect will be delayed until Chapter 9, when fault tree analysis is discussed in greater detail.

8.6 CONCLUSION

This chapter has outlined a modelling convention for handling the common mode aspects of utility supply failures. A procedure has been outlined for detecting the utility supply units in the configuration and for ensuring the consistency of the trees with respect to faults related to these units. The problem of logically consistent trees generating incorrect cutsets has also been introduced, along with a solution to the problem. A detailed discussion of this latter point is delayed until the next chapter.
9.0 FAULT TREE ANALYSIS

9.1 INTRODUCTION

This thesis has concentrated mainly on the synthesis of fault trees. Fault tree analysis is a research topic in its own right and was not a requirement of this project. However, a means of evaluation was needed to assess the consistency of the fault trees produced by FAULTFINDER. The primary criterion used to achieve this was the generation of minimal cutsets. Since automatic fault tree analysis is more advanced than automatic fault tree synthesis, a number of programs are available for evaluating fault trees. Two such programs: PREP [Ref 9-1] and FTAP [Ref 9-2], were used for determining the minimal cutsets.

9.2 MINIMAL CUTSETS

Cutsets are the combinations of events which are sufficient to give rise to the top event of the fault tree. Consider the two schematic representations given in Figure 9.1 below:

![Figure 9.1 Schematic Fault Tree Representation](image-url)
Fault tree (A) would have the following two cutsets:

a) A  
b) C and D

e.g. if either A occurs or if both C and D occur then the top event will occur.

Fault tree (B) would also have two cutsets:

a) A  
b) C and A

However, the second cutset would be deleted since it is non-minimal. The first cutset says that A alone is sufficient to cause the top event. Hence, the requirement for C to co-exist with A is not necessary.

The order of the cutset is also important. A first order cutset contains a single event, e.g. A; a second order cutset contains two events, e.g. C and D, and so on.

9.3 INTERFACE WITH THE ANALYSIS PROGRAMS

Data files are used for creating the input to FTAP and PREP and also for storing the output from these. Both programs expect data input in a fixed format. The nature of the input created by the FAULTFINDER synthesis algorithm is summarised below.

9.3.1 FTAP FORMAT

The synthesis program creates an input file for this program conforming to the following format:

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 21</th>
<th>Column 31</th>
<th>Column 41</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>+</td>
<td>BE1</td>
<td>G2</td>
</tr>
<tr>
<td>G2</td>
<td>*</td>
<td>G6</td>
<td>-BE2</td>
</tr>
<tr>
<td>G6</td>
<td>+</td>
<td>BE3</td>
<td>BE4</td>
</tr>
</tbody>
</table>

Table 9.1
FTAP Format

Each record contains the following information:

a) gate node name, e.g. G1  
b) logic indicator (+ - OR, * - AND, k - an integer in k-out-of-n redundancy)  
c) immediate subnodes, e.g. BE1 and G2.

The following conventions are adhered to:

a) 8 fields per record with 10 columns per field  
b) each field is left-justified  
c) node names are restricted to 8 or less characters
d) a dash in columns 20, 30, 40, 50, 60 or 70 indicates event complementation, e.g. -BE2

e) if a gate has more than 6 causes, then these are continued on to the next record. In this case fields 1 and 2 are left empty.

Additionally, at the end of the file after defining the tree structure it is necessary to define various parameters to control the action of FTAP. At the very minimum the following three instructions are required:

a) ENDTREE
b) *XEQ
  c) ENDJOB

The first instruction indicates the end of the fault tree definition; the second instructs FTAP to commence the generation of cutsets; and the final stops further processing.

9.3.1.1 OPTIONS WITH FTAP

The following two options were used during the course of this project:

a) MAXSIZE N, this restricts the generation of cutsets to a maximum order of N.

b) PROCESS G1,G2... . By default FTAP generates all cutsets for the top event of the fault tree. By specifying gate nodes using this command, cutsets can be generated for sections of the tree. This feature is useful for the SEQUENTIAL OPERATIONS fault trees, where cutsets need to be evaluated separately for each SEQ-F-AT Step branch.

9.3.2 PREP FORMAT

PREP actually comprises two different programs: TRIAD and CUTSET. The fault tree synthesis program thus creates two files for input to these two programs and conforming to the specified format.

The input to TRIAD defines the fault tree structure and has a format similar to FTAP. Each record defines one level of the tree structure; the contents of each record are outlined below:

a) 1st field - output gate node
b) 2nd field - gate type, either AND or OR or k-n
c) 3rd field - number of transmissive inputs to this gate
d) 4th field - number of basic event inputs to this gate
e) Fields 5-9 - up to 5 inputs to the gate

If more than five inputs exist to any given gate, then these are continued on to the next line, in which case fields 1-4 would be left empty. It should also be noted that PREP does not allow event complementation.

The final record in this file contains the text END, signifying the end of the tree structure.

The TRIAD program processes this file to create a suitable representation of the fault tree for use by the CUTSET program.
The fault tree synthesis program also creates an input file to the CUTSET program. This file specifies the options to be used in determining the cutsets. Only the minimum necessary information for PREP to be able to generate cutsets was provided. The structure of this file is as follows:

a) 1st record - name of the configuration under study
b) 2nd record - * DATA, indicating start of options data
c) 3rd record - minimum and maximum order of cutset to be found
d) 4th record - END, indicating end of file

TRIAD and CUTSET need to be executed sequentially and a large number of intermediate files are generated. To facilitate this process a command procedure has been written, which automatically manages the files and executes the programs.

9.3.3 INTERFACE PROBLEMS

For both FTAP and PREP it was necessary to modify the programs to accept input using data files. A further problem was that both the fault tree analysis programs allowed a maximum of eight characters for the specification of events in the tree. However, FAULTFINDER requires up to twelve characters for defining these events.

In the case of PREP the program was further adapted to overcome this restriction. However, for FTAP the indices of the events, as used by the synthesis program, were used to define the tree structure. A post-FTAP procedure was implemented to subsequently convert the indices into actual event names. This latter approach was adopted, since other post-cutset determination manipulations were also needed.

Generally during the course of this project FTAP was used for generating the cutsets.

9.4 POST-CUTSET ANALYSIS ALGORITHMS

This section describes some features implemented to overcome conflicts between the FAULTFINDER fault trees and the limitation of the FTAP package.

9.4.1 MUTUALLY EXCLUSIVE EVENTS

9.4.1.1 INTRODUCTION

The problems inherent with this will be illustrated with reference to a system that was modelled during this project.

One application of FAULTFINDER was to generate rules for alarm analysis. The flow and decomposition diagrams for this system are given in Figures 9.2a and 9.2b, respectively. The purpose of the plant is to maintain a constant head of fluid in the open tank for an unspecified downstream unit. The level is controlled by manipulating the flow of liquid into the tank using the control valve (Unit 3), which in turn is directed by the reverse-acting controller (Unit 13). The controller receives its level signal from Unit 11, which is connected through a shared sampling port via the vessel port splitter to the tank.
The alarm logging system is represented by Unit 16, which monitors the tank level via the two level sensors and also the flow of the manipulated stream via the flow sensor. The purpose of the second level sensor is to provide redundancy in the system.

9.4.1.2 MODELLING AN ALARM COMBINATION

The following combination of events was modelled via the top event: a low level signal from Unit 15 and a high level signal from Unit 11. The start of this fault tree was as indicated in Figure 9.3 below:

![Fault Tree Diagram]

9.4.1.3 THE PROBLEM

The two branches S9 HI and S9 LO will contain mutually exclusive events; however, because of the way that the top event has been specified the fault tree itself is logically consistent. The fault tree synthesis algorithm cannot, and indeed should not, delete events from either branch. This is because the events do not occur immediately beneath the AND gate. Only the combination of events in the S9 HI and S9 LO branches is inconsistent; the combination of all other branches, e.g. SEN-F-LO UNIT 15 and S9 HI is valid.

Nevertheless as the fault tree stands, the standard fault tree analysis packages, FTAP and PREP, will not appreciate that some of the events in the fault tree are mutually exclusive and will consequently give rise to incorrect cutsets.

9.4.1.4 PROPOSED SOLUTION

One possibility to overcome this problem would be to re-organise the fault tree structure, to make the mutual exclusivity explicit. This is outlined in Figure 9.4.
However, this technique was not pursued as it was found not to be generally applicable. This type of tree rearrangement was possible in this case since the mutual exclusivity occurred at the top of the tree and was quite obvious. In more complicated examples the rearrangement of the tree would have proved quite messy and would have resulted in rather opaque trees.

The solution adopted was to utilise the ability of FTAP to handle event complementation. Essentially, event complementation entails the inclusion of those events which must not occur, for the output to be true. This is equivalent to the use of NOT logic in the tree. Hence, when writing the FTAP file for the above tree, it would automatically alert FTAP so as not to AND the causes of the S9 LO branch with those of the S9 HI branch. Although the fault tree itself is not modified, the FTAP file would be equivalent to the tree structure given in Figure 9.5.

It should be noted that any valid cutsets below the complemented event will also include this event. However, this was overcome by removing the complemented events from the final cutsets by a post-cutset generation algorithm. This is possible because the FAULTFINDER methodology does not itself utilise event complementation.

9.4.1.5 DETERMINATION OF MUTUALLY EXCLUSIVE EVENTS

The main problem arises in determining the mutually exclusive branches so that these can be flagged in the FTAP input file.

The outline algorithm for achieving this is as follows:
Figure 9.5
Removal of Mutual Exclusivity Using Event Complementation.

a) Search down the tree from the top event for AND gates.

b) If an AND gate is found then compare the events in each branch with those in all the other branches under this gate. Flag those events which are deemed to be mutually exclusive.

c) When writing the FTAP input file, consult the above flagged events and introduce event complementation at the appropriate points.

