System dependency modelling

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

Additional Information:


Metadata Record: https://dspace.lboro.ac.uk/2134/34918

Publisher: © Huiling Sun

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at: https://creativecommons.org/licenses/by-nc-nd/4.0/

Please cite the published version.
University Library

Author/Filing Title .................. S.U.N. H

Class Mark .......................... T

Please note that fines are charged on ALL overdue items.

FOR REFERENCE ONLY
System Dependency Modelling

By

Huiling Sun

A Doctoral Thesis
Submitted in partial fulfilment of the requirements for the award of
Doctor of Philosophy of Loughborough University

February 2007

© by Huiling Sun 2007
Abstract

It is common for modern engineering systems to feature dependency relationships between its components. The existence of these dependencies render the fault tree analysis (FTA) and its efficient implementation, the Binary Decision Diagram (BDD) approach, inappropriate in predicting the system failure probability. Whilst the Markov method provides an alternative means of analysis of systems of this nature, it is susceptible to state space explosion problems for large, or even moderate sized systems.

Within this thesis, a process is proposed to improve the applicability of the Markov analysis. With this process, the smallest independent sections (modules) which contain each dependency type are identified in a fault tree and analysed by the most efficient method. Thus, BDD and the Markov analysis are applied in a combined way to improve the analysis efficiency. The BDD method is applied to modules which contain no dependency, and the Markov analysis applied to modules in which dependencies exist.

Different types of dependency which can arise in an engineering system assessment are identified. Algorithms for establishing a Markov model have also been developed for each type of dependency.

Three types of system are investigated in this thesis in the context of dependency modelling: the continuously-operating system, the active-on-demand system and the phased-mission system. Different quantification techniques have been developed for each type of system to obtain the system failure probability and other useful predictive measures.

Investigation is also carried out into the use of BDD in assessing non-repairable systems involving dependencies. General processes have been established to enable the quantification.
Acknowledgements

Firstly I would like to thank my supervisor Professor John Andrews for his guidance and support throughout the course of my Ph.D, and especially for his understanding and belief in me through my difficult times.

Thanks also to the members of the Department of Aeronautical and Automotive Engineering, particularly Rachel, Jeni and Emma in the same group, for their understanding, encouragement and good laughs we had together.

I would also like to thank my friends I have made since I came to Loughborough. They have made my experience as an overseas student in Loughborough more enjoyable and memorable.

Finally, a special thank you to my parents, for their love, support, encouragement and always being there for me.
Contents

1. Introduction  
   1.1 Introduction to Risk and Reliability Assessment  
   1.2 General Concepts in System Failure Quantification  
   1.3 System Reliability Modelling  
      1.3.1 Fault Tree Analysis  
      1.3.2 Binary Decision Diagram  
   1.4 System Dependency Modelling  
      1.4.1 The Markov Method  
      1.4.2 Dynamic Fault Tree Analysis (DFTA)  
   1.5 Research Objectives  

2. Overview of Conventional Reliability Assessment Techniques  
   2.1 Fault Tree Analysis (FTA)  
      2.1.1 Introduction  
      2.1.2 Definition of the Fault Tree structure  
      2.1.3 Qualitative FTA  
         2.1.3.1 Relevant Boolean Laws of Algebra  
         2.1.3.2 Two ways of Obtaining Minimal Cut Sets  
      2.1.4 Quantitative FTA  
         2.1.4.1 Reliability Models for Component Failures  
         2.1.4.2 Top Event Probability  
         2.1.4.3 Top Event Frequency  
            2.1.4.3.1 Unconditional Failure Intensity of Minimal Cut Set  
            2.1.4.3.2 Calculation of Top Event Frequency  
            2.1.4.3.3 Approximation for the System Unconditional Failure Intensity  
            2.1.4.3.4 Expected Number of System Failures  
         2.1.4.4 Importance Measures  
            2.1.4.4.1 Deterministic Measures  
            2.1.4.4.2 Probabilistic Measures  
      2.1.5 Summary of Fault Tree Analysis
2.2 Binary Decision Diagram – BDD
   2.2.1 Introduction
   2.2.2 Properties of BDD
   2.2.3 Formation of BDD
      2.2.3.1 Construction of BDD
         2.2.3.1.1 Construction of BDD Using Structure Function
         2.2.3.1.2 Construction of BDD Using If-Then-Else
   2.2.4 Quantitative Analysis of Binary Decision Diagram
      2.2.4.1 Top Event Probability/System Unavailability \( Q_{\text{sys}}(t) \)
      2.2.4.2 Unconditional System Failure Intensity \( w_{\text{sys}}(t) \)
      2.2.4.3 Unconditional System Repair Intensity \( v_{\text{sys}}(t) \)
      2.2.4.4 Criticality Function of Each Basic Event
      2.2.4.5 Worked example
   2.2.5 Summary of Binary Decision Diagram

3. System Dependency Modelling Techniques
   3.1 Markov Method
      3.1.1 Basic Elements of Markov Models
      3.1.2 Markov Property – the Underlying Assumption
      3.1.3 Different Types of CTMC
         3.1.3.1 Homogeneous CTMC
         3.1.3.2 Non-homogeneous CTMC
         3.1.3.3 Semi-Markov Model
         3.1.3.4 Comparison between 3 Types of CTMCs
      3.1.4 Additional Issues concerning the Markov Method
         3.1.4.1 Model Generation and Validation
         3.1.4.2 Model Size
            3.1.4.2.1 State Reduction Technique – State Lumping
            3.1.4.2.2 State Reduction Technique – State Truncation
      3.1.5 Summary of the Markov Method
         3.1.5.1 Advantages of the Markov Method
         3.1.5.2 Disadvantages of the Markov Method
   3.2 Dynamic Fault Tree Analysis
      3.2.1 Dynamic Fault Tree Gates
3.2.1.1 Functional-dependency Gate (FDEP) 45
3.2.1.2 Cold-spare Gate (CSP) 47
3.2.1.3 Sequence-enforcing Gate 48
3.2.1.4 Demand-dependency Gate (DDEP) 49

3.2.2 Application of Dynamic Fault Tree Method 52
3.2.2.1 Example System Description 52
3.2.2.2 Construction of Dynamic Fault Tree 54
3.2.2.3 Quantitative Analysis 56
  3.2.2.3.1 Calculation of System Unavailability 56
  3.2.2.3.2 Calculation of System Unreliability 57

3.2.3 Summary of Dynamic Fault Tree Technique 60
  3.2.3.1 Advantages of Dynamic Fault Tree Technique 60
  3.2.3.2 Disadvantages of the dynamic fault tree technique 60

4. Different Types of Dependency in the Systems 61
  4.1 Types of Dependency 61
    4.1.1 Maintenance Dependency 61
    4.1.2 Standby Dependency 61
    4.1.3 Sequential Dependency 62
    4.1.4 Sequence-enforcing Dependency 62
    4.1.5 Secondary-failure Dependency 63
    4.1.6 Initiator-enabler Dependency 65
    4.1.7 Revealing Dependency 68
    4.1.8 Test Dependency 69
    4.1.9 Functional Dependency 69
    4.1.10 Switching Dependency 70
  4.2 The Representation of Dependency Relationships 71
  4.3 The Representation of System Failure Logic 77
  4.4 The Representation of Component Failure and Repair Parameters 78

5. A New Method for Solving Dynamic Fault Trees with Dependencies 80
  5.1 Overview 80
  5.2 Pre-processing of Fault Tree Structure 81
5.2.1 Re-organize the Dependency Information 83
5.2.2 Fault Tree Simplification 85
5.2.3 Form the Dependency Information 90
5.2.4 Combination of Dependent Events 91
5.2.5 Modularisation 93
5.2.6 Update the Dependency Information 97
5.2.7 Re-modularise for Each Dependency Relationship 98

5.3 Discussion 101

6. The Generation of Markov Models 102

6.1 Introduction 102

6.2 The Generation of Markov Models for Different Types of Dependency Relationship 103
  6.2.1 Model Generation for the Maintenance Dependency 103
  6.2.2 Model Generation for Standby Dependency 113
  6.2.3 Model Generation for the Sequential Dependency 124
  6.2.4 Model Generation for the Sequence-Enforcing Dependency 127
  6.2.5 Model Generation for Secondary-Failure Dependency 128
  6.2.6 Model Generation for Initiator-Enabler Dependency 131
  6.2.7 Model Generation for Revealing Dependency 136
  6.2.8 Model Generation for Test Dependency 139

6.3 Transition Rate of Multiple-repair Transitions 141
  6.3.1 Approximation through Comparison 141
  6.3.2 Approximation from Real System Data 142

6.4 Model Generation for Multiple Dependency Relationships 142
  6.4.1 Maintenance Dependency and Initiator-Enabler Dependency 143
  6.4.2 Sequential Dependency and Initiator-Enabler Dependency 146
  6.4.3 Initiator-Enabler Dependency and Secondary-Failure Dependency 148

7. Quantitative Analysis with the Combined Application of FTA and Markov Method 150

7.1 Quantitative Analysis of Modules Involving a Dependency Relationship 150
  7.1.1 Module Failure Probability $Q_M(t)$ 151
  7.1.2 Module Unconditional Failure Intensity $w_M(t)$ 155
  7.1.3 Module Unconditional Repair Intensity $v_M(t)$ 155
7.1.4 Module Conditional Failure Rate $\lambda_M(t)$ 156
7.1.5 Module Conditional Repair Rate $\nu_M(t)$ 156
7.1.6 Criticality Information of the Basic Events 156

7.2 Quantitative Analysis for Modules Containing no Dependencies 159
7.2.1 Conversion of Fault Tree Structure to BDD 159
7.2.2 Quantitative Analysis Using BDD Method 161

7.3 System Quantitative Analysis 162
7.3.1 System Reliability Parameters 162
7.3.2 Basic Event Criticality 163

8. Dependency Modelling of Continuously-Operating Systems 169

8.1 Introduction 169
8.2 The Dependency Modelling Process for Continuously-Running Systems 169
8.3 The Software Development 169
8.3.1 Data Input 170
8.3.2 Pre-processing 171
8.3.3 Quantification 175
  8.3.3.1 Preparation 175
  8.3.3.2 The Quantification Process 177
8.3.4 Output 178

8.4 Application to an Example Fault Tree 181
8.4.1 System Information Input 181
8.4.2 Pre-processing 183
8.4.3 Quantification 183
  8.4.3.1 Preparation for Quantification 183
  8.4.3.2 The Quantification Process 186
8.4.4 Output 187
  8.4.4.1 System Reliability Predictions 187
  8.4.4.2 Importance Measures of Basic Events 192
8.4.5 Contrast with the Case Where Dependencies are Ignored 192
  8.4.5.1 Outputs Assuming Independence 192
  8.4.5.2 Comparison at Module/Factor Level 195
9. Dependency Modelling of Active-on-demand Systems

9.1 Dynamic Fault Tree Method and Active-on-demand Systems

9.2 Static-Dynamic Two-Phase Approach

9.2.1 Algorithm

9.2.2 The Analysis Process

9.2.3 Implementation of the General Process

9.2.3.1 System Information Input

9.2.3.2 Pre-processing

9.2.3.3 Quantification

9.2.3.3.1 Calculation of System Unavailability \( Q_{sys}(t) \)

9.2.3.3.2 Calculation of System Unreliability \( F_{sys}(T_r) \)

9.2.3.4 Calculation of Importance Measures

9.2.3.4.1 Importance Measures in the System Static Phase

9.2.3.4.2 Importance Measures in the System Dynamic Phase

9.2.3.5 Output

9.3 Application of the Static-dynamic Two-phase Approach to the Firewater Deluge System

9.3.1 Description of the Firewater Deluge System

9.3.1.1 The Fire Detection System and the Deluge System

9.3.1.2 Firewater Supply and Distribution System

9.3.1.3 AFFF Supply and Distribution

9.3.2 Reliability Assessment of the Firewater Deluge System

9.3.2.1 System Failure Information

9.3.2.1.1 Construction of Fault Trees

9.3.2.1.2 Dependency Files

9.3.2.1.3 Basic Event File

9.3.2.2 Pre-processing

9.3.2.3 Quantification

9.3.2.3.1 System Unavailability \( Q_{sys}(t) \)

9.3.2.3.2 System Unreliability \( F_{sys}(T_r) \)

9.3.2.3.3 Calculation of Importance Measures

9.3.2.4 Output

9.3.3 Review of Static-dynamic Two-phase Approach

9.3.4 Comparison with Results Assuming Independence
10. Review of Phased-mission System Analysis

10.1 Introduction
10.2 Existing Methods for Phased-mission Systems
10.2.1 Non-repairable Phased-mission Systems
   10.2.1.1 Transformation of a Multi-Phased Mission to an Equivalent Single-Phase Mission
   10.2.1.2 Approximation Techniques for Mission Unreliability
   10.2.1.3 Laws of Boolean Phase Algebra
   10.2.1.4 Binary Decision Diagrams
   10.2.1.5 Non-coherent Fault Tree Analysis
   10.2.1.6 The Markov Method
10.2.2 Repairable Phased-mission Systems
   10.2.2.1 Combinatorial Approaches
   10.2.2.2 Markov Methods
   10.2.2.3 Minimal Markov Method
10.2.3 Review of Existing Solutions to Phased-mission Analysis

11. Dependency Modelling in Phased-mission Systems

11.1 Introduction
11.2 Phased-mission Systems Involving Dependencies
11.3 Generic Modified Phase Algebra Method
   11.3.1 Repair Process and Phase Algebra
   11.3.2 General Process of Modified Phase Algebra Method
   11.3.3 Importance Measures in Phased-mission Systems
      11.3.3.1 Phase Criticality Function $G^p_q(q(t))$
      11.3.3.2 Phase Criticality Measures
         11.3.3.2.1 In-phase Criticality Measure $I^p_{i,j}$
         11.3.3.2.2 Transition Criticality Measure $I^T_{i,j}$
11.3.3 Mission Importance Contribution
11.3.4 Fussell-Vezeley Measure of Importance
11.3.4 Summary of the Modified Phase Algebra Method

12. Application of Modified Phase Algebra Approach
12.1 Example Phased-mission System
12.2 Application Process
12.2.1 Phase and Mission Unreliability
12.2.2 Importance Measures

13. Dependency Modelling Using Binary Decision Diagrams
13.1 Introduction
13.2 Dependency Modelling Using BDD for Non-repairable Systems
13.2.1 Dependency Modelling for Initiator-enabler Dependency
13.2.2 Dependency Modelling for Sequential Dependency
13.2.3 Dependency Modelling for Standby Dependency
13.2.3.1 Generic Process
13.2.3.2 Application
13.2.4 Dependency Modelling for Stress-related Dependency
13.2.4.1 Generic Process
13.2.4.2 Application

14. Conclusions and Further Work
14.1 Summary
14.2 Conclusions
14.3 Further Work

References

Appendix A Markov Model for Module in Chapter 8
Appendix B Quantification Models for Modules in Chapter 9
Nomenclature

\( A(t) \) \hspace{1cm} \text{Availability function}
\( F_{\text{sys}}(t) \) \hspace{1cm} \text{System unreliability function}
\( f(t) \) \hspace{1cm} \text{Failure probability density function}
\( C_i \) \hspace{1cm} \text{Minimal cut set } i
\( q_C(t) \) \hspace{1cm} \text{Component unavailability}
\( G_i(q(t)) \) \hspace{1cm} \text{Criticality function for event } i
\( I_i \) \hspace{1cm} \text{Criticality measure for component } i
\( w_i(t) \) \hspace{1cm} \text{Component unconditional failure intensity}
\( v_i(t) \) \hspace{1cm} \text{Component unconditional repair intensity}
\( \lambda_i \) \hspace{1cm} \text{Component conditional failure rate}
\( u_i \) \hspace{1cm} \text{Component conditional repair rate}
\( Q_M(t) \) \hspace{1cm} \text{Module failure probability}
\( w_M(t) \) \hspace{1cm} \text{Module unconditional failure intensity}
\( v_M(t) \) \hspace{1cm} \text{Module unconditional repair intensity}
\( \lambda_M(t) \) \hspace{1cm} \text{Module conditional failure rate}
\( u_M(t) \) \hspace{1cm} \text{Module conditional repair rate}
\( Q_{\text{sys}}(t) \) \hspace{1cm} \text{System failure probability}
\( w_{\text{sys}}(t) \) \hspace{1cm} \text{System unconditional failure intensity}
\( W_{\text{sys}}(t_0, t_1) \) \hspace{1cm} \text{Expected number of system failures during interval } [t_0, t_1)
\( G_{i,j}(q(t)) \) \hspace{1cm} \text{Phase criticality function for component } C \text{ in phase } j
\( I_{i,j}^p \) \hspace{1cm} \text{In-phase criticality measure for component } i \text{ in phased-mission system}
\( I_{i,j}^T \) \hspace{1cm} \text{Transition criticality measure for component } i \text{ in phased-mission system}
\( I_{i,j}^{FVI} \) \hspace{1cm} \text{Fussell-Vesely Measure of Importance for component } i \text{ in phase } j
\( Q_{\text{TRF},j} \) \hspace{1cm} \text{Phase transition failure probability for phase } j
Chapter 1. Introduction

1.1 Introduction to Risk and Reliability Assessment
History has witnessed many disastrous accidents due to the failure of industrial systems, such as the Chernobyl nuclear power plant disaster in 1986, the explosion on the Piper Alpha oil platform in 1988 and the Concorde disaster in Paris in 2000. In order to manage the safety of such systems, it is now common practice that the probability of system failure is quantified. It is always more effective to carry out the process of evaluating the system failure probability at the design stage when design changes can be made most effectively.

Significant advances have been made since the Second World War in the system failure assessment. Methods have been developed to enable the evaluation of the probability or frequency by which a specific hazardous event could occur. The Risk [1], or ‘expected loss’ of a specific incident, denoted by ‘R’, is defined as the product of the consequences of the event, C, and the probability or frequency of the event occurrence, P, as in equation 1.1

\[ R = C \times P \]  

1.1

Usually the consequences of an event are measured by the number of resulting fatalities and indicate the severity of the undesired event. The risk can therefore be reduced by either alleviating the consequences of the incident or reducing the associated incident probability or frequency. A risk assessment carried out at the system design stage can achieve both of these two aspects by identifying the potential improvements in the design.

With the risk of a specific event calculated according to equation 1.1, a three-zone approach is defined by the Health and Safety Executive to define the acceptable level of risk. As is illustrated in figure 1.1, the highest zone (High Frequency, High Consequence) represents unacceptable risk level, whilst the lowest zone (Low Frequency, Low Consequence) represents negligible risk levels which are considered to be acceptable. The intermediate zone is defined as ‘ALARP’ region [2] (As Low As Reasonably Practicable). In this case, the risk must be reduced to as low as possible.
To determine the risk level for a specific event, a quantified risk assessment is required. It is carried out in four basic stages:

1). Identification of potential safety hazards.
2). Estimation of the consequences of each hazard.
3). Estimation of the probability of occurrence of each hazard.
4). Comparison of the results of the analysis against the acceptability criteria.

The estimation of the consequences of a hazardous event is very much industry-dependent. In contrast, assessment techniques applied to investigate the probability or frequency of the event occurrence are generic and thus implemented across many different industries. Methods which have been frequently used include Failure Mode and Effect Analysis (FMEA) [3], Fault Tree Analysis (FTA) [3], Reliability Block Diagram [3], Binary Decision Diagram (BDD) [4] and the Markov Analysis [5].

1.2 General Concepts in System Failure Quantification
When predicting the reliability performance of a system, the quantification is carried out using appropriate techniques by drawing on component failure probabilities. Two parameters are frequently used as the measure of the system performance: Availability and Reliability.

- **Availability**
  Availability is a relevant and appropriate measure for systems which can be repaired and thus are tolerant of failures. It is defined as:
The fraction of the total time that a system (or component) is able to perform its required function.

Regarding a specified time point $t$, this parameter can also be interpreted as:

The probability that a system (or component) is working at time $t$.

The complement of Availability is Unavailability, which is obtained in:

$$\text{Unavailability} = 1 - \text{Availability}$$

The unavailability of a system or component represents the probability that the system or component is failed at time $t$, usually denoted by $Q_{sys}(t)$ and $q_c(t)$ respectively.

- **Reliability**
  The reliability measure is concerned with the system performance over a continuous period of time, and is defined as:
  
  The probability that a system (or component) will operate without failure for a stated period of time under specified conditions.

  Correspondingly, the probability that a system or component fails to function successfully over a stated period of time under specified conditions is defined as the Unreliability, denoted by $F(t)$, where:

  $$\text{Unreliability} = 1 - \text{Reliability}$$

  The Reliability is a more relevant measure for systems where failures cannot be tolerated and thus it is vital that the system should function successfully for a specified time duration. For systems or components which are non-repairable, if it is working at time $t$, then it must have worked without any failure over the time duration $[0, t)$. In this case, the unreliability during $[0, t)$ is equal to the unavailability at time $t$.

  An important parameter which is required to obtain the unavailability or unreliability of a system or component is its hazard rate or conditional failure rate, $h(t)$. This parameter indicates the
frequency of transition from a working state to a failed state of a component or system, and is defined as:

The probability that a component or system fails in the interval \([t, t+dt]\) given that it has not failed in \([0, t]\).

It is generally considered that for components the hazard rate can be modelled by a 'reliability bath-tub curve' [3], illustrated in figure 1.2.

During the first phase in figure 1.2, the hazard rate reduces as the weak components are eliminated. During the final phase, the hazard rate increases as the components start to wear out. During the second phase, the hazard rate remains approximately constant, and the reliability assessment is typically performed on components which are considered to be in this phase. With the constant hazard or conditional failure rate \(\lambda\), the reliability of a non-repairable system can be expressed in equation 1.2:

\[ R_{sys}(t) = e^{-\lambda t} \quad \text{1.2} \]

Further component and system quantification techniques are presented in [3].

1.3 System Reliability Modelling
Some of the most commonly used methods are discussed in the following sections.

1.3.1 Fault Tree Analysis
The concept of fault tree was first introduced by H.A. Watson in the 1960's. In the risk and reliability assessment, the fault tree establishes a deductive analysis using a 'what can cause this' approach. A fault tree diagram provides a visual representation of the combination of component failure events resulting in the occurrence of a specific system failure mode.
In a fault tree diagram, the particular system failure mode of concern is termed ‘top event’, which is the starting point of the fault tree development. A top-down approach is carried out from the top event to investigate the root causes. The development is not completed until component failure events (basic events) have been identified to define the causes of all the intermediate events. Through the logic operators which link all the component failures (AND, OR, NOT), the fault tree analysis is able to produce different combinations of component failure events which may lead to the occurrence of the top event.

The quantitative analysis of a fault tree is based on Kinetic Tree Theory, which was developed in the early 1970's by Vesely [6]. This technique enables the calculation of the time-dependent probability and frequency of the top event occurrence, and contributions to the system failure, known as importance measures of component failures [3] can also be produced. The main disadvantage of the fault tree analysis is that the fault tree quantification can become computationally demanding and even intractable for large fault trees. To tackle this problem, approximations have been developed.

1.3.2 Binary Decision Diagram

The use of the binary decision diagram (BDD) as a means for evaluating the top event probability of a fault tree was developed by Rauzy [4]. A binary decision diagram is a directed acyclic graph in which each node corresponds to a basic event in the fault tree. The BDD is constructed from the fault tree based on a particular basic event ordering.

Both qualitative and quantitative analysis can be performed on a BDD and exact solutions can be obtained without the need for approximations. However, one disadvantage with the BDD method is that the chosen variable ordering influences the size of the resulting BDD. The wrong choice of ordering scheme can result in a large BDD and thus increased analysis time.

1.4 System Dependency Modelling

Many systems feature a high level of inter-dependence between components. Some types of the inter-dependence between the components will introduce statistical dependency during the quantification process. In this case, the fault tree analysis and BDD technique are no longer appropriate means of analysis as they are based on the assumption of the independence between component failures in the system. Techniques such as dynamic fault tree analysis and the Markov method can be used to implement the quantification of systems which involve dependency relationships.
1.4.1 The Markov Method
The Markov analysis is based on a state-space approach. It analyses the reliability and availability of a system by determining the probability that a system resides in each possible state at a specified time point $t$. Component failures and repairs introduce transitions between states which occur at specified constant rates in the model. As the behaviour of each component in the system can be tracked in the Markov model, the Markov analysis is able to take into account the dependency relationships between components during the quantification process.

One big disadvantage of the Markov method is that the model size can grow exponentially with the number of components in the system, thus rendering it intractable for large or even medium-sized systems.

1.4.2 Dynamic Fault Tree Analysis (DFTA)
Dynamic fault tree was developed by J.B. Dugan [7, 8] to overcome the limitation of conventional fault tree analysis. By introducing new types of gate structures, such as the Functional-dependency Gate (FDEP), Cold-Spare Gate (CSP), Warm-spare Gate (WSP), Sequence-enforcing Gate (SEQ) and Demand-dependency Gate (DDEP), the dynamic fault tree is able to embrace, in its own structure, some types of dependency relationship as well as represent the system failure logic appearing in a conventional fault tree.

The quantitative dynamic fault tree analysis is carried out by converting sections below independent gates to a BDD and sections below dependency gates to a Markov model. By containing the dependencies under the particular dependency gates, the dynamic fault tree technique alleviates, to some extent, the model size problem which restricts the applicability of the Markov method. However, disadvantages also come with the dynamic fault tree analysis. When repeated basic events are included in different dependency gate structures, they will be repeatedly accounted for during the quantification process. Also, as it is very difficult to establish a dependency structure for some types of dependency, the number of dependency relationships which the dynamic fault tree is able to represent are limited.

1.5 Research Objectives
The aim of this research is to consider analytical techniques for the efficient representation and generic solution of systems which involve dependency relationships. A range of different types of dependency relationships are investigated in detail. Three distinct types of system will be examined – continuously-operating systems, active-on-demand systems and phased-mission
systems. The focus is placed upon the development of appropriate quantification techniques for each type of system. Also attempts have been made in this research to solve non-repairable systems involving dependencies using the BDD approach. The specific objectives are listed as follows:

- Identify typical types of dependency in all types of systems.
- Establish a generic mechanism to represent different types of dependency.
- Establish an algorithm for developing a Markov model for each type of dependency relationship.
- Improve the efficiency of the dependency modelling by applying the Markov method only to the smallest independent fault tree sections where dependences are self-contained.
- Consider the appropriate quantification technique for each type of system.
- Investigate how the BDD technique can be applied to non-repairable systems involving dependency relationships.
2.1 Fault Tree Analysis (FTA)

2.1.1 Introduction
Since its conception in the 1960s, fault tree analysis has been widely used in the safety and reliability assessment of engineering systems. Its deductive feature helps provide a clear picture of the causal relationships between combinations of component failures and a specific system failure mode. Compared with other assessment methodologies, fault tree analysis features an articulate and organized documentation of the failure logic.

2.1.2 Definition of the Fault Tree structure
A fault tree is a structure by which a particular system failure mode can be expressed in terms of combinations of component failure modes and operator actions [3]. The system failure mode to be considered is termed the 'top event' (TE). The construction of a fault tree is carried out in a deductive or backward way. That is, the system failure mode (TE) is broken down or developed into subsystem failures, which are in turn further developed into lower resolution events or failures. This process is continued until no further development can take place and the limit of resolution is encountered [9]. The development of the fault tree features two types of elements: 'events' and 'gates'. In figure 2.1 and 2.2 below, different types of gates and events are illustrated.

- AND gate: output event occurs if all input events occur simultaneously (number of inputs>=2)
- OR gate: output event occurs if at least one of the input events occurs (number of inputs>=2)
- k-out-of-n gate (voting gate): output event occurs if at least k out of the n input events occur
Priority AND gate: output event occurs if all input events occur in the order from left to right

NOT gate: output event occurs if the input event does not occur

Figure 2.1 Gate types and corresponding symbols

Top/Intermediate event: to be further developed by a logic gate

Basic event: usually representing a specific failure mode of the basic component

House event: either occurring or not

Figure 2.2 Event symbols

Figure 2.3 illustrates a fault tree to demonstrate how the events and gates are connected to each other to present the fault tree structure. In a system analysis each of the text boxes which are at the gate outputs would have a full text description of the events they represent.

Figure 2.3 Example fault tree
2.1.3 Qualitative FTA

A specific system failure (TE) can be caused in different ways. Each unique way is a system failure mode and will involve the failure of individual components or combinations of components. Therefore to analyse a system and to eliminate the most likely causes of failure will first require that each failure mode is identified. Here the concept of the cut-set will be defined to represent the system failure modes.

A cut-set is a collection of basic events such that if they all occur the top event also occurs.

For most industrial engineering systems, there exist a very large number of cut sets each of which can consist of many component failure events. To work out all the cut-sets will be time-consuming and may actually be impractical. Therefore, in FTA, people are only interested in lists of component failure modes which are both necessary and sufficient to produce system failure. And this idea is embodied in the concept of a minimal cut-set, which is a cut set such that if any basic event is removed from the set the top event will not occur, i.e. a minimal cut set is the smallest combination of component failures, which if they all occur will cause the top event to occur.

Any fault tree will represent a finite number of minimal cut sets which are unique for the specific top event. And two fault trees drawn using different approaches are logically equivalent if they produce identical minimal cut-sets. The minimal cut set expression for the top event, TE, can be written in the form:

\[ TE = K_1 + K_2 + K_3 + \ldots + K_n \]  \hspace{1cm} 2.1

where \( K_i, i=1, \ldots, n \), are the minimal cut sets and ‘+’ represents logical OR relationship between \( K_i \).

Each minimal cut set consists of a combination of component failures and a k-component cut set can be expressed as \( K_i = X_1 \cdot X_2 \cdot \ldots \cdot X_k \), where \( X_i \) are basic component failures in the system and ‘\( \cdot \)’ represents logical AND. It should be noted that if the NOT gate is included in the fault tree, the concept of ‘cut set’ will be represented by another term ‘implicant’ [3], and correspondingly, minimal sets of implicants are called prime implicants.

To work out all the minimal cuts is the focus of the qualitative fault tree analysis. This process is carried out by employing the relevant Laws of Boolean Algebra. Since the basic gate types in fault trees, such as the OR gate, the AND gate and the NOT gate, combine events in exactly the
same way as the Boolean operations of ‘union’, ‘intersection’, and ‘complementation’, there is a one-to-one correspondence between Boolean algebraic expressions and the fault tree structure.

2.1.3.1 Relevant Boolean Laws of Algebra

Distributive Laws:

- \( A + (B \cdot C) = (A + B) \cdot (A + C) \)
- \( A \cdot (B + C) = (A \cdot B) + (A \cdot C) \)

Idempotent Laws:

- \( A + A = A \)
- \( A \cdot A = A \)

Absorption Laws:

- \( A + (A \cdot B) = A \)
- \( A \cdot (A + B) = A \)

Complementation:

- \( \overline{A} = 1 - A \)
- \( A \cdot \overline{A} = 0 \)
- \( (A) = A \)

De Morgan's Law:

- \( (A + B) = \overline{A} \cdot \overline{B} \)
- \( (A \cdot B) = \overline{A} + \overline{B} \)

2.1.3.2 Two ways of Obtaining Minimal Cut Sets

The fault tree in figure 2.3 is used here to illustrate the two approaches of calculating the minimal cut sets.

- 'Top-down' Approach:
  With this method, the qualitative analysis will begin with the top event, and develop further by substituting in the Boolean events appearing lower down in the fault tree and simplifying according to Boolean rules until the expression remaining includes only basic events. Then a group of minimal cut sets will be obtained. Take for instance the fault tree in figure 2.3:

\[
T = G1 \cdot G2 \\
= (G3 + A) \cdot (C + G4) \\
= G3 \cdot C + G3 \cdot G4 + A \cdot C + A \cdot G4
\]
\[(B+C).C+(B+C).(A.B)+A.C+A.(A.B)\]
\[=C+(A.B+A.B.C)+A.C+A.A.B\]
\[=C+A.B+A.C+A.B\]
\[=C+A.B\]

Therefore the minimal cut sets of the fault tree in figure 2.3 are \{C\} and \{A.B\}.

- 'Bottom-up' Approach:
  The bottom-up method uses the same substitution, expansion, and reduction methods as the 'top-down' approach. The difference from the 'top-down' approach is that the Boolean operations commence at the base of the fault tree, i.e. the basic events, and work towards the top event. Equations containing only basic failures are successively substituted into each gate. And finally the minimal cut sets will be evaluated.

Of the minimal cut sets, the one-component minimal cut-sets (first-order minimal cut sets) represent single failures, while two-component minimal cut sets (second order) represent double failures which together will cause the top event to occur. In general, the lower-order cut sets contribute most to system failure and effort should be concentrated on the elimination of these in order to improve system performance.

2.1.4 Quantitative FTA
Fault tree quantification will result in predictions of the system performance in terms of component level performance data (probability of failure or frequency of failure). These system performance indicators usually include the following items:

(a) top event probability (unavailability or unreliability);
(b) top event unconditional failure intensity;
(c) top event failure rate;
(d) expected number of top event occurrences in a specified time period;
(e) total system downtime in a specified time period.

Since the fault tree quantification is based on the parameters at component level, the following section will give a brief introduction to the reliability characteristics of basic components in the system.
2.1.4.1 Reliability Models for Component Failures

In engineering systems basic components will feature types of failure which will have different models. Three different types of failure models will be discussed here: fixed, revealed and dormant, which have different inherent and functional characteristics.

- Fixed Failure Model

Basic components which are incorporated in the 'fixed' failure model category are parameterised with a constant unavailability (probability of failure) and unconditional failure intensity, which are usually represented by $q_i$ and $w_i$.

- Revealed Failure Model

With this model 'revealed' means the failure of the basic component will be immediately detected and identified and repair will be carried out on the component straight away. In terms of components falling in this category, a constant conditional failure rate ($\lambda$) and a conditional repair rate ($u$) will be assigned accordingly. The unavailability and unconditional failure intensity of the component will be derived from the following equations [3]:

$$q(t) = \frac{\lambda [1-e^{-\lambda \cdot t}]}{\lambda + u}$$  \hspace{1cm} 2.2

$$w(t) = \lambda [1-q(t)]$$  \hspace{1cm} 2.3

- Dormant Failure Model

The 'dormant' failure model means that the failure of the component is not evident and will not be identified right after it occurs. In reality, most protective/safety system features such a failure mode. With regard to the basic component of this category, constant unrevealed failure rate ($\lambda$), Mean Time To Repair ($\tau$) and scheduled inspection interval ($\theta$) will be assigned. Any failure which occurs between two inspections will remain dormant and unattended till the next inspection. And the average unavailability of the basic component can be approximated by the following equation:

$$q_{AV} = \lambda \cdot (\frac{\theta}{2} + \tau)$$  \hspace{1cm} 2.4
2.1.4.2 Top Event Probability

The Top event probability, also referred to as the system unavailability, provides the likelihood that the system is residing in a state defined by the top event at any specific point of time. Usually, the top event probability is denoted by $Q_{sys}(t)$.

- The general method of calculating $Q_{sys}(t)$

This general approach utilizes the minimal cut sets derived from the previous qualitative analysis. The underlying algorithm is that if a fault tree has $n_C$ minimal cut sets $K_i$, $i=1, 2, \ldots, n_C$, then the top event exists when at least one minimal cut set exists. Translated into the logic function, the algorithm can be represented by:

$$TE = K_1 + K_2 + \ldots + K_{n_C}$$

Therefore:

$$Q_{sys}(t) = P\left(\bigcup_{i=1}^{n_C} K_i \right)$$

By applying the Inclusion-exclusion expansion:

$$Q_{sys}(t) = \sum_{i=1}^{n_C} P(K_i) - \sum_{i=1}^{n_C} \sum_{j=1}^{n_C} P(K_i \cap K_j) + \ldots + (-1)^{n_C-1} P(K_1 \cap K_2 \cap \ldots \cap K_{n_C})$$

where $P(K_i)$ is the probability of the existence of the minimal cut set $K_i$.

Take, for example, the fault tree in figure 2.3, by applying the equation 2.4 to the two minimal cut sets $\{C\}$ and $\{A, B\}$, the system unavailability would be obtained as:

$$Q_{sys}(t) = P(K_1) + P(K_2) - P(K_1 \cap K_2)$$

$$= P(C) + P(A, B) - P(A, B, C)$$

$$= q_c(t) + q_A(t).q_B(t) - q_A(t).q_B(t).q_C(t)$$

For moderate-sized systems, calculating each term in the inclusion-exclusion expansion in equation 2.4 could be very time-consuming and for large systems, could be an intractable task. Since the first term is numerically more significant than the second one, the second one more significant than the third and so on, truncation and approximation techniques can be employed to simply the process of calculating the system unavailability.
- Approximation of system unavailability

1). Upper and lower bounds for system unavailability

In equation 2.5, truncation from the second term will give a lower bound of the system unavailability and the truncation from the first term will give an upper bound of the top event probability.

\[
\sum_{i=1}^{n_e} P(K_i) - \sum_{i=2}^{n_e} \sum_{j=1}^{i-1} P(K_i \cap K_j) \leq Q_{sys}(t) \leq \sum_{i=1}^{n_e} P(K_i)
\]

lower bound  exact  upper bound

The upper bound used here is known as the Rare Event Approximation since it would be very close to the exact system unavailability if the component failure events are rare.

2). Minimal cut set upper bound

Another more accurate upper bound for top event unavailability is the minimal cut set upper bound. This upper bound is derived from:

\[
P(\text{system failure}) = P(\text{at least 1 minimal cut set occurs})
\]

\[
= 1 - P(\text{no minimal cut set exists})
\]

since:

\[
P(\text{no minimal cut set exists}) \geq \prod_{i=1}^{n_e} P(\text{minimal cut set i does not occur})
\]

(equality achieved when no basic event appears in more than one minimal cut set)

Therefore:

\[
P(\text{system failure}) \leq 1 - \prod_{i=1}^{n_e} P(\text{minimal cut set i does not occur}),
\]

i.e.

\[
Q_{sys}(t) \leq 1 - \prod_{i=1}^{n_e} [1 - P(K_i)]
\]

For the system unavailability it shows that:

\[
Q_{sys}(t) \leq 1 - \prod_{i=1}^{n_e} [1 - P(K_i)] \leq \sum_{i=1}^{n_e} P(K_i)
\]

exact  minimal cut set  rare event  upper bound  approximation
2.1.4.3 Top Event Frequency

The system unconditional failure intensity, which is denoted by \( w_{sys}(t) \), is defined as the probability that the top event occurs per unit time at \( t \) given the system was working at \( t=0 \). Therefore \( w_{sys}(t)dt \) stands for the probability that the top event occurs during the time interval \([t, t+dt)\) given that it did not exist at \( t=0 \).

The procedure to calculate the top event frequency is the same as that to calculate the system unavailability. First component failure parameters will be calculated. The minimal cut set parameters will then be obtained and finally the top event frequency will be derived based on the minimal cut set parameters. The following section illustrates the detailed process of calculating the unconditional failure intensity of the minimal cut set.

2.1.4.3.1 Unconditional Failure Intensity of Minimal Cut Set

The unconditional failure intensity of minimal cut set, denoted by \( w_C(t) \), is the probability that the minimal cut set occurs per unit time at \( t \). Correspondingly, \( w_C(t)dt \) represents the probability that the minimal cut set occurs during the time interval \([t, t+dt)\). This definition implies that the minimal cut set does not exist at time \( t \). Therefore under the assumption that during a very short time interval \( dt \) only one component in the minimal cut set will fail, the minimal cut set frequency can be expressed by equation 2.8 below:

\[
 w_C = \sum_{i=1}^{n} \{ w_i(t) \prod_{j=1}^{n} Q_j(t) \} 
\]  

where \( w_i(t) \) represents the frequency of basic event \( i \) included in the minimal cut set, \( Q_j(t) \) represents the probability of failure of basic event \( j \) at time \( t \), and \( n \) is the number of events in the minimal cut set.

Since \( w_i(t) \) implies that at time \( t \) basic event \( i \) does not exist, equation 2.8 ensures that at time \( t \) minimal cut set does not exist. Take for instance the fault tree in figure 2.3, the frequency of its two minimal cut sets can be calculated as follows:

Minimal cut set 1: \{C\}

\[
 w_{C_1}(t) = w_C(t) 
\]

Minimal cut set 2: \{A, B\}

\[
 w_{C_2}(t) = q_B(t)w_A(t) + q_A(t)w_B(t) 
\]

16
2.1.4.3.2 Calculation of Top Event Frequency

The definition of top event frequency implies that no minimal cut set can exist at time \( t \), and during the time interval \([t, t+\Delta t)\) at least one minimal cut set \( C_i \) must occur. The corresponding numerical expression can be written as:

\[
w_{sys}(t)\Delta t = P\left[ A \bigcup_{i=1}^{n_c} C_i \right]
\]  \hspace{1cm} \text{(2.9)}

where \( A \) is the event that no minimal cut set exists at time \( t \) and \( \bigcup_{i=1}^{n_c} C_i \) is the event that one or more minimal cut sets \( C_i \) occur during the interval \([t, t+\Delta t)\).

Since \( P(A)=1-P(\overline{A}) \), equation 2.9 can also be written as:

\[
w_{sys}(t)\Delta t = P\left[ \bigcup_{i=1}^{n_c} C_i \right] - P\left[ A \bigcup_{i=1}^{n_c} C_i \right]
\]  \hspace{1cm} \text{(2.10)}

Let \( \mu_i \) represent the event that the \( i \)th minimal cut set does not exist at time \( t \). So:

\[
\overline{A} = \bigcup_{i=1}^{n_c} \overline{\mu_i}
\]

Denote the two terms on the right hand side of equation 2.10 by \( w_{sys}^{(1)}(t)\Delta t \) and \( w_{sys}^{(2)}(t)\Delta t \) respectively, so equation 2.10 can be written as:

\[
w_{sys}(t)\Delta t = w_{sys}^{(1)}(t)\Delta t - w_{sys}^{(2)}(t)\Delta t
\]  \hspace{1cm} \text{(2.11)}

where \( w_{sys}^{(1)}(t)\Delta t \) represents the contribution to system failure from the occurrence of at least one minimal cut set and \( w_{sys}^{(2)}(t)\Delta t \) represents the contribution of minimal cut sets occurring while other minimal cut sets already exist which should be deducted from the first term.

Then each of these two terms can be further developed separately. Expanding the first term will give the following series expression:

\[
w_{sys}^{(1)}(t)\Delta t = P\left[ \bigcup_{i=1}^{n_c} C_i \right]
\]  \hspace{1cm} \text{(2.12)}

\[
= \sum_{i=1}^{n_c} P(C_i) - \sum_{i=2}^{n_c} \sum_{j=1}^{i-1} P(C_i \cap C_j) + \ldots + (-1)^{n_c-1} P[C_1 \cap C_2 \cap \ldots \cap C_{n_c}]
\]
where $\sum_{i=1}^{n_c} P(C_i)$ is the sum of the probabilities that minimal cut set $i$ occurs during $[t, t+dt)$. All other terms involve the simultaneous occurrence of two or more minimal cut sets which results from the repeated basic events contained in more than one minimal cut set.

The second term in equation 2.11 has a more involved expansion:

$$w_{sys}^{(2)}(t)dt = P[\bigcup_{i=1}^{n_c} C_i]$$

$$= \sum_{i=1}^{n_c} P(C_i \cap \overline{A}) - \sum_{i=2}^{n_c} \sum_{j=1}^{i-1} P(C_i \cap C_j \cap \overline{A}) + ...$$

$$+ (-1)^{e-1} P(C_1 \cap C_2 \cap ... \cap C_{e-1} \cap \overline{A})$$

and each term in equation 2.13 can be further expanded by expanding the element $\overline{A}$.

Take for example the fault tree in figure 2.3, with the two minimal cut sets $\{C\}$ and $\{A, B\}$, the top event frequency is calculated as follows:

$$w_{sys}(t)dt = w_{sys}^{(1)}(t)dt - w_{sys}^{(2)}(t)dt$$

$$w_{sys}^{(1)}(t)dt = w_{C_1}(t)dt + w_{C_2}(t)dt - P[C_1 \cap C_2]$$

$$= w_{C_1}(t)dt + w_{C_2}(t)dt$$

$$w_{sys}^{(2)}(t)dt = P(C_1, \overline{A}) + P(C_2, \overline{A}) - P(C_1, C_2, \overline{A})$$

$$= P(C_1, \mu_1) + P(C_2, \mu_2) - P(C_1, \mu_1, \mu_2) +$$

$$P(C_2, \mu_1) + P(C_2, \mu_2) - P(C_2, \mu_1, \mu_2) - 0$$

$$= w_{C_1}(t)dtq_c(t) + w_{C_2}(t)dtq_c(t)$$

2.14.3.3 Approximation for the System Unconditional Failure Intensity

As with the calculation of the top event probability, to calculate each term of the expanded version of equation 2.10 could turn out to be intractable for large systems. Therefore, approximation is applied in order that an acceptably accurate top event frequency will be produced using less computational resources.

By approximating the second term in equation 2.11 by zero, the upper bound for $w_{sys}(t)$ can be obtained:

$$w_{sys_{max}}(t)dt = w_{sys}^{(1)}(t)dt$$
By applying the rare event approximation technique, one upper bound can be expressed as in equation 2.15:

$$w_{sys}(t)dt = P[n_C = P\left[\bigcup_{i=1}^{n_c} C_i \right] \leq \sum_{i=1}^{n_c} w_{C_i}(t)dt]$$  \hspace{1cm} (2.15)

Alternatively another upper bound for top event frequency can be derived:

$$w_{sys}(t) \leq \sum_{i=1}^{n_c} w_{C_i}(t) \prod_{j \neq i} \left[1 - Q_{C_j}(t)\right]$$  \hspace{1cm} (2.16)

In equation 2.16 the term on the right hand side represents that the top event occurs during \([t, t+dt]\) due, in turn, to the occurrence of minimal cut set \(C_i\). All other minimal cut sets have not occurred at that time. The equality will hold if all minimal cut sets are independent from each other.