Two basic criteria are used for determining the mutually exclusive events:

i) analysis of variable deviations. Since a variable deviation defines the status of the process streams, then these can be used to determine the mutually exclusive plant states. Each variable has a discrete number of states within the FAULTFINDER methodology; consequently two different deviations of a variable at the same location represent mutually exclusive states. Hence, the variable deviations in the different branches are analysed and if they have the same variables and stream numbers, BUT different deviations, then they are flagged as mutually exclusive. This criterion would thus flag the S9 HI and the S9 LO branches as being mutually exclusive.

ii) analysis of special intermediate events. The above criteria would be adequate if the fault tree consisted entirely of variable deviation events. This is, however, not the case.
As far as mutual exclusivity is concerned, intermediate events are also of interest, particularly those which are at the head of the control loop templates. These events represent failure modes of the control loop. The different branches in any given template are by their very definition mutually exclusive, e.g. CL-STK and CL-F-HA.

This latter case will be highlighted by reference to the tank level control system of Figure 9.2 again. A different alarm condition is modelled in this instance: an indicated high flow into the tank and an indicated low level in the tank. Since the level in the tank is the control loop regulated variable, controlled by manipulating the flow into it, then these two events can only persist if the control loop has failed. In general the control loop failures associated with a high flow into the tank are contradictory to those for a low level in the tank, e.g. a high flow of the manipulated stream will be caused by CL-F-HA whereas a low level would be caused by CL-F-LA. The resultant fault tree for this system would be of the form given in Figure 9.6 below:

Figure 9.6
Start of Fault Tree for Second Alarm Combination
Once again unless special treatment is afforded, mutually exclusive failure branches of the fault tree will be ANDed together by the fault tree analysis packages, e.g. the CL-F-HA branch would be ANDed with both the CL-F-LA and CL-STK branches.

To overcome this problem, the control loop failure states which were considered to be mutually exclusive were defined and applied during the determination of mutually exclusive events for FTAP. The branches which may be ANDed together between the regulated and manipulated variable templates are summarised in Table 9.2 below:

<table>
<thead>
<tr>
<th>Regulated Template</th>
<th>Manipulated Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL-F-HA</td>
<td>CL-F-HA</td>
</tr>
<tr>
<td>CL-F-LA</td>
<td>CL-F-LA</td>
</tr>
<tr>
<td>Sensed Variable Branch + CL-STK</td>
<td>Manipulated Variable Branch + CL-STK</td>
</tr>
<tr>
<td>Overload Branch (CL-O-LD)</td>
<td>Sensed Variable Branch (CL-ACT)</td>
</tr>
</tbody>
</table>

Table 9.2
Branches Which May Be ANDed Together

Entries on the same row in the above table may be ANDed together.

The first two entries in Table 9.2 allow the ANDing together of the appropriate control loop spontaneous failures. The third entry allows the combination of the control loop latent failures. The fourth entry allows the ANDing together of failures in the process streams which may co-exist; the CL-O-LD branch will trace faults in the manipulated stream, e.g. no flow, whereas the CL-ACT branch will trace the causes of the level deviations in the tank. The variable deviation mutual exclusivity checks will be able to prevent the ANDing together of any inconsistent events in these branches.

9.4.2 COMBINATORIAL EXPLOSION

There may be instances when more than one control loop is controlling a process stream and there is no slave or master relationship. An example of this is the British Gas Pressure Reduction Installation (see Chapter 10 for details), where one control loop is able to correct for failures in another control loop. For a two control loop scheme the required fault tree form would be of the format given in Figure 9.7.

Essentially this says that the top event will occur only if both control loops fail (in case of control loop latent failure there must also be a deviation in the process stream). The FAULTFINDER methodology differentiates between three different failure modes for the control loop: CL-STK, CL-F-HA and CL-F-LA. In any given application only one of CL-F-HA and CL-F-LA will be meaningful. Hence, as in the above example the control loop template contains two of these three failure modes.

In general the number of combinations of these two failure modes is given by: \(2^N\), where N is the number of control loops. Hence, the above two control loop system has a total of four combinations for the control loop failure branches.

The decomposition of the British Gas Control System yielded five control loops which would have required thirty-two branches in the fault tree. This would clearly have made the fault tree structure rather opaque. To overcome this problem, the structure of the
Figure 9.7
Template for Multiple Control Loops (2 Loops)
Figure 9.8
Modified Form of Fault Tree Given in Figure 9.7
fault tree was simplified by the use of combined OR and AND logic as shown in Figure 9.8.

Comparing this with Figure 9.7, it can be seen that the location of the AND and OR gate has been exchanged. Doing this allows the number of branches to be reduced to the number of control loops. However, there is one problem with this approach. When the cutsets are now generated for the Spontaneous Failure Branch there will be the following four combinations, as opposed to the three indicated in Figure 9.7:

1) CL-F-HA 1  CL-F-HA 2
2) CL-F-HA 1  CL-STK 2
3) CL-STK 1  CL-F-HA 2
4) CL-STK 1  CL-STK 2

It can be seen that the additional combination is the last one. When the minimum cutsets for this combination are generated all those from the Latent Failure Branch in Figure 9.8 will be rejected since the presence of the Deviation in Sensed Variable Branch makes them non-minimal. Hence, the price paid for increased fault tree transparency is that there will always be one combination which is invalid (combination of CL-STK modes).

There is thus a need to flag this in the fault tree and to prevent the cutset packages yielding incorrect cutsets. To achieve this, the fault tree structure of Figure 9.9 was adopted.

In this case the spontaneous failure modes are all ANDED with a Dummy Basic Event: Unit 0. This can be distinguished from the other basic events since it has an invalid unit number (0) associated with it. Hence, when the cutsets are generated for this branch, CL-STK 1 AND CL-STK 2 from the previous example, will now in fact become CL-STK 1 AND CL-STK 2 AND DUMMY Unit 0. Hence, the package will not reject the cutsets from the Latent Failure branch on the grounds of non-minimality.

Of course, all the cutsets generated from the spontaneous failure branch will now contain the Dummy Unit 0 event in them. A post-cutset generation algorithm was therefore written to enable the 'normalisation' of these cutsets. Two main criteria were used to achieve this:

a) remove the event Dummy Unit 0 from all cutsets
b) remove any cutsets which are subsets of other cutsets.

An example of the former is:

CL-F-HA 1 AND CL-F-HA 2 AND Dummy Unit 0 becomes
CL-F-HA 1 AND CL-F-HA 2

An example of the latter is that

CL-STK 1 AND CL-STK 2 AND Dummy Unit 0 would become
CL-STK 1 AND CL-STK 2

but it is in fact rejected, since it is a subset of the cutset arising from the latent failure branch which will give a cutset of the form:

CL-STK 1 AND CL-STK 2 AND Process Variable Deviation
Figure 9.9
Modified Fault Tree Structure for Fault Tree of Figure 9.8
9.5 CONCLUSION

This chapter has described how FAULTFINDER was interfaced to two standard Fault Tree Analysis Programs. The problem of incorrect cutsets being produced for logically consistent trees by the mechanistic use of the Fault Tree Analysis Programs has been highlighted. Some techniques for generating correct cutsets, whilst retaining a transparent fault tree structure, have been outlined.
10.0 WORKED EXAMPLES

10.1 INTRODUCTION

This chapter presents a set of worked examples which were modelled during the course of this project. The first example gives a detailed description of each step to illustrate the fundamental fault tree construction algorithm. Subsequent examples will then only highlight the significant new features of that system.

10.2 LAPP-POWERS SYSTEM

The flow diagram, decomposition diagram and the input required by FAULTFINDER are given in Appendix B. This section therefore, illustrates the steps involved in synthesising the fault tree for the hazardous top event of a high temperature at the outlet of the temperature sensor, Unit 5 in the decomposition diagram.

The top event model for this is given below:

```
OVRTEMP
  OR
  DTROW1
    AND
    Q2OUT REV U1 IN HI
  DTROW2
    AND
    T2OUT HI Q2OUT SOME
```

Since this is being applied to the outlet of the temperature sensor model, it would be translated into the configuration-specific nomenclature as shown in Figure 10.2.

In order to facilitate the discussion each branch will be developed in turn. The development of the left hand, reverse flow effects, branch of Figure 10.2 will be described first.
10.2.1 REVERSE FLOW EFFECTS BRANCH

The causes of both Q5REV (Q2OUT REV) and U4HI (U11N HI) are to be found in the temperature sensor model. When these are added the tree will have the following format:

The causes of G5REV and U5HI lie in the unit downstream of the temperature sensor. From the configuration array, the algorithm will note that this unit is the dummy tail, Unit 6. At this stage both G5REV and U5HI will be flagged as diamond events as they have crossed the system boundary. They need further development; however this is not possible as the system configuration is undefined.

The event LK-LP-EN is a basic event and so does not require further development.
Therefore, the only event in this branch requiring further development is G4REV. The causes of this are given by the minitree for G11N REV for the temperature sensor unit. When this minitree is added, this section of the tree will become:

\[
\text{DTROW1} \\
\quad \text{AND} \\
\quad \text{Q5 REV} \\
\quad \quad \text{U4 HI} \\
\quad \quad \text{OR} \\
\quad \quad \text{U5 HI (diamond)} \\
\longleftarrow \text{G4 REV} \\
\quad \text{LK-LP-EN Unit 5} \\
\quad \text{G5 REV (diamond)} \\
\quad \text{OR} \\
\quad \text{Q4 REV} \\
\quad \text{LK-HP-EN Unit 5} \\
\quad \text{Q5 REV (deleted)}
\]

Figure 10.4
Development of the Reverse Flow Effects Branch

It should be noted that at this point Q5 REV will be deleted since it has already occurred in this branch. The event LK-HP-EN Unit 5 will also be deleted since it is a not allowed fault of Q5 REV (it should actually cause Q5 HI).

The only remaining event therefore is Q4REV; the causes of this will lie in the unit upstream of the temperature sensor. This unit will be determined by the methodology by looking at the configuration array. This unit will be the pipe (Unit 4).

By applying similar logic to that described above this branch will be expanded as shown in Figure 10.5.

It should be noted that the section of tree in Figure 10.5, represented by the dotted line will be deleted. The start event for TL-FN-F, and hence trip SHAC will be noted. TL-FN-F will be afforded special treatment later, see Section 10.2.7. Similarly the TL-FN-F events will be removed from any other events propagating through the trip valve.

There are no further events in this branch requiring development. Hence, returning to the sub-tree of Figure 10.2, the only events awaiting development are T5HI and Q5 SOME from the normal flow effects branch. The development of this branch will now be described.
10.2.2 NORMAL FLOW EFFECTS BRANCH

The development of the Q5 SOME event will be considered first. There are no new principles involved in developing this; Figure 10.6 gives the section of the fault tree related to this event.

However, the methodology will delete the whole of the sub-tree of Figure 10.6, since the SOME flow events at dummy heads and dummy tails are deleted as CERTAIN events. Deletion will continue up the tree until an AND gate is found. The effect of this is to delete events under a common OR gate, since if one event under an OR gate is certain to occur, then the inclusion of the other events is superfluous. (Note: the AND gate below Q2 SOME will have been converted to an OR gate due to the removal of the TL-FN-F event as described above).
The net effect of the above is to reduce the right hand, normal flow effects branch of Figure 10.2 to the following:
The development of the T5 HI branch will now be considered. The methodology will note that the T variable at this location has been specified as falling within the domain of protection of the control loop. Hence, the control loop operator for the deviation in the regulated variable of a feedback control loop will be applied thus:

```
DTROW2
  OR
    T5 HI
      OR
        C(DUMMY)
          Loop 1
        E(DUMMY)
          Loop 1
        F(DUMMY)
          Loop 1
        OR
          Spontaneous Failures
          D(DUMMY)
            Loop 1
          AND
            CL-STK
              Loop 1
            Deviation In Sensed Variable
```

Figure 10.8
Application of the Control Loop Operator
At this point there are a number of unknowns in the above template. These are:

- it is not known whether CL-F-HA or CL-F-LA would give rise to the event T5 HI.
- which deviation in the sensed variable will give rise to the event T5 HI.
- whether any events would overload the control loop.

The key to filling in these branches is:

a) trace the causes of the current event, T5 HI, adding these causes to the C(DUMMY) branch.

b) this branch will eventually trace through the control valve. The control valve model will give the necessary information on whether CL-F-LA or CL-F-HA is the cause of the event T5 HI. This event will then be moved to the spontaneous failure branch and be developed in the normal way. The control loop latent failure branch CL-STK will also be found from the control valve model.

c) the control loop sensed variable deviation, which gives rise to the event T5 HI will be found by noting the event at the inlet of the sensor, which would give rise to the spontaneous failure branch. This is illustrated by the following schematic representation:

![Figure 10.9](image)

If the event being developed is CL-F-HA and the controller is reverse-acting then the deviation at the inlet of the controller causing this is a low signal, which will eventually be traced to a low deviation at the inlet of the sensor.

d) the methodology assumes that the only events which can overload the control loop are the None and Reverse deviations in the flow of the manipulated stream. Hence, whenever the propagation path is along a manipulated stream, then if the variable is flow G/Q, and the deviation is NONE/REV, then the event will be transferred to the overload branch and be developed there in the normal manner.

Hence, returning to the event T5 HI, it is necessary to develop this and add the causes of this to the C(DUMMY) branch. The development of the tree up to the heat exchanger is quite straightforward and presents no new problems. This tree is given in the following figure:
The events EXT-HEAT, FOULING and LK-LP-EN are all basic events and require no further development. Of the other events: T2 HI, T9 HI and G2 HI can be developed fairly readily and pose no new problems. The resulting tree when these three events are fully developed is given in the following figure:

![Diagram](image)

**Figure 10.10**
Start of the C(DUMMY) Branch
The development of E(DUMMY) Unit 3 is more complicated. When the minitree for this event is extracted from the heat exchanger model and added to the tree above, it will be expanded thus:
At this point the methodology will note that stream 9 has been specified as having the flow manipulated in it, and that both G9 REV and G9 NONE are events which will overload the control loop. Hence, both these events will be moved to the control loop overload branch and developed there. G9 LO will therefore be the only event requiring further development. The causes of this will be developed thus:
At this point the propagation chain has reached the control valve and the causes of the current event are related to the spontaneous failure and latent failure of the control loop. The events CL-F-LA and CL-STK will be moved to the spontaneous failure and latent failure branches, respectively, and developed there. Therefore the only event requiring further development, in Figure 10.13, is G8 LO. This can be developed in the normal way and will yield the following tree.

At this stage the C(DUMMY) branch is fully developed. The start events of the spontaneous and latent failure branches have been found, and two events have been identified for the overload branch. It is next necessary to develop these branches. Firstly, the spontaneous failure branch will be developed.
10.2.3 SPONTANEOUS FAILURE BRANCH

The start event for this is CL-F-LA and the causes of this are to be found in the control valve model. The sub-tree for this event will be developed thus:

![Spontaneous Failures Diagram]

The event T4 HI has traced the spontaneous failure branch to the inlet connection of the sensor. Hence this is the deviation, HI, which is required for the sensed variable deviation branch. Furthermore the event T4 HI will be removed from this tree; the control loop spontaneous failure branch contains failures in the control loop components only; the event T4 HI has traced causes into the process stream.

Next the latent failure branch will be developed.

10.2.4 LATENT FAILURE BRANCH

In a similar manner to the spontaneous failure branch, the latent failure branch will be developed. The sub-tree for this event is given in the following figure:
Next the two events relating to control loop overload will be developed.

10.2.5 CONTROL LOOP OVERLOAD BRANCH

The development of this branch poses no new problems. The two events causing control loop overload, G9 REV and G9 NONE, have been identified above during the development of the C(DUMMY) branch. The resulting sub-trees for these two events are presented in Figures 10.17 and 10.18.

Now the only event awaiting development from Figure 10.8, is the deviation in the sensed variable. The sensed variable is known to be T; the development of the spontaneous failure event has revealed that the deviation in this event is HIGH. Hence, the event requiring development is T4 HI.

It will however be noted that T4 HI has already been developed under the C(DUMMY) branch. There is no need to develop this event again.

The next step is a fault tree rationalisation step.
Figure 10.17
Sub-tree for No Flow in Control Loop Overload Branch
Figure 10.18
Sub-tree for Reverse Flow in Control Loop Overload Branch
10.2.6 FAULT TREE RATIONALISATION

For the current system, the events in the C(DUMMY) branch are compared with those in the sensed variable branch. Any events which are common to both are deleted from the C(DUMMY) branch. In this case all the events under the C(DUMMY) branch would be deleted since the start of this branch is actually T4 HI and so it does not contain any other failures.

This procedure is necessary since there may be circumstances in which the C(DUMMY) branch will contain events which the control loop cannot sense but which can give rise to the top event. These would normally be events which mislead the control loop. However, no such events exist in this system.

The next thing to do, is to add the trip loop functional failure events. In order to do this it is necessary to develop both the trip 'should activate' sub-tree and also the trip loop functional failure sub-trees.

10.2.7 ADDITION OF THE TL-FN-F EVENTS

The start events for trip loop functional failure (TL-FN-F) and trip should activate (SHAC) have already been noted when events propagated through the trip valve (see Section 10.2.1). Firstly, the trip loop functional failure event will be developed. This will have the following causes:

![Sub-tree for Trip Loop Functional Failure](image)

In a similar manner to the control loop spontaneous/latent failure branches, the sub-tree for TL-FN-F will be prevented from tracing any causes into the process stream.

The SHAC tree will then be developed. In this instance there will not be any need to accord any special treatment to any sub-systems, e.g. control loops. The purpose of the SHAC tree is to simply trace all the basic failure events which should cause the trip to activate.

Note: to find the relative location of the C(DUMMY) and sensed variable branches, refer to Figures 10.10 and 10.21a, respectively.
Since this trip system is feedforward, it is necessary to compare all the causes of the top event of the main tree with the basic and diamond event causes in the SHAC tree. Any events which are common to both trees would then be ANDed with TL-FN-F in the main tree.
Since the trip system is designed to act upon No Flow of the manipulated stream, then the SHAC branch will only have events in common with the control loop overload branch. Figure 10.21, thus shows the net effect of including the TL-FN-F events in the causes of the G9 NONE event from the control loop overload branch.

It should be noted that all the causes of control loop overload as a result of No Flow of the manipulated stream, other than LK-LP-EN Unit 10, can be protected by the trip system. The trip system cannot protect against LK-LP-EN Unit 10, since although it
may be a cause of no flow of the manipulated stream through the heat exchanger, it will actually tend to indicate a high flow at the flow sensor, Unit 9.

10.2.8 OVERALL TREE

The overall tree is too large to present in one figure. Hence, Figure 10.21a below, presents an outline of the overall tree. References are made to sub-trees which have been developed above, thus enabling this tree to be expanded into the overall tree.