2.1.4.3.4 Expected Number of System Failures

The expected number of system failures, i.e. the expected number of top event occurrences, during time \(t_I\), \(W_{sys}(0, t_I)\), is obtained by integrating the system unconditional failure intensity over the time interval \([0, t_I]\):

$$W_{sys}(0, t_I) = \int_{0}^{t_I} w_{sys}(t)dt$$  \hspace{1cm} (2.17)

The expected number of system failures is an upper bound for the system unreliability \(F(t)\). When system failure is rare, this approximation is a close upper bound [3].

2.1.4.4 Importance Measures

The importance measure is another very useful piece of information which can be extracted from a system reliability assessment. The importance analysis will identify weak areas of the system and signify the role each component plays in producing system failure. Importance measures will also provide useful information for maintenance and diagnosis purposes.

Importance measures can be categorized in two ways:

(a) Deterministic
(b) Probabilistic
2.1.4.4.1 Deterministic Measures

Deterministic measures assess the importance of a component to the system operation without considering the components probability of failure.

The structural measure of importance is one of the deterministic measures, defined for a component $i$ as:

$$I_i = \frac{\text{number of critical system states for component } i}{\text{total number of states for the } (n-1) \text{ remaining components}}$$  \hspace{1cm} 2.18

where the critical system state for component $i$ refers to a state for the remaining components such that the failure of component $i$ causes the system to go from a working to a failed state.

2.1.4.4.2 Probabilistic Measures

In contrast with the deterministic measure, probabilistic importance measures take into account the component failure likelihood.

- **Birnbaum's Measure of Importance**

  This importance measure is also known as the criticality function. The criticality function for a component $i$ is denoted by $G_i(q)$ and is defined as the probability that the system is in a critical state for component $i$. Therefore it is the sum of the probabilities of occurrence of the critical system states for component $i$ [1].

  There exist two expressions for this importance measure:

  1). $G_i(q) = Q_1(q) - Q_0(q)$  \hspace{1cm} 2.19

  where:

  $Q_1(q)$ is the probability of system failure given component $i$ is failed, i.e. the failure probability of component $i$ is set as 1.

  $Q_0(q)$ is the probability of system failure given component $i$ is working, i.e. the failure probability of component $i$ is set as 0.

  This equation gives the probability that the system fails only if component $i$ fails.

  2). $G_i(q) = \frac{\partial Q(q)}{\partial q_i}$  \hspace{1cm} 2.20

  This expression is actually the same as the one given in 2.19 as:
The definition of Birnbaum’s measure of importance has formed the basis for many other probabilistic importance measures.

- **Criticality Measure of Importance**
This importance measure is defined as the probability that the system is in a critical state for component $i$, and $i$ has failed (weighted by the system unavailability):

$$I_i = \frac{G_i(q(t))q_i(t)}{Q_{sys}(q(t))}$$

2.22

The criticality measure of importance implies the situation where component $i$ fails at the point of time and the combination of other component states forms the critical state for component $i$, and in such a system state, the system fails. The critical state here means that if component $i$ gets repaired at that time, the system will also go back to working state because the failure or working of the system at that time is totally determined by the state of component $i$. Therefore, a practical interpretation of the criticality measure is that when it is known the system is failed at time $t$, it gives the probability that the system will be restored by repair only component $i$. That is, this importance measure provides useful information to the maintenance engineer when the system is failed at time $t$, about which component should be attended first to reduce the system down-time to as shortest as possible.

- **Fussell-Vesely Measure of Importance**
This measure is defined as the probability of the union of the minimal cut sets containing component $i$ given that the system has failed:

$$I_i = \frac{P(\bigcup_{k \in C_i} C_k)}{Q_{sys}(q(t))}$$

2.23

This importance measure is numerically very similar to the criticality measure of importance and also provides the assessment of the contribution of the failure of component $i$ to the system failure.

- **Fussell-Vesely Measure of Minimal Cut Set Importance**
Different from other probabilistic importance measures discussed above, this measure provides the ranking system for minimal cut sets rather than the individual basic
components. This measure is defined as the probability of occurrence of minimal cut set \( i \) given that the system has failed:

\[
I_i = \frac{P(C_i)}{Q_{sy}(q(t))}
\]

2.1.5 Summary of Fault Tree Analysis
Fault tree analysis provides an excellent means to represent the failure logic of a specific system failure mode. Its diagrammatic form helps greatly to articulate and communicate the analysis. The quantification performed on the fault tree structure provides a range of system reliability parameters which are essential in the process of design and safety evaluation.

However, for large fault trees, the analysis can be very time-consuming and require considerable computing power. To overcome this weakness, approximation techniques are introduced which simplify the quantification process at the expense of accuracy. Much research has been carried out to find more efficient and accurate analysis tools, of which Binary Decision Diagram (BDD) is very powerful and popular.

2.2 Binary Decision Diagram – BDD

2.2.1 Introduction
The very early application of Binary Decision Diagrams (BDDs) can be found in Lee’s work [10] in which the BDD was used to represent switching circuits. Its use in reliability assessment was predominantly prompted by Rauzy [4] and later the BDD has proven to be a more efficient and accurate technique for performing fault tree analysis [11, 12].

The BDD approach draws on the fault tree structure, although it does not analyse the fault tree directly. The application of the BDD method requires the conversion of the fault tree structure to a binary decision diagram, which represents the same system failure logic. As the BDD is a more explicit representation of the Boolean equation, it is much better suited to mathematical analysis than the fault tree structure. Capable of performing both qualitative and quantitative analysis, the BDD can work out the system reliability solutions very efficiently without having to sacrifice the accuracy of the results.
2.2.2 Properties of BDD

A BDD is a directed acyclic graph composed of terminal and non-terminal vertices (also called nodes) connected by branches. The non-terminal vertices encode the basic events of the fault tree and the terminal vertices correspond to the final state of the system. Non-terminal vertices have two outgoing branches. By convention, the left-hand branch is a ‘1’ branch, corresponding to the occurrence of the basic event (i.e. the component has failed); the right-hand branch is a ‘0’ branch corresponding to the basic event non-occurrence (i.e. the component is working). Terminal vertices have a value of either ‘1’ or ‘0’, corresponding to top event occurrence (i.e. the system failure) and non-occurrence (i.e. the system works) respectively.

Below is an example BDD to illustrate the basic structure of binary decision diagram:

![Example BDD Diagram](image)

In this binary decision diagram, by following the paths from the root vertex leading to the terminal node ‘1’, we can get the cut sets representing the combinations of component failures which result in the system failure:

1. \{a, b\}
2. \{a, c\}

Since in this case these cut sets cannot be simplified, they represent the minimal cut sets. It should be noted that only when the BDD is in its minimal form \[4\], can the qualitative analysis always produce the corresponding minimal cut sets.

2.2.3 Formation of BDD

The process of forming a BDD is essentially to convert the fault tree to a corresponding BDD. The first stage of this conversion process is to establish a proper variable ordering for the collection of basic events included in the fault tree. By determining the order in which the
variables, i.e. basic event nodes, appear in the BDD, the ordering can have a crucial effect on the size of the resulting BDD and thus the complexity of the calculation required for its construction and solution.

Take for instance the BDD in figure 2.4, it can be identified from the structure of this BDD that the variable ordering between the three basic events is $a < b < c$. If the ordering is changed to $b < a < c$, the resulted BDD will be established as shown in figure 2.5. The difference in the size of resulting BDD for this trivial example is not large. However for large systems this effect becomes very much more influential.

![Figure 2.5 Example BDD of different ordering](image)

There exist different schemes for solving the variable ordering problem [13-18]. Each approach has its own strengths and weaknesses. There is not such a thing as 'the best for all'. The structural features of the fault tree which is to be converted has to be fully understood so that the more appropriate ordering scheme can be chosen.

### 2.2.3.1 Construction of BDD

#### 2.2.3.1.1 Construction of BDD Using Structure Function

This method of forming the BDD uses the structure function of the corresponding fault tree. Once the variable ordering has been determined, values of '1' and '0' will be substituted into the structure function according to the ordering to replace the basic event which represents each node in the BDD. The fault tree in figure 2.3 is used here to illustrate this process. With the two minimal cut sets \{C\} and \{A, B\}, the structure function of the fault tree is:

$$\varphi = 1 - (1-x_C)(1-x_A x_B)$$
The variable ordering is chosen as $A < C < B$. The ordering means that during the process of the BDD construction, event ‘A’ will be considered first, then event ‘C’ and finally event ‘B’ until the BDD has been fully established. The resulting BDD with each node accompanied with the corresponding structure function is shown in figure 2.6:

$$1 - (1 - x_C)(1 - x_A x_B)$$

![Figure 2.6 BDD construction using structure function](image)

The cut sets this BDD gives are $\{A.C\}$, $\{A.B\}$ and $\{C\}$. As the BDD is not in its minimal form, it includes cut set $\{A.C\}$ which should be deleted to obtain the minimal cut sets.

### 2.2.3.1.2 Construction of BDD Using If-Then-Else

This method of BDD construction was developed by Rauzy [4] and proceeds by applying the if-then-else (ite) structure to each gate in the fault tree. The ite structure is expressed as $\text{ite}(X_1, f_1, f_2)$, where $X_1$ is the Boolean variable, i.e. the basic event node in the BDD, and $f_1$ and $f_2$ are logic functions. This structure means that the state of $X_1$ will be examined and if $X_1$ fails, look into function $f_1$, else look into $f_2$. Correspondingly in the BDD, $f_1$ is connected to the node encoding ‘$X_1$’ by its ‘1’ branch and $f_2$ connected by its ‘0’ branch. This is illustrated in figure 2.7.

![Figure 2.7 ite structure in BDD](image)
By referring to the chosen variable ordering, the following procedure can be carried out to construct the BDD using ite method:

- Each basic event $X_i$ is assigned the ite structure $\text{ite}(X_i, 1, 0)$.
- For each gate (starting in a bottom-up progression of the fault tree), Let $J = \text{ite}(X, f_1, f_2)$ and $H = \text{ite}(Y, g_1, g_2)$, then:
  - If $X < Y$ (i.e. variable $X$ appears prior to $Y$ in the ordering)
    \[ J <\text{op}> H = \text{ite}(X, f_1<\text{op}>H, f_2<\text{op}>H). \]
  - If $X = Y$, then: $J <\text{op}> H = \text{ite}(X, f_1<\text{op}>g_1, f_2<\text{op}>g_2)$

Where $<\text{op}>$ represents the logic operator corresponding to the gate type in the fault tree, and the following simplification can be implemented according to the value of the logic operator $<\text{op}>$:

- If $<\text{op}> = '+$': $1<\text{op}>H = 1; 0<\text{op}>H = H$
- If $<\text{op}> = '.'$: $1<\text{op}>H = H; 0<\text{op}>H = 0$

It can be seen that an advantage of the ite method is that the ite structure of each gate and basic event can be determined in advance and once an ite structure has been calculated, it can be stored for later reference so the process does not need to be repeated. Such a characteristic not only reduces the computer memory requirement but also increases the calculation efficiency.

The procedure described above can be demonstrated by applying the ite method to the fault tree in figure 2.3:

The chosen variable ordering is $A < C < B$, by translating each gate of the fault tree into the ite structure in the bottom-up way, we can finally get the ite structure for the top event:

- $G_3 = C + B$
  \[
  = \text{ite}(C, 1, 0) + \text{ite}(B, 1, 0)
  = \text{ite}(C, 1, \text{ite}(B, 1, 0))
  
  \]
- $G_4 = A.B$
  \[
  = \text{ite}(A, 1, 0) \cdot \text{ite}(B, 1, 0)
  = \text{ite}(A, \text{ite}(B, 1, 0), 0)
  
  \]
- $G_1 = A + G_3$
  \[
  = \text{ite}(A, 1, 0) + \text{ite}(C, 1, \text{ite}(B, 1, 0))
  = \text{ite}(A, 1, \text{ite}(C, 1, \text{ite}(B, 1, 0)))
  
  \]
\[ G2 = C + G4 \]
\[ = \text{ite}(C, 1, 0) + \text{ite}(A, \text{ite}(B, 1, 0), 0) \]
\[ = \text{ite}(A, \text{ite}(C, 1, 0) + \text{ite}(B, 1, 0), \text{ite}(C, 1, 0)) \]
\[ = \text{ite}(A, \text{ite}(C, 1, \text{ite}(B, 1, 0)), \text{ite}(C, 1, 0)) \]

\[ \text{TE} = G1.G2 \]
\[ = \text{ite}(A, 1, \text{ite}(C, 1, \text{ite}(B, 1, 0))). \text{ite}(A, \text{ite}(C, 1, \text{ite}(B, 1, 0)), \text{ite}(C, 1, 0)) \]
\[ = \text{ite}(A, \text{ite}(C, 1, \text{ite}(B, 1, 0)), \text{ite}(C, 1, \text{ite}(B, 1, 0))). \text{ite}(C, 1, 0)) \]
\[ = \text{ite}(A, \text{ite}(C, 1, \text{ite}(B, 1, 0)), \text{ite}(C, 1, 0)) \]

The next step is to construct the BDD according to the ite structure calculated for the top event of the fault tree. From the ite structure basic event 'A' is the first variable to be considered and so forms the root node of the corresponding BDD. Its left branch and right branch will be constructed respectively according to the 'then' and 'else' term included in the corresponding ite structure. For example, the first variable to be considered for the left branch of node 'A' is basic event 'C' as shown in the ite(C, 1, ite(B, 1, 0)). Such a process will be repeated till each path of the BDD reaches the terminal node. The resulting BDD is shown in figure 2.8.

![Figure 2.8 BDD construction using ite method](image)

2.2.4 Quantitative Analysis of Binary Decision Diagram

2.2.4.1 Top Event Probability/System Unavailability \( Q_{\text{sys}}(t) \)

The underlying algorithm of constructing a BDD determines that the structure function for the top event can be expressed through the ite (if-then-else) structure as:

\[ f(x) = \text{ite}(x_i, f_1, f_2) \]
\[ = x_i.f_1(x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n) + \bar{x}_i.f_2(x_1, x_2, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) \]
where: \( x_i \) is the pivoting variable;

\( f_1 \) and \( f_2 \) are Boolean functions with \( x_i = 1 \) and \( x_i = 0 \) respectively.

Let \( f(x) \) represent the root vertex of the BDD which encodes the basic event \( x_i \), then the expectation value of \( f(x) \) is actually the top event probability. To evaluate the probability of the top event represented by \( f(x) \), the expectation value of \( f_1 \) and \( f_2 \) needs to be worked out. Interestingly, \( f_1 \) and \( f_2 \) which are encoded as two discrete nodes in the BDD can be expressed in the same way as \( f(x) \) through the decomposition of the ‘1’ branch and ‘0’ branch respectively. Such decomposition process could be repeatedly applied until the \( \text{ite} \) structure of each non-terminal vertex in the BDD is decided. In this way, the expectation value of \( f(x) \) can then be decided by summing the probabilities of all paths through the BDD. Each path represents a combination of working and failed components that leads to system failure.

Equation 2.25 can be transformed into equation 2.26 to get the ‘probability value’ of each node in the BDD. The ‘probability value’ of the root vertex of the BDD is equivalent to the top event probability, i.e. system unavailability. For any other node in the BDD it is simply used for the calculation purpose and has no physical significance. For any BDD node, \( F = \text{ite}(x_i, J, K) \), the probability value is given by:

\[
P[F] = q_i(t).P[J] + (1-q_i(t)).P[K]
\]

Equation 2.26 is applied to the BDD in a bottom-up manner. The probability value of terminal ‘1’ and ‘0’ vertices are simply 1 and 0 respectively. The values are then worked up through the BDD until the top event probability is obtained. Take for instance the BDD in figure 2.8:
In the application of equation 2.26, because the expectation value of the terminal node ‘0’ is set as 0, the process of carrying out the calculation is actually equivalent to summing the probabilities of all the paths which lead to the occurrence of the top event, i.e. the terminal node ‘1’. For example, in the BDD in figure 2.8, there exist 3 paths which end with the terminal node ‘1’. The top event probability can then be expressed as the sum of the probabilities of these 3 paths:

\[ Q_{sys}(t) = q_A(t).q_C(t) + q_A(t).[1-q_C(t)].q_B(t) + [1-q_A(t)].q_C(t) \]

\[ = q_C(t) + [1-q_C(t)].q_A(t).q_B(t) \]

2.2.4.2 Unconditional System Failure Intensity \( w_{sys}(t) \)

For the calculation of \( w_{sys}(t) \), the criticality function for each component is calculated and used in equation 2.27. The original definition of the criticality function is given in section 2.1.4.4.2.

\[ w_{sys}(t) = \sum_i G_i(q(t)).w_i(t) \tag{2.27} \]

where \( G_i(q(t)) \) is the criticality function for component \( i \)

\( w_i(t) \) is the component unconditional failure intensity

The method of calculating the criticality function in the BDD will be discussed in the later section 2.2.4.4.

2.2.4.3 Unconditional System Repair Intensity \( v_{sys}(t) \)

The unconditional system repair intensity, \( v_{sys}(t) \) is obtained from BDD by referring to the same underlying algorithm. The only difference here is the ‘repair criticality function’, \( GR_i(q(t)) \), is required. By referring to the definition of the criticality function represented in equation 2.19, repair criticality function for component \( i \) can be given by:

\[ GR_i(q(t)) = (1-Q(0_i, q(t))) - (1-Q(1_i, q(t))) \tag{2.28} \]

where: \( 1-Q(0_i, q(t)) \) – the probability of system working with the failure probability of component \( i \) being 0;

\( 1-Q(1_i, q(t)) \) – the probability of system working with the failure probability of component \( i \) being 1
By simplifying the right-hand side of the equation 2.28, it can be concluded that the repair criticality function is equivalent to the criticality function.

On investigation of the term ‘critical state’ in the definition of the criticality function, it can be seen that, regarding one particular component, the system is on the brink of state change, whether its current state is working or failed. If the emphasis is placed on the failure process, the criticality function will be interpreted as the probability that the system will fail due to the failure of component i. However, if it is the repair process that is being examined, the criticality function is equivalent to the probability that the system will be restored due to the repair of component i.

The system unconditional repair intensity can be given by:

\[ v_{sys}(t) = \sum_i G_i(q(t)).v_i(t) \]

where \( G_i(q(t)) \) is the criticality function for component i

\( v_i(t) \) is the component unconditional repair intensity

2.2.4.4 Criticality Function of Each Basic Event

Since the criticality function, \( G_i(q(t)) \), is required in the calculation of both system unconditional failure intensity and repair intensity, it is important to work out an efficient method of evaluating the criticality function for each component in the BDD.

The criticality function is calculated from the BDD by considering the probabilities of the path sections in the BDD up to and after the relevant nodes representing the indication variable for component i. For instance, as shown in figure 2.10, the basic event \( x_i \) occurs at two intermediate nodes in the BDD:

![Figure 2.10 BDD section containing event \( x_i \)](image-url)
Q(1, q(t)) and Q(0, q(t)) can be calculated for x_i by:

\[ Q(1, q(t)) = \sum_n pr_{x_i}(q(t)) \cdot po^1_{x_i}(q(t)) + Z(q(t)) \]  
\[ Q(0, q(t)) = \sum_n pr_{x_i}(q(t)) \cdot po^0_{x_i}(q(t)) + Z(q(t)) \]  

where: 
- \( pr_{x_i}(q(t)) \) - the probability of the path section from the root vertex to the node \( x_i \) (set to 1 for the root vertex). 
- \( po^1_{x_i}(q(t)) \) - the probability value of the node beneath the ‘1’ branch of a node encoding event \( x_i \). 
- \( po^0_{x_i}(q(t)) \) - the probability value of the node beneath the ‘0’ branch of a node encoding event \( x_i \). 
- \( Z(q(t)) \) - the probability of paths from the root vertex to the terminal ‘1’ node that do not go through a node encoding \( x_i \). 
- \( n \) - the number of nodes encoding \( x_i \) in the BDD.

By substituting equation 2.30 and 2.31 into equation 2.19, the criticality function for each event can then be expressed as:

\[ G_i(q(t)) = \sum_n pr_{x_i}(q(t)) \cdot [po^1_{x_i}(q(t)) - po^0_{x_i}(q(t))] \]  

The summation implies that the algorithm must calculate \( pr_{x_i}(q) \), \( po^1_{x_i}(q) \) and \( po^0_{x_i}(q) \) for each node which encodes \( x_i \).

The values of \( pr[F], po^1[F] \) and \( po^0[F] \) (known collectively as the ‘path probabilities’) are calculated during one depth-first pass of the BDD, which explores the structure beneath the ‘1’ branch of any node before returning to consider the ‘0’ branch. Starting with the root vertex, values of \( pr[F] \) are evaluated and assigned to each node as the branches are descended. Once the foot of a branch is reached, the procedure continues by working back up through the BDD calculating values of \( po^1[F] \) and \( po^0[F] \) for each of the nodes.

It must be noted that in the BDD one node may be reached by more than one path due to sub-node sharing. In this case, its value of \( pr[F] \) needs to include the probabilities of all the possible path sections from the root vertex to that node. To enable such a summation in the depth-first process, a label must be assigned to each node in the BDD to record and indicate if the node has been visited before. If the node has not been visited, its value of \( pr[F] \) will be established as the equivalent of the probability of the path which links the root vertex to the node in question. If the
node has been visited, the value of $pr[F]$ will be updated by adding the probability of the path currently being traced.

For instance, as shown in figure 2.11, the node encoding $x_1$ can be reached by two paths:

![Figure 2.11 BDD section reachable by two different paths](image)

In the depth-first pass, node $F_1$ will be reached first via path a and the value of $pr_a$ will be assigned to $pr[F_1]$. And by following the ‘1’ branch of $F_1$, $pr[F_2]$ can be obtained as $q_1.pr_a$. Node $F_1$ will then be reached via path b for the second time. Since $F_1$ has been visited before, the value of $pr[F_1]$ then will be updated to $(pr_a+pr_b)$. When node $F_2$ is visited again, the final value of $pr[F_2]$ will be $q_1.(pr_a+pr_b)$.

Let $F_1$ and $F_2$ be the nodes which are respectively under the branch ‘1’ and branch ‘0’ of node $F$, then $po^1[F]$ and $po^0[F]$ actually refer to the probability value of node $F_1$ and $F_2$, i.e. $P[F_1]$ and $P[F_2]$. In this way, the calculation of the probability value $P[F]$ of each node and the path probabilities $pr[F]$, $po^1[F]$ and $po^0[F]$ can then be performed in one first depth-first pass through the BDD.

After the values of the three terms $pr[F]$, $po^1[F]$ and $po^0[F]$ of all the nodes in the BDD have been evaluated, another depth-first pass is carried out to calculate the criticality function for each event according to equation 2.32 by identifying all the nodes which represent each of the basic events.

2.4.4.5 Worked example

An example is given here to illustrate the whole process of obtaining the top event probability and the criticality function for each event. Take for instance the fault tree in figure 2.3, with
variable ordering $A < B < C$, the corresponding BDD will be constructed as shown in figure 2.12:

![BDD representation of the fault tree in figure 2.](image)

Figure 2.12 BDD representation of the fault tree in figure 2.

$pr[F1] = 1$;

Top event probability $= P[F1]$;

$P[F1] = pr[F2] = pr[F1].q_A = q_A$  
$F2$ visited

$po^1[F1] = P[F2]$;

$pr[F3] = q_A.(1-q_B) + (1-q_A).pr[F1]$  
$= 1-q_A.q_B$  
$po^0[F1] = P[F3]$;

$P[F1] = q_A.po^1[F1] + (1-q_A).po^0[F1]$  
$= q_A.q_B + q_A.(1-q_B).q_C + (1-q_A).q_C$  
$= q_A.q_B + q_C - q_A.q_B.q_C$

$po^1[F2] = 1$;

$pr[F3] = pr[F2].(1-q_B) = q_A.(1-q_B)$  
$F3$ visited

$po^0[F2] = P[F3]$;

$P[F2] = q_B.1 + (1-q_B).P[F3]$  
$= q_B + (1-q_B).q_C$

$po^1[F3] = 1$;

$po^0[F3] = 0$;

$P[F3] = q_C.1 + (1-q_C).0 = q_C$

Criticality function for basic event $A$:

$G_A(q(t)) = pr[F1].(po^1[F1] - po^0[F1])$  
$= 1.(q_B + (1-q_B).q_C - q_C)$  
$= q_B.(1-q_C)$
Criticality function for basic event B:
\[ G_B(q(t)) = \text{pr}[F2].(p^1[F2] - p^0[F2]) = q_A.(1 - q_C) \]

Criticality function for basic event C:
\[ G_C(q(t)) = \text{pr}[F3].(p^1[F3] - p^0[F3]) = 1 - q_A.q_B \]

2.2.5 Summary of Binary Decision Diagram

The major advantage of the BDD lies in that it provides both an efficient and an accurate means of analysing the system failure probability, without the need for the approximations previously used in the conventional methods of Kinetic Tree Theory. This explains the increasing popularity and wide use of BDD in the system reliability assessment.

However, BDD is not perfect. The difficulty with this technique is that it requires a variable ordering during the process of converting the fault tree to the BDD. Although many variable ordering schemes have been developed and become increasingly mature in their application, the difficulty still exists in constructing the BDDs for some large and complex fault trees. Apart from selecting the most appropriate ordering scheme by identifying the unique feature of the fault tree structure, another effective solution to this problem is to simplify and to modularise the fault tree before the conversion is implemented. This process will be discussed in detail in Chapter 5.
System dependency modelling refers to the process of analysing and evaluating the reliability of a system in which an inter-dependence between the basic components exists in terms of their failure or/and repair. In reality, many complex engineering systems feature this type of dependent relationship between components.

Some types of dependency relationship arise due to the fact that the reliability characteristics of one component will affect those of another component. A typical example of this type of dependency can be identified wherever cold or warm standby redundancy [5] is employed. Depending on the state of the duty component, the failure rate of the standby component will vary. It will experience a higher failure rate when called upon to function than when inactive in standby. Another type of dependency occurs in systems for which the Priority AND gate will be involved in the corresponding fault tree structure. Different from the normal AND gate, the Priority AND gate gives rise to a dependency relationship between basic events grouped under it and requires a specific order of their occurrences to cause its output. A more detailed description and illustration of each type of dependency can be found in chapter 4.

As far as the systems are concerned which feature such a dependency property, some techniques, such as fault tree analysis and the binary decision diagram method, would be inadequate because the quantification algorithms assume that the basic events occur independently. These techniques are not able to take into account the dependencies. The consequence of ignoring or being unable to identify the dependency in the system would result in the incorrect system reliability prediction. Therefore an appropriate modelling technique is required in order to solve the problems resulting from the dependency during the process of evaluating system risk and reliability. Such method is the Markov method.

### 3.1 Markov Method

#### 3.1.1 Basic Elements of Markov Models

Each Markov model is composed of two basic elements:

- A set of states;
- A set of transitions between the states;
The Markov model operates in the following way: the system is envisioned as being in one of the states at all times throughout the time period of interest. The system can be in only one state at a time, and from time to time it makes a transition from one state to another state by following one of the set of inter-state transitions. Depending on how the transitions can occur regarding the time domain, a general categorization can be implemented to distinguish two different types of Markov models. If the transitions can occur at any real-valued time, the model is called a Continuous Time Markov Chain (CTMC). Decided by the inherent probabilistic characteristic embedded in the process of system risk assessment, it is the Continuous Time Markov Chain that will be applied in the analysis. Unless specifically noted, all the Markov models discussed in this thesis refer to the Continuous Time Markov Chain.

As far as the CTMC is concerned, another element is required to make a complete and meaningful Markov model apart from the two elements mentioned above. This third element is the series of transition rates, each of which corresponds to every specific transition and governs the length of the time that elapses before the system moves from the originating state to the target state of the transition, i.e. the state holding time.

When Markov models are used in the process of evaluating the system risk and reliability, the states frequently represent every possible combination of the status of each component, working or failed. Each state will then determine the system status such as whether the system is operational, failed, undergoing recovery or repair, operating in a degraded mode, having experienced some specific sequence of events, etc. The set of transitions then define where it is possible to go directly from one state to another. And the transition rate, assigned to each transition, decides how long the system can reside in one state before it transfers into another one.

Figure 3.1 and figure 3.2 illustrates Markov models for a single component featuring the Revealed and Dormant failure model respectively.
In Figure 3.1, this Markov model is composed of two states which represent the situations where the component is functioning normally and failed undergoing repair respectively. These two states, state 1 and state 2, are linked by the two transitions which represent the failure and repair process. These two transitions are then labelled by the corresponding transition rates, the component’s conditional failure rate \( \lambda \) and conditional repair rate \( \nu \).

As can be noticed, the big difference in the Markov model shown in Figure 3.2 is the introduction of state 3 and the transition from state 2 to state 3 represented by the dotted line. Such a difference results from the unique characteristic of the dormant-failure model that the component features. As is described in Chapter 2, the failure of the components of this specific type will not be detected, thus will not be considered for repair, until the failure is detected on inspection. It means that the failed component will dwell in state 2 during the operational phase and transfer to state 3 as soon as it is discovered when the inspection on the component is carried out. Unlike the transitions from state 1 to state 2 and state 3 to state 1 to which a transition rate can be assigned, the transition from state 2 to state 3 is compulsory at a set time point. That is, at the time the inspection is performed, the transition from state 2 to state 3 will occur while at other times this transition is not allowed to occur.

3.1.2 Markov Property – the Underlying Assumption

Markov property is the fundamental underlying assumption for the solution of all the Markovian models. In the most general discrete-state stochastic process, the probability of arriving in a state \( j \) by a certain time \( t \) depends on conditional probabilities which are associated with sequences of states (paths) through which the stochastic process passes on its way to state \( j \). What’s more, it also depends on the times \( t_0 < t_1 < t_2 < \ldots < t_n < t \) at which the process arrives at those intermediate states. A complete accounting for all possible paths and all possible combinations of times would be very complex and is usually not feasible. The problem of evaluating all of the state probabilities in the resulting stochastic process generally is not tractable.
To overcome this problem and ensure most of the Markov models to be solvable, the Markov property has been established which allows a simplification both in the defining of the stochastic process (i.e. the specification of the conditional probabilities) and in the evaluation of the state probabilities by assuming that the probability of arriving in a state $j$ by the time $t$ is dependent only on the state immediately preceding state $j$ on the transition path instead of on the entire path. That is, the future behaviour of the Markov model is determined only by the present state and not by how the process has arrived in the present state.

The Markov property assumption is concisely represented by the following equation:

$$P[X_t = j] = P[X_t = j \mid X_0 = k, X_{1t} = m, ..., X_m = i] = P[X_t = j \mid X_m = i]$$  

where $k, m, i, j$ represent different system states in the Markov model; 

$P[X_t = j]$ refers to the probability that the system resides in state $j$ at time $t$

### 3.1.3 Different Types of CTMC

In the category of the CTMC, there exist 3 different types of Markov models, each of which has unique characteristics. In the following part of this section, these 3 types of Markov model will be defined, explained, and compared with each other using the single-component system.

#### 3.1.3.1 Homogeneous CTMC

Homogeneous CTMC is the simplest and most commonly used Markov model in analysing and evaluating systems which involve the inter-dependence between basic components.

In this type of CTMC, the 'Markov' property, which has been discussed in the pervious section, will always hold. Other important properties of the homogeneous CTMC are that the state holding times are exponentially distributed and the rates of the transitions between states are constant. These properties altogether lead to the fourth property, which is that the time to the next transition is not influenced by the time already spent in the state. This means that, regardless of whether the system has just entered state $i$ or has been in state $i$ for some time already, the probability that the next transition will occur at or before some time $t$ units into the future remains the same [5]. This property is called 'memoryless property' [19]. The Markov model shown in figure 3.1 is a typical homogeneous CTMC with the component's conditional failure rate $\lambda$ and conditional repair rate $v$ both being constant.
3.1.3.2 Non-homogeneous CTMC
A non-homogeneous CTMC can be obtained by easing the restriction on one of the properties characterized by the homogeneous Markov models, which is the 'constant transition rate' feature. In the non-homogeneous Markov models, the rates of the transitions between the system states are not necessarily constant. Instead, the transition rates can be functions of time measured by the 'global clock', which starts to count as the system begins to function for a period of time and refers to the elapsed mission time.

Apart from this difference, the other properties of homogeneous CTMC still hold for the non-homogeneous Markov models. The Markov property is still valid in the non-homogeneous CTMC, which prescribes that the selection of the transition to the next state depends only on the present state the system is in and not on the previous transition path that has led the system to be in the present state. As far as the state holding times are concerned, they also do not depend on previous or future transitions [5]. The probability that the system transfers from state $i$ to state $j$ per unit time at $t$, i.e. the probability that the system transfers from state $i$ to state $j$ during the period of time $[t, t+\Delta t)$, is determined by the elapsed system mission time as the transition rate between state $i$ and state $j$ can be any function of the 'global time', i.e. the elapsed mission time.

In real engineering systems, non-homogeneous Markov models are likely to be established when the failure rate of some components may tend to either increase or decrease as the system proceeds to function.

3.1.3.3 Semi-Markov Model
The behaviour of the semi-Markov model is the same as the other two discussed above in that the selection of the transition to the next state does not depend on the previous transition path that brought the system to the present state. However, it differs from the other two types of CTMC in that when the system enters a new state, as the consequence of the generally distributed state holding times, the inter-state transition rates can be functions of time as measured by 'local clock' [5]. This refers to how long the system has been residing in the specific state. Such a model is called a homogeneous semi-Markov process. When the theory of homogeneous semi-Markov models are further generalized to allow the state holding times to depend also on the time when the state was entered and on the number of transitions preceding when the state was entered [20], a new model type will be obtained, called the non-homogeneous semi-Markov process. Approximation techniques are used to solve the semi-Markov models [5].
3.1.3.4 Comparison between 3 Types of CTMCs

As can be concluded from the previous sections, homogeneous Markov models are the simplest and lowest on the scale of modelling power because it requires the system to fulfil the assumption of the constant inter-state transition rates. Non-homogeneous Markov models are more complex as it allows the transition rates to be non-constant by accepting the transition rates being the functions of global time. Therefore, non-homogeneous CTMCs are able to model more complex system behaviours than can be accommodated by homogeneous models. Semi-Markov models are similar to non-homogeneous CTMCs in that they also feature non-constant transition rates. The most significant difference here lies in that in semi-Markov models the inter-state transition rates are functions of state-specific local time instead of the system global mission time, as was the case for non-homogeneous CTMCs. Usually semi-Markov models will be generated when detailed fault/error handling has to be reflected in the resulting Markov model because the rates of the transitions which represent the fault handling processes often depend on the time elapsed since the fault occurred and handling/recovery commenced rather than on the elapsed mission time [5].

Semi-Markov models might be considered to be more sophisticated than non-homogeneous CTMCs since they are able to model system behaviour which is in some senses more complex. However, it does not mean that semi-Markov models are an encompassing generalization of non-homogeneous CTMCs because there are still some system behavioural characteristics that non-homogeneous CTMCs can model whereas semi-Markov models cannot. The example of the type of model which does encompass both non-homogeneous and semi-Markov models is one which has inter-state transition rates being functions of global time and local time both within the same model. This type of model is non-Markovian and is very difficult to solve numerically in a direct way, thus requiring more flexible evaluation techniques like simulation [5].

3.1.4 Additional Issues concerning the Markov Method

3.1.4.1 Model Generation and Validation

It is generally difficult to construct and validate Markov models, especially for systems which involve complex dependency relationships.

Usually an analyst is faced with two options for producing the Markov model of a system. The most basic way is to draw the model by hand directly from the system. This is potentially an error-prone method and is practical only for very small systems of which the resulting Markov
model should contain no more than 50 states. The next best option is to write a customised computer program to generate the Markov model directly from the information about the system. This may also be a troublesome method because of the difficulty in debugging the program and ensuring the resulting model correctly embraces the system behavioural characteristics. Besides, to generate a Markov model directly from the system description does not help the analyst to gain an insight into the system failure logic. To overcome the shortcomings of the above two approaches, generalized reliability analysis programs have been developed during the last two decades. One method for automatically generating a Markov model is to automatically convert a model of a different type into an equivalent Markov model. One example of the programs using this method is the Hybrid Automated Reliability Predictor (HARP) program [21], which is now part of the HiRel (Hybrid Automated Reliability Predictor, HARP, Integrated Reliability) program [22]. HARP converts a dynamic fault tree model into an equivalent Markov model [23]. Such an approach provides an advantage if the alternate model type offers a more concise representation of the system failure logic. A second method for automatically generating a Markov model is to use a specialized computer programming language for describing transition criteria [5]. This approach is used by the ASSIST (Abstract Semi-Markov Specification Interface to the SURE Tool) program, which uses a rule-oriented language to automatically generate input files for the SURE (Semi-Markov Unreliability Range Evaluation) program [24].

3.1.4.2 Model Size

The Markov method analyses the system at the basic component level by considering all possible system states in terms of different combinations of component state. It has a high level of flexibility and is able to embrace most of the types of dependency involved in the system. However, the Markov method has an inherent weakness which may render the method inapplicable to large-scale systems. This weakness refers to the model size problem. The number of system states contained in the resulting Markov model will increase exponentially as the number of components in the system grows. That is, a system composed of $n$ components which just ‘work’ or ‘fail’ in theory will result in a Markov model with a maximum of $2^n$ system states. To address the model size problem, several techniques have been developed to reduce the number of system states in the Markov model. The following sections give a detailed illustration of how these techniques are applied.

3.1.4.2.1 State Reduction Technique – State Lumping

State lumping [25] is a technique which, in most cases, can effectively reduce the total number of states in the Markov model by combining some states in the model into one composite state.
Take a system composed of 3 identical components to illustrate how the lumping process is carried out.

Assume these are non-repairable components. In figure 3.3, on the left-hand side is the original Markov model of the example system which contains 8 system states as numbered. Each state represents a state of the 3 components with '0' representing the component is working and '1' representing the component is failed. In contrast, on the right-hand side of figure 3.3 is the reduced Markov model after the lumping process which contains only 4 system states, half of the original model size. With the state lumping technique, states are created by considering only the number of working and failed components.

State lumping has to be carried out carefully depending on the form of the model, the meaning of the states, and system behaviour of interest that must be represented in the model. For example, as for the above example system, if what the analyst is concerned about is not only the number of operating components but also the detailed accounting of each operational configuration, the lumping would be inappropriate because the resulting reduced model is not able to provide the required information. What's more, close attention must be paid with regard to which states can be lumped together. During the lumping process, states which are to be combined into one must meet certain requirements [25]. For example, these states must have common properties which in some sense reflect the same system configuration. Consider the left-hand model in figure 3.3, states 2, 3 and 4 all represent the system configuration that 2 components are working while 1 is failed and they all have the same outgoing transition rate $\lambda$, which is determined by the same conditional failure rate of the 3 components.

Figure 3.3 Illustration of state lumping process
3.1.4.2.2 State Reduction Technique – State Truncation

State truncation is an approximation technique capable of reducing the number of states in the Markov model [5]. When this technique is applied, a limit for the number of component failures represented by system states is decided before the model construction is started. Then during the process of constructing the corresponding Markov model, states which represent configurations with a larger number of component failures than the predetermined limit are lumped into one aggregate state. In general, the aggregate state contains both states which represent the system working and states which represent the system failed. Therefore, the approximated system unavailability/unreliability can be obtained through a bounded interval which is generated by assuming that the aggregate state represents first only failure states and then only operational states in turn. The first assumption results in an underestimated system availability/reliability because it fails to take into account the probability of the operational states included in the aggregate state, while the second assumption results in an overestimated system availability/reliability because it fails to filter the failure states in the aggregate one. An equation illustration of this approximation technique is:

\[ R_{\text{trunc}}(t) \leq R_{\text{real}}(t) \leq R_{\text{trunc}}(t) + P_{T}(t) \]  

where:
- \( R_{\text{trunc}}(t) \) is the truncated system reliability obtained by assuming the aggregate state includes only failure states.
- \( P_{T}(t) \) is the aggregate of the probabilities of states included in the aggregate state.

The relationship between \( R_{\text{trunc}}(t) \), \( R_{\text{real}}(t) \) and \( P_{T}(t) \) is explored in a \( n \)-state Markov model. Assume that the truncation is carried out on system states where \( k \) or more component failures occur, and of the \( n \) system states, state \( 1 \) – state \( m \) feature \((k-1)\) or less component failures, while state \((m+1)\) – state \( n \) meet the condition for the truncation. When the truncation is implemented, state \((m+1)\) – state \( n \) form the aggregate state, leaving the truncated Markov model containing \((m+1)\) system states. Then \( R_{\text{trunc}}(t) \), \( R_{\text{real}}(t) \) and \( P_{T}(t) \) can be expressed as:

\[ R_{\text{trunc}}(t) = \sum P_{i}(t) \ (i \in (1, m)) \]

where \( P_{i}(t) \) refers to the probability of system state \( i \) at \( t \), in which the system is working.

\[ R_{\text{real}}(t) = \sum P_{i}(t) + \sum P_{j}(t) \ (j \in (m+1, n)) \]

where \( P_{j}(t) \) refers to the probability of system state \( j \) at \( t \), in which the system is working.

\[ P_{T}(t) = \sum P_{s}(t) \ (s=m+1-n) = \sum P_{j}(t) \ (j \in (m+1, n)) + \sum P_{i}(t) \ (l \in (m+1, n), l \neq j), \]
where \( j+1 = n-m \), and \( P_A(t) \) refers to the probability of system \( l \) at \( t \), in which the system is failed. Therefore, \( R_{real}(t) \) can be expressed in terms of \( R_{trunc}(t) \) and \( P_F(t) \) as follows:

\[
R_{real}(t) = R_{trunc}(t) + \sum P_f(t)
\]

\[
R_{real}(t) = R_{trunc}(t) + (P_F(t) - \sum P_f(t))
\]

By assuming that the aggregate state includes only failed system states, the lower bound of approximated \( R_{real}(t) \) is then obtained with \( \sum P_f(t) \) equal to 0. Alternatively, by assuming that the aggregate state includes only working system states, the upper bound of approximated \( R_{real}(t) \) can be worked out as illustrated in equation 3.2 with \( \sum P_f(t) \) equal to 0.

As can be concluded from equation 3.2, the smaller the interval, i.e. \( P_F(t) \) is, the more effective the truncation technique will be. Since in the Markov model, the probability passes through the inter-state transitions from the initial state which contain no component failures to states which includes more and more component failures, it implies that the state truncation is most effective for systems for which the mission time is relatively short and failure rates of components are very small because in this type of system, through the mission time, \( P_F(t) \) will be negligible.

### 3.1.5 Summary of the Markov Method

#### 3.1.5.1 Advantages of the Markov Method

- The component-level investigation enables the Markov model to reflect the repair scheme in a natural way.
- It makes it possible to include the fault/error handling and recovery at a detailed level.
- The Markov method is able to model most types of dependency embedded in the system, like the standby spares and sequential dependency.

#### 3.1.5.2 Disadvantages of the Markov Method

- The Markov property assumption which is fundamental to the application of this method may not hold for some systems.
- The construction and validation of the Markov model can be very difficult for some complex systems.
- The model size can always be a big problem for large-scale systems even if the state reduction techniques are available.
Therefore, the analyst must carefully balance the advantages and disadvantages of the Markov method against the behavioural characteristics of the system in order to ensure that it is appropriate to apply the Markov modelling technique to the system risk and reliability assessment.

3.2 Dynamic Fault Tree Analysis

As has been discussed at the beginning of this chapter, the conventional fault tree analysis method is inadequate as far as the evaluation of systems which involve inter-dependency between components is concerned.

To overcome this limitation, the Dynamic fault tree [7, 8, 26-33] has been developed, based on the original fault tree structure, to enable the representation of some types of dependency in the system. That is, the dynamic fault tree is not only able of representing the system failure logic but also able to exhibit, in its own structure, types of dependency included in the system. The dependency representation is realized by adding new types of gates to the original fault tree structure.

3.2.1 Dynamic Fault Tree Gates

Apart from the usual OR, AND and NOT gates, new types of gate have been introduced which are able to represent the particular types of dependency to generate the dynamic fault tree structure. Four types of 'dependency' gates will be described in detail in this section.

3.2.1.1 Functional-dependency Gate (FDEP)

The key to the existence of the functional dependency is the trigger event whose occurrence causes other functionally-dependent components to become inaccessible or unusable. That is, when the trigger event has occurred, these functionally-dependent components are disabled, and from the function point of view, these components are effectively failed. The structure of the functional-dependency gate (FDEP) is shown in figure 3.4.

![Figure 3.4 Structure of functional-dependency gate](image)
where

- the trigger event can be either a basic event representing a basic component failure or the output of another gate in the tree.
- dependent events are basic events which represent the failure of the components which are functionally-dependent on the trigger event.
- the non-dependent output conforms to the status of the trigger event

When the functional dependency exists in the system, it will affect the generation of the Markov model in the way that when a state is generated in which the trigger event has occurred, all the corresponding dependent events are regarded as having occurred. However, it must be noted that only in a non-repairable system can this case be true. When the system is repairable during its mission time, the occurrence of the dependent events still needs to be taken into account while the trigger event has occurred, because the repair of the dependent events will have influence on the system repair intensity and the system failure probability as well.

Functional dependency can be found in many systems. One example could be the communication between the components in the computer system. The components communicate with each other through some network interface elements, where the failure of the network elements isolates the connected components. Therefore, as far as the communication failure is concerned, the failure of the network element is the trigger event and the connected components are the dependent events.

In the above example, it can be noted that the event 'communication failure' can also be represented by an 'OR' gate. Assume the system is composed of two components A and B, which are connected through the network element, the communication failure between A and B can be expressed by the fault tree in figure 3.5:

![Fault Tree Diagram](image)

Figure 3.5 Conventional fault tree structure of FDEP gate
The transformation of the FDEP gate into an ‘OR’ gate indicates that the functional dependency is actually not a statistical dependency, therefore the mere existence of the functional dependency in the system does not require the use of the Markov method during the analysis process. Rather, the system can be analysed and evaluated using the conventional fault tree method. However, it is still useful to identify the functional dependency when the trigger event and dependent events are included in a Markov model due to the existence of other types of statistical dependency as the functional dependency will help to delete the impossible inter-state transitions.

3.2.1.2 Cold-spare Gate (CSP)

The cold-spare gate is structured to embrace one of the standby dependency modes, cold standby dependency. Standby dependency consists of three different modes, cold spare, warm spare and hot spare, distinguished by the failure rate featured by the standby inactive component. In the cold spare mode, the standby component is characterized by a zero conditional failure rate, which means the component will never fail when it is residing in the inactive standby status. Alternatively, in the hot spare mode, it is assumed that whatever state the component dwells in, working or standby, the component failure is governed by the same conditional failure rate. Warm spare mode lies in between the cold spare and hot spare as the standby component features a non-zero conditional failure rate which is smaller than the normal active failure rate. A dormancy factor, which is associated with the standby component, has been introduced to distinguish the three different modes. This dormancy factor, denoted by \( \alpha \in [0,1] \), is a multiplicative factor to the active failure rate to produce the spare failure rate \([8]\), i.e. \( \lambda_s = \alpha \lambda \). So when \( \alpha \) is 0, the spare is a cold spare; when \( \alpha \) is 1, the spare is a hot spare; and when \( \alpha \) lies in between 0 and 1, i.e. \( 0<\alpha<1 \), the spare is a warm spare.

Conventional fault tree techniques can't be used to model systems which contain cold spares because of the dependency it introduces. As is shown in figure 3.10, the CSP gate is composed of one primary input and one or more alternate inputs. All inputs are basic events. The primary input refers to the failure of the duty component which is initially powered on, and the alternate input(s) represent the components that are used as replacements and will be activated when the primary input occurs. The CSP gate has one output which becomes true after all the input events occur.
The CSP gate can be used in the case of pooled spares, where spare components are shared between active units. Then the basic event representing the cold spare forms inputs to more than one CSP gate. The spare is available only to one of the CSP gates, depending on which of the primary units fails first. There might also be a priority or order in which the spares are used, which can be reflected through the order of the alternative units from left to right in the CSP gate structure. The CSP gate can also interact with the functional dependency gate to reflect the relationship between the components in the system.

The warm-spare gate (WSP) can also be introduced to model systems where warm spares are used. The structure of the WSP gate is exactly the same as the CSP gate. When CSP gate or WSP gate is converted to a Markov chain, the special feature of cold spare or warm spare can then be taken into account.

### 3.2.1.3 Sequence-enforcing Gate

The sequence-enforcing gate, as is shown in figure 3.7, is introduced to represent the situation where the input events are constrained to occur in the left-to-right order as they appear under the SEQ gate.

It must be noted that the SEQ gate is different from the Priority-AND gate which is used to represent sequential dependency. The input events under the Priority-AND gate can occur in any order, but only when they occur in the specified left-to-right order can the output of the Priority-
AND gate become true. However, the input events under the SEQ gate can occur only in the specified order.

During the process of generating the Markov model from a dynamic fault tree which contains a SEQ gate, any state which represent the different order of occurrence of the events under the SEQ gate will never be produced. The sequence-enforcing gate can be used where pooled cold spares are included in the system.

3.2.1.4 Demand-dependency Gate (DDEP)

The demand-dependency results from the unique characteristic of most safety/protection systems which are incorporated in safety-critical systems to prevent specific undesirable or disastrous consequences. Generally speaking, the operation of safety/protection systems can be divided into two phases: standby/inactive and functioning/active. For most time of its life, the safety/protection system dwells in the inactive state, and during these inactive periods, the system will be tested and maintained at regular intervals. The safety/protection system will respond and activate when a demand arises. Usually such a demand refers to a specific undesirable initiating event which, if not suppressed successfully, could lead to severe or even catastrophic consequences. In most cases, in order to mitigate the specific initiating event, the safety/protection system is required to function successfully through a minimum period of time, and during this functioning period, no maintenance is possible.

Demand dependency exists in most safety/protection systems and arises when the system transfers from the standby mode to the active mode. Take for example the fire sprinkler systems installed in many buildings, when the intensity of the smoke reaches a certain level, it will be detected and trigger the water sprinklers to put out the fire. In this simple example system, the pump is the main part whose functionality will be controlled by the smoke sensor. If the smoke sensor has failed before the smoke intensifies, the pump will not be started and no water will come out of the sprinkler. Therefore, the demand dependency exists between the demand components (the pump) and the support system (the smoke sensor) which is supposed to detect the demand or to start the protection system. The demand-dependency gate is the dynamic fault tree construct which has been introduced to represent and model the demand-dependency. The structure of the demand-dependency gate is shown in figure 3.8. When the input represents the failure of the support subsystem, a fault tree diagram will be used to model its failure.
A computer-controlled sprinkler system (see Figure 3.9) is used here to illustrate the advantages of the DDEP construct in analysing the safety/protection systems.

As is shown in figure 3.9, this system is composed of three sensors, two pumps and one controller. Each pump has a support stream composed of valves and filters.

The sensors send signals to the controller, and when the temperature readings at two of the sensors are above the threshold, the controller activates the pump. And in order to start, the pump requires the support stream (valves and filters) to be operational. Pump 1 is the duty pump which will be activated first, and pump 2 is the cold spare and will be started by the controller when pump 1 fails. The controller will start pump 2 when the temperature reading at the sensors keeps rising due to the failure of pump 1.

If the demand dependency is not considered and neither the distinction between the system standby and active phases, the failure of the sprinkler system can be represented by the following dynamic fault tree structure shown in figure 3.10.
In figure 3.10, it can be seen that functional dependency gates (FDEP) are used to capture the dependency of the pumps on their support streams. However, the disadvantage of ignoring the standby/active feature in the analysis can be revealed when the system goes through a thorough examination.

In this sprinkler system, consider the valves and filters. These passive components can fail when they get blocked when the system dwells in the inactive state. It is assumed, however, when the system starts, i.e. when there is water flowing through them, they will not get blocked, i.e. they won’t fail.

Therefore, the dynamic fault tree structure in figure 3.10 does not embrace the failure characteristics discussed above as it groups all component/subsystem failures under the top event without identifying the different operational features of these components during the system standby and functioning mode. This problem can be tackled by introducing the demand-dependency gate.
In the dynamic fault tree in figure 3.11, it can be seen that only the basic event representing the controller failure and the CSP gate are directly related to the top event. Other events, like the failure of the sensor system and the support streams contribute to the occurrence of the top event through the DDEP gates, which are able to reflect the different effects of the sensors and the support streams on the failure of the system in the standby and active phases. It also indicates that during the system functioning period, only the controller, pump1 and pump2 need to be taken into account with regard to whether the system is able to operate for the required length of time.

3.2.2 Application of Dynamic Fault Tree Method

By introducing the dependency gates, the dynamic fault tree structure enables the representation of several types of dependency in the system. Accordingly, the quantification method which is based on the dynamic fault tree construct differs from the conventional quantitative fault tree analysis. The Markov method will be applied to the sections of the dynamic fault tree which are headed by the dependency gates. In this way, the limitation of the Markov method caused by the model size problem will be eased.

In the following section, an example system is used to illustrate how the dynamic fault tree analysis can be combined with the Markov method.

3.2.2.1 Example System Description

The example system used here is a water deluge system as is shown in figure 3.12. The features of this system are typical of water spray protection systems used on many different off-shore
platforms. This water deluge system is composed of three subsystems: pump system, computer control system and pressure sensing system. Four pumps, two electrical pumps EP1 and EP2 and two diesel pumps DP1 and DP2, are used to provide the water demand to the ringmain, which transports the water to the take-off points around the platform to protect against the hazards posed by hydrocarbon fires and explosions. When the system resides in the standby mode, the ringmain pressure is maintained by a jockey pump. When the take-off valves open, and water is delivered to the spray nozzles, the ringmain pressure drops. The pressure sensing system constantly monitors the ringmain pressure and transmits this to the computer controlling system by three pressure sensors: PS1 – PS3. When two of the three transmitters indicate a low ringmain pressure, main pumps are activated. Two pumps are required to function to ensure the water can be delivered at the required rate to satisfy demand. The four pumps will be started in the order of EP1, EP2, DP1, and DP2. That is, EP1 and EP2 are the default duty pumps, while DP1 and DP2 provide warm redundancy shared by EP1 and EP2. When either EP1 or EP2 fails, the ringmain pressure drops. The change will be transmitted by the sensors to the computer system which will start DP1 or DP2 accordingly. The features on each pump stream are identical. Because the water supply is direct from the sea, a filter is fitted on each stream. Isolation valves are located on either side of the pump. A pressure relief valve provides protection for the pump, and a test valve on each line enables individual pumps to be tested without fully activating the deluge system. Electric power supply and diesel supply are respectively connected to EP1, EP2 and DP1, DP2.

Figure 3.12 Diagram representation of the water deluge system
3.2.2.2 Construction of Dynamic Fault Tree

The construction of a rational and correct dynamic fault tree must start with the identification of the various dependency relationships between the components in the system. Firstly, in the pump subsystem, the warm spare dependency exists between EPl and DP1, DP2 and also between EP2 and DP1, DP2. This means that two warm spare dependency gates (WSP) are needed in the dynamic fault tree structure which share the same two standby alternatives DP1 and DP2. Secondly, functional dependency exists between the electric power supply and two electrical pumps EPl and EP2 and also between the diesel supply and two diesel pumps DP1 and DP2. It is self-evident that when the electric power supply fails, the two electrical pumps are unable to operate. It is the same case with the diesel supply and diesel pumps. Finally, the demand dependency (DDEP) is involved when the system undergoes the transition from the standby mode to the active mode. In this system, the demand dependency lies in various aspects. For example, demand dependency exists between each pump and its support stream consisting of filters and valves. If the stream fails, the pump can’t start. Also the demand dependency exists between the pumps and the computer controlling system and the pressure sensors. Either the failure of the computer control system or the failure of the pressure sensor system will cause the failure to start the relevant pumps. What’s more, similar to the sprinkler system discussed in section 3.2.1.4, each pump stream can fail only when the pump is not started.

Interestingly, functional dependency and demand dependency are similar to each other in that they both are introduced to represent the situation where the successful operation of one component depends on the state of other components. However, the difference between these two types of dependency lies in that the demand dependency does not exist through the whole operation of the dependency component, and it occurs only when the transition takes place. When the transition has been completed, the demand dependency no longer exists between the support components and the dependent component. For example, the computer control system is required to be working in order to start the pump, while after the pump has been started, the failure of the control system would have no effect on the operation of the pump. In contrast, the electric power supply is the prerequisite of the operation of the electrical pump. The pump needs the electric power to get started and also it needs the electric power to keep it running. That is, the functional dependency exists all the time between the support components and the dependent component.
After having identified the different types of dependency existing in the system, the dynamic fault tree can be established according to the system failure logic. Figure 3.13 shows the dynamic fault tree structure for the failure of the water deluge system.

![Dynamic fault tree for water deluge system failure](image)

**Figure 3.13 Dynamic fault tree for water deluge system failure**

The fault trees for sensor system failure and pump stream failure are shown in figure 3.14(1) and figure 3.14(2) respectively.

![Fault tree for pressure sensor system](image)

**Figure 3.14(1) Fault tree for pressure sensor system**

![Fault tree for pump stream failure](image)

**Figure 3.14(2) Fault tree for pump stream failure**
3.2.2.3 Quantitative Analysis

The safety/protection systems can have two different failure modes with regard to its expected function. The first is the system fails to respond to demand in the first place. The second is the system fails to operate for the required period of time, i.e. it fails during running. Correspondingly, the first failure mode is concerned about the system unavailability, while the second failure mode is considered in relation with the system unreliability. Both of the parameters are very useful when the assessment is carried out on the safety/protection system.

3.2.2.3.1 Calculation of System Unavailability

In this water deluge system, that the system is unavailable on demand is caused by the failure to start 2 out of the 4 pumps. That is, when 3 out of 4 pumps have failed at the time the demand arises, the system fails to react. Let events $\text{EP1SF}$, $\text{EP2SF}$, $\text{DP1SF}$ and $\text{DP2SF}$ respectively represent the failures to start of the 4 pumps. Then the system unavailability can be represented by the following combination of the occurrences of these events:

\[
\text{EP1SF.EP2SF.DP1SF OR EP1SF.EP2SF.DP2SF OR EP1SF.DP1SF.DP2SF OR EP2SF.DP1SF.DP2SF}
\]

For each of the events, its logic can be derived from the dynamic fault tree structure shown in figure 3.18. For example, the occurrence of the event $\text{EP1SF}$ can be traced through the 3 dependency gates and 1 functional dependency gate in figure 3.18. That is, either the control system failure (CTR), or the sensor system failure (PSS), or the pump stream failure (PSM) or the electric power supply failure (EPS) can result in the occurrence of the event $\text{EP1SF}$. Since the redundancy features a warm spare feature, the occurrence of the event $\text{EP1SF}$ can also be ascribed to the standby failure of pump 1 ($\text{EP1S}$). This means the event $\text{EP1SF}$ can be represented in the following way:

\[
\text{EP1SF} = \text{CTR} + \text{PSS} + \text{PSM} + \text{EPS} + \text{EP1S}
\]

The events PSS and PSM on the right-hand side of the equation can be replaced respectively by combinations of basic events obtained from figure 3.14(1) and figure 3.14(2).

The same algorithm can be applied to events $\text{EP2SF}$, $\text{DP1SF}$ and $\text{DP2SF}$. Finally the failure logic of the system unavailability can be represented in the form of minimal cut sets, therefore conventional quantitative FTA or binary decision diagram methods can be employed to work out the system unavailability.
3.2.2.3.2 Calculation of System Unreliability

Due to the existence of the warm spare dependency during the system functioning period, the Markov method has to be used to investigate the system unreliability which is the primary concern during the period of time the system is running.

As has been mentioned in section 3.1, the application of the Markov method is implemented through the correctly established Markov model. As one of the key elements of the Markov model is the system state which represents a specific combination of the state of basic events, the first thing is to decide which basic events need to be included in the Markov model from the preceding unavailability analysis. This can be done by considering which basic events still matter with regard to the system active operation. As can be concluded from the example of the sprinkler system in section 3.2.1.4, the pump streams don’t need to be taken into account as they are unlikely to fail during the system functioning period. The four pumps, as they assume the crucial function, have to be included in the model. Accordingly, the electric power supply and the diesel supply which determine whether the pumps are able to operate are also included. Pressure sensors and control system also have to be included as they determine if the standby pumps is required if the duty pumps fail during running.

When it has been established which basic events need to be included in the analysis, the process of developing the corresponding Markov model can start. Usually, the construction of a Markov model starts with an initial state. For many continuously operating systems, the CTMC has an initial state probability of 1 in which all the components are set as working. However, it can be a different case for safety/protection systems. In this water deluge system, due to the redundancy feature, the system can start in various different configurations with different initial probabilities. Therefore, to ensure that the Markov analysis is right, the initial states and the initial state probabilities (ISP) have to be established. By referring to the system description, one can derive the series of initial states in which the system can be activated. For a better understanding, we look at the system in the form of 2 subsystems. One subsystem is composed of 4 pumps (EPs and DPs), the electric power supply (EPS) and the diesel supply (DPS). The other is composed of the computer control system (CTR) and pressure sensor system (PSS).

Table 3.1 gives a list of initial states in which the first subsystem can start in order that the whole system activates successfully.

<table>
<thead>
<tr>
<th>Initial states</th>
<th>EP</th>
<th>DP</th>
<th>EPS</th>
<th>DPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 working</td>
<td>2 standby</td>
<td>working</td>
<td>working</td>
</tr>
<tr>
<td>2</td>
<td>2 working</td>
<td>2 standby</td>
<td>working</td>
<td>failed</td>
</tr>
<tr>
<td>3</td>
<td>2 working</td>
<td>1 standby</td>
<td>working</td>
<td>working</td>
</tr>
<tr>
<td></td>
<td>2 working</td>
<td>1 failed</td>
<td>working</td>
<td>failed</td>
</tr>
<tr>
<td>---</td>
<td>-----------</td>
<td>----------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>4</td>
<td>1 standby</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2 failed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2 working</td>
<td>2 failed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 working</td>
<td>1 working</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1 failed</td>
<td>1 failed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2 working</td>
<td>2 working</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2 failed</td>
<td>failed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1 Initial states for pump subsystem

Now look at the other subsystem. Table 3.2 gives the initial states for the control subsystem which enable the system to respond to the demand.

<table>
<thead>
<tr>
<th>Initial states</th>
<th>Computer control system</th>
<th>Pressure sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>working</td>
<td>3 working</td>
</tr>
<tr>
<td>2</td>
<td>working</td>
<td>2 working, 1 failed</td>
</tr>
</tbody>
</table>

Table 3.2 Initial states for the controlling subsystem

Each initial state given in table 3.1 can be combined with each of the initial states in table 3.2 to form a complete list of initial states for the whole system, which means the system can start in any of 20 initial states. The initial probabilities of the 20 states can then be obtained according to the combinations of the component states represented by each initial state.

With the initial states established, the whole Markov model can be generated by developing new system states from the initial states. As the size of a Markov model consisting of 6 basic events can be relatively big, the following figure provides part of the Markov model for the water deluge system only for illustration purpose.
In this Markov model, several state transitions reflect the existence of the functional dependency and demand dependency. For example, during the transition from state 1 to state 2, the failure of the electric power supply renders both of the electrical pumps unusable, therefore the two diesel pumps are activated. During the transition from state 3 to state 7, although the pressure sensors detect the pressure drop in the ringmain caused by the failure of one of the electrical pumps, the control system can't start the backup diesel pump because of the diesel supply failure. Demand dependency occurs during the transition from state 11 to state 12. The failure of one of the electrical pumps causes the ringmain pressure to drop, but due to the failure of the computer control system, the backup diesel pump is not activated. State 7 and state 12 represent the situation where the system fails during the functioning. As there is no maintenance available when the system is functioning, these two states are self-absorbing.

When the process of developing the Markov model is completed, the quantitative Markov analysis can be carried out on the model to arrive at the required system reliability parameters. The quantitative Markov analysis will be described and illustrated in detail in Chapter 7.
3.2.3 Summary of Dynamic Fault Tree Technique

3.2.3.1 Advantages of Dynamic Fault Tree Technique

- The dynamic fault tree technique enables the explicit representation of the dependency relationship between the components in the system.
- The newly introduced dependency gate structures not only reflect the dependency relationship between the components but also indicate how they are linked with each other.
- As all dependent components are grouped under the dependency gates, irrelevant independent components would be filtered out and won’t be included in the resulting Markov model, which reduces the model size.

3.2.3.2 Disadvantages of the dynamic fault tree technique

- The dynamic fault tree technique is unable to deal with the case where repeated basic events are included under dependency gates. With the Markov method applied to the sections which are led by the dependency gates, the dependency caused by the repeated occurrence of the basic events somewhere else in the fault tree will undermine the rationality of the analysis and result in incorrect system reliability prediction.
- The number of types of dependency that the dynamic fault tree is able to embrace might be limited because it might be very difficult to establish a dependency gate structure for some types of dependency existing in the system.
- The dynamic fault tree analysis will require a lot of manual work. Due to the structure of the dynamic fault tree, a lot of manual analysis is required prior to the quantification to determine respectively which component failures need to be investigated for the evaluation of system unavailability and system unreliability of active-on-demand systems.

The above weaknesses could severely restrict the application of the dynamic fault tree method. An alternative solution will be brought forward in the following chapters to address these problems.
In the previous chapter, several types of dependency have been introduced, like functional dependency, cold/warm spare dependency, sequence-enforcing dependency and demand dependency. In real systems there exist other types of dependency than have so far have been discussed. In this chapter, the main types of dependency which are frequently encountered in many engineering systems will be examined in a systematic way. A new way of representing the dependency relationships in the system, other than dependency gate structures, is introduced to overcome one of the limitations of the dynamic fault tree technique. The structures of data input for the system failure logic and component failure information are also demonstrated.

4.1 Types of Dependency

4.1.1 Maintenance Dependency

*Maintenance dependency* rises from the situation where one maintenance engineer or team of engineers has to take responsibility for a group of components usually of the same or similar type. In this case when one component has failed and is under repair, components from the same group which fail subsequently have to go into a queue waiting for repair until the engineer has restored the first component. It may be that some failures are more critical than others and so are given higher priority by the maintenance team. Either way the queuing means that the failure of one component affects the repair times and therefore the probability of failure of other components. Finally, the influence of the maintenance dependency affects the whole system, resulting in different system reliability prediction from those produced where no dependencies are encountered. In real engineering systems, the maintenance dependency is common.

4.1.2 Standby Dependency

In most safety-critical systems, redundancy or diversity is a very popular design strategy and used frequently to reduce the chance of system failure. Redundancy in the design can be achieved by the use of *standby systems*, where the failure of the primary system activates the standby to take over the primary system duty. That the standby component or system can reside in different states determined by the primary duty component or system is the root cause of standby dependency. As is mentioned in chapter 3, if the redundancy design is embraced in the
system in the form of a cold or warm spare, it implies that the likelihood of the standby component failure increases as it experiences the operational load due to the failure of the primary operating component. Such a difference represents a dependency relationship between the standby component failure probability and the operating component status (working or failed) [8, 29]. Therefore failure to take into account the standby dependency will produce incorrect system unavailability and failure intensity.

4.1.3 Sequential Dependency

*Sequential dependency* refers to the situation where a certain event will take place only when its causes occur in a specific sequence. Different from other forms of dependency, sequential dependency is the only type which can be represented in the conventional fault tree structure. The ‘Priority AND gate’ [3], as is shown in figure 2.1, is used in the conventional fault tree structure to represent the sequential dependency. With the priority AND gate, only when input events occur in the order from left to right, will the output event occur.

4.1.4 Sequence-enforcing Dependency

As has been discussed in chapter 3, *sequence-enforcing dependency* [7, 8] exists between the basic events which are constrained to occur in a specific order to contribute to system failure. Such a restriction might result from the different nature of the components’ function or from the different nature of the component’s failure mode. The difference between the sequence-enforcing dependency and the sequential dependency has been expounded in chapter 3.

The conventional assessment techniques are unable to take into account the sequence-enforcing dependency because the whole assessment process does not include a mechanism which can reflect the strict order. Failure to account for the sequence-enforcing dependency will result in inaccurate system failure intensity. The following example considers this in more detail. Consider the situation where system failure will only result if the two basic events A and B occur in the order of A then B. With the conventional assessment method, the unconditional system failure intensity is given by:

\[ w_{\text{sys}}(t) = q_A(t) . w_B(t) + q_B(t) . w_A(t) \]

However, it is obvious that the term ‘\( q_B(t) . w_A(t) \)’ should not be included on the right hand side of
the above equation as event B is not 'allowed' to occur prior to event A.

With the dynamic fault tree approach, the sequence-enforcing dependency is represented by the sequence-enforcing gate as is shown in figure 3.12. In the dynamic fault tree, the sequence-enforcing gate is defined as a gate of AND logic. However, in some cases, intermediate gate types may be required to represent the sequence-enforcing dependency relationship for the system failure logic. The logic relationship between the basic events in such circumstances may be too complex, for a mere AND logic gate to cover.

4.1.5 Secondary-failure Dependency

Secondary failure dependency exists in the system when the same component failure mode can be caused by either the primary failure of this component or a secondary failure. Primary failure refers to the failure of the component operating under the expected operating condition. It might result from the poor quality of the component or wear and tear related to the component itself. Alternatively, the secondary failure is due to the failure of other components in the system which then result in the component under examination operating in a way which it was not designed to. Frequently, the secondary failure will result in a change in the working environment of the primary component, which will cause the primary component to fail.

In the fault tree structure, the secondary-failure dependency can be represented through an OR gate with primary failure, secondary failure and sometimes command faults as input [34] (see figure 4.1 below).

![Figure 4.1 Fault tree structure for secondary-failure dependency](image)

The secondary failure dependency mainly affects the repair process related to the failure of the primary component. When the primary failure causes the component failure, the repair process is performed as usual to rectify the fault, as the primary component will be examined and restored.
However, when the secondary failure occurs, the repair needs to be carried out not only on the components which has contributed to the secondary failure but also on the primary component which also fails as a consequence of the secondary failure. In the conventional assessment methods, the repair of the primary component won’t be included in the process to rectify the system when the secondary failure occurs. Frequently the repair of the primary component can take much longer than the secondary components, therefore to neglect the secondary-failure dependency can lead to an underestimation of the system unavailability [35].

Figure 4.2 gives an example tank system, in which the tank rupture can result from both primary failure and secondary failure.

![Figure 4.2 Example tank system](image)

In this system, the pressure relief valve acts as a safety valve, which will open to prevent the possible rupture when the pump surge happens and results in an abnormally high tank pressure. For illustrative purposes, the analysis of this tank system will be focused on the safety aspect, and the top event is defined as 'Tank ruptures'. The corresponding fault tree is shown in figure 4.3.

![Figure 4.3 Fault tree for the example tank system](image)
As is shown in figure 4.3, with regard to the top event of ‘tank rupture’, the basic event TRP represents the primary failure, meanwhile the gate event ‘Tank ruptures due to abnormal high pressure’ represents the secondary failure which leads to the same consequence as the primary failure, the tank rupture. The significance of identifying and taking into account the secondary-failure dependency in this system is substantial. When the secondary failure occurs, repair needs to be implemented on the tank itself as well as on the pump and the pressure relief valve. Since the time that it takes to restore the tank following its rupture is much longer than to repair the pump and the relief valve, failure to account for the secondary-failure dependency may result in an overestimation of the system availability.

4.1.6 Initiator-enabler Dependency

In this section, ‘initiator’ and ‘enabler’ respectively refers to the initiating event and enabling event. A good understanding of this type of dependency has to start with the definitions of these different types of events. As defined in [3] and [36], initiating events:

perturb system variables and place a demand on control/protective systems to respond,

whilst enabling events:

represent the failure of inactive control/protective systems which permit initiating events to cause the system failure state.

The requirement to distinguish between initiating and enabling events and the initiator-enabler dependency usually occurs in the analysis of safety systems. The initiating event is the event which the safety system is designed to mitigate, and the occurrence of the initiating event usually results in the safety system being activated. Enabling events represent the failure of the components which compose the safety system. The failure of the safety system is a general enabling event which produces the conditions under which the initiating event can cause a hazard.

The prerequisite for accounting for the initiator-enabler dependency in the system analysis is to identify the initiating events and enabling events in the system. For a complex system, this may prove to be a confusing and difficult task. A feature of the enabling event, which can distinguish it from initiating events, is that the occurrence of the enabling events alone can never bring about the system hazard.
Take for example the tank system in section 4.1.5. To give a better illustration, the system is amended by enhancing its safety measures. As is shown in figure 4.4, three pressure sensors (PS1, PS2, PS3) are incorporated into the tank to monitor the tank pressure level. All these sensors provide an input to a computer controller which can control the pump. The pump is tripped on a 2-out-of-3 voting system for the pressure sensors. The pressure relief valve acts as the first-line safety measure. In the event of a high pressure, if the pressure relief valve fails stuck, the pump control sub-system works to prevent the explosion. As long as 2 sensors report the abnormal high pressure in the tank, the controller will switch off the pump.

![Amended tank system diagram](image)

Figure 4.4 Amended tank system

Assume the analysis is centered on the tank over-pressurization, the corresponding fault tree is then developed as is shown in figure 4.5.

![Fault tree diagram](image)

Figure 4.5 Fault tree for tank over-pressurization
According to the definition of initiating and enabling events, in this system, the abnormal pump surge PS is an initiating event. All the safety measures are aimed to prevent this event from developing into the tank rupture. The failure of the safety measures is the general enabling event which consists of individual enabling events representing the failure of individual safety components. Assume the pump surge occurs prior to any other event, the over-pressurization in the tank will cause the pressure relief valve to open to reduce the tank pressure. In this case, the top event will not occur. However, if the pressure relief valve fails stuck before the pump surge, the controller system forms the last protection. If the controller system has also failed by the time the pump surge happens, the top event occurs, i.e. the tank ruptures due to the over-pressurization. If the controller system is working normally when the pump surge happens, the controller will stop the pump on receiving the high pressure signal from the sensors, and the top event will not occur.

From the above analysis of the tank system, two interesting points can be identified with regard to the initiator-enabler dependency. Firstly, it is the order of the occurrence of the initiating and enabling events which is important. Only when the enabling event occurs first, can the occurrence of the initiating event lead to the specific consequence, usually represented by the top event. Alternatively, if the initiating event occurs first, the corresponding safety function will mitigate the initiating event or mitigate its undesired effect so that the top event will not occur. As such, the initiator-enabler dependency is similar to the sequential dependency as it also imposes a certain occurrence order. What makes it distinct from the sequential dependency is that when there exist more than one enabling event to protect against the certain initiating event, usually there is no restriction on the occurrence order of these enabling events. If a system, which involves the initiator-enabler dependency, is processed with the conventional assessment methods, the system failure intensity will be, to some extent, overestimated.

Secondly, the analysis of the tank system identifies two different characteristics featured by different enabling events. The enabling event PRS falls into the first category. It can be noted that once the pressure relief valve has opened due to the high pressure, it is unlikely to get stuck until the pressure is brought down to the normal level and the relief valve closes. That is, if the initiating event PS occurs first, the enabling event PRS will not occur until the initiating event is
solved. Some basic events, which represent the same failure mode of valves, can be grouped into this category. For example, many pump systems will involve the use of different types of valves, like isolation valves and check valves. These valves may contribute to the system failure by sticking closed. But usually when the system is activated and there is fluid circulating through them, the valves will not consequently close against the pressure. The valves and filters in the sprinkler system in section 3.2.1.4 are examples which fit this case.

The second category of enabling events is different in that the enabling events can still occur after the occurrence of the initiating event, but their occurrence then will not have any effect on the top event. For example, the basic events CTR, PS1F, PS2F and PS3F in figure 4.5 can still occur after the initiating event happens. Most sensors and controllers which are fitted in the system as safety measures belong to this category. Although the subsequent occurrence of the enabling events fulfills the AND logic which mirrors the failure logic of the top event, it must be noted that due to the existence of the initiator-enabler dependency, the top event will not occur. The necessity of distinguishing the two different types of enabling events mainly exists during the process of developing the corresponding Markov model and will be explored and illustrated in detail in Chapter 6.

In most systems, all enabling events represent dormant failures. It means that the component failure represented by these enabling events will be inspected and maintained at certain regular intervals. An initiating event often represents a revealed failure. In most cases, the top event represents an undesired hazardous consequence, its occurrence will often be discovered immediately. In the tank system discussed above, if the enabling events occurred (undetected) during their inspection interval, a subsequent pump surge will result in the tank rupture and consequently reveal the failure of these safety components. In this way, the initiator-enabler dependency, for non-catastrophic top event, also has an influence on the down time of some enabling components in the system as their occurrence is revealed.

4.1.7 Revealing Dependency

Revealing dependency exists in systems, where combined component failures, which include dormant failures, lead to the occurrence of the certain event which will be discovered instantly
and in return reveal the relevant dormant failures. In this respect, revealing dependency is similar to the initiator-enabler dependency. However, unlike the initiator-enabler dependency, the revealing dependency has nothing to do with the restriction on the occurrence order of events.

In the fault tree structure, the revealing dependency is always related to an intermediate event which represents a self-revealing occurrence. Usually this intermediate event is further developed by the AND gate under which dormant failures are involved.

4.1.8 Test Dependency

In many systems, some component failures will be in a dormant mode. To reduce the time these components reside in the failed state they are tested or inspected at interval, regular maintenance is carried out as appropriate. However, in some systems, such regular inspection is not only implemented on individual components but also on sub-systems which assume a relatively independent function in the whole system. The inspection of the sub-system is carried out by testing if the sub-system can fulfill the required function. If the test fails, the maintenance team will identify and repair the failed components. The inspection interval for the sub-system can either be same as or different from the inspection interval for its components. When there is a difference, a dependency exists as the inspection on the sub-system will reveal the dormant failures of some components and thus change their actual downtime. Different from the types of dependency which have been discussed so far, test dependency does not give rise to statistical dependency between component failures. The influence of test dependency lies in the failure probability of each individual component, and does not result in the inter-dependence between component failures. In spite of the non-statistical dependent characteristic, test dependency still needs to be identified and specified in the analysis in order that accurate system reliability measures are obtained.

4.1.9 Functional Dependency

*Functional dependency* [7, 8] has been described and illustrated in detail in chapter 3. Although the word ‘dependency’ is used here, the functional dependency does not refer to the real dependency relationship since it entails no statistical dependency but just reflects the functional dependency relationship between some components in the system. However, this doesn’t
necessarily mean that functional dependency doesn’t need to be explicitly identified. Instead, in some cases, functional dependency does need to be identified and explicitly accounted for during the analysis.

Take an electric pump system for example, in which pump 1 is set as the duty pump and pump 2 serves as the cold standby. The system failure logic can be expressed as ‘pump 1 fails’ AND (‘pump 2 fails to activate’ OR ‘pump 2 fails following the activation’). Due to the existence of the cold-standby dependency, the Markov model has to be established to carry out the analysis. Two situations are considered, with regard to the functional dependency existing between the electric power supply and the pumps. In the first situation, the two pumps are powered by different electric supplies. When the electric supply to pump 1 fails, pump 2 needs to be activated, despite that pump 1 itself is still functional. In this case, if the functional dependency between the electric supply and pump 1 hadn’t been specified, the need to activate pump 2 would be overlooked, thus the resulting Markov model would be incorrect. In contrast, in the situation where the two pumps share the same electric power supply, the mere failure of the electric supply is sufficient to bring the system to fail. Therefore, there is no need to specify the functional dependency in the second situation. The analyst needs to decide according to the system configuration characteristics whether a particular functional dependency relationship needs to be specified.

4.1.10 Switching Dependency

As with the functional dependency, the switching dependency [7, 8] is identified and introduced with the dynamic fault tree method as the ‘demand dependency’ in chapter 3. Switching dependency is related to active-on-demand systems or active-on-demand components. It identifies that the activation of some component is dependent on the state of other components. See section 3.2.1.4.

Strictly speaking, as another form of functional dependency, switching dependency involves no statistical dependency. It just reflects the functional dependency relationship between the support components and the demand component at the point of time when the demand rises. Similar to the functional dependency, if a particular switching dependency relationship needs to specified in
the analysis depends on the system configurations. Usually in most standby systems, a controller sub-system is included in the system to activate the standby component when required. If the controller sub-system has failed by the time the demand for the activation rises, the standby component can't be switched in. Such a relationship between the standby component and the controller sub-system can be identified as a switching dependency. However, if the system is investigated as a whole from a different perspective, the failure of the duty component can be regarded as an initiating event, while the failure of the control system to activate the standby component can be defined as the corresponding enabling event. When the control system fails prior to the duty component, the failure of the duty component would immediately bring the system to fail. In this case, the switching dependency existing between the control system and the standby component wouldn't need to be explicitly accounted for in the analysis as the impact of this particular switching dependency relationship can be fully reflected by the initiator-enabler dependency. However, in the reliability analysis of systems in which the system configuration is complicated such that it is very difficult to fully reflect impact of the switching dependency relationship through other types of statistical dependency, the switching dependency will need to be explicitly specified to ensure that the correct Markov model be established.

4.2 The Representation of Dependency Relationships
In chapter 3, the dynamic fault tree approach provides a way to represent some types of dependency by introducing additional gate structures, such as functional dependency (FDEP), cold-spare or warm-spare dependency (CSP/WSP), sequential dependency (SEQ) and demand dependency (DDEP). However, this way of representation is appropriate for analysis of a limited number of types of dependency. A new method is proposed here to establish a systematic framework for the representation of all types of dependency.

The dependency information is provided to the software to predict system reliability. The framework is realized through the dependency file which together with the fault tree structure file and basic event file will be input to and stored in the computer program as the data source for the analysis. The dependency file takes the form where each record represents one specific dependency relationship. The structure of the dependency file is illustrated in table 4.1.
The first two fields in table 4.1 provide the same kind of information for different dependency relationships. The 'dependency group number' is an integer number assigned to each specific dependency relationship to distinguish it from others. The 'dependency type' gives which type of dependency this specific relationship belongs to. This piece of information is required for the effective and proper development of the Markov model in the later stage of the analysis. The information that is contained in other 4 fields in table 4.1 varies according to the type of the dependency relationship. The following sections will describe in detail what information is included in each field in view of the different types of dependency.

- Maintenance Dependency
The code 'mtnc' is entered to stand for the maintenance dependency and is input in the field 'Dependency type'. The events to which the maintenance dependency applies include only basic events which represent the failures of system components. To make the information complete, the fields 'Number 1' and 'List 1' respectively give the number of basic events and the name of the basic events involved in this dependency relationship. Field 'Number 2' indicates how many repair engineers or teams are available for the maintenance of the components represented in 'List 1'. Obviously, the value of field 'Number 2' is less than that of field 'Number 1' for maintenance dependency.

It should be noted that if the difference between the value of field 'Number 2' and field 'Number 1' is no less than 2, it means that in some system states there would be more than one component queuing for repair. It is possible that the priority issue will arise with regard to the order in which these components will be repaired. In this case, the maintenance team usually will predetermine the repair priority of each component which is involved in the same maintenance group, in view of the criticality and functional significance of each component. The order of the priority is reflected by the order that the corresponding basic events are listed in 'List 1'. The component which appears earlier in 'List 1' has a higher repair priority. And field 'List 2' is not applicable...
here. Alternatively, the ‘First Come First Serve (FCFS)’ principle can be applied to the repair involving maintenance dependency. When this is the case, field ‘List 2’ will contain the string of characters ‘FCFS’ to indicate which repair policy has been taken with regard to this maintenance dependency.

- **Standby Dependency**

  In view of the different modes the standby system can take, there are 2 categories of standby dependency. The first category is represented by the code ‘sdby1’ which indicates that the duty component and the standby component don’t alternate with each other. More exactly, in the case where the standby component is started due to the failure of the duty component, once the duty component is restored, it will be immediately returned to function while the standby component will then resume the standby state. The second category of standby dependency is encoded as ‘sdby2’. In contrast to the first mode, the preset duty component and the standby component alternate with each other working as the primary component. That is, when the working component fails, the standby component will take over and work as the duty component. When the original working component has been repaired, it will be put into standby mode and will only enter service again when the currently working component, i.e. the original standby component, fails. ‘sdby1’ and ‘sdby2’ are values that will be entered into the field ‘Dependency type’ with regard to the standby dependency.

  There exist two types of element in the standby dependency, the duty component(s) which are always set to function when the system is started, and the standby components. Field ‘Number 1’ and ‘Number 2’ respectively represent the numbers of the duty components and standby components. Correspondingly, ‘List 1’ and ‘List 2’ give the name of the basic events which respectively relate to the failure of duty components and standby components. In theory, the relationship between the value of ‘Number 1’ and ‘Number 2’ can be $1 - 1, 1 - n, m - 1$ and $m - n$, where $m$ is the number of pre-determined duty components and $n$ is the number of pre-determined standby components. In the last two cases, it is usually called ‘pooled spare’.

- **Sequential Dependency**

  The code ‘sq’ is used to represent the ‘Dependency type’ for the sequential dependency. In the
fault tree structure, sequential dependency can be identified through the Priority AND gate. Field 'Number 1' provides the number of the immediate descendants under the Priority AND gate. These immediate descendants can include both basic events and intermediate, gate, events. Field 'List 1' holds the name of the intermediate event which features the Priority AND logic. Field 'List 2' gives the names of the immediate descendants under the Priority AND gate. These events are listed in 'List 2' according to the required order of occurrence.

The sequential dependency relationship can be input in two ways. It can be entered as a record in the dependency file or it will be automatically added to the dependency file by the analysis software when Priority AND gates occur in the corresponding fault tree structure.

• **Sequence-enforcing Dependency**

For the sequence-enforcing dependency, the value of the field 'Dependency type' is set as 'sqef'. As is evident, the sequence-enforcing dependency can only involve basic events. These basic events don't have to be grouped under one gate and might be distributed throughout the fault tree. For the sequence-enforcing dependency, field 'Number 1' indicates the number of basic events involved, and field 'List 1' gives the names of these basic events in the order that they are obliged to occur one after another.

• **Secondary-failure Dependency**

For secondary-failure dependency, the value of the field 'Dependency type' is predetermined as 'scnf'. Since this dependency relationship concerns the primary failure and secondary failure, both of which are single events, field 'Number 1' and field 'Number 2' both automatically take the default value '1'. Field 'List 1' gives the name of the event which represents the primary failure of the component in question. According to the definition, it can only be a basic event that relates to the primary failure. Correspondingly, field 'List 2' gives the name of the event which represents causes of the secondary failure. The event which relates to the secondary failure can be either a basic event or an intermediate event.

With the existence of the secondary-failure dependency, the repair policy must be clarified. Whether it is the primary failure or the secondary failure which occurs, the failure is revealed by
the failed state of the primary component. When the component failure is discovered, a problem rises as to what has caused the component failure. If the secondary failure is self-revealing, the maintenance team will know it is a secondary failure and the primary component and secondary components can come under repair at the same time. However, if the secondary failure is a dormant failure mode, an inspection will be conducted on the secondary components to decide the cause of the component failure as the repair is carried out on the primary component. The purpose of such an inspection is to avoid the situation where the system fails again due to the unrevealed secondary failure after the primary component has been repaired and put back into function. In this sense, secondary-failure dependency may have an overlap with the revealing dependency. When the secondary components feature a dormant failure mode and the primary component failure is self-revealing, the specific repair policy implies that the revealing dependency exists under the secondary-failure event.

- **Initiator-enabler Dependency**

In terms of the initiator-enabler dependency, the abbreviation ‘ie’ is used as the predetermined value for ‘Dependency type’. Similar to secondary failure dependency, the main elements in the initiator-enabler dependency are the initiating event and enabling event. Field “List 1” and field “List 2” respectively give the names of the corresponding initiating event and enabling event. Both of the two events can either be basic events or intermediate events. For example, in the original tank system shown in figure 4.2, the basic event ‘PRS’ is the single enabling event. Whereas, in the amended tank system in figure 4.6, the intermediate event ‘safety measures fail’ in the fault tree in figure 4.7 refers to the general enabling event against the initiating event ‘PS’. In the same way, the initiating event can be an intermediate event. For example, in the tank system, if the event ‘Pump surge’ is developed in more detail to its root causes, it will be developed as an intermediate gate event.

When the enabling event occurs first and the initiating event follows, the specific undesired event will consequently take place. In this case, the repair will be conducted on all failed components at the same time. By contrast, the maintenance team may have 2 options when the initiating event occurs prior to enabling event. The first option is that whilst the initiating event is being repaired, inspection will be carried out on components which relate to the corresponding
enabling events and if any failure is identified, repair will be implemented. This repair policy is aimed to reduce the top event failure intensity by bringing down the failure probability of the enabling event. The second option is that the maintenance team will only restore the initiating event and do nothing about the enabling events. The maintenance team may select which of the different options in view of the system characteristics. The value of the field ‘Number 1’ is set as ‘1’ or ‘2’ according to the selected repair policy.

• Revealing Dependency
With regard to the revealing dependency, the value of the field ‘Dependency type’ is set as ‘revl’. The most important element in the revealing dependency is the specific intermediate gate event, whose occurrence will lead to revealing the dormant failure of some components under it. Field ‘List 1’ gives the name of this intermediate event. Field ‘Number 1’ is designed to indicate the number of the dormant basic events under this intermediate event and ‘List 2’ is used to hold the names of these dormant basic events.

• Test Dependency
The identification label of the field ‘Dependency type’ for the test dependency is simply ‘test’. Since the test dependency is mainly concerning a group of components which compose an independent sub-system or sub-function, the information required is focused on these components. Field ‘Number 1’ indicates how many components are included in the same test group. Field ‘List 1’ gives the names of the basic events which represent the individual component failures. Field ‘Number 2’ gives the inspection interval of the sub-system consisting of the listed components.

• Functional Dependency
When a particular functional dependency relationship needs to be explicitly accounted for in the analysis, it is also included as a record in the dependency file. The abbreviate ‘func’ is entered in the field ‘Dependency type’. With respect to the functional dependency, two factors need to be considered. The first one is the functionally controlling component or sub-system, and the second one is the functionally dependent component(s). Regarding the functional dependency, each dependency relationship centers around the control system and includes only one
functionally controlling component or subsystem. Therefore, field 'Number 1' has the default value '1', and field 'List 1' gives the name of the event which corresponds to the failure of the functionally controlling component or subsystem. This event can be either a basic event or an intermediate event. In contrast, there may exist in the system more than one component which is functionally dependent on the same component or subsystem. Field 'Number 2' gives the information of how many components are dependent on this controlling component. And field 'List 2' gives the names of the basic events which represent the failure of the functionally dependent components.