![Outline of the Overall Tree for Lapp-Powers System](image)

**Figure 10.21a**
Outline of the Overall Tree for Lapp-Powers System

10.2.9 CONCLUSION

This example has highlighted the basic working of the fault tree synthesis algorithm. In particular the following aspects have been demonstrated:

- linkage of minitrees by propagating faults from one unit to another.
- consistency checks.
- application of the control loop operator.
- diagnosis of trip loop functional failure.
10.3 PIPELINE MIXER SYSTEM

This system is based on that used by Taylor to illustrate the RIKKE code. Figures 10.22 and 10.23 give the flow and decomposition diagrams, respectively. The purpose of the plant is to regulate the mixture of the two streams, such that the downstream composition is unaffected. It can be seen that the system comprises a complex, though quite common control strategy. There are two control loops. One is feedback, since it acts by sensing the downstream composition and then manipulates the flow of one of the feeds. The other is feedforward, since it acts by sensing the flow of one feed stream and then manipulates the flow of the other feed stream.

Additionally both control loops share a common controller and control valve.

10.3.1 INPUT REQUIRED BY MASTER PROGRAM

The following is the input required by the MASTER program to specify the control loops in the above configuration.

Control Loops

- Number: 1
- Sensed Variable: X
- Variable Sensed in Unit: 11
- Control Valve Unit Number: 3
- Other Units in Control System: 14, 15, 16
- Variable X Regulated in Connections: 9, 10, 11, 12
- Flow Manipulated in Connections: 1, 2, 3, 4
- Loop is Not of Feedforward Type
Number: 2
Sensed Variable: Q
Variable Sensed in Unit: 8
Control Valve Unit Number: 3
Other Units in Control System are: 14 15 16
Variable X Regulated in Connections: 9 10 11 12
Flow Manipulated in Connections: 1 2 3 4
Loop is of the Feedforward Type

It is important to note that the feedback control loop is able to correct for failures in the feedforward control loop, whereas the converse is not true. Hence, the feedback control loop has been specified first.

10.3.2 FAULT TREE

Figure 10.24 gives the overall fault tree structure for this system. The main branches have been numbered. Example faults for each of these branches are given in the following table:

<table>
<thead>
<tr>
<th>Branch Number</th>
<th>Start Event for Branch</th>
<th>Example Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Control Loop Overload Loop 1</td>
<td>COMP-BLK Unit 4</td>
</tr>
<tr>
<td>2</td>
<td>CL-F-LA Loop 1</td>
<td>CV-F-LA Unit 3</td>
</tr>
<tr>
<td>3</td>
<td>CL-STK Loop 1</td>
<td>CV-STK Unit 3</td>
</tr>
<tr>
<td>4</td>
<td>C(DUMMY) Loop 2</td>
<td>XB5 HI/ XB1 HI</td>
</tr>
<tr>
<td>5</td>
<td>Control Loop Overload Loop 2</td>
<td>COMP-BLK Unit 4</td>
</tr>
<tr>
<td>6</td>
<td>CL-F-LA Loop 2</td>
<td>CV-F-LA Unit 3</td>
</tr>
<tr>
<td>7</td>
<td>CL-STK Loop 2</td>
<td>CV-STK Unit 3</td>
</tr>
<tr>
<td>8</td>
<td>Q6 HI</td>
<td>PART-BLK Unit 4</td>
</tr>
</tbody>
</table>

Table 10.1 Example Faults for Branches of Fault Tree of Figure 10.24

10.3.3 DISCUSSION OF TREE

Whilst the overall structure of the fault tree produced by the MK1 methodology was correct, the detail of the tree contained some inaccuracies. These arose as a result of the sequential application of the control loop operators. Two main problems were encountered. These related to:

- both control loops manipulating the same stream.
- both control loops sharing common components.

Each of these is described below.

10.3.3.1 MANIPULATED STREAM

Figures 10.25 and 10.26 give the fault tree sections for the control loop overload branches for control loop 1 and 2, respectively.
Figure 10.24
Overall Fault Tree Structure for Pipeline Mixer System
Figure 10.25
Control Loop Overload Branch for Loop 1

Figure 10.26
Control Loop Overload Branch for Loop 2
It has already been stated that control loop overload is caused by either none or reverse flow of the manipulated stream. For the current system, since both control loops are manipulating the same stream, then the only differences in these branches are going to lie in the control loop components, giving rise to the control loop failing such that there is no aperture for flow.

Since the application of the operator for control loop 1 states that control loop overload is sufficient to give rise to the top event, then it is necessary to eliminate the common events in Branches 1 and 5 (in Figure 10.24) from Branch 5. Hence, Branch 5 will only contain failures specific to control loop 2. This is illustrated in Figure 10.27 below:

10.3.3.2 COMMON COMPONENTS

It is necessary to apply the operator for Loop 1 prior to the operator for Loop 2, since the former can correct for faults in the latter. Hence, spontaneous failure of Loop 2, i.e. CL-F-LA Loop 2 is correctly ANDed with latent failure of Loop 1, i.e. CL-STK Loop 1.
However, since both control loops share common components, then the two branches mentioned above will have contradictory failures for the same component, e.g. the CL-F-LA branch will contain CV-F-LA Unit 3, whereas the CL-STK branch will contain CV-STK Unit 3. The fault tree sections for these two branches are given in Figures 10.28 and 10.29, respectively.
To overcome this problem a fault tree rationalisation step was added. This was based upon the following criteria.

Since the operator for Loop 1 is applied first, then this dictates what failure modes that the common component(s) can have. Subsequently any failures contradictory to this, appearing in any control loop operators below this would be removed from the tree. Hence, in the tree of Figure 10.29 the failure CV-F-LA Unit 3 in the CL-F-LA Loop 2 branch would be deleted, since the branch CL-STK Loop 1 imposes the condition that the valve failure state for this unit is CV-STK Unit 3. The modified branch for CL-F-LA Loop 2 is given in Figure 10.30.

![Figure 10.30 Modified Fault Tree Section for CL-F-LA Loop 2](image)

10.3.4 MUTUALLY EXCLUSIVE EVENTS

The criteria used for determining mutually exclusive events has been described in Chapter 9. The main principle used is to examine the variable deviations. In the current system the start event for CL-STK Loop 1 is S 15 NCHA, whereas the start events for CL-F-LA Loop 2 and CL-F-NA Loop 2 are S 15 LO and S 15 NONE, respectively. Hence, the criteria described in Chapter 9 will prevent the ANDing together of events from the CL-STK Loop 1 branch with those from the CL-F-LA Loop 2 and CL-F-NA Loop 2 branches.

However, this is not entirely accurate as only some of the events between the branches are mutually exclusive. Hence, it is necessary to modify the sub-trees of Figures 10.30 and 10.28 as illustrated in Figures 10.31 and 10.32, respectively.
Figure 10.31
Modified Sub-tree for CL-F-LA Loop 2

Figure 10.32
Modified Sub-tree for Control Loop Overload Loop 2
10.3.5 CONCLUSION

This example has outlined the problems presented by the sequential application of the control loop operators. The solution to these problems has also been outlined. In particular the following topics have been addressed:

- problems associated with shared components between the control loops.
- problems associated with shared manipulated stream.
- problem of mutually exclusive events.
10.4 BUTANE VAPORISER SYSTEM

The flow diagram for this system is given in Figure 10.33. The plant section under study is used at times of high demand to produce a butane-air mixture for supplementing the supply of natural gas in the British Gas distribution system.

The pump (P1) is used to deliver the liquid butane from storage to the vaporiser (VP1), where it is converted to the vapour by two independently controlled natural gas fired burners. The vaporised butane is then mixed with natural gas by flow ratio control.

Dependent upon demand, the pressure control valve (V1) is adjusted to regulate the delivery pressure of the liquid butane between 70 and 110 psig. Since the pump is capable of delivering 300 psig and the vaporiser coils are rated at 250 psig, overpressure protection is provided by the slam shut facility on the pressure control valve (V1), the slam shut valve (V2) and the pressure relief valves (VV1 and VV2). There is a further slam shut valve (V3) which activates on detection of liquid butane downstream of the vaporiser. This condition is represented by the presence of low temperature at the vaporiser outlet. All the slam shut valves are of the air-to-open type and activate by the rapid venting action of the three-way valves.

The hazardous event of interest is high pressure liquid butane venting through VV2; the venting of high pressure butane vapour was not considered. Therefore the top event of the fault tree was specified as 'low temperature at the outlet of the vaporiser', representing the passage of liquid butane through the coils.

10.4.1 SYSTEM DECOMPOSITION

The decomposed configuration diagram is given in Figure 10.34. In general there is a one-to-one correspondence between the units in this and the system flow diagram of Figure 10.33. The differences are described below:

a) since the plant downstream of the butane vaporiser does not contain any failure modes relevant to the current top event, it has been omitted from the decomposed configuration.

b) the pressure switches PSW1 and PSW2 are represented as two units, a switch and a sensor (Units 5 and 22, 7 and 16, respectively).

c) the dummy head/tail and set-point units are used, as described in Chapter 3.

d) the divider (Unit 6) is needed to represent the splitting of the process streams.

e) the pressure relief valve VV1 behaves in a similar manner to a Closed Valve Trip System as described in Chapter 7. In order to treat this as a special sub-system, the ancillary units to a trip valve have also been included, i.e. the sensor (Unit 11), trip switch (Unit 14) and set-point (unit 15).
Figure 10.33
Flow Diagram for Butane Vaporiser System
10.4.2 INPUT TO MASTER

The input required by the MASTER program to fully specify the control loops and trip systems in the above configuration, is given below:

Control Loops

Number: 1
Sensed Variable: P
Variable Sensed in Unit: 4
Control Valve Unit Number: 3
Other Units in Control System are: 23 25 26
Variable P Regulated in Connections: 3 4 5 6 7 8
Variable Q Regulated in Connections: 1 2 3 4 5 6 7 8
Loop is Not of Feedforward Type

Open Valve Trip Systems

Number: 1
Trip Valve Unit Number: 3
Other Units in Trip System are: 23 22 24 5

Number: 2
Trip Valve Unit Number: 8
Other Units in Trip System are: 7 16 17 18 19

Closed Valve Trip Systems

Number: 1
Trip Valve Unit Number: 12
Other Units in Trip System are: 11 14 15
Connections Having Flow When Valve Opens: 10 11 12

The significant points to note from the above information is that the valve (unit 3) is specified as being part of both the control loop and the trip system.

Also both the pressure P and the flow Q are specified as being regulated by the control loop.

10.4.3 PROBLEMS PRESENTED TO FAULTFINDER MK 1

Two main problems were encountered:

a) the control loop contains an inter-relationship between the absolute pressure (P) and the flow (Q) variables. The FAULTFINDER MK 1 methodology treated these two as separate variables.

b) the valve (Unit 3) is common to both the trip system and the control loop.

The second problem and its solution have already been discussed in Chapter 7. Hence, only the first problem will be considered here.
10.4.4 SOLUTION TO PROBLEM

Since the control loop is regulating the flow in stream 8, the template of Figure 10.35 will be imposed.

![Diagram](image)

Figure 10.35
Control Loop Operator for Butane Vaporiser System

It has already been described for the Lapp-Powers system how this template will be filled in. The basic points of note in the current context are that the C(DUMMY) branch will be developed to find the start points of the other branches. The contents of this branch will then be compared with the sensed variable branch, and the tree structure rationalised accordingly.

The problem arose in this rationalisation step, since the C(DUMMY) branch was developed in terms of G and Q variables and the sensed variable deviation branch in terms of the P variable. The consistency checks were incomplete since they were based on the modelling conventions of keeping the G/Q and P paths separate. However, in this instance a requirement has arisen where it is necessary to compare events from these two separate branches.

The consistency checks were therefore updated to reflect this overlap between the domains of the three variables, so that the fault tree synthesis algorithm could automatically rationalise events between the C(DUMMY) and the sensed variable branches.

The complete fault tree for this system is given in Figure 10.36.
Figure 10.36
Complete Fault Tree for Butane Vaporiser System
10.4.5 DISCUSSION OF THE FAULT TREE

The fault tree indicates that there are three potential causes of liquid butane passing through the vaporiser:

- **T1 LO** - a cold source could cause this if it were beyond the capacity of the vaporiser to compensate. However, no such source was found and so T1 LO is flagged as a diamond event.

- **G9 HI** - a large leakage downstream could be sufficient to cause a surge through the vaporiser. Again it is flagged as a diamond event as the downstream plant is not being considered.

- **P1 HI** - a genuine high pressure upstream causing an increased flow through the system. Since the three trip loops and the control loop can protect against this, this event is ANDed with failures in the protective devices.

One further point is worthy of note. Although the pressure relief valve has been modelled as a Closed Valve Trip System, with its associated components, e.g. sensor, trip switch etc., the fault tree does not contain any failures related to these components. This is because the user has the capability to prune irrelevant/insignificant events from the tree, once it has been synthesised. Faults related to these components were pruned, since these components do not really exist in the system; they were introduced to model the relief system as a trip system.

10.4.6 CONCLUSION

This example has addressed the following points:

- handling combined control/trip loops
- modelling pressure relief systems as closed valve trip systems
- rationalisation of events between pressure variable P, and the flow variable G/Q branches.

A detailed solution for this example is presented by Mullhi, Ang, Lees and Andrews [Ref 10-1], where the tree generated by FAULTFINDER is compared with that generated manually by workers at British Gas.
10.5 BRITISH GAS BURNER CONTROL SEQUENCES

A number of examples were provided by British Gas to validate the code. This section presents the fault tree for one of these systems - the manual system. It has been included since it demonstrates the use of the sequential operations algorithm of FAULTFINDER.

10.5.1 MANUAL SYSTEM

The flow diagram for the manually operated system is given in Figure 10.37.

The burner is to be ignited by means of a hand held torch in a series of steps. MV, TV, and ISV are all manually operated valves; GOV is a governor included in the inlet line to the torch and the burner to prevent high flow, which would cause a loss of flame. The operating procedure is as follows:

a) when the burner is shutdown, all valves will be closed.
b) open the oven doors and leave for the required purge period.
c) open the main isolation valve (ISV).
d) open the torch valve (TV) and immediately light the torch.
e) insert the torch into the oven in correct location.
f) open the main valve (MV).
g) check that the main flame has lit correctly.
h) close the torch valve (TV) and remove the torch.
i) close the oven doors.
j) to shutdown, close the main and isolation valves.

The undesired event is the release of unburnt gas from the system during the ignition, subsequent operation, or shutdown of the system.

The decomposition diagram for this system is given in Figure 10.38.
The following four sequence steps were identified.

a) open the isolation valve (ISV).

b) open the torch valve (TV) and then light the torch.

c) open the main valve (MV) and ignite the burner by inserting the torch at the correct location. Remove the torch and close TV.

d) to shutdown, close ISV and MV.

It is necessary to specify the unit models to be used. The basic configuration is specified as normal. This is given in the following table:

<table>
<thead>
<tr>
<th>TOPOLOGY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Number</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

Table 10.2
Topology Information for Manual Burner System

However, in addition to this, it is necessary to specify the new models to be used at each step in the sequence. The units affected by these configurational changes are given in the following table:
<table>
<thead>
<tr>
<th>Sequence Step</th>
<th>Unit Number</th>
<th>Model Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>77</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>79</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>75</td>
</tr>
</tbody>
</table>

Table 10.3
Configurational Changes for the Manual Burner System

The hazardous top events were specified as:

a) Q6 SOME for Steps 1 and 4, representing gas leakage from the main burner.
b) Either Q6 SOME or Q8 SOME for Steps 2 and 3, representing gas leakage from the main burner or torch valve.

A condensed form of this tree is presented below:

![Figure 10.39](image)

General Structure of Sequential Operations Fault Tree for Manual Burner System
Presented below are the sub-trees for each step in the sequence:

**Figure 10.40**
Fault Tree Section for SEQ-F-AT Step 1

SEQ-F-AT
Step 1

OR

HV-D-OP
Unit 5

HV-F-OP
Unit 5

**Figure 10.41**
Fault Tree Section for SEQ-F-AT Step 2

SEQ-F-AT
Step 2

OR

Q6 SOME

OR

HV-D-OP
Unit 5

Q8 SOME

OR

HV-F-OP
Unit 5

TOR-NLIT
Unit 8
SEQ-F-AT Step 3

OR

Q6 SOME

OR

Q8 SOME

INSUF-PU Unit 6

P-FL-STA Unit 6

G5 HI P-X-LIT Unit 6

T-N-INS Unit 6

OR

TOR-NLIT Unit 8

TV-L-OP Unit 8

GOV-F-FO Unit 3

Figure 10.42
Fault Tree Section for SEQ-F-AT Step 3

SEQ-F-AT Step 4

AND

A(DUMMY) Unit 5

OR

Q4 SOME

OR

HV-D-OP Unit 5

HV-F-OP Unit 5

HV-D-OP Unit 2

HV-F-OP Unit 2

Figure 10.43
Fault Tree Section for SEQ-F-AT Step 4
10.5.2 PROBLEMS PRESENTED

Two main problems were encountered in modelling this system using the MK1 methodology. Each of these is described below:

a) Incorrect deletion of a Duplicated Event under an AND gate. This is exemplified by the following example:

```
F HV-F-OP: A(DUMMY)
O HV-D-OP: A(DUMMY)
V Q1IN SOME: G1IN SOME
V Q2OUT SOME: Q2OUT SOME
V G1IN SOME I A(DUMMY) T Q2OUT SOME
V Q2OUT SOME I A(DUMMY) T G1IN SOME
```

These would yield the following section of tree for the event SOME flow to propagate through this unit (considering the upstream causes only):

![Fault Tree Section for SOME Flow through Hand Valve Model of Figure 10.44](image-url)
The second occurrence of Q2OUT SOME needs to be removed from the tree. However, it is insufficient to remove just Q2OUT SOME; it is also necessary to remove the other branch(es) from the AND gate above this event. In the MK1 methodology this was not done and thus yielded incorrect cutsets.

b) the top event needs to be carefully modelled, when the hazard of interest is the SOME state. This is illustrated by the following modified section of plant from the above system:

If the event of interest is SOME FLOW in the pipe at the outlet of the burner then the sub-tree for this will be:

Since SOME FLOW at the Dummy Tail is regarded by the methodology to be a normal(certain) event, then the right hand event will be deleted as certain. Since one branch under an OR gate is bound to occur then the left hand branch will also be deleted as certain.

Consequently what is required is an AND gate to commence the synthesis thus:
10.5.3 CONCLUSION

This example has addressed the following points:

- the application of the sequential operations algorithm for FAULTFINDER
- the deletion of events under an AND gate
- the need to model SOME deviation with care.
10.6 PUMPS IN PARALLEL PROVIDING REDUNDANCY

10.6.1 INTRODUCTION

The flow and decomposition diagrams for this system are given in Figures 10.49 and 10.50, respectively.

![Schematic Representation of Pump Bank]

Each pump is capable of providing 50% of the required throughput. The general problems presented by this sort of system, and the solution of these, have already been outlined in Chapter 5. This section presents the input required from the user along with the fault tree produced by the system.

Firstly, the input required by the MASTER program will be discussed. The most important thing to note is how the three legs of the system have been represented by the use of the two-legged divider and header models, as described in Chapter 5.

Having achieved this there is no problem in specifying the topology and configuration information to the MASTER program. As illustrated in Chapter 5, however, there is a need to specify the throughput that each leg is able to provide. The MASTER program will automatically detect that there are two dividers and two headers in this system. For each of these it will prompt for the required information as follows (the actual user input is given in italics):

For each divider/header enter flow capacity of each leg.
You may input: 0, 50, 100.