• Switching Dependency
Similar to functional dependency, switching dependency will only be included in the dependency file when the analysis requires its explicit representation. For switching dependency, the field 'Dependency type' is entered with 'swch'. The structure of the switching dependency takes the same definition as that of the functional dependency. That is, field 'Number 1' is assigned the default value '1'. Field ‘List 1’ gives the name of the event which represents the failure of the supporting component or subsystem. Field ‘Number 2’ indicates the number of the active-on-demand components of which the activation is dependent on the corresponding supporting component. And field ‘List 2’ gives the names of the basic events which represent the failure of the dependent active-on-demand components.

4.3 The Representation of System Failure Logic
The system reliability analysis is based on the knowledge of the system failure logic, which is usually represented in a fault tree structure. To enable the analysis to proceed, the ‘fault tree structure file’ is established which contains the system failure logic.

As has been defined in Chapter 2, a fault tree is a structure which is composed of two elements: events and gates. In a fault tree there exist three types of events: top event, intermediate event and basic event. Gates are used to connect these events in a structure which will represent the system failure logic. As a structure, gates form the framework of the fault tree. The fault tree structure file is actually a specification which is centered around each gate in the fault tree. Table 4.2 displays the structure of the fault tree file.
<table>
<thead>
<tr>
<th>Gate name</th>
<th>Gate logic</th>
<th>Number of immediate gate descendants</th>
<th>Number of immediate basic event descendants</th>
<th>List of immediate gate descendants</th>
<th>List of immediate basic event descendants</th>
</tr>
</thead>
<tbody>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

Table 4.2 The structure of fault tree file

Each record in the fault tree file holds the information of each specific gate in the fault tree. ‘Gate name’ is a list of characters which is assigned to distinguish the specific gate from others. Usually ‘gate name’ refers to the name of the event (top event or intermediate event) to which the corresponding gate is attached to develop the deeper causes. The field ‘gate logic’ reflects how the immediate descendants under the gate contribute to the occurrence of the output event. The value of ‘gate logic’ usually include AND, OR, VOTE and Priority AND. In non-coherent fault trees [37, 38], it may also include the NOT logic, which has not been considered to this point. The third and forth fields respectively give the number of immediate gate descendants and basic event descendants under the specific gate. The value of these two fields can range from 0 to a limited integer. The fifth and sixth fields respectively hold the names of the intermediate events and basic events which form the immediate descendants of the specific gate.

4.4 The Representation of Component Failure and Repair Parameters

In order for the quantification process to proceed and produce accurate system reliability measures, accurate and comprehensive component failure and repair data are required. In the fault tree structure representing the system failure logic, the basic event is used to represent the failure of a system component in a certain mode. Corresponding to the fault tree structure file, a ‘basic event file’ is established, in which each record provides information of the reliability characteristics for a system component.

The structure of the basic event file is shown in table 4.3. It must be noted that the information required in the basic event file may vary in view of different characteristics of different systems. The structure shown in table 4.3 is applied to typical continuously-operating systems. A different structure of basic event file for safety/protection systems, which is characterized by the active-on-demand feature, will be described in Chapter 9.
where: $q$ – constant failure probability; $w$ – constant unconditional failure intensity; $\lambda$ – constant conditional failure rate; $v$ – constant conditional repair rate; $\tau$ – mean time to repair; and $\theta$ – inspection interval.

Table 4.3 The structure of basic event file

<table>
<thead>
<tr>
<th>Name of basic event</th>
<th>Failure model</th>
<th>Reliability parameters</th>
<th>Enabler</th>
<th>Number of dependency groups involved in</th>
<th>List of dependency groups</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>$q, w$</td>
<td></td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>$\lambda, v$</td>
<td></td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>$\lambda, \tau, \theta$</td>
<td></td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

The 'name of basic event' is an identity label which distinguishes the basic event. The 'failure model', as has been described in chapter 2, defines which reliability parameters are applicable. The value of the field 'failure model' ranges from 1 to 3, respectively standing for the fixed, revealed and dormant failure model of the corresponding basic component. Correspondingly, the contents of the field 'reliability parameters' varies according to the different failure model, as is indicated in table 4.3. The field 'enabler' is used when the initiator-enabler dependency exists in the system, and the modeling process needs to identify which basic events are enabling events. The value of 'enabler' ranges from 0 to 2. When it equals 0, it means the basic event is not an enabling event. When it equals 1, it means the basic event is an enabling event and it features the characteristic that once the corresponding initiating event has occurred, it can never occur. When the value is 2, it means the basic event is an enabling event and it can still occur after the initiating event occurs. This different characteristic of the enabling event was discussed in section 4.1.6. The fifth field indicates if the basic event is involved in any dependency relationship in the system. A value is 0 means the basic event is not explicitly included in any dependency group. When the value is greater than 0, the sixth field gives the list of the numbers of the dependency groups in which the basic event is explicitly included. It should be noted that if a basic event is included in a functional dependency or switching dependency group, the group number won’t be included due to that these two types of dependency involve no statistical dependency.
Chapter 5. A New Method for Solving Dynamic Fault Trees with Dependencies

5.1 Overview

As has been discussed in the preceding chapters, the Markov method offers a better alternative to other methods with regard to the assessment of systems which include dependency relationships between components. However, the Markov method lacks the ability to present and document the system failure logic, which is possible in other reliability assessment methods such as fault tree analysis. Therefore an issue arises with regard to how to combine these two different approaches in order to retain the best features of both. The Dynamic fault tree method provides a means to link the fault tree analysis with the Markov method. However, its applicability is significantly weakened due to that it can only take a limited number of dependency types into account and may result in a large Markov model for complex systems.

To overcome these problems, a new approach is proposed here aimed to provide an all-around solution to the assessment of systems which contain dependency relationships. This solution is not only able to combine the fault tree analysis and the Markov method but also able to improve the efficiency of the analysis by generating the smallest possible Markov model for each dependency relationship in the system.

The solution is based on the development of analysis software. System information, including the system failure logic, component reliability parameters and dependency relationships existing between system components, is passed to the software through the input of the fault tree structure file, basic event file and dependency file. After acquiring the basic system information, the analysis software will implement pre-processing on the fault tree structure to obtain a simplified and modularized fault tree structure which is reported in the results. Among the resulting modules, those which include the dependency relationship between the basic events will be represented in their most concise form. These modules will be modeled using Markov models generated according to the specific dependency relationship(s) included. Then the quantification process starts, conventional fault tree assessment methods are applied to modules which involve no dependency relationship. In this way, the conventional fault tree analysis (FTA) and the Markov method are integrated naturally during the analysis process. The strengths of the
fault tree approach are supplemented by the ability of the Markov method to address the dependency problem. Each stage of this approach will be described and illustrated in detail in this chapter and following chapters.

5.2 Pre-processing of Fault Tree Structure – the Identification of the Smallest Independent Fault Tree Modules for Fault Trees Containing Dependencies

It is one of the weaknesses of the Markov method that the model size can become very large, growing exponentially with the number of components. The dynamic fault tree approach to some extent provides the means to overcome the problem, but is limited as it can’t ensure the complete independence of the resulting models. In the solution presented here, the fault tree is preprocessed to identify the smallest independent modules which contain dependency relationships. The whole process can be broken down into the following stages [39]:

1) Re-organize the dependency information
2) Fault tree simplification
3) Form the dependency information
4) Combination of dependent events
5) Modularization
6) Update the dependency information
7) Re-modularize for each dependency relationship

Each of the stages will be explained and illustrated in detail separately in the following sections. For a better understanding, an example fault tree is introduced to illustrate how the preprocessing is implemented and generates the desired result. As is shown in figure 5.1, the example fault tree contains 27 gates and 24 basic events, of which 10 are repeated basic events and appear in the fault tree more than once. To make this example more typical and representative, four types of dependency are assumed to be involved in this fault tree. Maintenance dependency exists between basic events 5 and 8, and between basic events 18, 19 and 20. Sequential dependency exists between basic events 18, 19 and 20. Initiator-enabler dependency exists under gate G17.
Figure 5.1 The Example Fault Tree
Corresponding to these dependency relationships, the dependency file for the example fault tree is established as shown in Table 5.1 below:

<table>
<thead>
<tr>
<th>Dependency group number</th>
<th>Dependency type</th>
<th>Number 1</th>
<th>Number 2</th>
<th>List 1</th>
<th>List 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mtnc</td>
<td>2</td>
<td>1</td>
<td>5 and 8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>mtnc</td>
<td>3</td>
<td>1</td>
<td>18, 19 and 20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>sq</td>
<td>3</td>
<td></td>
<td>G14</td>
<td>18, 19 and 20</td>
</tr>
<tr>
<td>4</td>
<td>ie</td>
<td></td>
<td></td>
<td>G18</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 5.1 Dependency file for the example fault tree

5.2.1 Re-organize the Dependency Information

In terms of the first 8 types of dependency described in Chapter 4 which involve statistical dependency, some concern only basic events, such as maintenance dependency, standby dependency, sequence-enforcing dependency and test dependency. Whereas, other types of dependency relationship involve the intermediate gate events such as sequential dependency and revealing dependency. Initiator-enabler and secondary-failure dependency are both likely to include intermediate gate events too. These intermediate events are listed in the dependency file as part of the dependency relationships. Therefore, the investigation needs to be carried out to decide if these intermediate events should be replaced in the dependency file with their own basic event descendants to more effectively represent the corresponding dependency relationship. The following is the general principle regarding this issue.

In terms of the initiator-enabler dependency, if the initiating event refers to an intermediate event, there is no need to expand it into a list of the basic events under it, since the initiating event is usually self-revealing and can be an independent section in the fault tree structure. Alternatively, when the enabling event appearing in the dependency file is represented by an intermediate event, it is necessary to investigate all its basic event descendants. As this gate is expanded into the cause events, those intermediate descendants can be retained in the list if they contain no basic events whose failure is a dormant failure. The reason for this is that these intermediate descendants could be the gate event which heads an independent module itself since it won’t be involved in a dependency relationship due to its repair.

The secondary-failure dependency is treated in a similar manner to the initiator-enabler
dependency, as the secondary-failure event might also be represented by an intermediate gate event in the fault tree. When this is the case, there exists the need to replace the original secondary-failure event in the dependency file with all its basic event descendants. The reason for this is that the specified repair policy may result in the revealing dependency under the secondary-failure event (See section 4.2). Again in this process, those intermediate gate descendants under the secondary-failure event can be retained in the list if they contain no basic events whose failure is dormant and can be considered as a separate module.

In terms of the sequential dependency, if there exists an intermediate event under the Priority AND gate, it shall be retained as it may form an independent module in the fault tree.

When considering the revealing dependency, no change is necessary if one of more of the dormant failures included in ‘List 2’ in the corresponding dependency file is the immediate descendant of the intermediate gate event specified in ‘List 1’. If this is not the case, then it is possible that during the modularizing process the basic events shown in ‘List 2’ will all be included in a separate module which doesn’t contain the specific intermediate gate event displayed in ‘List 1’. In this case, there would be no way to account for the corresponding revealing dependency in the later stage of the analysis as the dormant failures have been separated from the self-revealing intermediate gate event. Therefore, when the self-revealing intermediate gate event includes none of the dormant-failure basic events as its immediate descendant, its immediate descendants should be added to ‘List 2’ so that these dormant failures will be included in the same module as the specific intermediate event during the modularization process.

All that has been discussed above gives only one reason for the re-organization of the dependency file. As can be seen from table 5.1, there may exist overlap between different dependency relationships, and the same event may appear in more than one record in the dependency file. In this case, to ensure that all the events which share a dependency relationship with the event repeated in several dependency categories will be included in the same module, the records that overlap with each other should be combined. The resulting record no longer represents a single dependency relationship, therefore the original dependency group number is
no longer an appropriate identity tag. Consequently, a 'dependency serial number' is allocated to distinguish the group of dependent events.

Take for example the dependency file in table 5.1, the re-organization will produce the new form of dependency information as is shown in table 5.2.

<table>
<thead>
<tr>
<th>Dependency serial number</th>
<th>Dependency group involved</th>
<th>Events involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5 and 8</td>
</tr>
<tr>
<td>2</td>
<td>2, 3</td>
<td>18, 19 and 20</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>G18 and 15</td>
</tr>
</tbody>
</table>

Table 5.2 Re-organized dependency information

From the re-organized dependency information, it can be established which events need to be included in the same module so that the corresponding dependency relationships can be taken into account during the analysis.

5.2.2 Fault Tree Simplification

Simplification of the fault tree structure is aimed at reducing the fault tree to its most concise form without changing the logic function it represents. The simplification stage is based on the reduction technique which is used in the Faunet code [40]. This provides a good framework to reduce the fault tree to its minimal form. This method consists of four stages:

- **Contraction**: subsequent gates of the same type are contracted to form a single gate. This structures the fault tree as an alternating sequence of AND gates and OR gates.

- **Extraction**: this looks for structures of the type illustrated in figure 5.2 and converts them as illustrated. The effect is to identify the common factor.
Factorisation: pairs of (independent) events that always occur together in the same gate type are identified and combined to form a single complex event.

To enhance the effectiveness of the simplification process an additional operation has been applied.

**Elimination**: this process uses the Boolean laws of absorption:

\[ a + (a \cdot b) = a \]
\[ a \cdot (a + b) = a \]

These laws correspond to the fault tree structures shown in Figure 5.3 below and can therefore be simply reduced to a single event ‘a’.

The absorption law can be extended to fault tree structures containing events that are repeated over any number of levels of a fault tree branch. This can be illustrated by the two examples shown in Figure 5.4:
The extended use of absorption law requires the definitions of primary and secondary gates. A primary gate is the gate at which the repeated variable is first encountered as an input. A secondary gate is a gate, below the primary gate, at which a second occurrence of the event appears as an input. In the two examples shown in figure 5.4, G1 is the primary gate. And G4 and G3 are respectively secondary gates which are the descendants of the primary gate and contain repeated event 'a'. The algorithm is that when the primary and secondary gates are of different types, the secondary gate can be eliminated from the fault tree, while if the primary and secondary gates are of the same type, the repeated event under the secondary gate can be eliminated.

Figure 5.4 Extended elimination
When the absorption results in gates that have only one input, these gates are replaced in the fault tree structure by their single input.

It should be noted that during the simplification process, exceptions may apply to obtain a better result. When a gate structure appears in the fault tree more than once, this gate structure should be skipped as the Contraction is carried out throughout the fault tree, because it might make an independent module and to contract it may be counter-productive and undermine the purpose of the simplification. Also, intermediate gate events, which are directly involved in the dependency relationships, should be kept intact during the simplification process. Besides, attention should be paid to repeated gate structures in the fault tree. The Elimination and Extraction operation may change the input of the repeated gate at a certain location in the fault tree, while the gate of the same name appearing somewhere else in the fault tree maintains the original structure. In this case, a different gate name should be assigned to the modified structure to distinguish it from the original one.

Considering the example fault tree in figure 5.1, the simplification process will result in the minimal form shown in figure 5.5. The reductions include the contractions between G1 and G2, G8 and G9, plus G20 and G23; eliminations of event 2 under G12, plus G24 under G22; extraction of event 10 out of G25 and G26; and finally factorisation of event 22 AND 23, plus event 21 OR 24. To distinguish factors from other elements in the fault tree, these factors are named from 3001 onwards.
Figure 5.5 Simplified fault tree structure

Since fault tree sections which feature interdependent events will be analysed with the Markov method at the later stage, the factorisation process ignores those interdependent components to avoid the need for re-expansion. Therefore complex factors are formed of only independent event types.
5.2.3 Form the Dependency Information

This step allocates the dependency serial numbers which each gate in the fault tree is dependent upon. The information forms the basis for the implementation of the next combination step. The dependency of each gate is defined by a list of all the dependency serial numbers to which basic events below it in the fault tree structure belong.

This step is conducted by traversing the fault tree to decide the dependency serial number that each gate features. This process is illustrated with a simple example (see Figure 5.6).

Assume that after the re-organisation of dependency information, events a and b bear dependency serial number 1, and event c, d and e belong to dependency serial number 2. Therefore the dependency of G1 is the dependency serial of its immediate descendants, i.e., serial 1, since both of its input events a and b feature in dependency serial 1. Similarly, the dependency of G2 is serial 2. And finally the dependencies of G0 is identified as serial numbers 1 and 2 because G1 has dependency serial 1 and both G2 and event e are characterized by dependency serial 2.

Therefore, according to this algorithm, by referring to table 5.2, the dependency serial information of each gate in the simplified fault tree in figure 5.5 is summarized in table 5.3.

<table>
<thead>
<tr>
<th>Gate</th>
<th>G0</th>
<th>G1</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>G6</th>
<th>G7</th>
<th>G8</th>
<th>G10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency serials</td>
<td>1, 2, 3,</td>
<td>1, 2, 3,</td>
<td>-</td>
<td>1, 2, 3,</td>
<td>1, 2</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gate</th>
<th>G11</th>
<th>G13</th>
<th>G14</th>
<th>G15</th>
<th>G16</th>
<th>G17</th>
<th>G18</th>
<th>G20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency serials</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5.3 Gate dependency serials
It should be noted that since G18 represents the general initiating event in dependency group 4, it features the dependency serial number 3 itself in the way like dependent basic events.

5.2.4 Combination of Dependent Events
The purpose of this step is to restructure the fault tree in a way which will separate those events with the same dependency serial into separate branches. Using the information generated in the previous phase, each gate will be examined in turn, additional gates of the same logic type as the gate being investigated, are added where necessary to group the input events (immediate descendants) of the same dependency serial. The reason for implementing the 'combination' phase is that the resulting new gates (numbered from 20001 upwards) are leading to a fault tree structure with the smallest independent sub-trees for each dependency.

For the example in figure 5.6, the application of the Combination will result in a new fault tree structure as shown in figure 5.7. G2 and basic event e, both of which feature dependency serial 2, are grouped under the new gate 20001, which consequently also bears dependency serial 2.

Similarly, when applied to the fault tree in figure 5.5, the 'combination' step will produce the restructured fault tree shown in figure 5.8 with new gates 20001 and 20002.
Figure 5.8. Combination
5.2.5 Modularisation

The task of this phase is defined as to identify modules in the fault tree. A module of a fault tree is a sub-tree that is completely independent from the rest of the tree. After the modularisation, each module will be replaced with a super-event in the original fault tree structure. The super-event has the same reliability characteristics as the fault tree section which it has replaced and is determined using Markov theory or fault tree theory depending on whether the corresponding module contains dependent basic events or not.

The algorithm developed by Rauzy and Dutuit [41] provides an efficient means to identify the modules, which mainly requires two depth-first traversals of the fault tree. The first performs a step-by-step traversal recording for each gate and event, the step number at which the first, second, and final visits to each node were made. It also records the number of appearances in the traversal which will be used in a later stage. In this first traversal, it must be noted that the graph under a vertex is never traversed twice [42]. Therefore when gates appear more than once in the tree, only its first appearance will be traversed completely, after this, its appearances elsewhere in the tree will be treated like a basic event.

In order to ensure that dependent basic events featuring the same dependency serial will end up in the same module, they are treated as a single basic event with the same label during the first traversal. All events in the same dependency group will be replaced with an id that characterizes the particular dependency serial. For example, in the fault tree in figure 5.7, both a and b will be replaced by label 10001, and c, d and e by 10002 (dependency event numbering starts at 10001).

The principal of the algorithm for modularisation is that if any descendant of a gate has a first visit step number smaller than the first visit step number of the gate, then it must also occur beneath another gate. Similarly, if any descendant has a last visit number greater than the second visit number of the gate, then again it must occur elsewhere in the tree. Therefore a gate can be identified as heading a module only if:

• the first visit to each descendant is after the first visit to the gate and
• the last visit to each descendant is before the second visit to the gate
Then the second pass through the fault tree assesses these conditions. The maximum (Max) of the last visits and the minimum (Min) of the first visits of all the descendants (any gates and events appearing below that gate in the tree) for each gate will be obtained based on the result of the first traversal.

Therefore, based on the fault tree in figure 5.8, the two traversals will provide the information given in tables 5.4 – 5.7.

<table>
<thead>
<tr>
<th>Visit number</th>
<th>Gate/ event</th>
<th>Visit number</th>
<th>Gate/ event</th>
<th>Visit number</th>
<th>Gate/ event</th>
<th>Visit number</th>
<th>Gate/ event</th>
<th>Visit number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G0</td>
<td>12</td>
<td>G7</td>
<td>23</td>
<td>G13</td>
<td>34</td>
<td>G14</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>20001</td>
<td>13</td>
<td>G8</td>
<td>24</td>
<td>G11</td>
<td>35</td>
<td>18(10002)</td>
<td>46</td>
</tr>
<tr>
<td>3</td>
<td>G1</td>
<td>14</td>
<td>G10</td>
<td>25</td>
<td>2</td>
<td>36</td>
<td>19(10002)</td>
<td>47</td>
</tr>
<tr>
<td>4</td>
<td>G3</td>
<td>15</td>
<td>3</td>
<td>26</td>
<td>G8</td>
<td>37</td>
<td>20(10002)</td>
<td>48</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>16</td>
<td>4</td>
<td>27</td>
<td>12</td>
<td>38</td>
<td>G14</td>
<td>49</td>
</tr>
<tr>
<td>6</td>
<td>G3</td>
<td>18</td>
<td>G11</td>
<td>29</td>
<td>5(10001)</td>
<td>40</td>
<td>G15</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>G4</td>
<td>19</td>
<td>1</td>
<td>30</td>
<td>6</td>
<td>41</td>
<td>6</td>
<td>52</td>
</tr>
<tr>
<td>8</td>
<td>20002</td>
<td>21</td>
<td>4</td>
<td>32</td>
<td>8(10001)</td>
<td>43</td>
<td>G15</td>
<td>54</td>
</tr>
<tr>
<td>9</td>
<td>20003</td>
<td>22</td>
<td>7</td>
<td>33</td>
<td>20002</td>
<td>44</td>
<td>G16</td>
<td>55</td>
</tr>
</tbody>
</table>

Table 5.4 Visit number for each event in the first traversal (referring to Figure 5.8)

<table>
<thead>
<tr>
<th>Event</th>
<th>9</th>
<th>10</th>
<th>3</th>
<th>4</th>
<th>1</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visit  1</td>
<td>5</td>
<td>6</td>
<td>15</td>
<td>16</td>
<td>19</td>
<td>22</td>
</tr>
<tr>
<td>Visit  2</td>
<td>5</td>
<td>52</td>
<td>15</td>
<td>21</td>
<td>19</td>
<td>22</td>
</tr>
<tr>
<td>Last visit</td>
<td>5</td>
<td>59</td>
<td>15</td>
<td>21</td>
<td>19</td>
<td>22</td>
</tr>
<tr>
<td>No. of visits</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event</th>
<th>2</th>
<th>12</th>
<th>10001</th>
<th>6</th>
<th>10002</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visit  1</td>
<td>25</td>
<td>27</td>
<td>29</td>
<td>30</td>
<td>35</td>
<td>42</td>
</tr>
<tr>
<td>Visit  2</td>
<td>25</td>
<td>27</td>
<td>32</td>
<td>41</td>
<td>36</td>
<td>42</td>
</tr>
<tr>
<td>Last visit</td>
<td>25</td>
<td>27</td>
<td>32</td>
<td>41</td>
<td>37</td>
<td>42</td>
</tr>
<tr>
<td>No. of visits</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.5 Event visit table (referring to Figure 5.8)

Note: when establishing the event visit table, G18 is treated as the dependent basic event with id 10003.
According to the conditions for a module, the gates marked in table 5.7 such as G0, 20002, G7, G8, G14, G17 and G18, are identified as heading the modules. To distinguish these modules from other events in the fault tree, these modules are assigned a unique id starting from 6001 onwards. Their structure is shown in figure 5.9.

![Figure 5.9 Modules identified – Module 6001](image-url)
Figure 5.9 Modules identified – module 6002 - 6007
5.2.6 Update the Dependency Information

By this stage, independent sub-trees have been identified. However, with the aim to find the smallest modules which contain dependent basic events, the task has not been accomplished yet. To attain the aim, two points must be made clear: the first is which modules contain which dependency serial; and the second is whether these modules are the smallest one. This step is designed to provide the information required to answer these two questions.

Slightly different from step 3, dependency information is updated establishing not only which dependency serials each gate contains but also its mutual dependency serials. The mutual dependency serial of a gate is a list of dependency serials which all of its immediate descendants feature.

Take module 6002 in figure 5.9 for example, it can be determined that gate G5 contains dependency serial 1 and since only one of its two input events features dependency serial 1, it has no mutual dependency serial. It is a different case for gate 20002: since both of its input events, gate G6 and event 8 features dependency serial 1, gate 20002 bears dependency serial number 1 as its mutual dependency serial.

Accordingly, table 5.8 below gives the dependency serial information of each of the modules shown in figure 5.9.

<table>
<thead>
<tr>
<th>Gate</th>
<th>G0</th>
<th>20001</th>
<th>G1</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>20002</th>
<th>G6</th>
<th>G7</th>
<th>G8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependencies contained</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Mutual dependency</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gate</th>
<th>G10</th>
<th>G11</th>
<th>G13</th>
<th>G14</th>
<th>G15</th>
<th>G16</th>
<th>G17</th>
<th>G18</th>
<th>G20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependencies contained</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mutual dependency</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Notes:

1. "-" means each of the immediate descendants of the gate contains different dependency. Therefore there is no mutual dependency for the gate.
2. Shaded gates are the top nodes of each module displayed in figure 5.9.

Table 5.8 Dependency information (referring to Figure 5.9)
5.2.7 Re-modularise for Each Dependency Relationship

The previous step has identified which module(s) contain interdependent events. This step determines whether a certain module is already the smallest module for a given dependency, or if not, to further identify the smallest independent sub-tree for the given dependency relationship(s) from within a module.

Firstly, it has to be determined whether a certain module is already the smallest one for the given dependency. This is accomplished by using the dependency information provided by the preceding step. If any module contains a given dependency serial number and its mutual dependency also includes this given dependency serial, it can be concluded that this module is already the smallest module and does not need further processing.

If the module does not satisfy this condition, the search for the smallest module of the given dependency serial continues.

Accordingly, in terms of the 7 modules in figure 5.9, judging from the dependency information in table 5.11, it can be concluded that module 6005 led by G14 and module 6006 led by G17 are already the smallest modules for dependency serial numbers 2 and 3 respectively.

After it has been decided that a certain module may not be the smallest one for a given dependency, the search continues within the module to find out the smallest independent sub-tree for the given dependency. This process can be generalized by the following steps:

a) Traverse the module from its top event, always following the gate which contains the given dependency and recording the path, until the gate is encountered whose mutual dependency also includes the given dependency.

For example, regarding module 6002 in figure 5.9, for dependency serial 1, the path will be: 20001, G1, G4, G5, 20002. Gate 20002 is the first gate encountered in this path whose mutual dependency includes dependency serial 1.
b) The gate identified in step (a) as having the correct mutual dependency serial would be leading the smallest independent sub-tree for the given dependency serial if it had been identified as a module. The fact that it is not a module indicates that some of its descendants must have occurred elsewhere in the module. Therefore, in this step, those preventing elements have to be identified which lie outside the fault tree section headed by the gate with the specified mutual dependency serial.

Those descendants which are preventing elements need to be identified. One solution is to see whether the number of appearances of any descendant under this gate is the same as the number of appearances of the specific descendant in the whole module. If it is different, that descendant is the preventing element.

For example, event 6 is the preventing element under gate 20002 because event 6 occurs only once under gate 20002 but occurs twice in module 6002 (see table 5.5).

c) After the preventing elements have been detected, the next thing is to identify a new module which includes those preventing elements. The way of doing this can be illustrated by the specific example of module 6002.

First some information should be listed:

The Path downwards to the mutual dependency event is:

20001, G1, G4, G5, 20002

The potential module: 20002

Preventing element: event 6 with another occurrence at location 41 in table 5.4.

From event 6 at location 41 the tree is traversed upwards and its antecedents listed:

Event 6, G15, G4

The search stops at G4 because G4 is the first gate which also appears in the descending Path for gate 20002. It indicates that G5 and G15 are both immediate descendants of G4 and that G15 contains the preventing element event 6. So the potential module is now updated to include the combination of G5 and G15.
In the new potential module, no preventing elements have been detected, therefore the combination of G5 and G15 is the new module labelled 6008 which is smallest for dependency serial 1. See figure 5.10.

If the new potential module contains any preventing elements, the procedures are repeatedly applied until a new module is identified.

d) If a module includes more than one dependency serial, these dependency serials shall be dealt with one after another in the same way.

Having progressed through the 7 stages, modules 6005, 6006, 6007 and 6008 are identified as the smallest modules for dependency serials 2, 3, 4, and 1 respectively. These four modules will be handled with the Markov method in the later stage of analysis which will be discussed in Chapter 8.
5.3 Discussion

The algorithm presented in this chapter will enable the efficient analysis of fault trees which contain dependent basic events. The method will allow the fault tree to be modularised into independent sections. Analysis of the sections will be by the Binary Decision Diagram (BDD) method or Markov method depending upon the existence of dependent events or not within the section. The key element in terms of the efficiency of this approach is to reduce the sections analysed using Markov method to the minimum possible size (number of basic events).
Chapter 6. The Generation of Markov Models

6.1 Introduction

As has been discussed in the previous chapter, the new solution to the assessment of systems which contain dependency relationships integrates the Markov method into conventional fault tree analysis. In the solution, the Markov method is applied to modules in which dependency relationships are involved.

The application of the Markov method is realized through the generation of a Markov model which represents the characteristics of the particular dependency relationships included in the module. The manual development of the Markov models, due to their size, can be error-prone. In the analysis method presented, the Markov models need to be generated automatically to avoid this pitfall. This chapter establishes the algorithm for developing the Markov model for each type of dependency relationship by examining their unique characteristics.

As is defined in chapter 3, a complete Markov model is composed of three elements: a list of system states, the transitions between these states and the corresponding transition rate. All these 3 elements will be considered during the model generation process. Usually the state transition is caused by the change in the state (failure or repair) of basic (component) events included in the system. The corresponding transition rate is the failure rate or repair rate of the relevant component. A basic assumption of the analysis method is that only one transition is possible in an infinitesimally small element of time.

During the model generation process, some operational details of the system need to be considered as they may have influence on the transitions between system states. At least two general system characteristics have to be clarified before the development of the Markov model. First, it has to be made clear if the system will be continuously operating or active-on-demand. Different assessment approaches are applied accordingly to assess different types of system. The development of the Markov model will also differ. Secondly, the analyst must be clear if it is possible to carry out maintenance during the system operational phase. This will not only affect the transitions between system states but also decide which types of dependency relationship
need to be accounted for. For example, if maintenance is not possible, the dependency relationships need not be considered which involve the repair process. In this chapter, the model generation is examined with regard to continuously operating systems for which maintenance is possible.

6.2 The Generation of Markov Models for Different Types of Dependency Relationship

6.2.1 Model Generation for the Maintenance Dependency

The process for generation of a Markov model, where components have a maintenance dependency, will be illustrated using an example. Module 6005 in figure 5.9 is used for this purpose. Assume there exists only a maintenance dependency between the basic events 18, 19, and 20. The following part of this section will look into the process of developing the Markov model where failure events are either revealed or dormant.

Each system state included in the Markov model represents a combination of possible states for the components included in the system or module. In terms of the basic events in the original fault tree structure, two potential states can be identified: one represents the basic event has occurred, and the other represents the basic event has not occurred. Usually these two states correspond to the failed state and working state of each component. When the component has failed, further division is necessary to distinguish if this occurrence is revealed or not. Therefore, during the process of model generation, integers '0', '1' and '3' are introduced to represent the three states of the component: 'working', 'failed revealed' and 'failed dormant' respectively. For components whose failure is revealed, only the first two states are relevant.

In view of the maintenance dependency, another state has to be identified which represents that the component is waiting for correction due to the temporary unavailability of the repair resource. When a basic event is tagged with the state '1', it means its failure of the corresponding components has been revealed and the component is under repair. In contrast, integer '2' is introduced to signify the 'queuing-for-repair' state of the basic event. It should be noted that only when the occurrence of the component failure has been noticed, can it join the queue for the repair.
Model Generation with Revealed Failures

Usually the development of a Markov model requires an initial system state to be established first. In most cases, for continuously running systems, the initial system state is configured with each component working normally. That is, in the initial system state, no basic events have occurred. With the initial state established, the Markov model can be gradually developed by systematically considering the transitions that make each component's state change. The algorithm is that the process starts from the initial state and investigates how each component state can change. A new state is generated for each possible transition. If this state is different from all currently existing states, it will be added to the list of system states in the Markov model. When each state in the Markov model has been considered, and all possible transitions from these states result in another state which is already included in the model, the Markov model is complete.

Table 6.1 and 6.2 below displays all the system states and the transitions between these states included in the Markov model for module 6005 in figure 5.9, which are established according to the algorithm presented above. It is assumed that only the maintenance dependency is involved and all components in the module have revealed failures.

<table>
<thead>
<tr>
<th>State number</th>
<th>Basic event state</th>
<th>Module state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0 0 1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1 2 0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1 0 2</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2 1 0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0 1 2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2 0 1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0 2 1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1 2 2</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>2 1 2</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>2 2 1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.1 Markov model states (referring to module 6005 in figure 5.9)
In Table 6.2, 'module state' indicates if the top event of the module occurs (1) or not (0) given the specific combination of the basic event states. It can be noted that since only 1 repair engineer is responsible for maintaining and repairing the 3 basic components (see Table 5.3 in Chapter 5), the subsequent component failures will not be corrected immediately and have to go into the queue for repair. This characteristic can be reflected in the transitions from state 2 to state 5 and 6, and from state 3 to state 7 and 8 etc. When the repair engineer becomes available by completing the repair of a failed component, they will then start working on the failed components which are queuing for repair. When there is more than one queuing component, the engineer then has to choose one from the queuing list according to the predetermined repair priority. For example, when the module is residing in state 11, the repair of the failed component 18 means the repair engineer can now commence working on components 19 or 20, since component 19 has been assigned a higher level of repair priority, it will come under repair prior to component 20. This leads to the transition from state 11 to state 8. Transitions from state 12 to state 6 and from state 13 to state 5, also mirror the same kind of situation.

By integrating the information provided in Table 6.1 and 6.2 and translating it into a graphic presentation, the Markov model for module 6005 regarding the maintenance dependency is shown in Figure 6.1.
In Figure 6.1, each circle represents a state listed in Table 6.1. The arrows between the circles stand for the transitions between states which are displayed in Table 6.2. The symbol attached to each arrow refers to the rate of each transition which is decided by the corresponding conditional failure or repair rate.

If the components whose failure is represented by basic events 18, 19 and 20 are all of the same type of component and feature exactly the same reliability parameters, in this case, the conditional failure rate, λ, and conditional repair rate, ν, the state lumping technique introduced in Chapter 3 can be applied here to reduce the size of the Markov model. Figure 6.2 below gives the reduced Markov model where the number of components in each of the possible states is used to define system states. 'W' means the component is working normally, 'F' means the component is failed revealed and under repair, and 'Q' means the component is queuing for repair.
It can be noted that each state in figure 6.2 is actually a reflection of the states in each column in figure 6.1. By identifying and combining states which represent the same system characteristics, the state lumping technique can effectively reduce the size of the resulting model. However, as has been mentioned in chapter 3, only when a series of conditions are fulfilled, can the state lumping be applied. For example, module 6008 in figure 5.10 also includes the maintenance dependency which exists between basic event 5 and 8. In this case, even if the corresponding components are of exactly the same type and characterized by the same reliability parameters, the state lumping technique is inapplicable to form the Markov model. The main reason is that, as can seen from the fault tree structure of module 6008 in figure 5.10, basic events 5 and 8 have different contributions with respect to the top event. More importantly, without being able to identify which specific basic event (5 or 8) has occurred in the Markov model, it is not possible to correctly determine if the top event has occurred or not. In contrast, in module 6005, since all of the 3 basic events are immediate descendants of the top event, they provide the same contribution to the system structure. Also in this case, with the same reliability parameters, the repair priority is only a nominal guide which helps the engineer choose the next component for repair, and has nothing to do with the component criticality.

- Model Generation with Dormant Failures

Now consider the circumstances where all the basic events 18, 19 and 20 in module 6005 have an unrevealed dormant failure mode. This means that at the time a component fails, the failure is unrevealed and will be discovered only when the regular inspection of the component is carried out. Assuming that the basic events 18, 19 and 20 all feature the same reliability parameters, the resulting reduced Markov model is established as follows in figure 6.3 where the state 'FU' means the component is failed but unrevealed:
Figure 6.3 The reduced Markov model with dormant failures (referring to module 6005 in figure 5.9)

In figure 6.3, it can be noted that some state transitions are symbolized by the dotted arrow instead of a solid-line arrow. These dotted arrows represent the state transitions which are caused by the inspection which reveals the dormant failure(s). Unlike other state transitions which are caused by component failure or repair and occur at a certain rate, the occurrence of these inspection transitions are mandatory and instantaneous at the specific points of time. If the system is residing in one of the states from which the dotted arrow reaches out, the transitions represented by the dotted arrow will not occur until the inspection is conducted.

In this particular example, because it is assumed that the 3 basic events all have same reliability parameters and feature the same inspection interval, the dormant failures will be revealed together. This can be seen from the transitions 3→6, 4→10 and 9→10. When the dormant failure is revealed, if the repair resource is available, the failed component will come under repair immediately, if there is no spare repair resource, the component will have to go into the queue for repair. When more than one dormant failure is revealed together, the repair resource will be assigned to the components according to their repair priority.
In many cases, different components which can fail in a dormant mode will have different inspection intervals. In these circumstances, inspections may coincide and multiple dormant component failures will be revealed together, but at other points of time only one or a fraction of the dormant failures are discovered. In this case, the problem arises as to how the different situations are represented through the state transitions in the Markov model. The following example is used to demonstrate how this can be achieved.

Assume that in module 6005, basic event 18 features a 2-month inspection interval, whilst basic events 19 and 20 are respectively assigned the 3-month and 4-month inspection interval. In the model generation process consider the development of a state in which the three basic events are all tagged with the state code ‘3’, i.e. ‘failed unrevealed’. Figure 6.4 and table 6.3 provide an insight into how the state transitions can occur in a different way depending on time.

Figure 6.4 State transitions with multiple dormant failures (referring to module 6005 in figure 5.9)

<table>
<thead>
<tr>
<th>State transitions</th>
<th>Points of time for occurrence (months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1 → State 2</td>
<td>$T = 2k$, $T \neq 3k$ or $4k$</td>
</tr>
<tr>
<td>State 1 → State 3</td>
<td>$T = 3k$, $T \neq 2k$ or $4k$</td>
</tr>
<tr>
<td>State 1 → State 4</td>
<td>$T = 4k$, $T \neq 3k$</td>
</tr>
<tr>
<td>State 1 → State 5</td>
<td>$T = 6k$, $T \neq 4k$</td>
</tr>
<tr>
<td>State 1 → State 6</td>
<td>$T = 12k$</td>
</tr>
</tbody>
</table>

Table 6.3 Time conditions for state transitions (referring to figure 6.4)

$k \in \{1, 2, 3, 4 \ldots \}$

For example, if currently the mission time that has elapsed is 18 months, this corresponds to an
inspection point for components 18 and 19 whose failure will be discovered resulting in the transition from state 1 to state 5. If the current time is 20 months, components 18 and 20 will be tested and the transition from state 1 to state 4 will occur. In the same way, different time conditions can obtained for transitions from state 2 in figure 6.4. See table 6.4:

<table>
<thead>
<tr>
<th>State transitions</th>
<th>Points of time for occurrence (months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 2 → State 4</td>
<td>T = 4k, T ≠ 3k</td>
</tr>
<tr>
<td>State 2 → State 5</td>
<td>T = 3k, T ≠ 4k</td>
</tr>
<tr>
<td>State 2 → State 6</td>
<td>T = 12k</td>
</tr>
</tbody>
</table>

Table 6.4 Time conditions for transitions from state 2 (referring to figure 6.4)

A systematic approach has been developed to ensure that each possible state transition will be considered when there exists more than one dormant failure. This approach is applied through the following steps:

- Calculate the common multiple \( \theta_e \) of the inspection interval of the basic events which in the current state are tagged with the state code '3'.
- Consider the first basic event with the current state '3': Start with its inspection interval \( \theta_e \). Establish the possible state transitions when \( T = \theta_e, 2\theta_e, 3\theta_e, \ldots \) till \( T \) reaches the \( \theta_e \). At each of the points of time, make the new state of the basic event to be revealed, i.e. '1'. Meantime record the time condition for each transition.
- Consider the next basic event with state '3'. Repeat step b until every basic event with state '3' has been examined. Points of time which have been considered with the previous basic events will be skipped.

Figure 6.5 Algorithm of establishing state transitions with multiple dormant failures

The application of the approach described in figure 6.5 is illustrated using the same example as illustrated in figure 6.4. Considering transitions from state 1 in figure 6.4, and referring to the algorithm presented in figure 6.5, the process of establishing the transitions and their corresponding time conditions, shown in figure 6.4 and table 6.3, can be represented by the flow chart in figure 6.6:
Figure 6.6 Illustration of the algorithm presented in figure 6.5

In figure 6.6, it can be seen that when $T = 10$, the same state transition occurs as when $T = 2$. It is the same case between $T = 4$ and $T = 8$, and between $T = 3$ and $T = 9$. In this case, the time conditions $T = 10$, $T = 8$ and $T = 9$ can be omitted. Therefore, the final list of time conditions will be as displayed in table 6.5 below:

<table>
<thead>
<tr>
<th>Time conditions</th>
<th>State transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T = 2$</td>
<td>State 1 $\rightarrow$ State 2</td>
</tr>
<tr>
<td>$T = 3$</td>
<td>State 1 $\rightarrow$ State 3</td>
</tr>
<tr>
<td>$T = 4$</td>
<td>State 1 $\rightarrow$ State 4</td>
</tr>
<tr>
<td>$T = 6$</td>
<td>State 1 $\rightarrow$ State 5</td>
</tr>
<tr>
<td>$T = 12$</td>
<td>State 1 $\rightarrow$ State 6</td>
</tr>
</tbody>
</table>

Table 6.5 Time conditions for state transitions (referring to figure 6.6)

For example, if the current time elapsed is 18 months, the biggest number in the time condition list which can divide 18 evenly is $T = 6$, which combines the state transitions for $T = 2$ and $T = 3$. The corresponding state transition which will occur is from state 1 to state 5. The algorithm generalized in figure 6.5 leads to the same result as is manifest in figure 6.4.
So far this section has illustrated the whole process of developing the Markov model which involves maintenance dependency for small specific example. This process has been generalized to account for any number of components of all revealed/unrevealed failure in figure 6.7.

```
establish_markov('mtnc')
{
    establish_initials(events);  /* all events are set as non-occurred in the initial state */
    while(initial state ... last system state)
    {
        while(first event ... last event)
        {
            if(current event state = 0)
            {
                if(failure model is revealed)
                    if(repair is available)
                        new event state = 1;
                    else
                        new event state = 2;
                }
                else
                    new event state = 3;
            }
            else
            {
                if(current event state = 1)
                    new event state = 0;
                else
                {
                    if(current event state = 3)
                        apply the algorithm proposed in figure 6.5;
                    else
                        ; /* when the component is queuing for repair, it is unlikely to change its state until the repair resource becomes available */
                }
            }
        }
        if (the newly created state produced by the state change of the specific event has existed)
        ;
        else
        {add the new state into the state list;
            decide the module state in the current system state;
        }
        record the transition and the transition rate;
    }
}
```

Figure 6.7 The algorithm for the generation of the Markov model involving maintenance dependency
6.2.2. Model Generation for Standby Dependency

The use of redundancy in design to enhance reliability is common in many engineering systems. This approach is aimed to enhance the system's ability to tolerate some key component failures and to ensure its continuous operation. In most cases, the standby component or subsystem is accompanied by a support subsystem which detects the need to activate the standby subsystem and to implement the activation. Therefore, according to the definition in chapter 4, it can be concluded that the existence of the support subsystem results in a switching dependency between the standby component and the support components. In most standby systems, the standby dependency exists together with the switching dependency. An additional complication for standby systems is that the existence of the switching dependency also introduces the initiator-enabler dependency. The water tank system in figure 6.8 provides a good example.

![Figure 6.8 Simplified water tank system](image)

The water tank system shown in figure 6.8 is a storage facility which provides water to other parts of a process. The level in the water tank has to be maintained at a certain height to ensure the system can function normally. To monitor the water level, a level sensor is fitted to the tank. Pump 1 and pump 2 transport water to the tank. When the system is started, pump 1 is set as the duty pump and pump 2 acts as the standby backup. In the event that pump 1 fails, the level monitor in the tank will detect the drop in the water level and send a signal to the controller which will accordingly switch on pump 2. Since pump 1 is set as the main duty pump, once repaired it will be put back into function immediately.

Assume the assessment focuses on 'insufficient water supply from the tank', then the corresponding fault tree can be drawn up as is shown in figure 6.9 below:
Insufficient water supply from the tank

![Fault tree structure for water tank system](image)

**Figure 6.9** Fault tree structure for water tank system (referring to figure 6.8)

It can be seen that in this system, the standby dependency exists between basic events P1 and P2. Pump 2 also features the switching dependency on the controller and level sensor. Now we will look into how the initiator-enabler dependency occurs in the system.