Header unit number: 9
Leg: 8
100
Leg: 18
50

Header unit number: 8
Leg: 7
50
Leg: 14
50

Divider unit number: 3
Leg: 3
100
Leg: 15
50

Divider unit number: 4
Leg: 4
50
Leg: 11
50

Since each leg is able to provide 50% of the required throughput, it should be noted that:

a) for the outer divider/header the leg leading to the inner divider/header has been specified as having 100% capacity, whereas the other leg has been specified as having 50% capacity.

b) for the inner divider/header each leg has been specified as having 50% capacity.

Having done this the system will generate the fault tree structure of Figure 10.51.

---

![Figure 10.51](image)

**Figure 10.51** General Form of Tree for Pump Bank System
10.6.2 DISCUSSION OF THE TREE

The fault tree has two main branches; the faults are split according to whether they are internal to the three legs of the system or external to them. It is necessary to do this since the three legs can provide redundancy, and so the combination of faults between the legs is important.

The internal branch is further split down between the leakage-type and blockage-type faults. As described in Chapter 5, the distinction between these faults is that, if a leakage occurs in any one leg, then that alone may be capable of producing low flow at the outlet, as the system may not be able to compensate for the loss of fluid. Hence, the leakage branches for the three legs are all linked by OR gates.

On the other hand, the blockage branch is connected by an r/n gate. The r/n gate is actually a 2/3 gate since each leg is capable of providing 50% of the required throughput.

Figure 10.52 gives the sub-tree for the external branch. Figures 10.53 and 10.54 contain the sub-trees for the leakage and blockage branches, respectively, for one leg of the above system. The sub-trees for the other legs would be similar.
10.6.3 CONCLUSION

This example has illustrated the special decomposition technique used to model parallel systems using the standard two-legged divider and header models. The resultant fault tree for such a system has also been presented.
10.7 PRESSURE REDUCTION INSTALLATION

10.7.1 INTRODUCTION

Figure 10.55 gives the flow diagram for a pressure reduction installation. It comprises two streams in parallel; one stream normally operating and the other providing 100% standby in the event of failure. In this study only one stream was considered - the normally operating one.

It can be seen that each stream contains three valves, a slam shut valve (SS), a pressure control valve (PCV) and a flow control valve (FCV).

In normal operation the FCV is driven by a volumetric controller (VC) which receives a signal from the station flow meter. The FCV maintains a constant flow. VC is an electrical-pneumatic controller. VC output is a high pressure for a high flow.

PCL and PCH control unacceptably low and high pressure respectively. PCL and PCH are reverse-acting controllers with pneumatic input and output. There is a signal selector for the FCV. The selector takes the highest signal from VC and PCL and the lowest signal from PCH and the signal selected from VC and PCL. The signal selector is a pneumatic selector relay.

The pressure control valve PCV reduces the pressure and is controlled by PC, a direct acting pneumatic controller. PCO is also a direct-acting pneumatic controller; however, it is an override controller. PCO is set at a value above PC such that if the FCV fails open then PCO will shut down PCV accordingly, since PCH is ineffective in this case. The slam shut valve will close if the pressure is unacceptably high at PZ.

The decomposition diagram for this system is given in Figure 10.56.

The following is a summary of the input required by the MASTER program. The topology and configuration information is omitted. Only the information relating to the sub-systems is given:

CONTROL LOOPS

Number: 1
Sensed Variable: P
Variable Sensed in Unit: 10
Control Valve Unit Number: 5
Other Units in Control System are: 6 15 16 22
Variable P regulated in Connections: 5 6 7 8 9 10 11 12
Loop is Not of the Feedforward Type

Number: 2
Sensed Variable: P
Variable Sensed in Unit: 9
Control Valve Unit Number: 7
Other Units in Control System are: 2 8 17 18 19 20 21
Variable P regulated in Connections: 7 8 9 10 11 12
Loop is Not of the Feedforward Type
Number: 3
Sensed Variable: P
Variable Sensed in Unit: 8
Control Valve Unit Number: 7
Other Units in Control System are: 2 9 17 18 19 20 21
Variable P regulated in Connections: 7 8 9 10 11 12
Loop is Not of the Feedforward Type

Number: 4
Sensed Variable: P
Variable Sensed in Unit: 6
Control Valve Unit Number: 5
Other Units in Control System are: 10 15 16 22
Variable P regulated in Connections: 5 6 7 8 9 10 11 12
Loop is Not of the Feedforward Type

Number: 5
Sensed Variable: P
Variable Sensed in Unit: 2
Control Valve Unit Number: 7
Other Units in Control System are: 8 9 17 18 19 20 21
Variable P Regulated in Connections: 7 8 9 10 11 12
Loop is of the Feedforward Type

OPEN VALVE TRIP SYSTEMS

Number: 1
Trip Valve Unit Number: 4
Other Units in Trip System are: 11 14

The points to note from the above decomposition are:

a) in accordance with the definition of a control loop, each sensor emitting a corrective signal to a control valve has been treated as a control loop in its own right. Hence, five control loops have been identified.

b) all the control loops have been specified as being able to regulate the pressure in this stream, since one control loop can compensate for faults in another. Hence, the order in which the control loops are specified does not matter, c.f. Pipeline Mixer System of Section 10.3.

c) control loop number 5, although it actually senses the flow variable has been specified as acting on the pressure variable. There is a bug in the program which prevented the use of the flow variable. There was insufficient time on this project to fix this problem. This approach was adequate in the current context since the causes/effects of the two variables were synonymous.
The top event of interest was specified as being a High Pressure at the outlet of the system. It has already been highlighted in Chapter 6 that, since more than one control loop can regulate a variable at one location and neither control loop is dominant, then the sequential application of the control loop operators is not correct. The solution to this problem has also presented in Chapters 6 and 9. The actual implementation is based on applying each control loop operator in turn and then rationalising the final structure of the tree to the required format.

In order to be able to do this, the system needs to know if there is a group of self-compensating loops, i.e. those which can correct for failures in one another. This is achieved by the fault tree synthesis program prompting for groups of self-compensating loops, if more than one loop exists in the system. For the current system Loops 1-5 were specified as being self-compensating.

The tree produced for this system was rather large, primarily due to the large number of common components in the different loops. Hence, only the top level of the overall tree structure is presented here in Figure 10.57.

10.7.2 DISCUSSION OF THE TREE

It can be seen from Figure 10.57, that the fault tree is dominated by failures in the control loop components, as exemplified by the CL-STK and CL-F-HA branches. The use of the A(DUMMY) Unit 0 event has been explained in Chapter 9.

The tree is large because the system has been decomposed as five control loops, and there are a large number of shared components. This arises because the sensor signals are compared before being passed to the control valve.

Hence, although the decomposition of this system, as described above in accordance with the rules of Chapter 6, yielded the correct cutsets, the resultant fault tree was rather opaque. There could be a case for treating this sort of system, where the signals are not independently fed to the control valve but are compared, as a special case and to decompose the control loops around the control valves as opposed to the sensors.

In the MK2 methodology, decomposition based on the sensors was supported as it was of general applicability. However, there is clearly a need to carry out further work to identify any similar special cases.

In actual fact the same problem was encountered by workers at British Gas, in the manual analysis of this system using the digraph technique. Two analysts independently studied this system. One applied the control loop operator five times, as did FAULTFINDER, whereas the other applied it twice for the two control valves. Both techniques produced the same cutsets, but obviously the latter yielded a smaller and more transparent tree structure.

10.7.3 CONCLUSION

This example has demonstrated how the system can handle multiple control loops, where each control loop is capable of correcting faults in another control loop.
Figure 10.57
Top-Level Fault Tree for Pressure Regulation System
11.0 CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

There is an on-going programme of research into fault tree synthesis at Loughborough University. The previous work had resulted in the creation of FAULTFINDER MK1, a suite of programs for computer-aided fault tree synthesis. However, this work also revealed certain problems.

The work described in this thesis has thus been involved in fundamental studies to improve the FAULTFINDER MK1 package. The approach taken has been to identify problems partly by review of the code facilities and partly by running case studies and then addressing the problems thus identified.

The output of this work has been FAULTFINDER MK2, a suite of programs written entirely in FORTRAN 77 and implemented at British Gas. Supporting user documentation has also been written and provided with the package.

One of the main thrusts has been to elucidate rules for fault tree synthesis, with emphasis on synthesising the tree. Rules have also been developed for other aspects, i.e. modelling and system decomposition.

Overall work has been undertaken in the following areas:

- model generation
- system decomposition
- fault tree synthesis
- fault tree analysis
- case studies
- developing the computer program

Each of these is reviewed below:

**Model Generation:** in the MK1 methodology the user had to specify a large number of event statements. Since this is a tedious approach, it is clearly advantageous to facilitate this task. Rules were developed which enabled the model generation algorithm to determine much of this information automatically. This was achieved by developing a more intelligent procedure for analysing propagation equations and by establishing a library of basic faults to handle fault initiation.

**System Decomposition and Fault Tree Synthesis:** the methodology has been considerably enhanced by facilitating the treatment of special sub-systems in the configuration. The following main advances have been made:

a) the need to explicitly specify divider header combinations has been eliminated. The package automatically identifies loops in the information flow structure and so can set up appropriate boundary condition checks in the fault tree synthesis algorithm. Rules have been defined for handling an arbitrary number of legs which may or may not have flow using a standard divider and header pair of models.

b) rules have been defined to assist the user in the decomposition of control loops and trip systems. A decomposition technique has been defined for modelling pressure relief systems as closed valve trip systems. The fault tree synthesis algorithm has been enhanced to handle complex control loops and combined control and trip systems. Rules
have been defined to ascertain the conditions that control and trip loops can protect against. The consistency checks have been updated, particularly in the case of combined flow and pressure regulation. The problems inherent in shared components between different loops has been addressed.

c) the problem of partial utility failures has been addressed.

Fault Tree Analysis: FAULTFINDER has been interfaced to FTAP and PREP. The problem of logically consistent trees which nevertheless yield incorrect cutsets due to the mechanistic application of standard fault tree analysis packages has been addressed by writing a modified input file to FTAP and carrying out post-analysis processing of the cutsets thus generated.

Case Studies: the application of the FAULTFINDER MK2 methodology has been illustrated using a large number of examples, which have been taken from three main sources:

a) from the literature, e.g. heat exchanger system of Lapp and Powers, composition regulation system of Taylor, hydrocarbon/oxygen reactor of Lihou and propane pipeline system of Lawley.

b) from British Gas studies, e.g. butane vaporiser system, burner control sequences and the pressure reduction installation.

c) a large number of contrived examples were set up to fully evaluate specific aspects of the methodology.

Developing the Computer Program: the MK1 system had been developed in a mixture of Basic and FORTRAN and resided on two different hardware environments. The MK2 system has been rationalised by conversion to FORTRAN 77 and implementation on a single machine (MicroVax II). It is easily portable to other environments.

The work described in this thesis has demonstrated that FAULTFINDER MK2 is capable of modelling realistic examples from the process industries. The main strengths of the package are:

- flexible component-based decomposition procedure which allows the easy description of the plant flow diagram to the system and facilitates the generation of failure models. Rules have been defined to assist the user in the decomposition of the plant flow diagram.

- the methodology can handle a wide range of systems consisting of quite complex control and trip loops.

- an extensive failure model library now exists, reducing the need to generate models for new studies.

- the system interfaces to standard fault tree analysis programs FTAP and PREP.
It is however not intended that FAULTFINDER should replace the fault tree expert; it should rather be viewed as a tool to assist the work of an experienced engineer in evaluating a number of design options. To this end FAULTFINDER should not be regarded as a "blind" package.

Although the primary role of FAULTFINDER will be to assist in the design phase, some time was also spent during this project evaluating its applicability to alarm diagnosis. It was intended that the cutsets from FAULTFINDER could be used to derive rules enabling the diagnosis of faults. Two main problems were encountered in achieving this:

- alarm combination trees could entail mutually exclusive branches, making their analysis difficult by standard fault tree analysis packages.

- fault trees by their very nature contain only failures. In diagnosing faults it is necessary to explicitly state the normal working states of plant items as well as failed states.

This topic is addressed in greater detail by Trenchard [Ref 11-1].

Although FAULTFINDER MK2 now offers a flexible tool for the interactive generation of fault trees, due to the limitation of time on this project there are inevitably further areas of work which should be undertaken to further enhance the package. These can be broken down into three main categories: fundamental studies on improving the underlying methodology, improved features in the package, and improvements to the code. Each of these is discussed below:

**Improved Methodology**

The emphasis to date in developing the FAULTFINDER methodology has been on producing logically consistent trees. Fault tree analysis has therefore tended to be overlooked. Some problems inherent in this have been discovered through the presence of mutually exclusive events in the fault tree. This is clearly an area which requires greater study and evaluation.

The rules for decomposing control loops and the application of control loop operators need further evaluation. The flow regulation system, although producing the correct cutsets, generated a rather opaque fault tree.

**Improved Features**

The following features of the system should be further enhanced:

- enhancement of the fault library for generating event statements (see Section 4.6.2).
- conflict resolution in generating event statements (see Section 4.6.4).
- enhancement of the algorithm for the detection of loops in the information flow structure (see Section 5.10.3).
- automatic detection/specification of control loops (see Sections 6.6.1.3, 6.6.1.5 and 6.7)
Improved Code

The following improvements should be made to the code itself:

- the man-machine interface is very basic and should be improved. The ultimate aim should be to integrate FAULTFINDER with standard CAD packages.

- the code has not been developed using any formal design methodologies. It is thus becoming rather unwieldy and would benefit from a re-write to a formal standard.

- the use of a conventional programming language, FORTRAN, has meant that much of the methodology of fault tree synthesis is embedded in the code and so is somewhat difficult to understand and upgrade. The use of knowledge based systems technology should be considered to enable a better control of the development and refinement of the rules relating to fault tree synthesis.
REFERENCES

This section gives the complete list of references used in this thesis. The numbering sequence relates to their use in the various chapters.


[Ref 2-6] Henley E.J. and Kumamoto H., "Reliability Engineering and Risk Assessment".


[Ref 5-1] same as [Ref 2-25].


[Ref 6-1] same as [Ref 2-25].

[Ref 6-2] same as [Ref 2-10].
[Ref 7-1] same as [Ref 2-25].


APPENDICES

APPENDIX A  SUITE OF FAULTFINDER PROGRAMS
APPENDIX B  LAPP-POWERS HEAT EXCHANGER SYSTEM
APPENDIX C  COMPENDIUM OF FAULTS
A.1 INTRODUCTION

The suite of FAULTFINDER programs have been written in FORTRAN 77 and implemented on a DEC MicroVax running VMS. However, the code should be portable to other systems with only minimal modifications. The main programs and the flow of information between these is shown in Figure A.1 below:
The key to the above programs is:

MODGEN  =>  Model Generation Program
EVTGEN  =>  Event Generation Program
MASTER  =>  Configuration Input Program
FAULT  =>  Fault Tree Synthesis Program
DRAW/PLOT  =>  Fault Tree Drawing Programs
FTAP/PREP  =>  Public Domain Fault Tree Analysis Programs

A.2  MAIN STEPS IN USING FAULTFINDER

The following section summarises the main steps involved in the synthesis of fault trees using FAULTFINDER.

a) decompose the plant representation to identify the unit and top event models to be used. Identify the special sub-systems in the configuration.

b) ensure that the unit and top event models exist in the respective libraries. If any models do not exist then the MODGEN/EVTGEN programs should be run. The user input to these can either be interactive or can come from a suitably created data file.

If these programs are run using interactive input, then a record is maintained of the input in a suitable data file for future reference.

A data file for a new model may be created using the template of an existing model.

c) run the MASTER program to input the plant configuration data. This input can either be interactive or can come from a data file.

If the program is run using interactive input, then a record is maintained of the input in a suitable data file for future reference.

A data file for a new configuration may be created using the template of an existing system.

d) run FAULT to synthesise the fault tree. The input to this program must be the output of the MASTER program. The output from this program is suitable to obtain either a pictorial representation of the fault tree or to carry out fault tree analysis.

e) run the fault tree drawing programs. These programs take their input from the data files created by the fault tree synthesis program and send their output to either a line printer, or a flat bed plotter.

f) run PREP/FTAP. The input to these programs is from data files created by the fault tree synthesis program. The output from these programs is also written to data files and comprises the listing of the minimal cutsets for the system.

Further details of the nature of user input required by these programs can be found in the main body of the thesis.
B.1 INTRODUCTION

This section presents an overview of the Lapp-Powers Heat Exchanger system and gives the complete decomposition information required by FAULTFINDER to synthesise the fault tree.

B.2 LAPP-POWERS SYSTEM FLOW DIAGRAM

The starting point for any analysis by FAULTFINDER is the system flow diagram. This is given in Figure B.1. The main points to note are that the system comprises two streams which interact in the heat exchanger. The hot nitric acid stream passes through the tubes of the heat exchanger and is cooled by the flow of coolant water through the shell-side of the exchanger.

The temperature at the outlet of the heat exchanger is monitored and regulated by manipulating the flow of coolant via a feedback control loop.

There is also a feedforward trip system. This is activated only if there is no flow of coolant. Under such conditions the flow of nitric acid is cut off. This is necessary since the control loop cannot compensate for deviations in the nitric acid stream if there is no flow of coolant.

Figure B.1
Flow Diagram for Lapp-Powers System
B.3 DECOMPOSITION DIAGRAM FOR LAPP-POWERS SYSTEM

The decomposition diagram corresponding to the flow diagram given in Figure B.1, is given in Figure B.2. A discussion of the decomposition diagram is presented in Chapter 3.2.

B.4 CONFIGURATION INPUT TO FAULTFINDER

When a new configuration is being input, it would normally be done interactively; the system prompts the user for input. The input would then be stored in a data file. This file is suitable for carrying out minor modifications to the configuration using a suitable screen editor. The complete input file required by FAULTFINDER to specify the configuration is given below.

NUMBER OF UNITS: 17  
NUMBER OF CONNECTIONS: 19  
NUMBER OF DIVIDER-HEADER COMBINATIONS: 0  
NUMBER OF CONTROL LOOPS: 1  
NUMBER OF TRIPS WITH OPEN VALVE: 1  
NUMBER OF TRIPS WITH CLOSED VALVE: 0

<table>
<thead>
<tr>
<th>TOPOLOGY - LIBRARY NUMBERS OF UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT NUMBER</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
</tbody>
</table>
Figure B.2. Decomposition Diagram for Lapp Powers System as Given in Figure B.1.
## CONFIGURATION - CONNECTIONS LINKING UNITS TOGETHER

<table>
<thead>
<tr>
<th>CONNECTION NUMBER</th>
<th>UPSTREAM</th>
<th>DOWNSTREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNIT</td>
<td>PORT</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>17</td>
<td>1</td>
</tr>
</tbody>
</table>

### CONTROL LOOPS

- **NUMBER:** 1
- **SENSED VARIABLE:** T
- **VARIABLE SENSED IN UNIT:** 5
- **CONTROL VALVE UNIT NUMBER:** 10
- **OTHER UNITS IN CONTROL SYSTEM ARE:** 12 13 16
- **VARIABLE T REGULATED IN CONNECTIONS:** 3 4 5
- **FLOW MANIPULATED IN CONNECTIONS:** 6 7 8 9 10
- **LOOP IS NOT OF THE FEEDFORWARD TYPE**

### OPEN VALVE TRIP SYSTEMS

- **NUMBER:** 1
- **TRIP VALVE UNIT NUMBER:** 2
- **OTHER UNITS IN THE TRIP SYSTEM ARE:** 9 14 15 16

The following notes accompany the above data input requirements:
a) the first section simply summarises some top-level information about the system. All systems comprise units and interconnections between these. In most systems these will interact to form sub-systems which require special treatment during fault tree synthesis. The FAULTFINDER MK1 methodology identifies four categories of sub-system: divider-header combinations, control loops, open valve trip systems and closed valve trip systems. Each of these is explained in Section 3.2 of Chapter 3. It will suffice here to note that the current system is flagged as having one control loop and one open valve trip system.

b) in the topology section it is necessary to identify the model library reference number of the failure model relating to each unit in the configuration. It is necessary that failure models exist in the library for all the units in the configuration.

c) in the configuration section it is necessary to identify how all the units are connected together. This is done by reference to the connections between the units. The terms upstream and downstream refer to the normal direction of flow.

d) for each sub-system it is necessary to specify additional information which is needed by the fault tree synthesis algorithm.