Assume that pump 1 fails while other components are still working normally. Pump 2 will then start to continue the water supply to the tank. In the event that the controller fails after it activates pump 2, then the basic events which exist include P1 and C. From the failure logic given in the fault tree structure in figure 6.9, the top event should have occurred in this case. However, in reality, since pump 2 has been successfully activated, the top event does not occur. Alternatively, if the controller fails prior to pump 1, then the subsequent failure of pump 1 will not result in the activation of pump 2 due to the unavailability of the controller. In this case, the top event occurs, as indicated by the failure logic presented by the fault tree structure in figure 6.9, P1 AND C. These two different cases indicate that the AND logic of gate G0 in the fault tree in figure 6.9 is not sufficient to totally prescribe the conditions of the system failure. Instead, the initiator-enabler dependency has to be introduced here to reflect the requirement on the specific occurrence order of basic events C, M and P1.

However, another problem rises when the basic events come under examination. Each component may have different failure modes and each basic event should represent only one failure mode of the specific component. Correspondingly, the reliability parameters of each basic
event are established for only one failure mode of the component. In the water tank system in figure 6.8, pump 2 may have two different failure modes: one is the dynamic failure which refers to the failure of pump 2 while it is running; the other is the static failure representing the failure of pump 2 when it is residing in the standby state. These two failure modes should be characterized by different reliability parameters and represented by two different basic events. However, in the fault tree structure in figure 6.9, only one basic event is used to represent the failure of pump 2.

To resolve this problem, one option is to distinguish between the different failure modes of pump 2 by introducing another basic event. The fault tree structure then becomes that shown in figure 6.10:

![Figure 6.10 Amended fault tree structure for the water tank system](image)

In this fault tree, basic events ‘P2’ and ‘P2S’ represent the dynamic and static failure of pump 2 respectively. This approach has two advantages.

Firstly, the different failure modes of pump 2 don’t have to be mixed together. The characteristics relevant to each failure mode can be clearly distinguished through the information provided in the basic event file. For example, the basic event ‘P2S’ will be identified as an enabling event, as will basic events ‘C’ and ‘M’. Gate G2 is the general enabling event against the initiating event ‘P1’.
Secondly, the new fault tree structure provides a more accurate representation of the system failure. It reflects the different phases of system operation. The analyst can tell directly from the fault tree structure if the standby is a cold-spare or not. When the basic event 'P2S' is not included in the fault tree, it indicates that pump 2 will not fail in standby and therefore it is a cold spare.

With regard to the new fault tree structure, the corresponding dependency information and the basic event information is given in table 6.6 and table 6.7.

<table>
<thead>
<tr>
<th>Dependency group number</th>
<th>Dependency type</th>
<th>Number 1</th>
<th>Number 2</th>
<th>List 1</th>
<th>List 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sdby1</td>
<td>1</td>
<td>1</td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>2</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>G2</td>
<td>P2</td>
</tr>
<tr>
<td>3</td>
<td>ie</td>
<td>2</td>
<td>-</td>
<td>P1</td>
<td>G2</td>
</tr>
</tbody>
</table>

Table 6.6 Dependency information for the water tank system (referring to figure 6.10)

<table>
<thead>
<tr>
<th>Name of Basic event</th>
<th>Failure model</th>
<th>Reliability parameters</th>
<th>Enabler</th>
<th>Numbers of dependency groups involved in</th>
<th>List of dependency groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>2</td>
<td>λ_p1, ν_p1</td>
<td>0</td>
<td>2</td>
<td>1 and 3</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>λ_p2, ν_p2</td>
<td>0</td>
<td>2</td>
<td>1 and 2</td>
</tr>
<tr>
<td>P2S</td>
<td>3</td>
<td>λ_p2', ν_p2, θ_p2</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>λ_c, τ_c, θ_c</td>
<td>2</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>M</td>
<td>3</td>
<td>λ_m, τ_m, θ_m</td>
<td>2</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Note: Switching dependency is only specified and included in the dependency information for illustration purpose. It doesn't need to be specified since it is fully reflected through the ie dependency.

Table 6.7 Basic event information (referring to figure 6.10)

In table 6.7, since basic event 'P2S' represents the static failure of pump 2, it is characterized by a dormant failure model and also features a different failure rate, \( \lambda_{p2}' \), from that of basic event 'P2', \( \lambda_{p2} \).

Markov analysis is used for the reliability prediction of the system. Due to the large size of the corresponding Markov model, part of the Markov model developed for the water tank system based on the fault tree structure in figure 6.10 is shown in figure 6.11 to illustrate key system states and transitions.
In the Markov model in figure 6.11, a new basic event state code ‘-1’ is introduced to represent the standby state (in this case applied to pump 2). The basic event $P_2$ has a state change from ‘-1’ to ‘0’ when pump 2 is activated. The inactive failure of the standby component is reflected through the change of the state from ‘0’ to ‘1’ or ‘3’ (‘failed revealed’ or ‘failed unrevealed’) for the basic event which represents the failure of the component in standby. For example, in figure 6.11, the state change from ‘-1’ to ‘0’ of basic event $P_2$ is caused by the failure of pump 1 as represented by the state transition from state 1 to state 2 for basic event $P_1$. The switching dependency existing between $G_1$ and ‘$P_2$’ is reflected through the state transitions from state 3 to state 8, from state 4 to state 9 and from state 5 to state 10. For instance, in state 3, the functionally controlling event $G_1$ has occurred, under this circumstance, the failure of pump 1
won't activate pump 2 because the pump 2 has been rendered inaccessible by the failure of the controller. Also the initiator-enabler dependency can be identified in these transitions. The occurrence of the initiating event 'P1' following the occurrence of the enabling events causes the top event which consequently reveals the dormant component failures, such as the controller, the sensor and the standby pump 2. The initiator-enabler dependency is also evident in state 6 and state 7 in figure 6.11. When the system is residing in either of these two states, the failure logic presented in the fault tree structure in figure 6.10, will indicate that the system will be failed. However, due to the identification of the initiator-enabler dependency, it can be concluded that the system has not caused the top event since the enabling event has occurred after the initiator.

This approach presented and illustrated above can also apply to the situation where pump 1 and pump 2 alternate as the duty pump on the condition that pump 1 and pump 2 feature exactly the same failure and repair characteristics, in which case it won't make any difference to the system reliability measures whether the two pumps rotate with each other or not.

However, when pump 1 and pump 2 rotate with each other as the duty pump with different failure and repair parameters, the approach presented above would be inadequate. The problem lies in modeling the change in the role of each pump. In this case, both pumps can fail in standby or when functioning. The pump failure can either be an initiating event or an enabling event with respect to each other, depending on which pump is currently working and which pump is serving as the standby. To obtain accurate system reliability measures, the fault tree structure must be able to capture the dynamic feature in the analysis brought about by the rotation between the two pumps as well as to correctly reflect the system failure logic. This cannot be achieved by simply distinguishing the different failure modes (standby/active) of the key components.

To overcome the inadequacy of the first approach, a second method is proposed here based on a different fault tree structure to tackle the rotating standby characteristic. In this method, the system failure is investigated in two different situations. The first situation occurs when the system operates with pump 1 set as the duty pump, and then fails according to the failure logic presented in the fault tree shown in figure 6.10. The other situation is where the system operates with pump 2 set as the duty pump, and then fails according to the failure logic presented in figure.
6.10 with the role of pump 1 and pump 2 swapped. These two situations are exclusive to each other and will never overlap. Figure 6.12 illustrates how the two system modes alternate with each other.

Figure 6.12 Alternation between two different system modes

In figure 6.12, in system mode 1, the failure of pump 1 causes pump 2 to activate. Whilst pump 1 is going under repair, pump 2 fulfills the role of a standby pump and ensures the system's continuous operation. Once pump 1 gets repaired, it is put into the standby position which swaps of the role of the two pumps. From this moment, pump 2 is prompted to a duty pump with pump 1 serving as a standby and the system switches to mode 2. The same process applies when the system is operating in mode 2. With the failure and the subsequent repair of pump 2, the system switches back into mode 1. The system operates in either mode 1 or mode 2 but never in both. According to such a characteristic, a new gate type 'Switching gate' is introduced to represent the alternating feature between its. Figure 6.13 illustrates how the switching gate fits into the fault tree structure.

Figure 6.13 Switching gate structure

In figure 6.13, the $n$ input events represent the system failure in $n$ different modes in which the
system can operate. These modes alternate with each other but never co-exist. The output event occurs when one of the input events occurs.

With the introduction of the switching gate, the fault tree structure is constructed as in figure 6.14 to represent the failure of the system, shown in figure 6.10, which features the rotating warm-spare dependency.

![Fault tree structure for rotating-standby dependency](image)

**Figure 6.14 The fault tree structure for rotating-standby dependency**

The corresponding dependency relationships existing between the basic events included in the fault tree in figure 6.14 are displayed in table 6.8.

<table>
<thead>
<tr>
<th>Dependency group number</th>
<th>Dependency type</th>
<th>Number 1</th>
<th>Number 2</th>
<th>List 1</th>
<th>List 2</th>
<th>Relevant in input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sdby1</td>
<td>1</td>
<td>1</td>
<td>P1</td>
<td>P2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Sdby1</td>
<td>1</td>
<td>1</td>
<td>P2</td>
<td>P1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>ie</td>
<td>2</td>
<td>-</td>
<td>P1</td>
<td>G5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>ie</td>
<td>2</td>
<td>-</td>
<td>P2</td>
<td>G6</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 6.8 Dependency relationships in rotating standby systems**

In table 6.8, the rotating standby dependency is broken down into two non-rotating dependency...
relationships between pump 1 and pump 2 corresponding to the two different system modes. The initiator-enabler dependency relationships are also identified in both system modes which determines if the dependency exists between any basic events in a specific system mode.

The Markov model is then established based on the fault tree shown in figure 6.14. To enable the model to correctly represent the system behavior, it is necessary to keep track of the alternation between the system modes during the process of developing the Markov model. Due to the size of the resulting Markov model, only part of the model, which displays typical model features, is shown in figure 6.15.

![Figure 6.15 Part of the Markov model for the rotating standby system](image)

The section of the Markov model shown in figure 6.15, highlights the alternation between the two different system modes. The system starts in mode 1 with pump P1 operational (basic event P1S is inapplicable). The system switches to mode 2 when pump 1 fails and is subsequently repaired and put back to the system as a standby. The transition from state 1 to state 2 represents the failure of P1 and the transition from state 2 to state 3 represents its repair and the transition of the system to mode 2. As a rule for the development of the Markov model, the system mode changes when the current duty pump gets repaired following its failure. For example, in state 4, the system is functioning in mode 2. According to the dependency file shown in table 6.8, the duty pump in mode 2 is pump 2. Consequently, the repair of pump 2 in state 4 results in the shift
of the system mode, illustrated by the state transition from state 4 to state 1. The necessity to distinguish the different system modes in the process of developing the Markov model can be highlighted by the transition from state 2 to state 7. With mode 1 identified as the system operating mode, the failure of the controller can be identified as an enabling event following the occurrence of the initiating event P1 and therefore will not lead to the fault tree top event. Knowledge of the current system operating mode is required to determine the correct dependency relationships between the basic events and determine if the system is failed or not in state 7. Transitions from state 5 to state 1 and from state 6 to state 1 occur due to the repair of pump 1 and pump 2 at the same time. Their transition rates would be determined separately as discussed in section 6.3.

In addition to the switching dependency which usually accompanies the standby dependency, the functional dependency is also frequently found in systems featuring the standby dependency. For example, again consider the water tank system shown in figure 6.9. Assume pump 1 and pump 2 are both electrical pumps with electric supplies 1 and 2 respectively. A functional dependency exists between power supply 1 and pump 1 and between power supply 2 and pump 2. The way in which the functional dependency interacts with the standby dependency can be illustrated by the Markov model shown in figure 6.16, where ES1 and ES2 represent the failure of power supply 1 and 2 respectively:

![Markov model diagram](image)

Figure 6.16 Illustration of interaction between functional and standby dependency
In figure 6.16, the transition from state 1 to state 2 is caused by the failure of power supply 1. Although the standby dependency exists directly between pump 1 and pump 2, due to the functional dependency between power supply 1 and pump 1, the failure of power supply 1 renders pump 1 unusable, which consequently leads to the activation of pump 2. The difference between the switching dependency and the functional dependency is also illustrated in figure 6.16. The switching dependency will affect the accessibility or usability of a component only when it is residing in the standby state, whereas the functional dependency works on the dependent component whatever state it resides in. For example, only when the state of basic event 'P2' is '-1', can the failure of the controller render pump 2 inaccessible. In contrast, from state 2 to state 4, when the state of 'P2' is '0', i.e. pump 2 has been activated, the failure of the controller has no effect at all on the functioning of pump 2. As with the electric supply to pump 2, the failure of ES2 will immediately render pump 2 unusable whether pump 2 is in the standby state or working.

A general algorithm is presented in figure 6.17 for the generation of a Markov model which involves the standby dependency together with the switching and functional dependency.
establish_model('sdby1' or 'sdby2')
{
  establish the initial system state;
  while (the initial state ... the last system state)
  {
    if (the system is failed in this state)
      {repair all revealed failures at one time;
        if (the created state is new)
          {add to the list of existing states;
            record the state transition and decide the transition rate;
          }
    }
    else
      {while (the first event ... the last event)
       {change=0; /* variable used to indicate if any change has been made to the current state */
        if (the current event is a basic event & relevant in the current system state)
          {switch (the current state of this basic event)
           {case 0:
            set the new state as 1 or 3; /* depending on the failure model: dormant or revealed */
            if (the duty component fails or becomes unusable or inaccessible) /* functional dependency */
              if (the standby component is available, accessible and usable) /* functional dependency & switching dependency */
              set the state of the corresponding basic event as 0;
            }
            change=1; break;
          case 1:
            set the new state as '0';
            if (is the duty pump in the current system mode)
              set the new state as '-1';
            change=1; break;
          case 3:
            the algorithm presented in figure 6.5 applies;
            change=1; break;
          }
        }
      }
      if (change==1)
        {decide the module state of the created system state; /* with regard to initiator-enabler dependency, refer to the algorithm presented in figure 6.23 */
         if (the newly created system state has not existed)
           {add to the list of system states;
            record the transition and the transition rate;
           }
        }
    }
  }
}

Figure 6.17 The algorithm of generating Markov model involving standby dependency

6.2.3 Model Generation for the Sequential Dependency

The sequential dependency needs to be identified and addressed when generating a Markov model for systems whose failure depends on the specific order of occurrence of certain events. The development of a Markov model for modules which contain a sequential dependency requires the order of the occurrence of each relevant event to be identified.
Figure 5.9 is again used as the example. To illustrate the model development, the maintenance dependency included in this module is ignored and only the sequential dependency is considered. The corresponding Markov model is shown in figure 6.18.

With regard to the model generation for the sequential dependency, several points are worth noting. First, the labels used to identify the state of the basic event include the position in the order of failure of the basic event. When the basic event has not occurred, its order is set as ‘0’.

For example, in figure 6.18, in the initial state – state 1, the state of all basic events is indicated as ‘0 – 0’. While in state 2, basic event 18 has occurred and is the first failure event, therefore the state ‘1 – 1’ is used to represent its current state, failed and occurred first.
Secondly, when a state is generated from one of the existing states, it needs to be decided if this state is a new state or one already existing in the Markov model. To perform this process, a comparison needs to be made between the combination of the basic event states of the newly created state and that of those already in the Markov model. The occurrence order in addition to the component states need to taken into account in the comparison.

Thirdly, when repair is possible extra attention should be paid to the state transitions caused by the repair. Take for instance state 13 in figure 6.18, when basic event 19 undergoes repair and returns to state ‘0 – 0’, the basic event 18 which originally occurred immediately after basic event 19 will now become the first of the failed events. Therefore, the order label of basic event 18 should be changed to ‘1’. In the same way, basic event 20 becomes the second failure event and its order label should consequently be reduced to ‘2’. This leads to the state transition from state 13 to state 6. Finally, when deciding which states will cause the occurrence of the Priority AND gate event, the order of occurrence of the basic events must be tested.

The general algorithm for the development of the Markov model which involves the sequential dependency is given in figure 6.19.
6.2.4 Model Generation for the Sequence-Enforcing Dependency

The existence of the sequence-enforcing dependency, different from the sequential dependency, does not allow the relevant events to occur in orders other than the one specified. For example, assume that a module contains three basic events a, b and c between which the sequence-enforcing dependency exists. If the top event of the module occurs only when all of the three events occur in the order a, b then c, then figure 6.20 displays the Markov model of this module assuming the repair is not possible.

![Markov model for sequence-enforcing dependency (repair unavailable)](image)

**Figure 6.20** Markov model for sequence-enforcing dependency (repair unavailable)
It can be seen in figure 6.20 that when the system is residing in state 1, neither basic event b nor basic event c can occur since basic event a has not occurred. In the same way, in state 2, basic event c is not allowed to occur because basic event b has not occurred.

In most cases, the sequence-enforcing dependency doesn’t exist on its own, it is usually involved as a by-product of other types of dependency. For example, when the cold-standby dependency is a feature of a system, the standby component will never fail unless the duty component has failed, which is a form of sequence-enforcing dependency.

6.2.5 Model Generation for Secondary-Failure Dependency
To illustrate this particular dependency relationship, assume the secondary-failure dependency relationship exists in a module which contains three basic events a, b and c, and that the module failure is represented by the fault tree structure shown in figure 6.21. In this example, basic event a represents the primary failure, while gate G1 represents the secondary failure.

![Fault tree structure for the example module](image)

Figure 6.21 Fault tree structure for the example module

Assume also that basic event a represents a revealed failure, whereas basic events b and c are dormant failure events and are inspected at the same inspection interval \( t \). Figure 6.22 below gives the Markov model of this module based on the repair policy presented in section 4.2 and the assumption that no component failures will occur after the primary failure is revealed.
The key points to establishing the Markov model which can correctly embrace the secondary failure dependency are summarized as follows:

- When the secondary failure event occurs, it leads to the failure of the primary component. Under this circumstance, the primary failure event should in effect be regarded as having occurred, and its repair accounted for to rectify the situation. In figure 6.22, transitions from state 3 and state 4 to state 7 and from state 6 and state 9 to state 7 both reflect such a feature.

- The repair process involving secondary-failure dependency needs extra attention. When the failure of the primary component occurs and gets revealed, inspection will be carried out immediately on the secondary components, which reveal any dormant failures of secondary components if they exist. Transitions from state 3 to states 5 and 7 and from state 4 to states 7 and 8 both represent this process. Repair will be conducted on all revealed failures at the same time, and the system will be restored to the initial state when all the repair processes are finished.
In this case, the corresponding transition rate resulting from multiple component repairs is determined in the way as discussed in section 6.3.

The general algorithm of establishing the Markov model which includes the secondary failure dependency is given in figure 6.23.

```
establish_markov('scnf')
{
establist the initial system state;
while(the initial state ... the last system state)
{if(the primary failure is revealed) /* the primary failure event has current state of '1' */
 {if(the general secondary failure event has occurred in the current state.)
  (record the transition from the current state to the initial state;
  decide the transition rate accordingly; /* apply the algorithm presented in the second point */
 )
else
 {set the new state of the primary failure event as '0';
  set the new state of secondary components with dormant failures as '1'; /* the current state is '3' */
  if(the newly created state has not existed)
   add to the list of system states;
  record the transition rate; /* the conditional repair rate of the primary failure event */
 }
}
else
{while(the first event ... the last event)
 {if(the current state = 0)
   new state = 1 or 3; /* depend on the failure model */
   if(secondary failure occurs in the resulting system state)
    set the new state of the primary failure event as '1' or '3'; /* depend on the failure model */
   else
    if(the current state = 1)
     new state = 0;
    else
     the algorithm presented in figure 6.5 applies;
   if(the newly created state has not existed)
    add to the list of system states;
   record the transition rate;
 }
}
}
```

Figure 6.23 The algorithm for the generation of the Markov model involving secondary-failure dependency
6.2.6 Model Generation for Initiator-Enabler Dependency

In section 6.2.2 the initiator-enabler dependency was discussed in relation to the standby dependency. In this section, it will be examined in more detail using an example system to illustrate the modeling principle. Consider the tank system shown in figure 4.6. To make the illustration simpler, whilst representing the relevant features, the failure of the sensing system which consists of three sensors will be represented by one basic event PSF. Accordingly the fault tree which represents causes of the over-pressurization of the tank is shown in figure 6.24.

![Fault tree structure for the example system](image)

Figure 6.24 Fault tree structure for the example system (referring to figure 4.6)

In this system, the initiator-enabler dependency exists between the initiating event PS and the general enabling event, gate G1. PRS, CTR and PSF are three enabling events featuring different reliability characteristics. Based on the fault tree structure in figure 6.24 and the dependency relationship between the basic events, the Markov model established for the top event is displayed in table 6.9 (model states) and table 6.10 (state transitions) since the model is too large to represent clearly in diagrammatic form. Assume repair policy 1 (see section 4.2) is adopted and basic events PRS, CTR and PSF are dormant failures with a common inspection interval \( \theta \).
<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>$\lambda_{PS}$</td>
<td>28</td>
<td>24</td>
<td>0 - 0</td>
<td>28</td>
<td>3</td>
<td>$v_{CTR}$</td>
<td>41</td>
<td>13</td>
<td>$v_{PRS}$</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>$\lambda_{PRS}$</td>
<td>14</td>
<td>17</td>
<td>$\lambda_{PS}$</td>
<td>28</td>
<td>48</td>
<td>$\lambda_{PSF}$</td>
<td>41</td>
<td>45</td>
<td>0 - 0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>$\lambda_{CTR}$</td>
<td>14</td>
<td>31</td>
<td>$\lambda_{PRS}$</td>
<td>29</td>
<td>33</td>
<td>$\lambda_{PS}$</td>
<td>42</td>
<td>5</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>$\lambda_{PSF}$</td>
<td>14</td>
<td>32</td>
<td>$\lambda_{CTR}$</td>
<td>29</td>
<td>48</td>
<td>$\lambda_{PRS}$</td>
<td>42</td>
<td>44</td>
<td>0 - 0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$v_{PS}$</td>
<td>14</td>
<td>1</td>
<td>$v_{PSF}$</td>
<td>29</td>
<td>5</td>
<td>$v_{CTR}$</td>
<td>43</td>
<td>44</td>
<td>$\lambda_{PS}$</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>$\lambda_{CTR}$</td>
<td>15</td>
<td>12</td>
<td>$v_{PS}$</td>
<td>29</td>
<td>12</td>
<td>0 - 0</td>
<td>43</td>
<td>29</td>
<td>$v_{PRS}$</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>$\lambda_{PSF}$</td>
<td>15</td>
<td>2</td>
<td>$v_{CTR}$</td>
<td>30</td>
<td>34</td>
<td>$\lambda_{PS}$</td>
<td>43</td>
<td>22</td>
<td>$v_{CTR}$</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>$\lambda_{PS}$</td>
<td>15</td>
<td>33</td>
<td>$\lambda_{PSF}$</td>
<td>30</td>
<td>49</td>
<td>$\lambda_{PRS}$</td>
<td>43</td>
<td>45</td>
<td>0 - 0</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>0 - 0</td>
<td>16</td>
<td>30</td>
<td>$v_{PS}$</td>
<td>30</td>
<td>14</td>
<td>$v_{CTR}$</td>
<td>44</td>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>$\lambda_{CTR}$</td>
<td>16</td>
<td>34</td>
<td>0 - 0</td>
<td>30</td>
<td>12</td>
<td>$v_{PSF}$</td>
<td>45</td>
<td>44</td>
<td>$\lambda_{PS}$</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>$\lambda_{PSF}$</td>
<td>17</td>
<td>14</td>
<td>$v_{PS}$</td>
<td>31</td>
<td>26</td>
<td>$\lambda_{PS}$</td>
<td>45</td>
<td>30</td>
<td>$v_{PRS}$</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>$\lambda_{PS}$</td>
<td>17</td>
<td>35</td>
<td>$\lambda_{CTR}$</td>
<td>31</td>
<td>27</td>
<td>0 - 0</td>
<td>45</td>
<td>27</td>
<td>$v_{CTR}$</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>$\lambda_{PRS}$</td>
<td>17</td>
<td>2</td>
<td>$v_{PSF}$</td>
<td>31</td>
<td>50</td>
<td>$\lambda_{CTR}$</td>
<td>45</td>
<td>24</td>
<td>$v_{PSF}$</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>0 - 0</td>
<td>18</td>
<td>9</td>
<td>$v_{PS}$</td>
<td>31</td>
<td>2</td>
<td>$v_{PSF}$</td>
<td>46</td>
<td>4</td>
<td>*</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$\lambda_{PSF}$</td>
<td>18</td>
<td>2</td>
<td>$v_{PRS}$</td>
<td>32</td>
<td>35</td>
<td>$\lambda_{PS}$</td>
<td>46</td>
<td>44</td>
<td>0 - 0</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>$\lambda_{PS}$</td>
<td>18</td>
<td>36</td>
<td>$\lambda_{CTR}$</td>
<td>32</td>
<td>50</td>
<td>$\lambda_{PRS}$</td>
<td>47</td>
<td>44</td>
<td>$\lambda_{PS}$</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>$\lambda_{PRS}$</td>
<td>18</td>
<td>37</td>
<td>$\lambda_{PSF}$</td>
<td>32</td>
<td>30</td>
<td>0 - 0</td>
<td>47</td>
<td>32</td>
<td>$v_{PRS}$</td>
</tr>
</tbody>
</table>

* Transitions result from multiple component repairs, thus the transition rates are determined according to section 6.3
In this Markov model, it can be noted that considering only the combination of the component states, state 38 and state 23 represent the same system state. However, this is without considering the event sequence. In state 38, the pump surge occurs prior to the failure of the protection features and has consequently been shut down, therefore preventing the top event. In state 23, due to the earlier failure of the safety measures, the pump surge will cause over-pressurization in the tank. It is the same situation between states 40 and 26, states 52 and 42, states 54 and 46, and states 53 and 44. The question is then how to distinguish between these two different situations which yield the same component states. Here the system/module state has to be taken into account as the crucial criterion for determining if the two system states are same. It can be
noticed in table 6.9 that ‘-1’ and ‘-2’ are used, along with ‘0’ and ‘1’, to represent the system state. In terms of the top event, ‘-1’ and ‘-2’ both indicate that the top event has not occurred. However, they both have more implications than ‘0’.

Module state ‘-1’ and module state ‘-2’ are introduced to indicate the different order of occurrence between the initiating event and enabling event. State ‘-1’ refers to the situation where the initiating event occurs prior to the general enabling event. Take for instance the transition from state 3 to state 8 in table 6.10, because in state 3 the enabling event ‘G1’ has not occurred, the occurrence of the initiating event ‘PS’, which leads to this transition, has taken place prior to the enabling event.

Alternatively, the module state ‘-2’ stands for the situation where the enabling event occurs prior to the initiating event. For example, in state 10 in table 6.9, the occurrence of the basic event ‘PRS’ and ‘CTR’ result in the occurrence of the enabling event ‘G1’. Since the initiating event has not occurred in this state, the module state of state 10 is set as ‘-2’. When the initiating event ‘PS’ occurs in state 10, the transition to state 23 takes place. Since this transition meets the requirement on the occurrence order of the initiating and enabling events, the top event has occurred in state 23. The principle of determining the module state in a certain system state can be expressed in the flow chart in figure 6.25 below:
It can also be noticed that the dormant failures in state 9 are revealed after the transition to state 23 as the occurrence of the top event makes the maintenance team aware of the occurrence of the enabling events. In addition, since the repair policy 1 is implemented in the system, the correction of the initiating event will be accompanied by the discovery of the dormant failures among the enabling events. For example, in the transition from state 16 to state 30, after the pump surge has been rectified, inspection is carried out on the components related to the enabling events, and consequently reveals the dormant failure of the controller and pressure sensor. Otherwise, if repair policy 2, with which only initiating events are rectified when it occurs prior to the enabling event, is to be practiced, the transition should take place from state 16 to state 13. The general algorithm of developing the Markov model for the system which contains the initiator-enabler dependency is established in figure 6.26 as follows:
establish_markov('ie')
{
    establish the initial system state;
    while (the initial state ... the last system state)
    {
        if (the top event occurs in the current system state) /* the module state is '1' */
            repair all revealed failures; /* set the new state of relevant basic events as '0' */
            record the transition and the transition rate;
        while (the first event ... the last event)
        {
            if (the current state is '0')
                set the new state as '1' or '3'; /* depending on the failure model */
            else
                the algorithm presented in figure 6.5 applies;
            if (there is any change to the current state)
                if (the newly created state has not existed)
                    add to the list of system states with module state = 1;
                    record the transition and the transition rate;
        }
    }
    else
    {
        while (the first event ... the last event)
        {
            if (the current state = 0)
                set the new state as '1' or '3'; /* depending on the failure model */
            else
                if (the current state = 1)
                    set the new state as '0';
                else
                    if (the event is the initiating event and the repair policy is '1')
                        reveal all dormant failures of enabling events; /* set the new state of relevant events as '1' */
                    else
                        the algorithm presented in figure 6.5 applies;
                    decide the module state of the newly created state; /* apply the algorithm in figure 6.22 */
                if (the newly created state has not existed)
                    add to the list of system states;
                    record the transition and the transition rate;
        }
    }
}

Figure 6.26 Algorithm for the generation of Markov model involving initiator-enabler dependency

6.2.7 Model Generation for Revealing Dependency

In this section, an example system is used to illustrate the process for developing the Markov model for situations which involve the revealing dependency. This example system is composed of components a, b and c, and the system failure logic can be expressed as a.(b+c). Since the
system failure is self-revealing whilst the failure of each of the three components are unrevealed or dormant, the revealing dependency exists under the top event between the three basic events. The corresponding Markov model for this system is shown in figure 6.27 below:

![Figure 6.27 The Markov model involving the revealing dependency](image)

In figure 6.27, the revealing dependency existing between the basic events a, b and c is reflected through the transitions from state 2 to state 6 and 7, from state 3 to state 6, from state 4 to state 7 and etc. In these transitions, the occurrence of the top event reveals the dormant component failures. For this dependency relationship, the specific repair policy needs to be clarified in order to construct the Markov model. In the example, the repair policy is that when the top event occurs, all revealed failures will be repaired at the same time and inspection will be carried out on other components to identify any dormant failures. The system is not made operational until all revealed failures have been cleared. This repair policy explains the transition from state 11 to state 10 and state 12 to state 8.
The algorithm of developing the Markov model for the revealing dependency relationship is presented in figure 6.28 below:

```c
Establish_markov('revl')
{
    establish the initial state;
    while( the initial state ... the last system state )
    {
        if( the top event occurs in the current state ) /* the module state of the current state is '1' */
            repair all revealed failures; /* set the new state of relevant basic events as '0' */
            reveal all dormant failures; /* set the new state of relevant basic events as '1' */
            record the transition and the transition rate;
        while( the first event ... the last event )
            if( the current state is '0')
                set the new state as '1' or '3'; /* depending on the failure model */
            if( the current state is '3')
                the algorithm presented in figure 6.5 applies;
            if( there is any change to the current state )
                if( the newly created state has not existed )
                    add to the list of system states with module state = 1;
                record the transition and the transition rate;
        }
    }
    else
    {
        while( the first event ... the last event )
            if( the current state = 0)
                set the new state as '1' or '3'; /* depending on the failure model */
            else
                if( the current state = 1)
                    set the new state as '0';
                else
                    the algorithm presented in figure 6.5 applies;
                decide the module state of the newly created state;
            if( the top event occurs in the newly created state )
                reveal all dormant failures in the newly created state; /* set the state of relevant events as '1' */
            if( the newly created state has not existed )
                add to the list of system states;
            record the transition and the transition rate;
        }
    }
}
```

Figure 6.28 Algorithm for the development of Markov model involving the revealing dependency
6.2.8 Model Generation for Test Dependency

The test dependency exists in the systems where there exists the discrepancy between the common inspection interval for a group of components and the inspection interval for individual components. Test dependency does not give rise to the statistical dependency as it neither results in the interaction between component failures nor changes the way that component failures contribute to the system failure. It is, however, necessary to identify test dependency existing in the system to obtain accurate system reliability measures as test dependency has an influence on the component failure probability.

Since test dependency does not involve the statistical dependency, its sole existence will not require the use of the Markov method. Instead, it can be tackled using a straightforward numerical solution presented through equation 7.16. However, when components involved in test dependency are also included in a type of statistical dependency relationship, the Markov model generation must account for the existence of the test dependency.

An example is used here to illustrate how the test dependency will affect the generation of the Markov model. Consider a system whose failure is caused by the failure of both components labeled a and b. Assume the test interval of the components a and b are respectively 2 and 3 months, and the test interval for the system is 5 months. Figure 6.29 below gives the Markov model of this system.

![Figure 6.29 The Markov model for the example system including revealing dependency](image-url)
The time conditions for the state transitions caused by the discovery of the dormant failures are listed in table 6.11.

<table>
<thead>
<tr>
<th>State transition</th>
<th>Time condition (months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 2 → State 4</td>
<td>T = 2k</td>
</tr>
<tr>
<td>State 3 → State 6</td>
<td>T = 3k</td>
</tr>
<tr>
<td>State 5 → State 7</td>
<td>T = 2k</td>
</tr>
<tr>
<td>State 5 → State 8</td>
<td>T = 6k or T = 5k</td>
</tr>
<tr>
<td>State 5 → State 9</td>
<td>T = 3k</td>
</tr>
<tr>
<td>State 7 → State 8</td>
<td>T = 3k or T = 5k</td>
</tr>
<tr>
<td>State 9 → State 8</td>
<td>T = 2k or T = 5k</td>
</tr>
</tbody>
</table>

Table 6.11 Time conditions for state transitions in figure 6.26

Time conditions listed in table 6.11 are established by following the algorithm presented in figure 6.5 in section 6.2.1 with one exception. It can be seen that for the time condition T = 5k, transitions from state 5, 7 and 9 to state 8 can take place. The reason for this lies in the existence of the test dependency. Since in state 5, 7 and 9 the top event occurs, when the elapsed mission time equals 5k months, the test of the system will reveal the dormant failures in these states. By adding the time condition T=5k to the original list of time conditions of the state transitions for which the source state means the occurrence of the top event, the Markov model will be produced for the test dependency with a complete set of state transitions and corresponding transition rate.

The general algorithm of developing the Markov model for test dependency is displayed in figure 6.30 as follows:

```c
establish_markov('test')
{
    establish the initial system state;
    while(the initial state ... the last system state)
    {
        while(the first event ... the last event)
        {
            if(the current state = 0)
                set the new state = '1' or '3'; /* depending on the failure model */
            else
                if(the current state = 1)
                    set the new state = '0';
        }
        else
            the algorithm applies presented in figure 6.5;
            if(the top event occurs in the current system state)
                add the time condition T = k0_system;
            if(the newly created system state has not existed)
                add to the list of system states;
                record the state transition and the transition rate;
        }
    }
}
```

Figure 6.30 Algorithm for the generation of the Markov model involving test dependency

140
6.3 Transition Rate of Multiple-repair Transitions

In the reliability assessment using the Markov method, it is assumed that during a very short period of time, only one component failure or repair would occur. This means that the rate of the transition resulting from the specific component failure or repair can be directly determined by the corresponding component conditional failure or repair rate. However, in the evaluation of systems which contain some types of dependency, situations may arise where multiple component failures have to be rectified at the same time to bring the system back to its normal function mode. For example, in the Markov model in figure 6.15, when the system is residing in state 5 or state 6, it is required to repair both pump 1 and pump 2 to bring the system back to state 1. This is also reflected in the transitions from state 5 and state 7 to state 1 in the Markov model in figure 6.22, some state transitions as shown in table 6.10 with respect to the initiator-enabler dependency, such as state 42 to state 5, state 44 to state 1, state 46 to state 4 and etc.. In this case, an investigation needs to be carried out to find an appropriate way to determine the transition rate for multiple repairs.

The Markov method used in this thesis is a homogeneous CTMC (see section 3.1.3.1) based on the exponential distribution of component failure and repair times. It is impossible to define a constant repair rate for multiple component repairs which is made up of repair rates of the individual components and can be directly applied in the Markov analysis, as the repair times of multiple component repairs will not be characterized by an exponential distribution. Therefore, an appropriate way to solve the multiple-repair transition rate is through approximation. Two approximation techniques are discussed as follows:

6.3.1 Approximation through Comparison

Consider the situation where \( n \) component failures need to be rectified at the same time with the conditional repair rate of each individual component being \( \nu_1, \nu_2, ..., \nu_n \). Then the mean time to repair for each individual component \( \tau_j \) can be obtained as the inverse of the corresponding conditional repair rate, i.e. \( \frac{1}{\nu_j} \). Considering from the practical perspective, the mean time to repair these \( n \) components would be determined by the component repair which takes the longest time. That is:
6.1

\[ r_n = \max(\tau_1, \tau_2, \ldots, \tau_n) \]

where \( r_n \) represents the mean time to repair the \( n \) components at the same time.

The transition rate for the \( n \) component repairs at the same time can then be determined as \( \frac{1}{r_n} \), i.e. \( \frac{1}{\max(\tau_1, \tau_2, \ldots, \tau_n)} \). It can be concluded that the greater the difference between \( \tau_j \) \( (j=1, 2, \ldots, n) \), the smaller the deviation would be produced through this approximation method.

6.3.2 Approximation from Real System Data

Another approximation method is to produce the approximated value based on real system data. In the process of developing the Markov model, when the system is residing in a state in which multiple component repairs need to be carried out at the same time, the process could inform the analyst on which specific component failures need to be rectified at the same time and ask the analyst to assign a particular rate for these multiple repairs. The analyst could derive the data from previous records if the same situation has arisen before. Alternatively, the analyst could seek advice from repair engineers who could give an estimation based on experience. The validity of this approximation method is mainly dependent on the quality of the system maintenance records and the expertise of repair engineers.

6.4 Model Generation for Multiple Dependency Relationships

In the analysis of engineering systems, several types of dependency relationship may be involved in the same model. With regard to multiple dependency relationships, there are two cases. One is that a certain type of dependency will always involve another type of dependency relationship. For example, in most systems which involve initiator-enabler dependency, enabling events usually represent dormant component failures whose occurrence, followed by the initiating event, will lead to the occurrence of a self-revealing event, which consequently reveals the dormant failures. Therefore, the revealing dependency exists between the self-revealing event and enabling events. Similarly, in systems where there exists the secondary-failure dependency and the primary failure event features a revealed failure mode, the revealing dependency can be identified between the primary failure event and the dormant secondary component failures.
which contribute to the occurrence of the primary failure. That is, the revealing dependency is the by-product of initiator-enabler and secondary-failure dependency relationships and always exists when the latter two types of dependency occur in a system. In this case where the existence of one type of dependency will automatically give rise to another, only the primary type of dependency relationship needs to be identified and specified. No extra attention is required with respect to the secondary dependency relationship since it has been taken into account in the algorithm for the model generation for the primary dependency relationship.

The other situation is where several dependency relationships exist in the system at the same time. For example, the maintenance dependency and test dependency can occur together with any other type of dependency. Also the sequential-failure dependency can exist within the initiator-enabler dependency. In this case with the top-level event of the sequential failures being either the initiating event or the general enabling event. Other examples include the Sequential failure dependency existing along with the secondary-failure dependency when sequential failure events contribute to the causes of the secondary failure. Also the initiator-enabler dependency and standby dependency can occur together when the standby is a warm-spare. To ignore any type of dependency will lead to the development of an incorrect model. In this case, the algorithms for the model generation for each type of dependency involved need to be integrated. For each combination of different types of dependency relationships, a unique algorithm is required for the model generation. It is impossible to go through every possible combination of different dependency relationships. Several typical combinations of dependency types are investigated to illustrate how the algorithms for each individual type of dependency can be integrated to establish the correct Markov model.

6.4.1 Maintenance Dependency and Initiator-Enabler Dependency

When the maintenance dependency and the initiator-enabler dependency occur together in a system, the algorithm for the model generation for maintenance dependency will be integrated into the algorithm for the initiator-enabler dependency. Apart from the introduction of state code '2' to indicate that the component is queuing for repair, the existence of the maintenance dependency will also influence the transition rate from states where multiple failures need to be rectified. For example, assume that the maintenance dependency exists between basic events
PRS, CTR and PSF in the fault tree shown in figure 6.22. In the Markov model displayed in table 6.9 and 6.10, it can be seen that the system is failed in state 26. Without the maintenance dependency between the pressure relief valve and the pressure sensing system, the system will be restored to state 1 from state 26 with an expected mean time to repair which is the maximum of the repair times for the three failed components if the first approximation method in section 6.3 is applied. However, with the maintenance dependency between basic events PRS, CTR and PSF, the pressure relief valve and the sensing system have to be repaired one after the other, while at the same time the repair is conducted on the pump. Therefore, in this case, if the first approximation method still applies, the mean time to restore the system back to state 1 is the longer of the mean time to repair of the pump and the total of the mean times to repair of the pressure relief valve and the sensing system. That is, with maintenance dependency between PRS and PSF, the rate for the transition from state 26 to state 1 should be expressed as

$$\frac{1}{\max(\tau_{PS}, (\tau_{PRS} + \tau_{PSF}))}$$.

The same principle applies when multiple repairs are carried out where the maintenance dependency is involved if the comparison approximation is adopted. On the other hand, if the second approximation method (see section 6.3.2) applies, the maintenance dependency should also be accounted for and reflected in the estimation given by repair engineers. Figure 6.31 illustrates the general algorithm for the model generation where the maintenance dependency is involved in the initiator-enabler dependency relationship.
establish_markov('ie' & 'mtnce')
{
    establish the initial system state;
    while(the initial state ... the last system state)
    {
        if(the top event occurs in the current system state) /* the module state is '1' */
        {
            repair all revealed failures; /* set the new state of relevant basic events as '0' */
            record the transition and the transition rate; /* rate decided by taking into account mtnce dep. */
            while(the first event ... the last event)
            {
                switch(the current sate)
                {
                    case '0':
                        if(dormant failure model): set new state = 3;
                        else: set new state = 1;
                        if(mtnce dep involved & no repair available): set new state = 2;
                        case '3': the algorithm presented in figure 6.5 applies;
                }
                if(there is any change to the current state)
                {
                    if(the newly created state has not existed)
                    {
                        add to the list of system states with module state = 1;
                        record the transition and the transition rate;
                    }
                }
            } else
            {
                while(the first event ... the last event)
                {
                    switch(current state)
                    {
                        case '0':
                            if(dormant failure model): set new state = 3;
                            else: set new state = 1;
                            if(mtnce dep involved & no repair available): set new state = 2;
                            case '1': set the new state as '0';
                            if(the event is the initiating event and the repair policy is 'I')
                            {
                                reveal all dormant failures of enabling events; /* set the new state of relevant events as '1' */
                                case '3': the algorithm presented in figure 6.5 applies;
                            }
                            decide the module state of the newly created state; /* apply the algorithm in figure 6.22 */
                            if(the newly created state has not existed)
                            {
                                add to the list of system states;
                                record the transition and the transition rate;
                            }
                        }
                    }
                }
            }
        }
    }
}

Figure 6.31 The algorithm for model generation for i-e dependency mixed with mtnce dependency
6.4.2 Sequential Dependency and Initiator-Enabler Dependency

The sequential dependency can be relevant in a system which also features the initiator-enabler dependency relationship. The dependency on a sequence of events can contribute to the cause of either the initiating event or the enabling event. When events combined in the form of the sequential dependency contribute to causes of the initiating event, they can usually be dealt with in an individual module, thus separating the sequential dependency relationship from the initiator-enabler dependency and allowing both to be modeled separately. In this section, the main focus is placed on the situation where sequential failure events feature a dormant failure and are included as part of the general enabling event. In this case, the sequential failure events can no longer be dealt with in an independent module. Instead, they have to be investigated in the same model with the initiating event and other enabling events. That is, the sequential failure dependency has to be accounted for in the same model as the initiating-enabling dependency. During the process of developing the Markov model which take into account both of the two dependency relationships, the order of occurrence of sequential failure events needs to be taken into consideration when deciding whether the enabling event has occurred. The algorithm for the model generation where sequential dependency and initiator-enabler dependency exist along with each other is displayed in figure 6.32.
establish_markov('ie' & 'sq')
{
    establish the initial system state;
    while(the initial state ... the last system state)
    {if(the top event occurs in the current system state) /* the module state is '1'*/
        {repair all revealed failures; /* set the new state of relevant basic events as '0' */
            record the transition and the transition rate;
        while(the first event ... the last event)
        {switch(the current state)
            {case '0':
                set new state = 1 or 3; /* depending on failure model */
                if(involved in sequential dep.): decide the order of occurrence.
                case '3': the algorithm presented in figure 6.5 applies;
            }
            if(there is any change to the current state)
                if(the newly created state has not existed): add to the list of system states with module state = 1;
                record the transition and the transition rate;
            }
        }
    }
    else
    {while(the first event ... the last event)
        {switch(current state)
            {case '0':
                set new state = 1 or 3; /* depending on failure model */
                if(involved in sequential dep.): decide the order of occurrence.
                case '1': set the new state as '0';
                if(involved in sequential dep.): decide the order of occurrence.
                case '3': the algorithm presented in figure 6.5 applies;
            }
            decide the module state of the newly created state; /* apply the algorithm in figure 6.22 & take into account of sequential dep. */
            if(the newly created state has not existed)
                add to the list of system states;
            record the transition and the transition rate;
        }
        }
    }
}
Figure 6.32 The algorithm for model generation for i-e dependency mixed with sq dependency
6.4.3 Initiator-Enabler Dependency and Secondary-Failure Dependency

The initiator-enabler dependency can occur in a system together with the secondary-failure dependency when component failures featuring in the initiator-enabler dependency contribute to the causes of the secondary failure. In this case, all relevant component failures will have to be investigated in the same model in order to obtain the accurate system reliability parameters. The fault tree structure shown in figure 4.3 provides an example where initiator-enabler dependency is embedded in the secondary failure dependency relationship. The existence of the initiator-enabler dependency prescribes that the order of occurrence of the initiating event and enabling event should be taken into account when deciding if a secondary failure has occurred. Figure 6.33 displays the algorithm for the model generation where the secondary failure dependency has to be solved together with the initiator-enabler dependency.
Figure 6.33 The algorithm for model generation where i-e dependency exists along with the secondary-failure dependency
Chapter 7. Quantitative Analysis with the Combined Application of FTA and Markov Method

For continuously-running systems a reliability assessment will predict the system failure probability and the system unconditional failure intensity at certain points of time over the lifetime of the system. Considering the transient failure characteristics of the components of which the system is composed, the system reliability measures are also a function of the elapsed mission time. System reliability predictions are obtained using numerical methods which progress the solution in small increments of time, \( dt \). The length of \( dt \) is specified by the analyst considering the system life span and the components' failure and repair characteristics. If \( dt \) is too small, it may require too much computational resources for complete system life assessment and reduce the analysis efficiency. However, the accuracy will increase if small values of \( dt \) are selected. The analyst has to consider the value of \( dt \) carefully to reach a compromise between accuracy and computational efficiency.

For systems with dependencies, the overall system assessment will be based on the combined application of the fault tree and Markov methods. This requires the solution of the modularized fault tree structures which are independent from each other. The way to achieve modularization has been described in chapter 5 as the pre-processing stage of the solution. Following the modularization the conventional fault tree analysis approach and Markov method can then be applied respectively to modules involving no dependency relationship and those containing inter-dependent events.

The following sections will look into how the quantitative analysis is carried out in the modules using different techniques and how the results of each module can be fed back into the overall fault tree structure to obtain the final top event probability and failure intensity. During the analysis process, the criticality function and the criticality measure of importance for each basic event will also be calculated.

7.1 Quantitative Analysis of Modules Involving a Dependency Relationship

A correct Markov model is required in order to conduct the analysis of modules which feature
some form of dependency. Chapter 6 has described how to establish the corresponding Markov model for modules accounting for the specific dependency relationships they contain. The Markov model is then quantified to obtain relevant module reliability parameters.

7.1.1 Module Failure Probability \( Q_M(t) \)

The Markov model consists of numerous system states with interaction between each other through the state transitions. The Markov model provides the exhaustive list of all possible states in which the system may reside. During the period of time that the system is working, the system can experience different states at different times. Analysis gives the probability of the system being present in any specific state at any point of time. At any point of time, the total of the probabilities of the system residing in all of the system states is equal to 1. System states can be split into two categories: those states in which the system functions normally; and states in which the specific system failure mode represented by the top event occurs. Therefore, the system failure probability or the top event probability can be obtained by summing up the probabilities of all the system states belonging to the second category; Alternatively, the probability of the system functioning is equal to the total of the probabilities of system states which are of the first category.

Consider a general Markov model consisting of \( n \) system states, of which states \( 1 \) - \( k \) are working system states and states \( (k+1) \) - \( n \) are failed system states. \( Q_i(t) \) is the probability of system residing in state \( i \) at time \( t \). \( a_{ij} \) is used to denote the rate of transition from state \( i \) to state \( j \). Therefore, \( P(\text{system in state } j \ \text{at } (t+dt) \mid \text{system in state } i \ \text{at time } t) = a_{ij} \cdot dt \). The module failure probability, \( Q_M(t) \), can then be expressed by the following equation:

\[
Q_M(t) = \sum_{i=k+1}^{n} Q_i(t)
\]

Consider the probability that the module is residing in state \( i \) at time \( t+dt \). Two situations should be taken into account: one is the module was residing in other state, \( j \), at time \( t \) and during the time interval \( dt \) the relevant state transitions occur which lead the module to state \( i \); the other situation is where the module was residing in state \( i \) at time \( t \) and during the interval \( dt \) it remains in state \( i \). Therefore:
\[ Q_i(t+dt) = \sum_j P(\text{module in state } j \text{ at } t \text{ and makes a transition to state } i \text{ in the next } dt) + P(\text{system in state } i \text{ at } t \text{ and remains in state } i \text{ after } dt) \]
\[ = \sum_{j \neq i} Q_j(t) a_{ji} dt + Q_i(t) [1 - \sum_{l \neq j} a_{lj} dt] \]

Rearranging gives:
\[ \frac{Q_i(t+dt) - Q_i(t)}{dt} = \sum_{j \neq i} Q_j(t) a_{ji} - Q_i(t) \sum_{l \neq j} a_{lj} \]

As \( dt \to 0 \):
\[ \frac{Q_i(t+dt) - Q_i(t)}{dt} \to \frac{dQ_i(t)}{dt} \]

Define \( a_{ii} \) as \(-\sum_{j=1}^{n} a_{ij} \), equation 7.3 can be expressed as:
\[ \frac{dQ_i(t)}{dt} = \sum_{j=1}^{n} Q_j(t) a_{ji} \quad \text{for } i = 1, 2, \ldots, n \]

In matrix form, equation 7.4 can be expressed as:
\[ [\frac{dQ_1}{dt}, \frac{dQ_2}{dt}, \ldots, \frac{dQ_n}{dt}] = [Q_1(t), Q_2(t), \ldots, Q_n(t)][A] \]

The matrix \([A]\) in equation 7.5 is called the state transition matrix with elements:
\[ a_{ij} = \text{transition rate from state } i \text{ to state } j \]
\[ a_{ii} = -\sum_{j=1}^{n} a_{ij} \]

The rate matrix \([A]\) can be formulated directly from the Markov model using the following rules:
- the dimensions of the matrix are equal to the number of states in the model;
- all rows sum up to zero;
- an off-diagonal element in row \( i \) column \( j \) represents the transition rate\(^1\) from state \( i \) to state \( j \);
- a diagonal element in row \( i \) column \( i \) is the transition rate out of state \( i \) (always negative).

\(^1\) When the transition occurs due to multiple component repairs, the transition rate is determined through approximation as discussed in section 6.3.
There exist two types of solutions to a Markov model. One is the steady-state solution, and the other is the transient solution [3].

• The Steady-State Solution

When the system has arrived in its 'steady-state', the probabilities of being in any of the states will not be change with time. These steady-state probabilities do not depend on the initial state from which the system starts. At steady-state:

\[ \text{As } t \to \infty, \quad \frac{dQ_i(t)}{dt} \to 0 \]

Therefore, according to equation 7.4:

\[ \sum_{j=1}^{n} Q_j(t) a_{ij} = 0 \text{ for } i = 1, 2, \ldots, n \]

These equations are not independent and so an additional equation is needed to formulate a solution. The extra equation is:

\[ \sum_{i=1}^{n} Q_i(t) = 1 \]

The steady-state solution of being in each state, \( i \), to the Markov model can then be obtained by:

\[
Q_i(\infty) = \begin{pmatrix}
0 & a_{12} & \cdots & a_{1,n-1} \\
& \ddots & \ddots & \ddots \\
& & \ddots & 0 \\
& & & \ddots & a_{n,n-1} \\
& & & & 0
\end{pmatrix}
\]

From equation 7.1, the module failure probability can be given by:
• The Transient Solution

The transient solution is aimed to provide the probability of system residing in any of the states at any specific point of time \( t \) by progressing from the initial system state in very small time steps, \( dt \).

Replacing \( -\sum_{i=1}^{n} a_{ij} \) with \( a_{ii} \) in equation 7.2 gives:

\[
Q_i(t+dt) = \sum_{j=1}^{n} Q_j(t) a_{ji} \cdot dt + Q_i(t) \cdot [1+a_{ii} \cdot dt]
\]

(i = 1, 2, ..., \( n \))

Equation 7.6 gives a scheme where the state probabilities can be evaluated at discrete time points given the step length \( dt \) and the initial state probability \( Q_i(0) \), \( i = 1, 2, ..., n \).

The transition probability matrix \([P]\) is then established so that the equation 7.6 can be applied to all system states in the matrix form. The probability matrix \([P]\) is a square \( n \times n \) matrix with elements:

\( p_{ij} = a_{ij} \cdot dt \), for \( i \neq j \); and \( p_{ii} = 1 + a_{ii} \cdot dt \)

Therefore, starting from time \( t = 0 \), the state probabilities at discrete time points with constant interval \( dt \) can be obtained as:

\[
[Q_1(dt), Q_2(dt), ..., Q_n(dt)] = [Q_1(0), Q_2(0), ..., Q_n(0)] \cdot [P]
\]

\[
[Q_1(2dt), Q_2(2dt), ..., Q_n(2dt)] = [Q_1(dt), Q_2(dt), ..., Q_n(dt)] \cdot [P]
\]
Accordingly, the module failure probability at any specific point of time \( k \) can then be obtained through equation 7.1.

When dormant failure(s) are involved in the Markov model, extra attention should be paid to the state transitions forced by the inspection(s). Assume in a Markov model, the transition from state \( i \) to state \( j \) occurs due to the revealing of a dormant failure by the inspection at the regular interval \( \theta \). Consider a specific point of time \( k \): 

If \( k = 0, 2\theta, \ldots, m\theta \)

\[
Q'(k) = Q(k) + Q'(k)
\]

where \( Q'(k) \) represents the updated state probability at the point in time immediately following the inspection.

7.1.2 Module Unconditional Failure Intensity \( w_M(t) \)

The definition of the module unconditional failure intensity, \( w_M(t) \), is the rate of failure of the module at time \( t \). The word 'unconditional' here is aimed to emphasize the difference from the module conditional failure rate \( \lambda_M \), which refers to the rate that the module fails at time \( t \) given the module is working at time \( t \). For the module to fail at time \( t \), two conditions must be both fulfilled. The first is that the module is residing in a working state at time \( t \); and the second is that the module then fails at that time \( t \). Using the general Markov model in section 7.1.1, the module unconditional failure intensity can be expressed as:

\[
w_M(t) = \sum_{i=1}^{k} Q_i(t) \cdot \left( \sum_{j=k+1}^{n} a_{ij}(t) \right)
\]

where state \( i \) is the working system state and \( a_{ij}(t) \) is the rate of the transition from state \( i \) to state \( j \) in which the module is failed.

7.1.3 Module Unconditional Repair Intensity \( v_M(t) \)

Following a similar argument to the unconditional failure intensity, the module unconditional repair intensity, \( v_M(t) \), is the rate that the module gets restored at time \( t \). It is given in the
following equation:

\[ v_M(t) = \sum_{i=k+1}^{n} \sum_{j=1}^{k} Q_j(t) \cdot \left( \sum_{j=1}^{k} a_{ij}(t) \right) \]  \hspace{1cm} 7.8

where state \( i \) is the failed system state and \( a_{ij}(t) \) is the rate of the transition from state \( i \) to state \( j \) in which the module is working.

### 7.1.4 Module Conditional Failure Rate \( \lambda_M(t) \)

The module conditional failure rate, \( \lambda_M(t) \), is the rate of module failure given the module is working at time \( t \). The module conditional failure rate is therefore given by:

\[ \lambda_M(t) = \frac{\text{the rate that the module fails at time } t}{\text{the probability that the module is working at time } t} \]

\[ = \frac{w_M(t)}{1 - Q_M(t)} \]  \hspace{1cm} 7.9

### 7.1.5 Module Conditional Repair Rate \( v_M(t) \)

Similar to the module conditional failure rate, the module conditional repair rate is also actually a value based on all failed system states. The definition of the module conditional repair rate is the rate that the module gets repaired at time \( t \) given the module is failed at time \( t \). The module mean conditional repair rate can be expressed as:

\[ v_M(t) = \frac{\text{the rate that the module gets repaired at time } t}{\text{the probability that the module is failed at time } t} \]

\[ = \frac{v_M(t)}{Q_M(t)} \]  \hspace{1cm} 7.10

### 7.1.6 Criticality Information of the Basic Events

- **Criticality Function \( G(q) \)**

The criticality function is an importance measure which in its own might be used to rank the contribution of the basic events to the system failure. The definition of the criticality function can be found in section 2.1.4.4.2. The way to calculate the criticality function for each basic event in the Markov model is to explore different system states in the model. Consider a Markov model.
which includes \( m \) basic events and consists of \( n \) system states. Since each system state represents a unique combination of the states of the basic events, the system states which represent the critical state for any specific basic event \( i \) can be easily found. For example, if the system transfers from state \( j \), a working state, to state \( k \), a failed state due to the occurrence of the basic event \( i \), then the combination of the states of other components in state \( j \) or \( k \) forms the critical state for basic event \( i \). However, the criticality function for basic event \( i \) is not simply the total of the probabilities of the critical system states for this component. The critical state for basic event \( i \) only concerns the combination of the states of the other \((m-1)\) basic events, the probability of basic event \( i \) has to be filtered out from the critical system states to obtain the correct criticality function.

There are two ways to achieve this. One can be given by:

\[
G_i(q(t)) = \frac{\sum_j Q_j(t)}{1 - q_i(t)} \tag{7.11}
\]

where state \( j \) is the system state in which the combination of the states of other \((m-1)\) basic events forms the critical state for basic event \( i \) and the basic event \( i \) has not occurred.

The other method of calculating the criticality function is given by:

\[
G_i(q(t)) = \sum_j Q_j(t) + \sum_k Q_k(t) \tag{7.12}
\]

where state \( j \) has the same definition as in equation 7.11 and state \( k \) represents the system state in which the states of all the basic events are exactly the same as in state \( j \) except that the basic event \( i \) is failed\(^2\).

The second method is more efficient than the first because the first calculation method requires the failure probability of basic event \( i \) to be calculated before \( G_i(q(t)) \) can be obtained, and if basic event \( i \) is involved in any dependency relationship, the calculation could be difficult. In the

\[^2\sum_j \text{ and } \sum_k \text{ represents the sum of probabilities of the states which have the properties as are specified for state } j \text{ and } k.\]
second method \( Q(t) \) and \( Q_{l}(t) \) can be easily obtained since the probability of each system state has been worked out to calculate the module failure probability. Therefore, equation 7.12 has been employed in this work to calculate the criticality function for each basic event in the Markov model.

- **Criticality Measure of Importance** \( I_{i} \)

The definition of this measure together with the formula to calculate it has been introduced in the section 2.1.4.4.2. However, the formula applies when component failures are independent and so it is inappropriate to apply the formula to calculating the criticality measure of the basic event in the Markov models developed. Therefore, the way to calculate the criticality measure of basic event \( i \) in the Markov model is to carry out the calculation looking at the contribution of the individual states. System states which conform to the definition of the criticality measure must fulfill three conditions. The first is that in these system states the top event occurs; the second is that in these system states the combination of the states of other components form the critical state for basic event \( i \); and the third is that in these states the component \( i \) is failed.

Then weighted by the module failure probability, the criticality measure of basic event \( i \) is given by:

\[
I_{i}(t) = \sum_{j} \frac{Q_{j}(t)}{Q_{M}(t)}
\]

where state \( j \) refers to the system state which fulfills the three conditions listed above.

Criticality measure must be distinguished from the Fussell-Veseley measure of importance given in equation 2.23 in chapter 2. When gauged by the state probabilities in the Markov model, the Fussell-Veseley measure can be given by:

\[
I_{F.V}(t) = \sum_{k} \frac{Q_{k}(t)}{Q_{M}(t)}
\]

where state \( k \) refers to the system state in which both the system and the component \( i \) is failed.

By adding up the probability of state \( k \), it cancels the effect of component failures which are not
included in the minimal cut sets for the system failure, because among the collection of state \( k \), there must exist two states corresponding to each other, in one of which the component which is not included in the minimal cut sets is working, while in the other it is failed.

It can be noticed that the difference between the criticality measure and Fussell-Veseley measure lies in that for the Fussell-Veseley measure, the states don't have to meet the second condition with regard to the criticality measure. That is, system states which are referred to for the calculation of the criticality measure of importance are part of the states which are used to obtain the Fussell-Veseley measure.

7.2 Quantitative Analysis for Modules Containing no Dependencies

With regard to the fault tree modules whose basic events are all independent, conventional techniques will be sufficient. As has been introduced in chapter 2, these techniques include the conventional fault tree analysis method which utilises the Kinetic Tree Theory (KTT) and also the binary decision diagram (BDD) approach. Compared with the KTT analysis, the BDD has proven to provide a more accurate and efficient solution [11, 12] and has been employed in this work to tackle the modules which include no dependency relationships.

7.2.1 Conversion of Fault Tree Structure to BDD

Binary decision diagram is generated through the conversion from the fault tree structure. As has been mentioned in chapter 2, there exist different mechanisms for the conversion. The basic events need to be placed in an ordering for the conversion process and the ordering mechanism is selected in view of the specific characteristics of each fault tree. The selection of the ordering mechanism forms another interesting research subject, which is however beyond the scope of the current research project. For convenience one method has been used for all fault trees. This is the modified top-down left-right mechanism. The 'top-down left-right' has been shown to give reasonable results for a large class of fault trees, which makes it an appropriate general choice. The fault tree structure can be viewed as a 'tower' structure consisting of different levels. The top event is the highest level at the top of the tower. The 'top-down' means that the basic events at a higher level should have the higher rank in the ordering. The 'left-right' means that at the same level, the rank of the basic events in the ordering will progress from the left to the right of the fault tree. The exception to this is that the basic events which have the larger number of
appearances at the same level should have the higher rank in the ordering. This latter condition incorporated in the ordering scheme produces the 'modified top-down left-right' process. An example fault tree is introduced in figure 7.4 to illustrate how the conversion is implemented by applying the modified top-down left-right approach.

Event e appears twice in the fault tree, while all other events only once. According the top-down left-right approach, the variable ordering for the example fault tree in figure 7.4 is a>b>c>f>e>d>g. Presented below in figure 7.5 is the algorithm of establishing the corresponding BDD according to the pre-determined variable ordering based on the given fault tree structure.
Start from the first basic event in the ordering list; establish a new node for it as the root node for the BDD, set the new node as the current node.

Consider the situation where the basic event in the current node occurs
and update the combination of the known basic event states

If the top event occurs with the current combination of the basic event states
Add the terminal node 'I' to the left branch of the current node.

Consider the situation where the basic event in the current node does not occur and update the combination of the known basic event states

If the top event will definitely not occur with the current combination of the basic event states
Add the terminal node '0' to the right branch of the current node.

Go back to the parent node

Add a new node for this basic event to the left branch of the current node; set the new node as the current node.

Add a new node for this basic event to the right branch of the current node; set the new node as the current node.

Pick the next basic event from the ordering whose state counts regarding the module state

Pick the next basic event from the ordering whose state counts regarding the module state

Figure 7.5 The algorithm of establishing the BDD according to the given variable ordering

7.2.2 Quantitative Analysis Using BDD Method

The quantitative analysis based on the BDD has been described in detail in chapter 2, and therefore won't be repeated in this section.

One thing that needs attention is the calculation of the probability of the basic event in the BDD which features the dormant failure model. The average failure probability of the dormant basic event given by formula 2.4 in chapter 2 is no longer applicable, as the reliability analysis is time dependent. The reliability parameters of the components should also be the function of the elapsed mission time. Consider the failure probability of the dormant-failure component at time \( t \).

Assume that \( n \) inspections on the component have already taken place by time \( t \). Therefore, the failure probability of the component at \( t \) is equivalent to its unreliability during the period of \([n\theta, \theta] \).
Therefore, the failure probability of the dormant-failure component at time $t$ is given by:

$$q_i(t) = 1 - e^{-\lambda(t-n\theta)}$$  \hspace{1cm} 7.15

where $\theta$ is the inspection interval of the component $i$ and the $n$th inspection is the last inspection carried out by time $t$.

This decides that the failure probability of a dormant-failure component as the function of time will take the pattern shown in figure 7.6.

![Figure 7.6 The pattern of failure probability of dormant-failure components](image)

According to the principle presented in equation 7.15, test dependency (see section 4.1.8) can be examined from a similar perspective. The effect that the test dependency has on the component failure probability can be accounted for by addressing the failure probabilities of the relevant components in the way represented in equation 7.16:

$$q_i(t) = 1 - e^{-\lambda(t - \text{max}(n_1\theta_1, n_2\theta_m))}$$  \hspace{1cm} 7.16

where:

- $t$ – the actual elapsed mission time;
- $\theta_1$ – the individual inspection interval for component $I$;
- $\theta_m$ – the common inspection interval for the group of components;
- $\text{max}(n_1\theta_1, n_2\theta_m)$ – means the component failure probability is considered from the point the most recent inspection (whether an individual or a common inspection) takes place.

### 7.3 System Quantitative Analysis

#### 7.3.1 System Reliability Parameters

The previous sections have illustrated how the module quantification can be realized where
dependencies exist or not. This section looks at how the module parameters can then be used in
the process of calculating the reliability measures of the whole system. Among all the modules in
the fault tree structure, the ‘top’ module is the module which is headed by the top event of the
original fault tree structure.

The quantification process is implemented in a bottom-up manner, starting from modules which
are located at the lowest level in the fault tree, i.e. the modules which contain no other modules,
up to the top module. If there exist dependency relationships in the top module, the
quantification will be carried out as is illustrated in section 7.1 using the established Markov
model. Otherwise, a binary decision diagram will be constructed to represent the top module
failure function and quantified. In most cases, the top module will include other modules in it as
descendants. These modules will be treated as super events in the top module during the
quantification process and may also form the parent module for other modules. If the parent
module is solved by the Markov method, the conditional failure rate and repair rate of its
descendant module(s) will be referred to when establishing the rate matrix for the parent module.
Similarly, when the parent module is handled using the binary decision diagram, the failure
probability of the descendant modules will be included in the calculation of the reliability
parameters of the parent module. In this way, the reliability parameters of the descendant
modules will be fed back into the quantification of the parent module. It should be noted that
since the module conditional failure rate and repair rate vary as mission time elapses, the Markov
model which includes descendant module(s) no longer feature constant-rate transitions and
therefore is actually a non-homogeneous model (see Chapter 3).

7.3.2 Basic Event Criticality
The previous sections introduced how the importance measures of each basic event, such as the
criticality function \( G_i(q) \) and the criticality measure of importance \( I_i \), can be obtained relative to
the module top event. This section will look at how the importance measures of the basic events
included in the modules can be calculated relative to the overall system failure.

First consider a top module \( M_0 \) and its descendant module \( M_i \), neither of which contains a
dependency relationship. Basic event \( i \) is included in module \( M_i \). Assume that two binary
decision diagrams have been established as are shown in figure 7.7 to facilitate the quantification of modules $M_0$ and $M_1$ respectively.

Module $M_0$  

\[
\begin{array}{c}
... \\
1 \\
0 
\end{array}
\]

Module $M_1$  

\[
\begin{array}{c}
... \\
1 \\
0 
\end{array}
\]

Figure 7.7 BDDs for example fault tree structure

These two BDDs can be merged to produce one BDD for the top module which includes the basic events originally included in the BDD of module $M_1$. This is implemented by taking the BDD of the top module and replacing the node which represents the descendant module with its whole BDD structure. In the new combined BDD, the node which, in $M_0$, was originally connected to the left branch of the node representing $M_1$, is now linked to the branch in the BDD of $M_1$ which terminates with a terminal node ‘1’; and the node which was originally connected to the right branch of the node representing $M_1$, is now linked to the branch in the BDD of module $M_1$ which terminates with a terminal node ‘0’. Therefore, the form of the new BDD is as shown in figure 7.8:

\[
\begin{array}{c}
... \\
1 \\
0 
\end{array}
\]

Figure 7.8 Incorporated BDD(referring to figure 7.6)

In the combined BDD, the criticality function of basic event $i$ can then be directly obtained according to the algorithm presented in chapter 2.

However, a problem rises when one or both of the parent module and descendent module
contains a dependency relationship and has to be solved with the Markov method. In this case, the BDD incorporation approach is no longer applicable, and an alternative procedure must be followed as demonstrated below.

Consider a parent module $M_j$ and the descendant module $M_k$ immediately included in module $M_j$. Basic event $i$ is included in module $M_k$. $G_{i/M_j}(q(t))$ and $G_{M_k/M_j}(q(t))$ denote the criticality function of basic event $i$ relative to module $M_k$ and the criticality function of module $M_k$ relative to module $M_j$ respectively. The solution will address $G_{i/M_j}(q(t))$, i.e. the criticality function of basic event $i$ relative to module $M_j$.

The criticality function for basic event $i$ is defined in equation 2.20, $G_i(q) = \frac{\partial Q(q)}{\partial q_i}$. Therefore:

$$G_{i/M_j}(q) = \frac{\partial Q_{M_j}(q)}{\partial q_i}$$

$$= \frac{\partial Q_{M_k}(q)}{\partial q_i} \times \frac{\partial Q_{M_k}(q)}{\partial q_i}$$

$$= G_{M_k/M_j}(q) \cdot G_{i/M_k}(q) \quad 7.17$$

Equation 7.17 is obtained by considering the influence of basic event $i$ on the failure probability of module $M_j$ through the intermediate module $M_k$. As basic event $i$ is not directly contained in module $M_j$, its influence on the failure probability of module $M_j$ is investigated by breaking it down into the influence of basic event $i$ on the failure probability of module $M_k$, i.e. $\frac{\partial Q_{M_k}(q)}{\partial q_i}$, and the influence of module $M_k$ on the failure probability of module $M_j$, i.e. $\frac{\partial Q_{M_j}(q)}{\partial Q_{M_k}(q)}$.

Equation 7.17 can also be obtained by looking at the criticality function given by equation 2.19, $G_{i/M_j}(q) = Q_{M_j}(1_i) - Q_{M_j}(0_i)$, where $Q_{M_j}(1_i)$ refers to the conditional failure probability of module $M_j$, given component $i$ is failed; and $Q_{M_j}(0_i)$ represents the conditional failure
probability of module $M_j$, given component $i$ is working.

Let $M_j F$, $M_k F$ respectively denote the failure of module $M_j$ and $M_k$, and $i F$ denote the failure of component $i$, $M_k W$ and $i W$ respectively denote that module $M_k$ and component $i$ is working. The two terms in the right-hand of equation 2.19 are investigated respectively as follows:

\[
Q_{M_j}(1_i) = \frac{P(\text{Module } M_j \text{ is failed AND component } i \text{ is failed})}{P(\text{component } i \text{ is failed})} = \frac{P(M_j F \text{ AND } i F)}{P(i F)}
\]

by taking into account the state of module $M_k$:

\[
Q_{M_j}(1_i) = \frac{P(M_j F \text{ AND } M_k F \text{ AND } i F) + P(M_j F \text{ AND } M_k W \text{ AND } i F)}{P(i F)}
\]

by considering the effect of component $i$ on module $M_j$ through module $M_k$:

\[
Q_{M_j}(1_i) = \frac{P(M_j F | M_k F)P(M_k F | i F)P(i F) + P(M_j F | M_k W)P(M_k W | i F)P(i F)}{P(i F)}
\]

\[
= P(M_j F | M_k F)P(M_k F | i F) + P(M_j F | M_k W)P(M_k W | i F)
\]

\[
= Q_{M_j}(1_{M_k}) \times Q_{M_k}(1_i) + Q_{M_j}(0_{M_k}) \times [1 - Q_{M_k}(1_i)]
\]

\[
Q_{M_j}(0_i) = \frac{P(\text{Module } M_j \text{ is failed AND component } i \text{ is working})}{P(\text{component } i \text{ is working})} = \frac{P(M_j F \text{ AND } i W)}{P(i W)}
\]

by taking into account the state of module $M_k$:

\[
Q_{M_j}(0_i) = \frac{P(M_j F \text{ AND } M_k F \text{ AND } i W) + P(M_j F \text{ AND } M_k W \text{ AND } i W)}{P(i W)}
\]

by considering the effect of component $i$ on module $M_j$ through module $M_k$:

\[
Q_{M_j}(1_i) = \frac{P(M_j F | M_k F)P(M_k F | i W)P(i W) + P(M_j F | M_k W)P(M_k W | i W)P(i W)}{P(i W)}
\]

\[
= P(M_j F | M_k F)P(M_k F | i W) + P(M_j F | M_k W)P(M_k W | i W)
\]
Therefore:

\[ G_{M_j}(q) = Q_{M_j}(1) - Q_{M_j}(0) \]

\[ = \{ Q_{M_j}(1) - Q_{M_j}(0) \} \times [Q_{M_j}(1) - Q_{M_j}(0)] \]

Assume \((k+1)\) modules \(M_0 - M_k\), where module \(M_0\) is the top module and module \(M_j\) is the immediate descendant of module \(M_j\) \((j = 1, 2, \ldots, k)\). Basic event \(i\) is included in module \(M_n\) \((1 \leq n \leq k)\). The criticality function of basic event \(i\) relative to the top module can then be given by:

\[ G_{M_n/M_0}(q(t)) = \prod_{p=1}^{n} G_{M_p/M_{p+1}}(q(t)) \]  \(7.18\)

Equation 7.18 can be interpreted as the 'chain rule' of the criticality function and reflects the 'transferability' of the basic event criticality function in the multi-level modularized fault tree structure.

The 'transferability' of the criticality function also applies to the criticality measure of importance of the basic event. Equation 2.22 given in chapter 2 illustrates how to obtain the criticality measure of importance by drawing on the criticality function. The criticality measure of basic event \(i\) included in the immediate descendant module \(M_k\) relative to the parent module \(M_j\) can be obtained as follows:

\[ I_{M_j} = \frac{G_{M_j}(q) \times q_j(t)}{Q_{M_j}(t)} \]

\[ = \frac{G_{M_j}(q) \times G_{M_j/M_j}(q) \times q_j(t)}{Q_{M_j}(t)} \]

\[ = \frac{G_{M_j}(q) \times G_{M_j/M_j}(q) \times q_j(t) \times Q_{M_j}(t)}{Q_{M_j}(t) \times Q_{M_j}(t)} \]
\[
\frac{G_{M_s/q}(q) \times q_i(t)}{Q_{M_s}(t)} \times \frac{G_{M_s/M_f}(q) \times Q_{M_f}(t)}{Q_{M_f}(t)} = I_{i/M_s} \times I_{M_s/M_f} \tag{7.19}
\]

Therefore, the criticality measure of the basic event relative to the top module can be obtained using the same process as for the criticality function:

\[
I_{i/M_s} = I_{i/M_s} \times \prod_{p=1}^{n} I_{M_p/M_{p-1}} \tag{7.20}
\]

where \(M_0, M_n, \) and \(M_p\) has the same meaning as in equation 7.18.

Using equations 7.18 and 7.20, the criticality function and criticality measure of the basic event included in descendant modules relative to the top module can be acquired in an accurate and efficient way.
Chapter 8. Dependency Modelling of Continuously-Operating Systems

8.1 Introduction
All aspects of system dependency modelling are combined together in this chapter to form an overview of the whole process. Two types of systems are considered in the thesis separately. In this chapter, the focus is placed upon the dependency modelling in the reliability assessment of continuously-operating systems. The process of developing the software to perform the assessment is also described. An example system is used to illustrate the software application.

8.2 The Dependency Modelling Process for Continuously-Running Systems
In the assessment of continuously-operating systems, the general process of dependency modelling is illustrated by the flow chart in figure 8.1.

![Flow Chart](image)

Figure 8.1 The general process of assessing the reliability of continuously-running systems involving dependency modelling

The following section will describe how each step is realized and linked to each other in the software development.

8.3 The Software Development
The software was developed using the C programming language. The software is structured in a framework which consists of 4 blocks: data input, pre-processing, quantification, and data output. These blocks are described separately in the following sections.
8.3.1 Data Input

The assessment of any system has to start with the representation of the causes of system failure. The ‘data input’ process is designed to provide the software with all the system information required to perform the evaluation of the system’s availability and reliability. The basic system information includes the system failure logic, the components’ reliability parameters and the dependency relationships. This can be represented using three different data files. In terms of the fault tree structure file, each different gate event in the fault tree is regarded as a different record in the file. For each record, the information specifies the name of the gate event (which consists of no more than 20 characters), the type of the gate (which include ‘OR’, ‘AND’, ‘Priority AND’ and ‘Voting’ respectively encoded as ‘+’, ‘*’, ‘&’ and ‘v’ in the input), the number of the immediate gate event descendants, the number of immediate basic event descendants, the list of the names of the immediate gate event descendants and the list of the names of the immediate basic event descendants. When the gate type is ‘Voting’, additional information is required to specify the minimum number of occurrences of its immediate descendants to cause the gate event output. Dependency information is accomplished by generating two dependency files. One is the so-called ‘normal dependency’ file which includes the dependency relationships discussed in sections 4.1.1 – 4.1.7, i.e. those which result in the statistical dependency during the evaluation process and the test dependency. The other dependency file is the ‘functional dependency’ file which only contain functional and switching dependencies. The dependency groups included in these two files share the same numbering system. For different types of dependency relationship, the information that the analyst provides will vary, as has been illustrated in detail in section 4.2. The advantage of distinguishing the functional and switching dependencies from other types of dependency is to enable the analyst to gain a clearer understanding of the different characteristics of the distinct types of dependency.

Corresponding to the third step in figure 8.1 is the provision of the basic event file, each record in the file represents a unique basic event in the corresponding fault tree structure. The basic event file contains the name of the basic event, the failure model (‘1’ for fixed, ‘2’ for revealed and ‘3’ for dormant), the corresponding reliability parameters relevant to each failure model (see chapter 2), and other dependency features. In terms of the dependency features, a code is given to define if a basic event is an enabling event or not. A value of ‘0’ means that the basic event is not an enabling event, and an input of ‘1’ or ‘2’ means the basic event is an enabling event and indicates respectively that the enabling event is unlikely or still likely to occur after the initiating event has occurred. The number of dependency groups in which this basic event is involved is also specified. When the input is ‘0’, it means this basic event is independent and is not involved
in any dependency relationship in the system. It must be noted that the dependency group numbers included in the basic event file can only come from the ‘normal’ dependency file.

8.3.2 Pre-processing

The principle underlying the pre-processing procedure has been described and illustrated in chapter 5. Through this process, the data provided to the software in the form of the three input files will be examined and refined to make the quantification process more efficient. The pre-processing can be broken down into the steps shown in figure 8.2.

![Diagram of pre-processing steps](image)

Figure 8.2 Procedures included in the pre-processing stage

The aim of the ‘proof-reading’ is to ensure that the fault tree structure file, the basic event file and the dependency files are correctly related and consistent with each other. Also error checking within the fault tree structure file is carried out, aimed to ensure that gate events which have the exactly same input events and appear more than once in the fault tree are assigned the same name. This process is implemented in a bottom-up manner, starting from based events in the fault tree.

In the second procedure, the data contained in the basic event file and dependency files is extracted and re-organized into the required different structures so that they can be processed by the program more efficiently. Each record in the basic event file is represented by a ‘Structure’ (a special data structure embedded in C programming language), and all the structures are linked one by one to form the basic event list. Numerical labels, starting from 1, are assigned to each basic event. The dependency files are then processed in the same way. A single dependency information list is established using information from both the normal dependency and the functional dependency files. Any basic events involved in the dependency files are identified using their numerical labels.

In the third step, the fault tree structure file is examined. A binary diagram is employed to represent the fault tree structure given in the file. In the binary tree structure, each node
represents an intermediate gate event or a basic event in the fault tree. In the process of establishing the binary fault tree, numerical labels (in the software, the numbering starts from 10000) are assigned to each different intermediate gate event. In addition to the numerical label of the gate event, each node contains the logic and two branches. The left branch is linked to the node representing its first immediate descendant and the right branch is linked to the node representing its sibling event under the same parent gate. When the node represents a basic event, both of its left branch and its logic are null since there is no input event to it. When the node represents an intermediate gate event, the logic refers to the type of its gate\(^1\). The right branch of the top event is null. Take for instance the fault tree in figure 5.6 in chapter 5, its corresponding binary tree structure is shown in figure 8.3.

![Binary tree representation of the fault tree in figure 5.6](image)

In the forth step, the re-organization of the dependency relationships is carried out using the established dependency information list. The aim is to establish the dependency serial list. This is done by first forming a temporary dependent event list in which each entry corresponds to a dependency group from the 'normal' type of dependency. In the dependent event list, each entry contains only the dependency group number and the list of events involved represented by their numerical labels. The dependency type doesn't need including. During the process of establishing the dependent event list, the intermediate gate events included in the corresponding dependency group may need to be replaced by its basic event descendants according to the principles given in section 5.2.1 depending on the dependency type. When the construction of the dependent event list is completed, the dependency serial list is established accordingly as is illustrated in section 5.2.1. Each entry in the dependency serial list includes the assigned serial

\(^1\) The 'AND' logic is encoded '1', the 'OR' logic is encoded as '2', the 'Priority AND' logic is encoded as '3', and the 'Voting' logic is encoded as the value of \((4+\text{number of votes})\).
number, the group number of the dependency groups involved and the list of all the relevant
events represented by their numerical labels. During the process of establishing the temporary
dependent event list, the test dependency is not taken into account since the test dependency does
not result in the statistical dependency and therefore does not require the use of the Markov
method. This means that the basic events which are involved in the test dependency relationship
do not need to be included in the same fault tree section in the analysis.

In the fifth step, the fault tree simplification is carried out. The simplification operations are
conducted on the established binary fault tree structure according to the algorithm illustrated in
chapter 5. With the exceptions discussed in section 5.2.2, contraction, extraction, elimination and
factorisation are carried out on the binary tree structure in a left-right depth-first manner,
generating a simplified binary fault tree structure. During the factorisation process, each factor is
assigned with a numerical label (starting from 3001) to distinguish it from other events in the
fault tree. A factor list is generated in which each entry represents a factor containing the factor
ID label and the list of the basic events included in the factor. Those basic events which are
grouped as factors are substituted with new node marked with the corresponding factor ID label
in the binary tree structure. Figures 8.4 and 8.5 contain the algorithms underlying the different
functions during the simplification process.

```
Figure 8.4 Algorithm underlying the contracting and eliminating operation in the
simplification process
```

```
The root node of the binary fault tree structure:
FTND = R (id, logic, *lptr, *rptr)

simplify_faulttree(R)
{contract_faulttree(R);
 eliminate_events(R);
 extract_events(R);
 factorise_events(R);
}

contract_faulttree(FTND *parent)
{
 FTND *ftwk;
 ftkw = parent->lptr;
 while(ftwk!=0)
 {if(ftwk->id represents a gate id)
  contract_faulttree(ftwk);
  ftkw=ftwk->rptr;
  }
 if(parent->logic < 3)
 { ftkw = Base->lptr;
  while(ftwk!=0)
   {if(ftwk->id represents a gate id)
    if(fkw->logic == parent->logic)
     add descendants of ftkw to parent;
     delete the gate pointed by ftkw;
    }
    ftkw=ftwk->rptr;
  }
 }
 eliminate_events(FTND *parent)

{FTND *ftwk, *ftwk1;
 if(parent->logic<3)
 {ftwk=parent->logic;
  while(ftwk!=0)
   {ftwk1=parent->lptr;
    while(ftwk1!=0)
     {if(ftwk1==ftwk && ftwk1 is a gate)
      search under ftwk1 in a top-down manner to see
      if the same event as pointed by ftwk exists under
      ftwk1;
      if(repeated event exists)
       {if(the logic of the secondary gate is the same as the primary gate)
        delete the repeated event under ftwk1;
        else /* opposite logic */
        delete the secondary gate;
        if(eliminate leaves the gate with only one input)
        delete the gate;
        contract_faulttree(parent of the deleted gate);
       }
       ftkw=ftwk1->rptr;
    }
    ftkw=ftwk->rptr;
    }
    ftkw=parent->logic;
    while(ftkw1!=0)
     {if(ftkw points at a gate)
      eliminate_events(ftkw);
      ftkw=ftwk->rptr;
     }
    }
 }`
Figure 8.5 Algorithm underlying the extracting and factorising operation in the simplification process

Finally, to carry out the sixth step in figure 8.2, a gate dependency serial list is established. In this process, the binary fault tree structure is traversed in a left-right depth-first manner so that the dependency serial numbers encountered under each intermediate gate event are identified and recorded. The gate dependency list is referred to for the combination process as is described in section 5.2.4. The combination operation is carried out directly on the binary fault tree structure by introducing new nodes which represent the intermediate events added to group the events together with same dependency serial numbers.

Following the above steps, the modularization is conducted on the binary fault tree structure. The two left-right depth-first traversals through the binary fault tree structure provide both the absolute and relative positions of the intermediate gate and basic events in the fault tree. According to the principle of deciding if an intermediate gate event is leading a module based on the relative positions of its descendants in the fault tree, modules can be identified. Numerical labels are allocated to each module (starting from 6001) such that modules can be distinguished.
from basic events, intermediate events and factors. A module list is formed in which each entry represents the identified module. A link is established between each entry in the module list and the corresponding module fault tree structure contained in the original binary fault tree, in the mean time, new nodes which represent the modules are introduced into the binary fault tree to replace the modules’ fault tree structures.

In the modularized binary tree structure, a left-right depth-first traverse is implemented to update the information contained in the established gate dependency list. Modules and factors will be treated as basic events in the traversal. The traversal extends into each module through the link contained in each entry in the module list. In this process, the dependency serial numbers mutual to all the immediate gate descendants of the intermediate event are also recorded in the gate dependency list. Modules which contain dependency relationships are identified through the information in the gate dependency list. Re-modularization is carried out on the fault tree structures of modules which contain dependencies according to the algorithm in section 5.2.7.

8.3.3 Quantification

8.3.3.1 Preparation
The gate dependency list tells which modules contain dependencies and which don’t. In the preparation stage, the binary decision diagrams and the Markov models are established for modules which contain no dependency and modules which do involve dependency relationships respectively.

- Binary Decision Diagrams Development
A binary decision diagram is required for each factor and module involving no dependency serial numbers. Each node in the binary decision diagram represents a basic event or a complex event (module or factor).

For each factor, the binary decision diagram is established in a straightforward way. When the basic events included in the factor are grouped with the OR logic, the binary decision diagram is constructed by linking each basic event through their right branches and connecting the terminal node ‘1’ to the left branch of each basic event. On the opposite, if the basic event contained in the factor are grouped with the AND logic, the binary decision diagram is constructed by linking each basic event through their left branches and connecting the terminal node ‘0’ to the right branch of each basic event.
For the process of establishing the BDD for modules, a modified top-down ordering mechanism is employed to establish the order in which the basic/complex events in the module should be considered. Figure 8.6 presents the algorithm of establishing the binary decision diagram based on the pre-determined event ordering.