B.5 MODEL FOR HEAT EXCHANGER

A schematic representation of this unit is given in Figure B.3. This model has been selected from the above configuration to illustrate the modelling conventions of FAULTFINDER. This model is discussed in Section 3.3 of Chapter 3.

Given below is the list of propagation equations and event statements used to specify the failure model for the heat exchanger.
<table>
<thead>
<tr>
<th>Number</th>
<th>Propagation Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G1IN=F(Q1IN,Q2OUT)</td>
</tr>
<tr>
<td>2</td>
<td>R1IN=F(R2OUT)</td>
</tr>
<tr>
<td>3</td>
<td>U1IN=F(U2OUT)</td>
</tr>
<tr>
<td>4</td>
<td>Y1IN=F(Y2OUT)</td>
</tr>
<tr>
<td>5</td>
<td>Q2OUT=F(G1IN,G2OUT)</td>
</tr>
<tr>
<td>6</td>
<td>T2OUT=F(G1IN,-G3IN,T1IN,T3IN)</td>
</tr>
<tr>
<td>7</td>
<td>X2OUT=F(X1IN)</td>
</tr>
<tr>
<td>8</td>
<td>P2OUT=F(P1IN)</td>
</tr>
<tr>
<td>9</td>
<td>G3IN=F(Q3IN,Q4OUT)</td>
</tr>
<tr>
<td>10</td>
<td>R3IN=F(R4OUT)</td>
</tr>
<tr>
<td>11</td>
<td>U3IN=F(U4OUT,G1IN)</td>
</tr>
<tr>
<td>12</td>
<td>Y3IN=F(Y4OUT)</td>
</tr>
<tr>
<td>13</td>
<td>Q4OUT=F(G3IN,G4OUT)</td>
</tr>
<tr>
<td>14</td>
<td>T4OUT=F(G1IN,-G3IN,T1IN,T3IN)</td>
</tr>
<tr>
<td>15</td>
<td>X4OUT=F(X3IN)</td>
</tr>
<tr>
<td>16</td>
<td>P4OUT=F(P3IN)</td>
</tr>
</tbody>
</table>

Table B.1  
List of Propagation Equations for Heat Exchanger Model of Figure B.3
<table>
<thead>
<tr>
<th>Number</th>
<th>Event Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F EXT-HEAT: T2OUT HI, T4OUT HI, U1IN HI</td>
</tr>
<tr>
<td>2</td>
<td>F EXT-COLD: T2OUT LO, T4OUT LO, U3IN LO</td>
</tr>
<tr>
<td>3</td>
<td>F FOULING: T2OUT HI, T4OUT LO, U1IN HI, U3IN LO</td>
</tr>
<tr>
<td>4</td>
<td>V G1IN LO : D (DUMMY)</td>
</tr>
<tr>
<td>5</td>
<td>V G3IN LO : E (DUMMY)</td>
</tr>
<tr>
<td>6</td>
<td>V G1IN NONE : D (DUMMY)</td>
</tr>
<tr>
<td>7</td>
<td>V G3IN NONE : E (DUMMY)</td>
</tr>
<tr>
<td>8</td>
<td>V G1IN REV : D (DUMMY)</td>
</tr>
<tr>
<td>9</td>
<td>V G3IN REV : E (DUMMY)</td>
</tr>
<tr>
<td>10</td>
<td>I D (DUMMY): T4OUT LO, U3IN LO</td>
</tr>
<tr>
<td>1</td>
<td>I E (DUMMY): T2OUT HI, U1IN HI</td>
</tr>
<tr>
<td>12</td>
<td>F LK-LP-EN: G3IN HI, G3IN SOME, Q4OUT LO, Q4OUT NONE, Q4OUT REV, R3IN HI, R3IN SOME, R3IN NOP, P4OUT NONE, P4OUT REV, T2OUT HI, T4OUT HI, U1IN HI</td>
</tr>
<tr>
<td>13</td>
<td>F INT-LK: G1IN HI, G1IN SOME, Q2OUT LO, Q2OUT NONE, Q2OUT REV, R1IN HI, R1IN SOME, R1IN NOP, P2OUT LO, P2OUT NONE, P2OUT REV, G3IN LO, G3IN NONE, G3IN REV, Q4OUT HI, Q4OUT SOME, Q4OUT REV, R3IN LO, R3IN NONE, R3IN REV, P4OUT HI, P4OUT SOME, P4OUT NOR, X4OUT HI, Y3IN HI, T2OUT LO, T4OUT HI</td>
</tr>
<tr>
<td>14</td>
<td>F PART-BLK: G1IN LO, Q2OUT LO, R1IN LO, P2OUT LO, T2OUT LO, T4OUT LO, U3IN LO</td>
</tr>
<tr>
<td>15</td>
<td>F COMP-BLK: G1IN NONE, Q2OUT NONE, R1IN NONE, R1IN NOP, P2OUT NONE, P2OUT NOR, T4OUT LO, U3IN LO</td>
</tr>
<tr>
<td>16</td>
<td>S NORMAL: U1IN LO, U3IN HI</td>
</tr>
</tbody>
</table>

Table B.2
List of Event Statements for Heat Exchanger Model of Figure B.3
APPENDIX C

This appendix contains the list of fault names, along with their meaning, which have been used in this thesis.

A(DUMMY) Intermediate event to structure the fault tree
AIR-LOCK Air lock in a pump
AN-BROKE Alarm annunciator broken
B(DUMMY) Intermediate event to structure the fault tree
CAVITATN Pump cavitating
C(DUMMY) Intermediate event to structure the fault tree
CL-ACT Normal control loop action branch
CL-F-HA Control loop has failed giving high aperture
CL-F-LA Control loop has failed giving a low aperture
CL-F-NA Control loop has failed giving no aperture
CL-O-LD Control loop overload
CL-STK Control loop stuck
CNT-F-HI Controller failed emitting high signal
CNT-F-LO Controller failed emitting low signal
CNT-MAN Controller is in a manual setting
CNT-STK Controller is stuck
COMP-BLK Complete blockage in pipe
CV-F-HA Control valve failed giving a high aperture
CV-F-LA Control valve failed giving a low aperture
CV-F-NA Control valve failed giving no aperture
CV-F-SH Control valve failed shut
CV-STK Control valve stuck
D(DUMMY) Intermediate event to structure the fault tree
DTROWn Intermediate event arising from decision table
DUMMY Intermediate event to structure the fault tree
EXT-COLD External cold source
EXT-HEAT External hot source
F(DUMMY) Intermediate event to structure the fault tree
FOULING Fouling in heat exchanger
GOV-F-FO Governor failed fully open
HEAT-FAIL Heater failed
HV-D-OP Hand valve driven open
HV-D-SH Hand valve driven shut
HV-F-OP Hand valve fails open
HV-F-SH Hand valve fails shut
HV-FT-SH Hand valve fails to shut
IAR-LOSS Loss of inert air supply
IMPLR-F Pump impeller failed
INSUF-PU Insufficient purge
INT-LK Internal leakage
LK-HP-EN Leak from a high pressure environment
LK-LP-EN Leak to a low pressure environment
OP-FAIL Failure of operator to act
OVRTEMP Over temperature
PART-BLK Partial blockage in pipe
P-F-STA Poor flame stability
POW-LOSS  Loss of power supply
PROTECTN  Start of trip loop functional failure
PUMP-FAIL  Pump failure
PUMP-SUR  Surge in pump
P-X-LIT  Poor cross lighting
SEN-F-HI  Sensor failed high
SEN-F-LO  Sensor failed low
SEN-F-NO  Sensor failed none
SEN-STK  Sensor stuck
SET-P-HI  Setpoint failed high
SET-P-LO  Setpoint failed low
SEQ-ABRT  Sequence aborts
SEQ-F-AF  Sequence fails after step n
SEQ-F-AT  Sequence fails at step n
SHUTDOWN  The unit has been shutdown
SIG-CB  Signal line is completely blocked
SIG-PB  Signal line is partially blocked
SV-FT-OP  Solenoid valve fails to open
TL-FN-F  Trip loop functional failure
TL-OP-F  Trip loop operational failure
T-N-INS  Torch not inserted
TOR-NLIT  Torch not lit
TSW-DIS  Trip switch disarmed
TSW-STK  Trip switch stuck
TV-FT-OP  Trip valve fails to open
TV-FT-SH  Trip valve fails to shut
TV-L-OP  Torch valve left open
UTL-FAIL  Utility failure
UTL-OK  Utility OK
UTL-SOME  Some utility exists
VV-FT-OP  Vent valve fails to open