```plaintext
build_module_bdd(the ordering list & the module fault tree structure)
{
    establish empty stack;
    establish_bdd(the ordering list, the module fault tree structure & the stack);
}

establish_bdd(the ordering list, the module fault tree structure & the stack)
{
    current event = pick_event(the ordering list & the stack);
    put the current event in the stack;
    add new node in the bdd representing the current event;
    consider the situation when the event occurs:
    if (the module state can not be decided according to the states of the events in the stack)
        mark the left branch of the current node;
        establish_bdd(the ordering list & the stack);
    else
        add the terminal node with value '1' to the left branch of the current node;
        consider the situation when the event does not occurs:
        if (the module state can not be decided according to the states of the events in the stack)
            mark the right branch of the current node;
            establish_bdd(the ordering list & the stack);
        else
            add the terminal node with value '0' to the right branch of the current node;
    if (the current event is the bottom one in the stack)
        link the corresponding module to the established bdd;
    else
        link the current node to the marked branch;
        take the event out of the stack;
}

pick_event(the ordering list & the stack)
{
    find the event in the ordering list which follows the event at the top of the stack;
    while (the current event ... the last event in the ordering list)
    {
        if (the state of the event in question is relevant regarding the module state based on the current states of the events in the stack)
        pick the event; break;
    }
}
```

Figure 8.6 The algorithm of establishing the BDD for modules

**Markov Model Development**

The Markov model is established for each fault tree module which contains dependencies according to the type of dependency involved in the module. When the test dependency is included in the modules which also contain other 'normal' statistical types of dependency, the algorithm presented in figure 6.28 should be referred to during the process of generating the Markov model. In the software, the development of the Markov model is realized by establishing
a list of system states and a list of state transitions which are both linked to the corresponding module in the module list. If the resulting Markov model contains $n$ system states. Then in theory, this will result in an $n \times n$ rate matrix which is sparse in nature. In the software, to save the memory space and to improve the computing efficiency, the rate matrix is simplified such that only non-zero entries are stored. The simplified rate matrix is composed of $n$ rows. The first element of each row represents the source state from which the transition takes place. Other elements in each row represent the destination states which the transitions take the system to from the source states and give the corresponding transition rate. When the transition is caused by the failure or repair of a complex event (a factor or a module), the rate is set as ‘-1’ in the first place. These elements will be different from the constant failure or repair rates of components, the complex event features a variable failure or repair rate as the function of time. The $n$ rows are connected through a list which links the first element of each row. The structure of the simplified rate matrix is illustrated in figure 8.7. Each rate matrix is linked to the corresponding module in the established module list.

\[ \begin{align*}
\text{Source state: } i & \rightarrow \text{destination state: } j & \rightarrow & \ldots & \rightarrow \text{destination state: } k \\
& \downarrow & & & \downarrow \\
& \ldots & & & \ldots \\
\text{Source state: } j & \rightarrow \text{destination state: } i & \rightarrow & \ldots & \rightarrow \text{destination state: } m \\
& \downarrow & & & \downarrow \\
& \ldots & & & \ldots \\
\text{Source state: } m & \rightarrow \text{destination state: } j & \rightarrow & \ldots & \rightarrow \text{destination state: } k \\
& \downarrow & & & \downarrow \\
& \ldots & & & \ldots 
\end{align*} \]

Figure 8.7 The structure of the simplified rate matrix for the Markov model

8.3.3.2 The Quantification Process

For continuously-operating systems, the availability and reliability assessment is carried out over a specified period of time $T$. In the software, the analyst will be asked to input the total period of time $T$ that the evaluation will cover. Also the analyst will be asked to specify the length of $dt$ (the time increment) by which the quantification process will progress.
During the quantification process, each module and factor will be investigated to obtain the factor/module unavailability, the factor/module unconditional failure and repair intensity, and also the factor/module conditional failure and repair rates. The criticality function of each basic event is also calculated as a function of the mission time. The quantification process starts in the top module in the modularized fault tree structure. The algorithm presented in figure 8.8 indicates how the quantification is implemented using both the BDD and the Markov method.

During the quantification of the modules which contain no dependencies, extra attention should be paid to components which are involved in test dependency, as equation 7.16 should be applied to calculating the component failure probability.

The process given in the algorithm presented above delivers the reliability parameters as a function of time for the top module. By implementing the procedure given in the algorithm presented in figure 8.9, the criticality function of each component with respect to the top event is obtained.

8.3.4 Output
When the quantification process is finished, two files will be generated as the final products from the assessment. One file is the ‘system prediction file’ which includes the system unavailability, system unconditional failure intensity and expected number of failures $\int w_{ys}(t)dt$ as functions of time. The analyst will be asked to input the time interval at which the system reliability parameters are saved in the file. The other file is the ‘importance measure file’ which holds the criticality function of each basic event in the system as the function of time. The analyst can also decide the time interval between each record in the file.
Quantify_system(the factor list, the module list, T, dt)
{
establish the criticality function list for each factor and module;
record the ID of the immediate parent factor/module of each event in the criticality function list;
for(count = = 0; count*dt<=T; count++)
quantify_module(the top module, the factor list, the module list, count, dt);
}

quantify_module(the current module, the factor, the module list, the basic event list, count, dt)
{
establish array param[5] with T'=count*dt for the module;
if(the module contains no dependency)
quantiﬁe_bdd(the current module, the factor list, the module list, the basic event list, count, dt);
else
quantify_markov(the current module, the factor list, the module list, the basic event list, count, dt);
}

quantify_bdd(current module, the module list, the basic event list, count, dt)
{
root vertex F = ite(x; J, K) of the BDD is linked to the module;
param[0] = calcyrobvalue(F, count, dt); /* the module/factor unavailability*/
calc_criticality_bdd(F, criticality function list, count, dt);
while(each basic/complex event i included in the current module)
param[1] = param[1] + G(q(T'))*w(T'); /* the module/factor unconditional failure intensity */
param[2] = G(q(T'))*v(T'); /* the module/factor unconditional repair intensity */
param[3] = param[1]/(1-param[0]); /* the module/factor conditional failure rate */
param[4] = param[2]/param[0]; /* the module/factor conditional repair rate */
}

Root vertex F = ite(x; J, K)
calcyrobvalue(F)
{
if(x; represents a factor)
quantify_factor(x;);
if(x; represents a module)
quantify_module(x;);
Consider 'I' branch:
if(J = 1)
pо'[F] = 1;
else
{ if(visited[J] = 1)
temp = param[J];
else
temp = 0;
pr[J] = q;pr[F];
po'[F] = calcyrobvalue(J);
}
Consider '0' branch:
if(K = 0)
po[F] = 0;
else
{ if(visited[K] = 1)
temp = pr[K];
else
}
temp = 0;
pr[K] = (1-q;).pr[F]
pо'[F] = calcyrobvalue(K);;
pr[K] = pr[K] + temp;
return(P[F]);
}

calc_criticality_bdd(F(x; J, K), criticality function list, count, dt)
{
if(J is not a terminal node)
calc_criticality_bdd(J, criticality function list);
if(K is not a terminal node)
calc_criticality_bdd(K, criticality function list);
if(event x; has not been visited)
Gi(q(T')) = pr[F]*p[J] - P[K];
else
Gi(q(T')) = Gi(q(T')) + pr[F]*p[J] - P[K];
}

Figure 8.8a Algorithm for quantification of continuously-operating systems using BDD
quantify_markov(the current module, the module list, the basic event list, count, dt)
{
    if(count == 0)
        set the value of the initial state probabilities;
        /* Q1(0)=1, Q2(0) = Q3(0) = .... = Qn(0) = 0 */
    else
        establish temporary state probability list temp;
        while(the first state ...... the last state in the system state list)
        {  temp[i] = 0;
            while(the first row ...... the last row in the rate matrix)
            {  if (the transition is caused by a factor x)
                quantify_factor(x);
                if (the transition is caused by a module x)
                    quantify_module(x);
                if (the row number is the state number)
                    temp[i] = temp[i] + Q(((count-1)*dt)*(1- \sum r\cdot dt));
                else
                    temp[i] = temp[i] + \sum Qj((count-1)*dt)*r\cdot dt;
            }
        }
        Q((count*dt)) = temp[i];
    }
}
while(the first state ...... the last state in the system state list)
{  if (the module is failed in the current state)
    param[0] = param[0] + Q,(count*dt));
}
adjust the state probabilities if the inspection is expected to occur at T=count*dt;
while(the first row ...... the last row in the rate matrix)
{  while(the first element ...... the last element in the row)
    {  if (the transition brings the system from working to failed state)
        param[1] = param[1] + Q((count*dt))*r\cdot / * the module unconditional failure intensity */
        if (the transition brings the system from failed to working state)
    }
    param[3] = param[1]/(1-param[0]); /* the module conditional failure rate */
    param[4] = param[2]/param[0]; /* the module conditional repair rate */
}
calc_criticality_markov(the current module, the criticality function list, count, dt);
}

calc_criticality_markov(the current module, the criticality function list, count, dt)
{
    while(the first basic/complex event ...... the last one in the criticality function list)
    {  while(the first row ...... the last row in the rate matrix)
        {  while(the first element ...... the last element in the row)
            {  if (the transition brings the system from working to failed state & the transition is caused by the occurrence of the event under investigation)
                Qj(q(T')) = Qj(q(T')) + Qj(T') + Qj(T');
                /* state \cdot is the source state & state \cdot is the destination state of the transition; */
            }
        }
    }
}

Figure 8.8b Algorithm for quantification of continuously-operating systems using Markov
generalize_criticality(the criticality function list)
{
    while(the first basic/complex event......the last one in the list)
    {
        while(the event's immediate parent is not the top module)
        {
            look for the event's immediate parent in the list;
            replace the event's original $G_i(q(t))$ with the multiple of $G_i(q(t))$ and the
            criticality function of its immediate parent event;
            replace the event's original immediate parent with its immediate grandparent;
        }
    }
}

Figure 8.9 The algorithm of calculating the criticality function in relation to the top event

8.4 Application to an Example Fault Tree
The fault tree shown in figure 5.1 represents the failure of a continuously-operating system. Dependency relationships exist in the system including a maintenance dependency between basic events $e_5$ and $e_8$, and between $e_{18}$, $e_{19}$ and $e_{20}$; a sequential dependency, as is indicated by the 'Priority AND' gate, under gate $G_{14}$ between basic events $e_{18}$, $e_{19}$ and $e_{20}$; an initiator-enabler dependency between $G_{18}$ and $e_{15}$, of which $G_{18}$ represents the general initiating event and basic event $e_{15}$ is the enabling event; and a test dependency between basic events $e_1$, $e_4$ and $e_7$ with a common inspection interval of 4 months. The following section describes how the software is applied to the example fault tree to predict the system reliability.

8.4.1 System Information Input

* Fault Tree Structure File
For the fault tree shown in figure 5.1, the corresponding fault tree structure file has the form shown in table 8.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Logic</th>
<th>Number of input gates</th>
<th>Number of input basic events</th>
<th>List of input gates</th>
<th>List of input basic events</th>
</tr>
</thead>
<tbody>
<tr>
<td>G0</td>
<td>*</td>
<td>3</td>
<td>0</td>
<td>G1, G20, G24</td>
<td></td>
</tr>
<tr>
<td>G1</td>
<td>+</td>
<td>2</td>
<td>0</td>
<td>G2, G19</td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td>+</td>
<td>2</td>
<td>0</td>
<td>G3, G4</td>
<td></td>
</tr>
<tr>
<td>G3</td>
<td>*</td>
<td>0</td>
<td>2</td>
<td></td>
<td>e9, e10</td>
</tr>
<tr>
<td>G4</td>
<td>*</td>
<td>3</td>
<td>0</td>
<td>G5, G15, G16</td>
<td></td>
</tr>
<tr>
<td>G5</td>
<td>+</td>
<td>2</td>
<td>1</td>
<td>G6, G14</td>
<td>e8</td>
</tr>
<tr>
<td>G6</td>
<td>*</td>
<td>1</td>
<td>2</td>
<td>G7</td>
<td>e5, e6</td>
</tr>
<tr>
<td>G7</td>
<td>+</td>
<td>1</td>
<td>1</td>
<td>G8</td>
<td>e12</td>
</tr>
<tr>
<td>G8</td>
<td>*</td>
<td>1</td>
<td>1</td>
<td>G9</td>
<td>e2</td>
</tr>
<tr>
<td>G9</td>
<td>*</td>
<td>2</td>
<td>0</td>
<td>G10, G11</td>
<td></td>
</tr>
<tr>
<td>G10</td>
<td>+</td>
<td>0</td>
<td>2</td>
<td></td>
<td>e3, e4</td>
</tr>
<tr>
<td>G11</td>
<td>+</td>
<td>2</td>
<td>0</td>
<td>G12, G13</td>
<td>e1, e2</td>
</tr>
<tr>
<td>G12</td>
<td>*</td>
<td>0</td>
<td>2</td>
<td></td>
<td>e4, e7</td>
</tr>
<tr>
<td>G13</td>
<td>*</td>
<td>0</td>
<td>2</td>
<td></td>
<td>e18, e19, e20</td>
</tr>
<tr>
<td>G14</td>
<td>&amp;</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

181
where:
•*• represents ‘AND’ logic; ‘&’ represents ‘Priority AND’ logic; and ‘+’ represents ‘OR’ logic.

Table 8.1 Fault tree structure file (referring to figure 5.1)

• Dependency File

The corresponding dependency file for the system is shown in table 8.2 (refer to section 4.2 for an explanation of the contents of the last 4 columns for each type of dependency).

<table>
<thead>
<tr>
<th>Dependency group number</th>
<th>Dependency type</th>
<th>Number 1</th>
<th>Number 2</th>
<th>List 1</th>
<th>List 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mtnl</td>
<td>2</td>
<td>1</td>
<td>5 and 8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>mtnl</td>
<td>3</td>
<td>1</td>
<td>18, 19 and 20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>sq</td>
<td>3</td>
<td>1</td>
<td>18, 19 and 20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ie</td>
<td>2</td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>test</td>
<td>3</td>
<td>4</td>
<td>1, 4, 7</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.2 Dependency file for the example fault tree

• Basic Event File

The basic event file, as is illustrated in section 4.4, for the example fault tree is shown in table 8.3.

<table>
<thead>
<tr>
<th>Name of basic event</th>
<th>Failure model</th>
<th>Reliability parameters</th>
<th>Enabler</th>
<th>Number of dependency groups involved</th>
<th>List of dependency groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>3</td>
<td>0.0001, 3, 3</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>e2</td>
<td>3</td>
<td>0.0006, 3, 2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e3</td>
<td>2</td>
<td>0.0008, 0.4</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e4</td>
<td>3</td>
<td>0.0005, 2, 3</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>e5</td>
<td>2</td>
<td>0.0007, 0.5</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e6</td>
<td>3</td>
<td>0.0001, 4, 6</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e7</td>
<td>3</td>
<td>0.0002, 2.5, 3</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>e8</td>
<td>2</td>
<td>0.0005, 0.4</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e9</td>
<td>2</td>
<td>0.0004, 0.25</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e10</td>
<td>2</td>
<td>0.0003, 0.2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e11</td>
<td>3</td>
<td>0.0001, 3, 2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e12</td>
<td>2</td>
<td>0.0004, 0.1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e13</td>
<td>2</td>
<td>0.0006, 0.4</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e14</td>
<td>2</td>
<td>0.0008, 0.5</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e15</td>
<td>3</td>
<td>0.0001, 4, 3</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>e16</td>
<td>2</td>
<td>0.0002, 0.2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e17</td>
<td>2</td>
<td>0.0002, 0.2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e18</td>
<td>2</td>
<td>0.0003, 0.4</td>
<td>0</td>
<td>2</td>
<td>2, 3</td>
</tr>
</tbody>
</table>
8.4.2 Pre-processing

After the error checking and the pre-processing of the information from the input files, a dependency serial list is established as is shown in figure 8.10.

Since the test dependency has been excluded from the dependency serial list, after the simplification, combination, modularization and re-modularization processes, 8 modules (Module 6001 – Module 6008) are identified from the original fault tree structure as shown in figures 5.9 and 5.10. In module 6008 test dependency exists between basic events e1, e4 and e7.

8.4.3 Quantification

8.4.3.1 Preparation for Quantification

- Conversion of the Fault Tree Structure into BDD

For each factor and module which contains no dependency serial, a BDD will be established based on the corresponding fault tree structure. According the algorithms presented in figure 8.6, the BDDs constructed for factors and modules are shown in figure 8.11:
• **Markov Model Development**

For modules 6005, 6006 and 6008, the Markov method will be employed for quantification. Therefore, the Markov model needs to be established for each of these modules depending on which type of dependency is involved.

From the dependency information given in table 5.8, module 6005 contains dependency serial number 2, which, as is indicated in the serial list in figure 8.10, includes both sequential and maintenance dependencies. According to the algorithms presented in figures 6.7 and 6.17, the corresponding Markov model is established for module 6005 as are represented by table 8.4 and table 8.5 together.
where the first state code of each event represents the component state (0 – working, 1 – failed revealed and 2 – queuing for repair); and the second state code represents the order of the component failure.

Table 8.4 States in the Markov model of module 6005

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>State 2</td>
<td>( \lambda_{18} )</td>
<td>State 5</td>
<td>State 11</td>
<td>( \lambda_{20} )</td>
<td>State 13</td>
<td>State 6</td>
<td>( \psi_{19} )</td>
</tr>
<tr>
<td>State 1</td>
<td>State 3</td>
<td>( \lambda_{19} )</td>
<td>State 6</td>
<td>State 4</td>
<td>( \psi_{18} )</td>
<td>State 14</td>
<td>State 18</td>
<td>( \psi_{19} )</td>
</tr>
<tr>
<td>State 1</td>
<td>State 4</td>
<td>( \lambda_{19} )</td>
<td>State 6</td>
<td>State 12</td>
<td>( \lambda_{19} )</td>
<td>State 15</td>
<td>State 5</td>
<td>( \psi_{20} )</td>
</tr>
<tr>
<td>State 2</td>
<td>State 1</td>
<td>( \psi_{18} )</td>
<td>State 7</td>
<td>State 2</td>
<td>( \psi_{19} )</td>
<td>State 16</td>
<td>State 19</td>
<td>( \psi_{20} )</td>
</tr>
<tr>
<td>State 2</td>
<td>State 5</td>
<td>( \lambda_{19} )</td>
<td>State 7</td>
<td>State 13</td>
<td>( \lambda_{20} )</td>
<td>State 17</td>
<td>State 20</td>
<td>( \lambda_{18} )</td>
</tr>
<tr>
<td>State 2</td>
<td>State 6</td>
<td>( \lambda_{20} )</td>
<td>State 8</td>
<td>State 14</td>
<td>( \lambda_{19} )</td>
<td>State 17</td>
<td>State 4</td>
<td>( \psi_{19} )</td>
</tr>
<tr>
<td>State 3</td>
<td>State 7</td>
<td>( \lambda_{18} )</td>
<td>State 8</td>
<td>State 4</td>
<td>( \psi_{19} )</td>
<td>State 18</td>
<td>State 4</td>
<td>( \psi_{18} )</td>
</tr>
<tr>
<td>State 3</td>
<td>State 1</td>
<td>( \psi_{19} )</td>
<td>State 9</td>
<td>State 15</td>
<td>( \lambda_{19} )</td>
<td>State 18</td>
<td>State 21</td>
<td>( \lambda_{19} )</td>
</tr>
<tr>
<td>State 3</td>
<td>State 8</td>
<td>( \lambda_{20} )</td>
<td>State 9</td>
<td>State 2</td>
<td>( \psi_{20} )</td>
<td>State 19</td>
<td>State 3</td>
<td>( \psi_{18} )</td>
</tr>
<tr>
<td>State 4</td>
<td>State 9</td>
<td>( \lambda_{18} )</td>
<td>State 10</td>
<td>State 16</td>
<td>( \lambda_{18} )</td>
<td>State 19</td>
<td>State 22</td>
<td>( \lambda_{20} )</td>
</tr>
<tr>
<td>State 4</td>
<td>State 10</td>
<td>( \lambda_{19} )</td>
<td>State 10</td>
<td>State 3</td>
<td>( \psi_{20} )</td>
<td>State 20</td>
<td>State 18</td>
<td>( \psi_{19} )</td>
</tr>
<tr>
<td>State 4</td>
<td>State 1</td>
<td>( \psi_{20} )</td>
<td>State 11</td>
<td>State 8</td>
<td>( \psi_{18} )</td>
<td>State 21</td>
<td>State 17</td>
<td>( \psi_{18} )</td>
</tr>
<tr>
<td>State 5</td>
<td>State 3</td>
<td>( \psi_{18} )</td>
<td>State 12</td>
<td>State 17</td>
<td>( \psi_{18} )</td>
<td>State 22</td>
<td>State 8</td>
<td>( \psi_{18} )</td>
</tr>
</tbody>
</table>

Table 8.5 State transitions in the Markov model of module 6005

Module 6006 involves the initiator-enabler dependency with module 6007 representing the initiating event and basic event \( e_{15} \) representing the enabling event. From the dependency file shown in table 8.2, it can be seen that repair policy 2 (only initiating events are rectified) is adopted. In the basic event file in table 8.3, it is indicated by the value of the ‘Enabler’ field that the enabling event \( e_{15} \) can still occur following the corresponding initiating event. Based on these characteristics, the Markov model, shown in figure 8.12, is developed for module 6006 according to the algorithm presented in figure 6.24. The transition from state 5 to state 1 is caused by the rectification of module M6007 and basic event \( e_{15} \) at the same time. In this case, the transition rate is consequently determined using the approximation method illustrated in section 6.3.1, which is obtained as \( \frac{1}{\max(\tau_{M6007}, \tau_{e_{15}})} \), i.e. \( \frac{1}{\max(\frac{1}{\psi_{M6007}}, \frac{1}{\psi_{e_{15}}})} \), equivalent to \( \min(\psi_{M6007}, \psi_{e_{15}}) \).
In figure 8.12, although state 5 and state 7 feature the same combination of states of M6007 and e15, they are different states as the state of the module top event is different. In state 5, the module top event is true because the initiating event M6007 occurs in state 3 in which the enabling event has occurred, whilst state 7 is generated from state 4 in which the initiating event occurs prior to the enabling event.

Module 6008 was identified as the smallest module for the maintenance dependency represented by dependency group 1 in table 8.2. Due to the large size of the resulting Markov model (180 system states and 1008 state transitions), representation of the states and transitions are contained in Appendix A, tables A.1 and A.2 respectively.

When the Markov models have been generated for each module which contains dependency relationship, the rate matrix will be established as is shown in figure 8.7 to facilitate the later quantification.

### 8.4.3.2 The Quantification Process

The lifetime of the example system is 6 years (i.e. 365×6×24 hours), and the time interval by which the quantification will be progressed (dt) is 0.5 hour. With the input of these variables, the quantification will be performed on the example fault tree through all the modules in the way illustrated in figure 8.13 according to the algorithm presented in figure 8.8.
where: the real arrow represents the calling-for relationship between the parent function and its sub-function; the dotted-line arrow represents the data flow.

Figure 8.13 The quantification process through the modules (referring to figure 5.9 and 5.10)

8.4.4 Output
Information has been requested at monthly (24×30=720 hours) interval.

8.4.4.1 System Reliability Predictions
The charts shown in figure 8.14, 8.15 and 8.16 provide diagrammatic summaries of the system unavailability, unconditional failure intensity and expected number of system failures respectively as the time elapses.
Figure 8.14 System Unavailability over a period of 6 years

Figure 8.15 System unconditional failure intensity over a period of 6 years
From figures 8.14 and 8.15, it can be seen that the system unavailability and unconditional failure intensity follow a similar pattern. In figure 8.14, the system unavailability repeats the pattern of increasing and plummeting at a fixed interval of 6 months. Since the system unavailability can be expressed as $Q_{sys}(t) = Q_{M6002}(t) \times Q_{F3002}(t)$, the pattern in the system unavailability is shaped by module 6002 and factor 3002. Figure 8.17 and 8.18 respectively give the failure probability of module 6002 and factor 3002 within the same period.
The pattern in the failure probability of factor 3002 is determined by the dormant failure characteristic of the components represented by basic events e21 and e24. According to equation 7.15, the failure probability of components featuring the dormant failure will increase from zero from the point of time immediately following the last inspection until dropping back to zero at the next one. Since the inspection intervals for components e21 and e24 are both 6 months, this explains the pattern in the failure probability of factor 3002. The failure probability of module 6002 also features a 6-month time cycle, which is partly due to the inspection interval of 2 months and 6 months respectively assigned to components represented by basic events e11 and c6. This differs from Factor 3002, as the failure probability of module 6002 increases over the inspection time period. With Module 6002 and Factor 3002 combined through the AND logic, it generates the pattern in the system unavailability illustrated in figure 8.14.

In terms of the system unconditional failure intensity, it can be expressed as: \( w_{sys}(t) = Q_{M6002}(t) \times W_{F3002}(t) + Q_{F3002}(t) \times W_{M6002}(t) \). Figure 8.19 and 8.20 respectively gives the unconditional failure intensity of module 6002 and factor 3002. It can be concluded that both \( Q_{F3002}(t) \), \( Q_{M6002}(t) \) and \( W_{M6002}(t) \) are characterized by the continuous increase every 6 months. Therefore, although \( W_{F3002}(t) \) features a pattern of the continuous decrease over the 6 months cycle, the combined effect determines that as time elapses, the system unconditional failure intensity increases over the 6-months time cycle as is displayed in figure 8.15.
With regard to the expected number of system failures $W_{sys}(0, T)$, the curve shown in figure 8.16 indicates, as expected, that it continuously increases as the time elapses, which is determined by the definition $W_{sys}(0, T) = \int_0^T w_{sys}(t) dt$. A further investigation into figure 8.16 will reveal that within every 6 months since $t=0$, $W_{sys}(0, t)$ increases at an increasing rate, which can be explained by the increasing $w_{sys}(t)$ over the same period, whilst the rate of increase drops greatly at the beginning of each 6-month interval.
8.4.4.2 Importance Measures of Basic Events
The criticality function of basic events e15, e18 and e19 at any time within 6 years is zero. It means that for component failures represented by these three basic events, there exists no critical state with regard to the combination of the states of other components. This is determined by the contribution that the corresponding component makes to the system failure. Basic event e15 is an enabling event involved in the initiator-enabler dependency. Because the occurrence e15 following the occurrence of the general initiating event gate G18 will never result in the failure of module 6006, there is no state of module 6007 which will satisfy the definition of a critical state for e15. It is the same case with basic events e18 and e19. Due to the specific order of occurrence imposed by the sequential dependency, there exist no such critical state in which the occurrence of e18 or e19 will make module 6005 pass from working to failed.

Also a comparison is carried out between the criticality function of the basic events at different points of time. The comparison is made between the basic event which are the top five in terms of the value of their criticality function, and illustrated in table 8.6. The criticality function of other basic events are of much smaller magnitude.

<table>
<thead>
<tr>
<th>Basic events</th>
<th>Time</th>
<th>18 months</th>
<th>36 months</th>
<th>54 months</th>
<th>72 months</th>
</tr>
</thead>
<tbody>
<tr>
<td>E13</td>
<td>2.3776E-05</td>
<td>3.5092E-05</td>
<td>5.1914E-05</td>
<td>7.6634E-05</td>
<td></td>
</tr>
<tr>
<td>E14</td>
<td>2.3776E-05</td>
<td>3.5092E-05</td>
<td>5.1914E-05</td>
<td>7.6634E-05</td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>5.4467E-04</td>
<td>8.0411E-04</td>
<td>1.1898E-03</td>
<td>1.7565E-03</td>
<td></td>
</tr>
<tr>
<td>E10</td>
<td>1.0925E-03</td>
<td>1.2675E-03</td>
<td>1.5275E-03</td>
<td>1.9097E-03</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.6 Comparison of criticality function between basic events

8.4.5 Contrast with the Case Where Dependencies are Ignored
In this section, the software is applied to the same example fault tree but assuming there is no dependency existing in the system. Then a comparison is carried out between the outputs of these two assessments. The effect on the system reliability prediction of failing to identify the dependency relationships is highlighted by the contrast.

8.4.5.1 Outputs Assuming Independence
Assuming that there is no dependency in the system, the simplification and modularization processes will turn the original fault tree structure into independent factors and modules as are shown in figure 8.21.
Since there is no dependency, all factors and modules will be solved using the BDD. Figure 8.22, 8.23 and 8.24 provides a comparison of system unavailability, system unconditional failure intensity and expected number of system failures between the two different situations.
From figure 8.22, it can be seen that the system unavailability $Q_{sys}(t)$ based on the independence assumption follows the same pattern within each 6-month cycle as the situation where various dependency relationships are recognized. However, the system unavailability where independence is assumed significantly underestimates the more accurate assessment accounting for the dependencies. The maximum value of each cycle also increases as time elapses. This results in a continuously increasing gap in system unavailability between the two situations at the end of each 6-month cycle.

The similar conclusion can be drawn when the comparison is carried out with regard to the system unconditional failure intensity $w_{sys}(t)$ in these two situations. Again assuming independence significantly underestimates the system failure intensity.
The gap between the two curves in figure 8.24 is explained by the difference between the average system unconditional failure intensity $w_{sys}(t)$ in these two situations.

### 8.4.5.2 Comparison at Module/Factor Level

The previous section has illustrated the influence that the dependency relationships existing in the system have on the system reliability assessment as well as the error which can stem from the failure to account for the dependencies during the assessment process. In this section, the comparison is extended to module/factor level to highlight the difference that each type of dependency relationship included in the example can make with regard to the reliability assessment.

- **Initiator-enabler Dependency**

Module 6006 in figure 5.9 and Factor 5005 in figure 8.20 represent the same fault tree section. The initiator-enabler dependency is taken into account in solving M6006, but ignored in the solution of F5005. Figure 8.25, 8.26 and 8.27 respectively illustrate the difference in the module unavailability, module unconditional failure intensity and expected number of module failures.

![Impact of initiator-enabler dependency - $Q_{sys}(t)$](image)

Figure 8.25 Illustration of the impact of initiator-enabler dependency on unavailability
The three-month cycle featured by both module unavailability $Q(t)$ and unconditional failure intensity $w(t)$ in figures 8.25 and 8.26 are caused by the dormant failure of basic event $e_{15}$ with the inspectional interval of 3 months. It can be seen that by taking into account the initiator-enabler dependency existing between the general initiating event represented by gate $G_{18}$ and the enabling event $e_{15}$, the module unavailability, unconditional failure intensity and expected number of failures are dramatically reduced. That is, the failure to account for the initiator-enabler dependency may lead to a considerable over-estimation in the system unavailability. This is especially true when the initiator-enabler dependency exists at a high level in the system.
• **Test Dependency**

The test dependency exists in module 6004 in figure 5.9. The exact counterpart which contains no dependency is module 7001 in figure 8.23. The three minimal cut sets for this module are e3.e1.e2, e4.e1.e2 and e4.e7.e2. Figures 8.28, 8.29 and 8.30 illustrate how the test dependency can influence the system reliability assessment.

![Figure 8.28 Illustration of the impact of test dependency on unavailability](image)

![Figure 8.29 Illustration of the impact of test dependency on unconditional failure intensity](image)
The difference in the module unavailability between the dependent and independent situations mainly lies in the different patterns with regard to the time cycle. In figure 8.28, it can be seen that the system unavailability with no dependency considered, i.e. represented by series 1, fluctuates in a time cycle of 6 months. It can be noticed that in each cycle, the increase within the third month is much smaller than that during the second month. This can be attributed to the dormant failure represented by basic event e2 with an inspection interval of 2 months which can be found in every minimal cut set. The plummet immediately following the end of the third month is explained by the 3-month inspection interval assigned to components whose dormant failures are represented by basic events e1, e4 and e7. When it reaches the end of the 6th month, the inspections are conducted on e2, e1, e4 and e7 at the same time, which explains why the module unavailability takes a abrupt drop. The module unavailability represented by series 2 takes a different pattern which features a time cycle of 12 months, as is displayed in figure 8.28. One difference lies in that the increase of the module unavailability during the 5th month on series 2 is much smaller than that on series 1. The reason for this is due to the extra mutual inspection conducted every 4 months on components represented by basic events e1, e4 and e7. And this extra mutual inspection also explains why the module unavailability at the end of the 6th month on series 2 is significantly lower than that at the same point of time on series 1 (see equation 7.16). Another obvious difference lies in the big drop during the 9th month on series 2 which is again caused by the extra mutual inspection coinciding with the inspection on e2. It can be noticed that this causes a significant gap at the end of the 9th month between the two situations. when it reaches the end of 12th month, all inspections are conducted at the same time revealing all dormant failures and bring the module unavailability down. A new cycle then begins. From figure 8.30, it can be seen that the test dependency also causes a gap in the expected number of module failures.
• Sequential Dependency
Module 6005 in figure 5.9 contains both a maintenance dependency and a sequential dependency. To gain an accurate and correct understanding of how these two types of relationship influence the reliability assessment individually, they should be investigated separately. An assessment is carried out on module 6005 for the same period of time assuming that there only exists the sequential dependency between the three basic events e18, e19 and e20. As is illustrated in figures 8.31, 8.32 and 8.33, a comparison is made with Factor 5002 in figure 8.23 to highlight the effect of sequential dependency.

![Impact of Sequential Dependency - Qsys(t)](image)

Figure 8.31 Illustration of the impact of sequential dependency on unavailability

![Impact of sequential dependency - wsys(t)](image)

Figure 8.32 Illustration of the impact of sequential dependency on unconditional failure intensity
Impact of Sequential Dependency - $W_{sys}(0, t)$

From these figures, it can be seen that the sequential dependency has a significant impact on the system reliability assessment as it results in a much smaller system unavailability, system unconditional failure intensity and expected number of system failures. This is ascribed to the strict occurrence order dictated by the sequential dependency.

- **Maintenance Dependency**

The assessment is carried out on Module 6005 assuming that only maintenance dependency is involved. Figures 8.34, 8.35 and 8.36 illustrates the differences in module unavailability, module unconditional failure intensity and expected number of failures from the situation where no dependency is taken into account.
The contrast made here is opposite to the case with regard to sequential dependency. From the above figures, it can be seen that maintenance dependency results in considerably larger values in the module reliability parameters. This is attributed to the fact that the maintenance dependency results in the increase in the mean time to repair of the relevant components, prolongs the components’ actual down time and thus leads to the increases in the probability that the system resides in a failed state. The maintenance dependency can have a significant effect on the system reliability assessment, especially when the components involved make bigger contributions to the system failure.
In the previous chapter, the process of evaluating the reliability of continuously-running systems is described in detail and illustrated with an example. In this chapter, the system dependency modelling is investigated from the perspective of active-on-demand systems. The static-dynamic two-phase approach is proposed to enable a more efficient analysis of active-on-demand systems. A firewater deluge system is used to illustrate the whole process.

9.1 Dynamic Fault Tree Method and Active-on-demand Systems

Active-on-demand systems have been introduced in chapter 3 to illustrate the application of the dynamic fault tree method. The sprinkler system and the water deluge system in chapter 3 are both typical examples of this type of system, in fact, most safety/protective systems come into this category. Protection systems are fitted on large and complex industrial systems, such as off-shore oil platforms, nuclear power plants and chemical process plants, to prevent potentially hazardous incidents from developing into the catastrophe. For example, a fire accident on an oil platform, if not immediately mitigated, can escalate to an explosion which will claim many lives as well as damage to valuable assets. The difference of this type of system from the continuously-operating systems is that they reside in the inactive/standby state until the incident in question occurs and generates the demand for the safety/protective system to activate. This usually lessens the hazardous impact of the incident, and may require the safety/protective system to function for a certain period of time, during which the repair of failed elements commonly can’t be carried out. Protection system failure during the required functioning time usually leads to the same consequence as being unable to respond in the first place. This means that most safety/protective systems have two failure modes: inactive failure and active failure. In terms of the assessment of the performance of the safety/protective systems, the two failure modes should be investigated separately to give the system unavailability and the system unreliability. The system unavailability $Q_{sys}(t)$ gives the probability that the system is unable to activate at time $t$ when the demand arises, whilst the system unreliability $F_{sys}(T_r)$ gives the probability that the system once started is unable to function through the required period of time $T_r$. 

202
In chapter 3, the dynamic fault tree method has been introduced as a means of performing the evaluation of the active-on-demand systems. Its general process can be summarized as follows:

Step 1. Identify different types of dependency existing in the system

Step 2. Construct the dynamic fault tree according to the system failure logic using the dependency gate structures.

Step 3. Calculate the system unavailability in terms of the failure characteristics of its components.

Step 4. Identify the components which contribute to the system failure whilst active. Establish the initial states for the relevant components, calculate the initial state probabilities and develop the Markov model. Calculate the system unreliability based on the corresponding Markov model.

However, as has been stated in section 3.2.3.2, the dynamic fault tree method has several significant disadvantages. In addition to its complex structure, the dynamic fault tree, in its current form, is unable to represent other types of dependency, such as the maintenance dependency, the test dependency and the revealing dependency. Component failures which are involved in these types of dependency may be well scattered around in the fault tree, and as such it can be very difficult to introduce new types of dependency gate structures to group these relevant basic events. Also, in addressing active-on-demand systems, the dynamic fault tree method investigates the two different system failure modes in one fault tree structure which does not provide a means to distinguish between the different contributions that component failures make to the distinct system failure modes.

9.2 Static-Dynamic Two-Phase Approach

9.2.1 Algorithm

To overcome the shortcomings of the dynamic fault tree method, the static-dynamic two-phase approach is proposed. Its principle is to investigate the failure of active-on-demand systems from an explicit two-phase perspective. The system failure is divided into static failure and dynamic failure, which are considered separately and refer to the system failure to respond to the demand and the system failure to function through the required period of time respectively. Two
conventional fault tree structures, ‘static-phase’ fault tree and ‘dynamic-phase’ fault tree, are established to model the two different system failure modes respectively, and the solution of each of the fault trees gives the prediction on the system unavailability and unreliability respectively. In this way, the effect of the different failure modes of components can be distinguished and component failures need not be considered in the analysis of the phase during which they have no influence and are irrelevant.

9.2.2 The Analysis Process

The general process of applying the static-dynamic two-phase approach to the assessment of active-on-demand systems is illustrated in the flow chart shown in figure 9.1.

![Flow chart of the analysis process](image)

Figure 9.1 The general process of application of static-dynamic two-phase approach to active-on-demand systems
9.2.3 Implementation of the General Process

A program has been developed to implement the static-dynamic two-phase approach in the reliability assessment of active-on-demand systems within the dependency modelling context. The following section describes how the implementation is realized in the program.

9.2.3.1 System Information Input

The assessment of active-on-demand systems requires inputs to define: system failure logic, the components' reliability parameters and the dependency relationships between components. In terms of system failure logic, two ‘fault tree structure files’ need to be established, representing the static system failure and dynamic system failure. These two files will be established in the same way as is described in section 8.3.1. The components’ reliability parameters are included in the ‘basic event file’. The ‘basic event file’ used in the assessment of active-on-demand systems is different from the continuously-operating systems, and features a distinct structure. Some components can be regarded as passive since they are not actually ‘running’, and don’t change the system state as the control components and pressure transmitters do. While for other components, there exist different phases throughout their service, during which they feature passive failures and active failures respectively. Usually the failure parameters also vary for these two different failure modes, which means two different basic events need to be introduced to distinguish the different failure modes. Apart from the different component failure modes, additional information is also required to identify in which system phase the component failure is relevant. The structure of the ‘basic event file’ used in the assessment of active-on-demand systems is displayed in table 9.1.

<table>
<thead>
<tr>
<th>Basic event mode represented</th>
<th>Relevant in phase</th>
<th>Counterpart failure event</th>
<th>Failure model</th>
<th>Parameters</th>
<th>Enabler</th>
<th>Number of dependency groups</th>
<th>List of dependency group numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>P: Passive</td>
<td>S: Static</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A: Active</td>
<td>D: dynamic</td>
<td>B: Both</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9.1 The structure of ‘basic event file’ for active-on-demand systems

In terms of the dependency information, as with continuously-operating systems, two dependency files are established to contain the normal dependency relationships and the functional dependency plus switching dependency relationships respectively. In addition to the information given in table 4.1 in section 4.2, for normal dependency file, an extra column
indicates in which system phase (static or dynamic) the dependency relationship in question is relevant. Functional dependency and switching dependency won't be relevant when the system resides in the inactive state, and only need considering when the system is functioning or when the system is going through the transition from inactive to active.

9.2.3.2 Pre-processing
This part of the process is similar to that in the assessment of continuously-operating systems (see section 8.2.2). Error checking is conducted to ensure the fault tree structure files, the basic event file and the dependency files are correctly related to each other. Two binary trees are constructed according to the fault tree structure files to represent the causes of system static and dynamic failures. Three basic event lists are formed respectively containing basic events which represent component failures relevant in only static system phase, dynamic system phase, or both. Two dependency information lists are established. One of them is established by extracting from the normal dependency file the dependency relationships which only need taking into account in the analysis of static system failure. The other contains the rest of dependency relationships included in the normal dependency file and all the functional and switching dependency relationships from the functional dependency file. Two dependency serial lists are formed, each based on the corresponding dependency information with the exception of the test dependency which are not considered. When extracting information from the functional dependency file, where gate events represent the functionally-controlling event, an expansion is carried out to express this intermediate gate event in terms of basic events which are linked to each other through OR logic. Each of these basic events will be the functionally-controlling event in a newly generated dependency relationship. For example, if an intermediate gate event, which consists of two basic events A and B related to each other through an OR logic (A+B), is involved in a functional dependency group as the functionally-controlling event, then this dependency relationship can be replaced by two new ones in which basic event A and basic event B are the functionally-controlling event against the same dependent component(s).

Then the simplification, combination, modularization and re-modularization processes are conducted on the static-phase fault tree structure in exactly the same way described in section 8.3.2. For the dynamic-phase fault tree, these processes will be implemented with extra factors.
considered. The dynamic-phase fault tree structure investigates the causes of the dynamic system failure, which implies the system has activated successfully on demand. It is therefore necessary to consider the states that the components may reside in when the system starts to function. Components states must be such that the system is able to start. That is, basic events included in the dynamic-phase fault tree which represent component failures relevant to the system static failure must be grouped in the same module. During the simplification process, contraction, extraction and elimination is conducted in the same way as illustrated in chapter 5. But in terms of the factorization, factors should be composed of basic events which are either all related to the system static failure or all relevant in the system dynamic phase. During the following combination, modularization and re-modularization processes, basic events which represent component failures relevant to the system static failure are regarded as from a special 'dependency group' separate from the dependency relationships identified in the normal dependency file. This special 'dependency group' will ensure that all component failures which are relevant to the static system failure are gathered in one module and thus can be considered together to determine the initial states from which the system activates.

9.2.3.3 Quantification

The reliability assessment of active-on-demand systems is performed over a specified period of time, T. The time increment for the calculation, dt, is also required for the quantification process to progress. Both system unavailability $Q_{sys}(t)$ and system unreliability $F_{sys}(T_r)$ are sought, the latter over the required period of operating time $T_r$. System unavailability and unreliability are solved in relation to each other according to the failure logic represented by the fault tree of static failure and dynamic failure respectively.

9.2.3.3.1 Calculation of System Unavailability $Q_{sys}(t)$

The process of calculating the unavailability of active-on-demand systems is the same as is described in section 8.3.3. In the preparation stage, BDDs and Markov models are constructed for modules which feature no dependency serial number and modules which involve more than test dependency respectively. Quantification is then performed on each module in a bottom-up manner to finally obtain the failure probability of the top event, i.e. the system unavailability. Given the frequency $(W_i(t))$ of the event that the system is expected to suppress or mitigate, the
expected number of accidents (due to the system failure to start) during the assessment period \( T \) can be obtained as \( \int \dot{Q}_{sys}(t)w_i(t)dt \).

9.2.3.3.2 Calculation of System Unreliability \( F_{sys}(T_r) \)

The definition of system unreliability, \( F_{sys}(T_r) \), implies that the system has activated successfully at a particular point of time \( t \). As described earlier, there exist two types of modules in the modularized dynamic-phase fault tree. One is the module which includes basic events solely relevant to the system dynamic failure. And the other is the module which includes basic events also related to the system static failure. Different quantification procedures are applied to these two types of modules.

- **System Dynamic Failure**
  Components included in these types of module make no contribution to the system static failure, which implies that they won’t fail during the system static phase. Therefore, for modules of this category, the quantification process is the same as in the system static phase. For modules which need to be addressed using the Markov method, the module starts with all components working with probability 1.

- **General Failure Event**
  Some components included in this type of module are related to the system static failure, which means that they are likely to fail during the system static phase and contribute to the system static failure. When the system activates, the system can do so from different initial states. The quantification of the module has to start with the identification of all the possible initial states defined by the components which are working and failed. This is achieved as follows:

  Step 1. Identify the initial system states which enables the system to activate.
  a) Identify basic events from the module which should be considered with regard to the initial states. Component failures included in the module which are not relevant to the static system failure are not considered. Basic events considered when determining the initial states must fulfil the condition that the basic event is related to a component whose either failure mode is
relevant in the system static phase. The relevant component information can be obtained from the basic event file.

b) Include basic events from static-phase fault tree. Some basic events included in the static-phase fault tree (but not in the dynamic-phase one) represent the failure of components which exist as the functionally-controlling part in corresponding functional and switching dependency relationships. They are relevant to the initial states for the system dynamic phase as their states determine if the functionally-dependent component is able to be activated. Basic events should be added to the list which represent the functionally-controlling components involved in any functional or switching dependency group with basic events from the existing list established in procedure a) as the functionally-dependent part.

c) Trim the list. Basic events which satisfy the following conditions will be deleted from the list.

- Represents functionally-controlling component; AND
- Only one component is functionally dependent on it.

The reason behind this procedure lies in that the state of the functionally-controlling component can be partly reflected by the state of the functionally-dependent component. For example, if component A is functionally dependent on component B, then when A is working or ready to start functioning, it implies that component B must be working as well. When A is failed unusable, it can be caused by the failure of A itself or the failure of B. In that case, the failure of component A and component B will be related to each other through an OR logic with regard to the system failure, the failed state of component A, whatever the cause, would be sufficient to decide the system state. What’s more, by carrying out this trimming procedure, the size of the resulting Markov model can be reduced to its smallest form.

d) Develop the list of initial states based on the chosen basic events. This process starts with the first initial state in which the state code of all basic events is set as ‘0’ representing the component is working. New states are created by changing the state code of each basic event from ‘0’ to ‘1’ (representing the component failed) in turn to create new states from existing states. This process is then applied to new states until the development is completed when no new states are generated. When a new state is generated, it won’t be added to the list if in this state the system is definitely failed and not able to start.
Step 2. Determine the probabilities that the system starts from different initial states.

Consider the general situation where \( n \) components are included in the \( m \) initial states. The state code of each component is denoted as \( s_i \):

\[
s_i = \begin{cases} 
0, & \text{component } i \text{ working} \\
1, & \text{component } i \text{ failed}
\end{cases} \quad i = 1, 2, \ldots, n
\]

\( q_i(t) \) is the probability that component \( i \) is failed at time \( t \), i.e. \( s_i = 1 \).

The initial state \( j \) is a function of the state codes of all the components:

\[
\text{state } j = f_j(s_1, s_2, \ldots, s_n).
\]

\( Q(t) \) is the probability of state \( j \) as an initial state at time \( t \), which means that the system must be available in state \( j \) at time \( t \):

\[
Q(t) = P(\text{the system is available in state } j \text{ AND the components exist in their relevant states represented by state } j)
\]

\[
= P(\text{the system is available } | f_j(s_1, s_2, \ldots, s_n))
\]

\[
\times \prod_{\text{such that } s_i = 1} q_i(t) \times \prod_{\text{such that } s_k = 0} (1 - q_k(t)) \tag{9.1}
\]

where all component states are independent

In equation 9.1, the first condition that 'the system is available in state \( j \)' must be explicitly accounted for. It must excludes all situations where in state \( j \) the system is failed due to the failure of the components which are not included for the consideration of the initial states. The sum of the initial state probabilities at time \( t \) is equal to the system availability at time \( t \).

When the probabilities of all initial states have been determined, the appropriate quantification model for the module is established.

Step 3. Establish the appropriate quantification model for the module.

> Establish BDD for the corresponding module if there is no dependency relationship involved.
Develop the Markov model based on the established initial states.

a) Add basic events to the model which are included in the module as only relevant to the system dynamic failure. For each of the established initial states, the state codes of these added basic events are set as '0'.

b) Develop the Markov model based on algorithms presented in chapter 6 according to the types of dependency involved. If it's the case that repair won't be carried out during the system dynamic phase, all states characterized by failed module state are absorbing, therefore no further development from these states is necessary. Also another technique is proposed aimed at reducing the size of the resulting Markov model. This technique is called 'No Further Influence'. The underlying algorithm is that in states where the module is working, components in the working state don't need to be considered to generate new system states if their failures are unlikely to make any further contribution to the module failure. For example, the module consists of three components linked through module failure logic (A+B).C. In the state with a combination \{A: failed, B: working, C: working\}, a transition caused by the failure of component B, which creates a new state as \{A: failed, B: failed, C: working\}, doesn't need to be included in the model because the failure of component B given component A has failed won't influence the failure conditions of the module failure. With the 'No Further Influence' technique, the size of the resulting Markov model can be dramatically reduced, which will greatly improve the efficiency of the quantification process. Before the development process starts, modules are established centring around each functionally-dependent event involved in functional or switching dependency relationships. The established modules include all the functionally-controlling events which are exclusive to the particular dependent component as well as the dependent event itself. When the module is failed, the dependent component is unusable or inaccessible.

Step 4: Module quantification

Quantification in BDDs

The system can start from any of the \( m \) initial states \( f(s_1, s_2, \ldots, s_n) \), \( i = 1, 2, \ldots, m \). Starting from each different initial state will result in a different module unreliability. For each initial state \( f(s_1, s_2, \ldots, s_n) \), a conditional module unreliability \( F_i(T_r) \) is obtained where:
\[ F_r(T_r) = P(\text{the module is unreliable over } T_r \mid s_1, s_2, \ldots, s_n) \]

where \( T_r \) is measured from the time of system activation.

The system unconditional unreliability is then obtained by:

\[ F_{sys}(T_r) = \sum_{i=1}^{n} F_r(T_r)Q_i \]

where \( Q_i \) refers to the probability of initial state \( i \) at the time of system activation.

- Quantification in the Markov model: the quantification is carried out through the established rate matrix.

When repair is not available during the system dynamic phase, the system unreliability over \( T_r \), \( F_{sys}(T_r) \), is equal to the system unavailability at the end of \( T_r \) from the system activation time. Since it is assumed that the failure to function for the required period of time \( T_r \) will bring about the same consequence as failure to activate, being unreliable also makes a contribution to the expected number of accidents by \( \int_0^{T_r} F_{sys}(T_r) \cdot w_i(t) dt \), where \( F_{sys}(T_r) \) stands for the system unreliability over \( T_r \) from the system activation time \( t \).

9.2.3.4 Calculation of Importance Measures

9.2.3.4.1 Importance Measures in the System Static Phase

The criticality function of each component \( G_i(q(t)) \) is calculated as an indicator of the component's contribution to the system static failure. The algorithm applied for this is as illustrated in section 7.1.6.

9.2.3.4.2 Importance Measures in the System Dynamic Phase

- Importance Measure for Initiating Events

In addition to the criticality function, the Barlow-Proshan measure of initiator importance (B-P measure) can also provide useful information indicating the contribution of each initiating event to the system failure. It is defined as the probability that initiating event \( i \) causes a system failure.
during the interval given the system is unreliable over the interval \([0, t)\). The B-P measure is expressed by:

\[
BP_i = \frac{\int \{Q(l_i, q(u)) - Q(0, q(u))\} w_i(u) du}{W(0, t)}
\]

9.3

The numerator gives the expected number of system failures during the interval due to the occurrence of initiating event \(i\) and is weighted by the total number of system failures during the same interval. This indicates the contribution that the initiating event \(i\) makes to the system unreliability during \([0, t)\). For active-on-demand systems, the period of time, \(T_n\), required for the system to effectively mitigate the incident is used as the time interval in equation 9.3.

The Barlow-Proschan measure evaluates the importance of the individual initiating event over a certain period of time. In this sense, it is a concept of time flow. To acquire an idea of how the failure of each individual component can contribute to the system failure as time elapses (i.e. at discrete points of time), the derivation of the numerator in equation 9.3 can be used, i.e. \(\{Q(l_i, q(t)) - Q(0, q(t))\} w_i(t)\).

When a B-P measure for the initiating event \(i\) is calculated in a Markov model, equation 9.3 can be transformed to enable a more efficient calculation. The transformation is carried out as follows:

\[
BP_i = \frac{\int G_i(q(u)). w_i(u) du}{W(0, t)} \\
= \frac{\int G_i(q(u)). [1 - Q_i(u)] \lambda_i du}{W(0, t)} \\
= \frac{\int \sum Q_j(u). \lambda_i du}{W(0, t)}
\]

9.4

where in state \(j\), the combination of the states of the other components form the critical state for the initiating event \(i\), thus the occurrence of the event \(i\) (with the conditional failure rate \(\lambda_i\)) will bring about the system failure.
Importance Measure for Enabling Events

For components whose failures can be defined as an enabling event, that is, the failure of the component can never immediately bring the system from working to failed state, the Barlow-Proshchan measure is zero as the criticality function of the enabling events is always zero. To correctly reflect the contribution that enabling events make to the system unreliability, an importance measure for enabling events is introduced. As is defined in [1], one measure of importance for the enabling event \( m \) at time \( t \) is expressed as:

\[
I_m(t) = \frac{\sum_{j=1}^{n_i} \sum_{k=1}^{n_m} P\left( \bigcup_{j} E_{j,k} \right) \cdot w_j(t)}{\sum_{i=1}^{n_i} \sum_{k=1}^{n_m} P\left( \bigcup_i E_{i,k} \right) \cdot w_i(t)}
\]

9.5

where:
- \( w_i(t) \) and \( w_j(t) \) represent the failure frequency (unconditional failure intensity) of initiating event \( i \) and \( j \) at time \( t \) respectively.
- \( E_{i,k} \) represents the event that a minimal cut set \( k \) containing the initiating event \( i \) occurs with event \( i \) removed
- \( E_{j,k} \) represents the event that a minimal cut set \( k \) containing both the initiating event \( j \) and the enabling event \( m \) occurs with initiating event \( j \) removed
- \( n_m \) represents the number of initiating events which are included in the same minimal cut sets with the enabling event \( m \)
- \( n_i \) represents the total number of initiating events in the system

The importance measure expressed in equation 9.5 represents a value at discrete points of time. To enable the measure to provide an indication of the contribution that each enabling event makes to the system unreliability in the dynamic phase of active-on-demand systems, the integration process is applied to both numerator and denominator in equation 9.5. This introduces the importance measure for the enabling event \( m \) in terms of the system unreliability over a certain period of time \( T_r \). This is shown in equation 9.6.
The importance measure for the enabling event in equation 9.6 is very similar to the Barlow-Proshan measure in equation 9.3. As \( P \left( \bigcup_{i \in k} E_{i,k} \right) \) is equal to the criticality function \( G_i(q(t)) \) of the initiating event \( i \), the denominator in equation 9.6 actually represents the expected number of system failures \( W(0, T_r) \), same as in the Barlow-Proshan measure. Therefore, if the initiating event \( i \) and the enabling event \( m \) always appear together in the minimal cut sets, which means that the numerator in equation 9.6 is the same as that in equation 9.3 (i.e. \( \sum_{j=1}^{n_s} P \left( \bigcup_{i \in k} E_{j,k} \right) = G_j(q(t)) \)), the Barlow-Proshan measure for the initiating event \( i \) and the importance measure \( I_m^E \) for the enabling event \( m \) will have the same value.

If the Markov model is adopted for the system dynamic-phase analysis, a transformation of equation 9.6 is required to enable the calculation of the importance measure for the enabling event without having to obtain the minimal cut sets. The transformation is shown in equation 9.7.

\[
I_m^E = \frac{\int_0^{t_r} \sum_{j=1}^{n_m} \sum_{k=1}^{m} Q_{k}(t) \lambda_j dt}{W(0, T_r)}
\]

where:
- \( \lambda_j \) represents the conditional failure rate of the initiating event \( j \)
- \( Q_k(t) \) represents the probability of state \( k \) at time \( t \) in which the enabling event \( m \) has occurred, and the initiating event \( j \) has not occurred, and the system state is a critical state for the initiating event \( j \).
- \( n_m \) represents the number of initiating events \( j \) which is involved in the same initiator-enabler dependency relationship with the enabling event \( m \).
9.2.3.5 Output

Three files are generated by the program to store the results of the reliability assessment carried out on the active-on-demand system. Two files are used to store the importance measures of relevant components respectively for the system static and dynamic phase. And the other file gives the predictions of system failures measured in both unavailability and unreliability.

9.3 Application of the Static-dynamic Two-phase Approach to the Firewater Deluge System

In this section, the firewater deluge system (FDS) fitted on an off-shore oil platform, which is a typical example of an active-on-demand system, is used to illustrate how the static-dynamic two-phase approach can be applied.

9.3.1 Description of the Firewater Deluge System

The basic features of the system are shown in figure 9.2. Its function is to supply, on demand when a fire occurs, water and foam at a controlled pressure to a specific area on the platform protected by a deluge system. As such, the FDS is composed of a fire sensing system, a deluge skid, firewater pumps, associated equipment and ringmains and Aqueous Film-Forming Foam (AFFF) pumps, associated equipment and ringmains. Each subsystem is described in more detail in the following sections.

9.3.1.1 The Fire Detection System and the Deluge System

The fire detection system consists of three fire and smoke sensors, which will send signal to the Main Fire and Gas Panel (MFGP) when a fire occurs. The deluge valve set including all associated equipment is mounted on a fabricated steel framework called a skid. Skids are situated on the processing platform where an incident can occur and its associated equipment act to spray water onto the affected area. The deluge valve set is composed of three main elements: the main distribution line, a water closing circuit and a control air circuit. Upon receiving a signal from the Main Fire and Gas Panel, the solenoid valves (SV1/2) are de-energised and open thus releasing air pressure from the control air circuit. The air pressure drop allows the valmatic release valve to open, and water from the water closing circuit runs to drain. This causes the pressure on the deluge valve diaphragm to fall. When the pressure on the diaphragm has fallen sufficiently, the firewater main pressure acting on the underside of the deluge valve clack
overcomes the load imposed by the diaphragm, allowing water to flow into the distribution pipes, through the nozzles and onto the hazard.

The deluge valve set is also fitted with an AFFF supply line. Instrument air pressure maintains the valmatic release valve and AFFF valve closed. When the air pressure drops in the control air circuit, due to the solenoid valves being de-energised (the same components as those used to activate the water deluge valve), the AFFF valve and valmatic release valve open simultaneously. As the water flows through the foam inductor in the main distribution line, foam concentrate is induced from the AFFF line via the foam proportioner. The solution of water and approximately 3% foam then feed into the distribution network, through the nozzles and onto the hazard.

9.3.1.2 Firewater Supply and Distribution System
The deluge systems are connected to a pressurized ringmain network. The ringmain pressure is maintained by a jockey pump drawing water from the sea. Falling pressure is detected by the pressure transmitters, which subsequently send a signal to the MFGP. In turn, the MFGP activates the firewater pumps to supply water direct from the sea at sufficient pressure to meet the deluge requirements. Pumps not needed remain in inactive standby.

The fire pumps are arranged in two sets, each including two pumps. one set is powered by electric supplies from the main electric power plant. One electric supply serves as the standby power. The other set is powered by their own dedicated diesel engines. The diesels have a day tank sized for a 24 hour supply with a low-level alarm fitted.

9.3.1.3 AFFF Supply and Distribution
The foam concentrate is stored in a stainless steel tank and is distributed through a stainless steel ringmain network. The tank has a low level alarm fitted. The foam system is kept at approximately the same pressure as the firewater system by a continuously running air driven jockey pump. Two AFFF pumps are fitted, with one motor driven power supplied from the platform power plant, and the other diesel driven but separate from the diesel supply to the firewater diesel pumps.
Figure 9.2 The Firewater Deluge System
When a fire is detected, the AFFF deluge valve opens and the pressure drops in the AFFF ringmain. The three pressure transmitters detect the pressure drop and send a signal to the AFFF control panel which activates the AFFF pump to supply the foam to the inductor nozzle. The pump not needed will remain in standby.

Additional system features and assumptions:

- The two electric firewater pumps and the electric AFFF pump are set as the duty pumps.
- 2 out-of-4 firewater pressure sensors and 2-out-of-3 AFFF pressure sensors are required to work to effectively detect the falling pressure.
- The distribution line on each firewater pump is identical, including a filter, two isolation valves, one check valve, one test valve and one pressure relief valve. The distribution line on each AFFF pump is the same as that of the firewater pump except the filter.
- System maintenance:
  - Each element of the fire detection system (consisting of 3 fire/smoke sensors), the firewater pressure sensing system (consisting of 4 pressure transmitters) and AFFF pressure sensing system (consisting of 3 pressure sensors) is maintained by one engineer.
  - The isolation valves IV1, IV2, IV12 and Iv13 and check valve CV5 as indicated in figure 9.2 are maintained by two engineers.
  - The inductor nozzle, the strainer and the spray nozzle are maintained by one engineer.
  - The water deluge valve, the AFFF deluge valve and the valmatic relief valve are maintained by one engineer.
  - The two solenoid valves are maintained by one engineer.
  - The two electric supplies to firewater electric pumps are maintained by one engineer.
  - Each of the valve sets on the four firewater pump streams (consisting of one filter, two isolation valves and one check valve) is maintained by one engineer.
  - The valve set on the AFFF electric pump stream (consisting of two isolation valves and one check valve) is maintained by one engineer.
- The valve set on the AFFF diesel pump stream (consisting of two isolation valves and one check valve) plus the isolation valve IV14 (the isolation valve located on the output of the AFFF diesel tank) are maintained by one engineer.
- All the isolation valves are tested every 6 months. All the check valves and filters are tested every 2 months. Each pump stream is tested through the corresponding test valve every 3 months.

• Human errors are ignored
• The repair won’t be carried out during the time the system is functioning
• The isolation valves, check valves and deluge valves won’t fail blocked when there is fluid flowing through them, i.e. the system has been activated.

9.3.2 Reliability Assessment of the Firewater Deluge System

9.3.2.1 System Failure Information

9.3.2.1.1 Construction of Fault Trees

• Static System Failure Fault Tree

The top event of this fault tree is defined as 'Firewater Deluge System Fails to Activate on Demand'. This top event will occur if either the firewater or AFFF pump mechanism fails to activate. Failure of these functions can be attributed to two causes: the control subsystem fails to activate the pump systems OR the firewater or AFFF pump subsystem themselves fail to activate when the need is detected, as is indicated in figure 9.3. The fault tree representing the static system failure is developed for each of these sub-events in figure 9.3.

The control subsystem fails to activate the firewater pumps

The firewater pumps will not activate if the main fire and gas panel itself fails to send the start-up signal or no pressure drop in the ringmain is detected. This occurs if the firewater pressure sensors fail to detect the falling pressure or the pressure doesn’t drop in the firewater ringmain. The latter event will occur when the water deluge valve does not open or the terminal distribution line is blocked. The water deluge valve cannot be opened either because the valve itself fails or because the pressure on the deluge valve diaphragm doesn’t fall. The latter event
can be further traced down to the failures of fire sensing system, the solenoid valves or the valmatic relief valve. Table 9.2 gives a summary of the events considered.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Event description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPF</td>
<td>Main fire and gas panel fails to send signal to firewater pumps</td>
</tr>
<tr>
<td>WPS1F – WPS4F</td>
<td>Firewater pressure sensors 1 – 4 fails to detect the pressure drop in the firewater ringmain</td>
</tr>
<tr>
<td>IV1 – IV2</td>
<td>Blockage of the locked open butterfly valves (Iso.v1 - Iso.v2)</td>
</tr>
<tr>
<td>INB</td>
<td>Blockage of inductor nozzle</td>
</tr>
<tr>
<td>SB</td>
<td>Blockage of strainer</td>
</tr>
<tr>
<td>NB</td>
<td>Blockage of spray nozzle</td>
</tr>
<tr>
<td>WDV</td>
<td>Firewater deluge valve (deluge v1) fails to open</td>
</tr>
<tr>
<td>WVS</td>
<td>Valmatic relief valve sticks closed on activation</td>
</tr>
<tr>
<td>SV1 – SV2</td>
<td>Solenoid activated valve fails to dump instrument air on receipt of the signal from MFGP</td>
</tr>
<tr>
<td>FS1F – FS3F</td>
<td>Fire and smoke sensors fail to detect the fire</td>
</tr>
</tbody>
</table>

Table 9.2 Events involved in the failure to activate the firewater pumps

The firewater pump system fails to activate

When 3-out-of-4 firewater pump streams do not activate, the firewater pump subsystem fails on demand. For each firewater pump stream, activation will not occur when the pump fails or the associated pipework and valve distribution line is blocked. Events considered are listed in table 9.3.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Event description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIF – E2F</td>
<td>Electric firewater pump 1 or 2 fails in standby</td>
</tr>
<tr>
<td>ESI – ES2</td>
<td>Electric power supply to electric firewater pumps fails</td>
</tr>
<tr>
<td>DIF – D2F</td>
<td>Diesel firewater pump 1 or 2 fails in standby</td>
</tr>
<tr>
<td>DS</td>
<td>Diesel supply to diesel firewater pumps fails</td>
</tr>
<tr>
<td>FB1 – FB4</td>
<td>Filter 1 – filter 4 blocked</td>
</tr>
<tr>
<td>IV3 – IV11</td>
<td>Isolation valve 3 – isolation valve 11 blocked</td>
</tr>
<tr>
<td>CV1 – CV4</td>
<td>Check valve 1 – 4 blocked</td>
</tr>
</tbody>
</table>

Table 9.3 Events involved in the static failure of firewater pump subsystem

The control subsystem fails to activate the AFFF pump

The causes of failure to activate the AFFF pump system are similar to these discussed above with regard to the failure to activate the firewater pump system. Events considered here are given in table 9.4.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Event description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP</td>
<td>AFFF panel fails to send signal to AFFF pumps</td>
</tr>
<tr>
<td>APS1F – APS4F</td>
<td>AFFF pressure sensors 1 – 3 fails to detect the pressure drop in the AFFF ringmain</td>
</tr>
<tr>
<td>IV12</td>
<td>Blockage of the locked open butterfly valve (Iso.v12)</td>
</tr>
<tr>
<td>CV5</td>
<td>Blockage of check valve 5</td>
</tr>
<tr>
<td>INB</td>
<td>Blockage of inductor nozzle</td>
</tr>
<tr>
<td>SB</td>
<td>Blockage of strainer</td>
</tr>
<tr>
<td>NB</td>
<td>Blockage of spray nozzle</td>
</tr>
<tr>
<td>ADV</td>
<td>AFFF deluge valve (deluge v2) fails to open</td>
</tr>
<tr>
<td>SV1 – SV2</td>
<td>Solenoid activated valve fails to dump instrument air on receipt of the signal</td>
</tr>
</tbody>
</table>
**The AFFF pump system fails to activate**

The AFFF pump system fails to activate if there is insufficient foam in the AFFF tank or both the AFFF electric pump stream and the AFFF diesel pump stream fail in standby. Events considered regarding the failure of AFFF pump streams are given in table 9.5.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Event description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEF</td>
<td>Electric AFFF pump fails in standby</td>
</tr>
<tr>
<td>AES</td>
<td>Electric power supply to electric AFFF pump fails</td>
</tr>
<tr>
<td>ADF</td>
<td>Diesel AFFF pump fails in standby</td>
</tr>
<tr>
<td>ADS</td>
<td>Diesel supply to diesel AFFF pump fails</td>
</tr>
<tr>
<td>IV13-IV18</td>
<td>Isolation valve 13 – isolation valve 18 blocked</td>
</tr>
<tr>
<td>CV6-CV7</td>
<td>Check valve 6 – 7 blocked</td>
</tr>
<tr>
<td>INAF</td>
<td>AFFF supply fails</td>
</tr>
</tbody>
</table>

Table 9.5 Events relevant to the static failure of AFFF pump system

![Fault Tree Diagram](image-url)
Figure 9.3b Static-phase fault tree for the failure of the control sub-system to activate the firewater pumps

[Diagram showing the fault tree with various nodes and edges detailing the failure conditions and their causes.]

223
Figure 9.3c Static-phase fault tree for the failure of firewater pump streams 1 and 2
Figure 9.3d Static-phase fault tree for the failure of firewater pump streams 3 and 4
Figure 9.3e Static-phase fault tree for the failure of the control sub-system to activate the AFFF pumps
AFFF pump system fails in standby

G29

AFFF pump stream 1 fails in standby

G30

Electric power supply 3 fails

AES

Electric AFFF pump fails in standby

AEF

AFFF distribution line 1 blocked

Isolation valve 15 blocked

IV15

Isolation valve 16 blocked

IV16

Check valve 6 blocked

CV6

Diesel supply to AFFF pump fails

G31

G33

Isolation valve 14 blocked

IV14

AFFF diesel supply insufficient

ADS

AFFF diesel pump fails in standby

G32

G34

Isolation valve 17 blocked

IV17

Isolation valve 18 blocked

IV18

Check valve 7 blocked

CV7

Insufficient AFFF from AFFF tank

G35

Insufficient AFFF on demand

IV13

INAF

Figure 9.3f Static-phase fault tree for the failure of AFFF pump system
• Dynamic System Failure Fault Tree

The top event of this fault tree is defined as ‘Firewater Deluge System Fails to Function throughout the Required Period’. The top event occurs when either the firewater sub-system or the AFFF sub-system fails during the required functioning period. The direct cause of the dynamic failure of the firewater sub-system is identified as less than two firewater pump streams working. In order for the firewater pump stream to fail once activated can be ascribed to two causes: the pump fails or the pressure relief valve opens under normal pressure which means the water is delivered to the firewater ringmain at a pressure lower than required. The direct cause of the dynamic failure of the AFFF sub-system can be developed in a similar way. Events considered with regard to the dynamic failure of the firewater deluge system are given in table 9.6.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Event description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1D – E2D</td>
<td>Electric firewater pump fails when functioning</td>
</tr>
<tr>
<td>ES1 – ES2</td>
<td>Electric power supply to electric firewater pumps fails</td>
</tr>
<tr>
<td>D1D – D2D</td>
<td>Diesel firewater pump fails when functioning</td>
</tr>
<tr>
<td>DS</td>
<td>Diesel supply to diesel firewater pumps fails</td>
</tr>
<tr>
<td>AED</td>
<td>Electric AFFF pump fails when functioning</td>
</tr>
<tr>
<td>AES</td>
<td>Electric power supply to electric AFFF pump fails</td>
</tr>
<tr>
<td>ADD</td>
<td>Diesel AFFF pump fails when functioning</td>
</tr>
<tr>
<td>ADS</td>
<td>Diesel supply to diesel AFFF pump fails</td>
</tr>
<tr>
<td>PRV1 – PRV6</td>
<td>Pressure relief valves 1 – 6 fails open under normal pressure</td>
</tr>
<tr>
<td>INAF</td>
<td>AFFF supply fails</td>
</tr>
</tbody>
</table>

Table 9.6 Events considered as the direct causes to the dynamic failure of firewater deluge system

The construction of the fault tree representing the dynamic system failure is accounts for the use of redundancy in the design of both firewater and AFFF sub-systems. The redundancy means that the system can activate with different starting configurations. For example, the system can start with two electric firewater pumps and one electric AFFF pump working whilst all the diesel pumps remain as standby. Alternatively it can start with two diesel firewater pumps and the diesel AFFF pump working because all electric pumps have already failed in standby. In the first situation, the Main Fire and Gas Panel, the firewater pressure sensing system, the AFFF control panel and the AFFF pressure sensing system can all contribute to the system failure since they would be required to start the standby pumps in the event that the duty pumps fail. However, in the second situation, the control sub-systems no longer contribute to the dynamic system failure. It is therefore very difficult to explicitly include the control sub-system failure in the fault tree structure representing the dynamic system failure. This is achieved by accounting for the failure...
of control sub-systems by considering the switching dependency relationship existing between them and the standby pumps. The resulting fault tree which represents the dynamic system failure is displayed in figure 9.4.

Figure 9.4b Dynamic-phase fault tree for the failure of the AFFF system
The firewater deluge system fails to function throughout the required period

The firewater subsystem fails to function through the required period

The AFFF subsystem fails to function through the required period

The firewater pump stream 1 fails during functioning

The firewater pump stream 2 fails during functioning

The firewater pump stream 3 fails during functioning

The firewater pump stream 4 fails during functioning

Figure 9.4a Dynamic-phase fault tree for the failure of the firewater pump system
9.3.2.1.2 Dependency Files

When establishing the dependency files, the static and dynamic system phase are investigated separately to identify any dependency relationships existing in the system.

- Establishment of the normal dependency file
  - Static phase: according to the functional characteristics of components included in the system and the system maintenance features, there exist two types of dependency relationship relevant to the static phase — maintenance dependency and test dependency. The latter is due to the inspections carried out on each pump stream.
  - Dynamic phase: the standby dependency exists in the dynamic system phase between the electric firewater pumps and the diesel pumps in the form of a pooled spare, and also between the electric AFFF pump and the diesel AFFF pump. Since the repair won't be conducted during the dynamic phase, there are no dependency relationships related to the repair process. This includes maintenance dependency, secondary failure dependency, test dependency and revealing dependency.

Accordingly, the normal dependency file is established and shown in table 9.7 corresponding to the fault tree structures in figures 9.3 and 9.4.

<table>
<thead>
<tr>
<th>Dependency group number</th>
<th>Dependency type</th>
<th>Number 1</th>
<th>Number 2</th>
<th>List 1</th>
<th>List 2</th>
<th>Relevancy in which phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mtnc</td>
<td>4</td>
<td>1</td>
<td>WPS1F, WPS2F, WPS3F, WPS4F</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>2</td>
<td>mtnc</td>
<td>3</td>
<td>1</td>
<td>APS1F, APS2F, APS3F</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>3</td>
<td>mtnc</td>
<td>3</td>
<td>1</td>
<td>FS1F, FS2F, FS3F</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>4</td>
<td>mtnc</td>
<td>5</td>
<td>2</td>
<td>IV1, IV2, IV12, IV13, CV5</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>5</td>
<td>mtnc</td>
<td>3</td>
<td>1</td>
<td>INB, SB, NB</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>6</td>
<td>mtnc</td>
<td>3</td>
<td>1</td>
<td>WDV, WVS, ADV</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>7</td>
<td>mtnc</td>
<td>2</td>
<td>1</td>
<td>SV1, SV2</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>8</td>
<td>mtnc</td>
<td>2</td>
<td>1</td>
<td>ES1, ES2</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>9</td>
<td>mtnc</td>
<td>4</td>
<td>1</td>
<td>FB1, IV3, IV4, CV1</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>10</td>
<td>mtnc</td>
<td>4</td>
<td>1</td>
<td>FB2, IV5, IV6, CV2</td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td>11</td>
<td>mtnc</td>
<td>4</td>
<td>1</td>
<td>FB3, IV7, IV8, CV3</td>
<td></td>
<td>Static</td>
</tr>
</tbody>
</table>
Table 9.7 The normal dependency file for the firewater deluge system

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dynamic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

• *Establishment of the functional dependency file*

It's evident that functional dependency exists between the electric power supplies and the electric pumps, and between the diesel supplies and the diesel pumps. Switching dependency exists between the firewater pumps and the main fire and gas panel, the firewater pressure sensing system plus the corresponding pump distribution lines. Similar dependencies exist between the AFFF pumps and the AFFF control panel, the AFFF pressure sensing system and the corresponding pump distribution lines. It should be noted that there is a special dependency relationship between the pump and the pressure relief valve on each pump stream. The pressure relief valve is assumed not to fail open when the pump stream is not activated. It therefore will not contribute to causes of the pump not activating successfully. It can however fail open under normal pressure, thus failing the functionality provided by the corresponding pump. Figure 9.5 illustrates the different types of functional dependency by distinguishing effects in terms of the states of the functionally-dependent component.
Limited functional dependency implies that the functionally-controlling component will not fail when the functionally-dependent component resides in the inactive state. The dependency relationship between the pressure relief valve and the pump falls into this category. The limited functional dependency is denoted as ‘ldfunc’ in the functional dependency file.

Table 9.8 gives the functional and switching dependency relationships existing in the system.

<table>
<thead>
<tr>
<th>Dependency group number</th>
<th>Dependency type</th>
<th>Number 1</th>
<th>Number 2</th>
<th>List 1</th>
<th>List 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>func</td>
<td>1</td>
<td>2</td>
<td>G13</td>
<td>E1D, E2D</td>
</tr>
<tr>
<td>24</td>
<td>func</td>
<td>1</td>
<td>2</td>
<td>D8</td>
<td>D1D, D2D</td>
</tr>
<tr>
<td>25</td>
<td>func</td>
<td>1</td>
<td>1</td>
<td>AES</td>
<td>AED</td>
</tr>
<tr>
<td>26</td>
<td>func</td>
<td>1</td>
<td>1</td>
<td>ADS</td>
<td>ADD</td>
</tr>
<tr>
<td>27</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV1</td>
<td>E1D</td>
</tr>
<tr>
<td>28</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV2</td>
<td>E2D</td>
</tr>
<tr>
<td>29</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV3</td>
<td>D1D</td>
</tr>
<tr>
<td>30</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV4</td>
<td>D2D</td>
</tr>
<tr>
<td>31</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV5</td>
<td>AED</td>
</tr>
<tr>
<td>32</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV6</td>
<td>ADD</td>
</tr>
<tr>
<td>33</td>
<td>swch</td>
<td>1</td>
<td>4</td>
<td>MPF</td>
<td>E1D, E2D, D1D, D2D</td>
</tr>
<tr>
<td>34</td>
<td>swch</td>
<td>1</td>
<td>4</td>
<td>G5</td>
<td>E1D, E2D, D1D, D2D</td>
</tr>
<tr>
<td>35</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>G14</td>
<td>E1D</td>
</tr>
<tr>
<td>36</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>G16</td>
<td>E2D</td>
</tr>
<tr>
<td>37</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>G19</td>
<td>D1D</td>
</tr>
<tr>
<td>38</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>G21</td>
<td>D2D</td>
</tr>
<tr>
<td>39</td>
<td>swch</td>
<td>1</td>
<td>2</td>
<td>IV11</td>
<td>D1D, D2D</td>
</tr>
<tr>
<td>40</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>G31</td>
<td>AED</td>
</tr>
<tr>
<td>41</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>G34</td>
<td>ADD</td>
</tr>
<tr>
<td>42</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>IV14</td>
<td>ADD</td>
</tr>
<tr>
<td>43</td>
<td>swch</td>
<td>1</td>
<td>2</td>
<td>APF</td>
<td>AED, ADD</td>
</tr>
<tr>
<td>44</td>
<td>swch</td>
<td>1</td>
<td>2</td>
<td>G24</td>
<td>AED, ADD</td>
</tr>
</tbody>
</table>

Table 9.8 The functional dependency file for the firewater deluge system

9.3.2.1.3 Basic Event File

For basic events included in the two fault tree structures shown in figure 9.3 and 9.4, the corresponding basic event file is shown in table 9.9, in the structure displayed in table 9.1. Since the initiator-enabler dependency does not exist in the system, the column ‘enabler’ is not included.
<table>
<thead>
<tr>
<th>Basic event name</th>
<th>Failure mode represented</th>
<th>Relevant in phase</th>
<th>Counterpart failure event</th>
<th>Failure model</th>
<th>Parameters</th>
<th>Number of dependency groups</th>
<th>List of dependency group numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>WPS1F</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000004, 5, 3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WPS2F</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000004, 5, 3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WPS3F</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000004, 5, 3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WPS4F</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000004, 5, 3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IV1</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.000014, 4, 6</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>IV2</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.000014, 4, 6</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>INB</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.000003, 4, 3</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>SB</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.000003, 4, 3</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>NB</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.0000027, 2, 3</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>WDV</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.000025, 4, 3</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>WVS</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.000003, 6, 3</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>SV1</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.00000083, 8, 3</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>SV2</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.00000083, 8, 3</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>FS1F</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.0000008, 8, 3</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>FS2F</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.0000008, 8, 3</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>FS3F</td>
<td>P S</td>
<td>-</td>
<td>3</td>
<td>0.0000008, 8, 3</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>MPF</td>
<td>P B</td>
<td>-</td>
<td>1</td>
<td>0.0001, 0.0002</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ES1</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000007, 3, 3</td>
<td>3</td>
<td>8, 15, 16</td>
<td></td>
</tr>
<tr>
<td>ES2</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000007, 3, 3</td>
<td>3</td>
<td>8, 15, 16</td>
<td></td>
</tr>
<tr>
<td>EIF</td>
<td>P S</td>
<td>E1D</td>
<td>3</td>
<td>0.000016, 10, 3</td>
<td>1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>E2F</td>
<td>P S</td>
<td>E2D</td>
<td>3</td>
<td>0.000016, 10, 3</td>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>D1F</td>
<td>P S</td>
<td>D1D</td>
<td>3</td>
<td>0.00005, 12, 3</td>
<td>1</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>D2F</td>
<td>P S</td>
<td>D2D</td>
<td>3</td>
<td>0.00005, 12, 3</td>
<td>1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>DS</td>
<td>P B</td>
<td>-</td>
<td>1</td>
<td>0.0002, 0.00001</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FB1</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000015, 2, 2</td>
<td>2</td>
<td>9, 15</td>
<td></td>
</tr>
<tr>
<td>FB2</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000015, 2, 2</td>
<td>2</td>
<td>10, 16</td>
<td></td>
</tr>
<tr>
<td>FB3</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000015, 2, 2</td>
<td>2</td>
<td>11, 17</td>
<td></td>
</tr>
<tr>
<td>FB4</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000015, 2, 2</td>
<td>2</td>
<td>12, 18</td>
<td></td>
</tr>
<tr>
<td>IV3</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000014, 4, 6</td>
<td>2</td>
<td>9, 15</td>
<td></td>
</tr>
<tr>
<td>IV4</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000014, 4, 6</td>
<td>2</td>
<td>9, 15</td>
<td></td>
</tr>
<tr>
<td>IV5</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000014, 4, 6</td>
<td>2</td>
<td>10, 16</td>
<td></td>
</tr>
<tr>
<td>IV6</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000014, 4, 6</td>
<td>2</td>
<td>10, 16</td>
<td></td>
</tr>
<tr>
<td>IV7</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000014, 4, 6</td>
<td>2</td>
<td>11, 17</td>
<td></td>
</tr>
<tr>
<td>IV8</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000014, 4, 6</td>
<td>2</td>
<td>11, 17</td>
<td></td>
</tr>
<tr>
<td>IV9</td>
<td>P B</td>
<td>-</td>
<td>3</td>
<td>0.000014, 4, 6</td>
<td>2</td>
<td>12, 18</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV01</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.00000014, 4, 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV02</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.00000009, 5, 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV01</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.0000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV02</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.0000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV03</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.0000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV04</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.000000014, 4, 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APS01</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.000000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APS02</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.000000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APS03</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.000000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APF</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>1</td>
<td>0.0001, 0.00002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV05</td>
<td>P</td>
<td>S</td>
<td>-</td>
<td>3</td>
<td>0.0000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV05</td>
<td>P</td>
<td>S</td>
<td>-</td>
<td>3</td>
<td>0.0000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADV</td>
<td>P</td>
<td>S</td>
<td>-</td>
<td>3</td>
<td>0.0000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV06</td>
<td>P</td>
<td>S</td>
<td>-</td>
<td>3</td>
<td>0.0000018, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INAF</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>1</td>
<td>0.0001, 0.00003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.0000007, 3, 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AEF</td>
<td>P</td>
<td>S</td>
<td>AED</td>
<td>3</td>
<td>0.00016, 10, 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADF</td>
<td>P</td>
<td>S</td>
<td>ADD</td>
<td>3</td>
<td>0.00005, 12, 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADS</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>1</td>
<td>0.0001, 0.00001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV07</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.0000009, 5, 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV08</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.000000016, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV09</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.000000016, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV06</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.000000016, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV07</td>
<td>P</td>
<td>B</td>
<td>-</td>
<td>3</td>
<td>0.000000016, 3, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1D</td>
<td>A</td>
<td>D</td>
<td>E1F</td>
<td>2</td>
<td>0.00006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2D</td>
<td>A</td>
<td>D</td>
<td>E1F</td>
<td>2</td>
<td>0.00006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1D</td>
<td>A</td>
<td>D</td>
<td>D1F</td>
<td>2</td>
<td>0.00001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2D</td>
<td>A</td>
<td>D</td>
<td>D2F</td>
<td>2</td>
<td>0.00001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AED</td>
<td>A</td>
<td>D</td>
<td>AEF</td>
<td>2</td>
<td>0.00006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>A</td>
<td>D</td>
<td>ADF</td>
<td>2</td>
<td>0.00001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRV1</td>
<td>P</td>
<td>D</td>
<td>-</td>
<td>2</td>
<td>0.0000016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRV2</td>
<td>P</td>
<td>D</td>
<td>-</td>
<td>2</td>
<td>0.0000016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRV3</td>
<td>P</td>
<td>D</td>
<td>-</td>
<td>2</td>
<td>0.0000016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRV4</td>
<td>P</td>
<td>D</td>
<td>-</td>
<td>2</td>
<td>0.0000016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRV5</td>
<td>P</td>
<td>D</td>
<td>-</td>
<td>2</td>
<td>0.0000016</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9.3.2.2 Pre-processing

• Information extraction from files

A 'static-phase' dependency serial list is established by extracting and processing the information from the normal dependency file. Only dependency relationships other than the 'test' dependency are relevant in the static phase. Given the information in table 9.7, each dependency serial in the 'static-phase' dependency serial list is corresponding to each of the dependency group numbers from 1 to 14. Normal dependency relationships existing during the system dynamic phase and the dependency groups from the functional dependency file form the dynamic-phase dependency information list. During this process, an extension is applied to the dependency group numbers 35, 36, 37, 38, 40 and 41 shown in table 9.8. New dependency groups are generated by breaking down the intermediate functionally-controlling events into their basic event descendants. Consequently, the dynamic-phase dependency information list will include the expanded functional and switching dependency relationships as are shown in table 9.10.

<table>
<thead>
<tr>
<th>Dependency group number</th>
<th>Dependency type</th>
<th>Number 1</th>
<th>Number 2</th>
<th>List 1</th>
<th>List 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>func</td>
<td>1</td>
<td>2</td>
<td>G13</td>
<td>E1D, E2D</td>
</tr>
<tr>
<td>24</td>
<td>func</td>
<td>1</td>
<td>2</td>
<td>DS</td>
<td>D1D, D2D</td>
</tr>
<tr>
<td>25</td>
<td>func</td>
<td>1</td>
<td>1</td>
<td>AES</td>
<td>AED</td>
</tr>
<tr>
<td>26</td>
<td>func</td>
<td>1</td>
<td>1</td>
<td>ADS</td>
<td>ADD</td>
</tr>
<tr>
<td>27</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV1</td>
<td>E1D</td>
</tr>
<tr>
<td>28</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV2</td>
<td>E2D</td>
</tr>
<tr>
<td>29</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV3</td>
<td>D1D</td>
</tr>
<tr>
<td>30</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV4</td>
<td>D2D</td>
</tr>
<tr>
<td>31</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV5</td>
<td>AED</td>
</tr>
<tr>
<td>32</td>
<td>ldfunc</td>
<td>1</td>
<td>1</td>
<td>PRV6</td>
<td>ADD</td>
</tr>
<tr>
<td>33</td>
<td>swch</td>
<td>1</td>
<td>4</td>
<td>MPF</td>
<td>E1D, E2D, D1D, D2D</td>
</tr>
<tr>
<td>34</td>
<td>swch</td>
<td>1</td>
<td>4</td>
<td>G5</td>
<td>E1D, E2D, D1D, D2D</td>
</tr>
<tr>
<td>35</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>FB1</td>
<td>E1D</td>
</tr>
<tr>
<td>36</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>IV3</td>
<td>E1D</td>
</tr>
<tr>
<td>37</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>IV4</td>
<td>E1D</td>
</tr>
<tr>
<td>38</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>CV1</td>
<td>E1D</td>
</tr>
<tr>
<td>39</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>FB2</td>
<td>E2D</td>
</tr>
<tr>
<td>40</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>IV5</td>
<td>E2D</td>
</tr>
<tr>
<td>41</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>IV6</td>
<td>E2D</td>
</tr>
<tr>
<td>42</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>CV2</td>
<td>E2D</td>
</tr>
<tr>
<td>43</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>FB3</td>
<td>D1D</td>
</tr>
<tr>
<td>44</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>IV7</td>
<td>D1D</td>
</tr>
<tr>
<td>45</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>IV8</td>
<td>D1D</td>
</tr>
<tr>
<td>46</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>CV3</td>
<td>D1D</td>
</tr>
<tr>
<td>47</td>
<td>swch</td>
<td>1</td>
<td>1</td>
<td>FB4</td>
<td>D2D</td>
</tr>
</tbody>
</table>
Table 9.10 Expanded functional and switching dependency relationships in the firewater deluge system

- The fault tree simplification

The simplification, combination, modularization and re-modularization processes are conducted on the static-phase fault tree, which results in the ‘static-phase’ fault tree modules displayed in figure 9.6.

Factors identified:

```
  F5001
    IV11  DS
  F5002
    AES  AEF
  F5003
    ADS  ADF
  F5004
    MPF  APF  INAF
```

Modules identified:

```
G0  M7019
    M7006  M7007  M7008  M7009  M7010  M7015  M7016  M7017  M7018  F5004
```

```
G11  M7006
    E1F  M7001  M7002  M7003  E2F  M7004  F5001  D1F  M7005  F5001  D2F
```
Figure 9.6 Modules identified in the static-phase fault tree for the firewater deluge system

In the dynamic-phase fault tree shown in figure 9.4, basic events $E_1$, $E_2$, $E_1D$, $E_2D$, $D_1$, $D_2$, $A_E$, $A_E D$, $A_D$, and $A_D D$ are all basic events relevant to static system failure, of which $E_1D$, $E_2D$, $D_1$, $D_2$, $A_E$ and $A_D D$ are related to static system failure through their static counterparts $E_1F$, $E_2F$, $D_1F$, $D_2F$, $A_E F$ and $A_D F$. Basic events $E_1$ and $E_2$ gathered under $G_{13}$ can be identified as a module, corresponding to module 7001 shown in figure 9.6. Based on the dependency information given in table 9.7 and table 9.8 and the principle discussed
in section 9.2.3.2, it can be concluded that the dynamic-phase fault tree as whole is the only module identified. It is assigned a module id M7020 with the structure displayed in figure 9.7.

Figure 9.7 Simplified dynamic-phase fault tree structure for the firewater deluge system
9.3.2.3 Quantification

For illustrative purposes, the reliability assessment of the FDS is to be performed over a period of 5 years. The numerical time increment, \( dt \), is set as 0.5 hours. Assume that for the deluge system to effectively mitigate the fire incident, it is required to function continuously with sufficient supply for 6 hours. The frequency of the fire incident (\( w_{\text{fire}} \)) is set as 0.000001 per hour.

9.3.2.3.1 System Unavailability \( Q_{\text{sys}}(t) \)

- **Preparation**

As shown in figure 9.6, modules M7006, M7012, M7014, M7015, and M7019 contain no dependency relationships other than the test dependency within them. For these modules, BDDs will be established to facilitate the quantification to obtain the module parameters during the system static phase. Modules M7001, M7002, M7003, M7004, M7005, M7007, M7008, M7009, M7010, M7011, M7013, M7016, M7017 and M7018 involve dependency relationships corresponding to the dependency group 8, 9, 10, 11, 12, 1, 7, 3, 2, 13, 14, 4, 5 and 6 respectively. A Markov model needs to be developed for each of these modules to enable reliability prediction. Figures B.1 and B.2 in Appendix B respectively show the BDDs for modules M7006, M7012, M7014, M7015, and M7019, and reduced Markov models for modules M7001 and M7007-M7010. Tables B.1-B.3 in Appendix B display the Markov models constructed for modules M7011, M7017 and M7018. For other modules, the established Markov models are too big to reproduce and therefore not included.

Components included in each of the modules M7007, M7008, M7009 and M7010 all feature the same failure parameters and have the same structural contribution to the module failure. As such a simplified Markov model can be constructed for these modules by employing the 'State Lumping' technique illustrated in chapter 3.

- **Calculation of System Unavailability**

For the firewater deluge system, its unavailability at a specific point in time is obtained by investigating the reliability parameters of each module and factor displayed in figure 9.6. The
quantification is implemented on the established BDDs and Markov models in the way illustrated in chapter 7.

9.3.2.3.2 System Unreliability \( F_{sy}(T) \)

Module 7020 is the top module for the dynamic-phase fault tree. Some basic events included in the module are related to the system static-phase failure. According to the process described in section 9.2.3.3.2, the unreliability of the firewater deluge system during its dynamic phase is obtained through the following steps.

Step 1. Identify the initial states from which the firewater deluge system is able to activate.

a) Identify basic events from the dynamic-phase fault tree shown in figure 9.4 which should be considered when determining the initial states. Basic events E1D, E2D, DS, D1D, D2D, AES, AED, ADS, ADD, INAF and module M7001 are chosen as they are all related to component failures which contribute to the system static-phase failure.

b) Basic events MPF, APF, FB1 – FB4, IV3 – IV11, CV1 – CV4, IV14 – IV18, CV6 – CV7 and modules M7007 and M7010 are also added to the list of initial component states to consider since they contribute to the system static failure and represent the functionally-controlling components as are indicated in table 9.8.

c) Basic events which fulfil the two conditions explained in step 1(c) in section 9.2.3.3.2 will be deleted from the list. The basic events to be eliminated are AES, ADS, FB1 – FB4, IV3 – IV10, CV1 – CV4, IV14 – IV18, CV6 – CV7. Consequently, the final list of events which need to be examined to establish the initial states from which the system may start is: E1D, E2D, DS, D1D, D2D, IV11, AED, ADD, INAF, MPF, APF, M7001, M7007 and M7010.

d) The full list of initial states, based on the chosen basic events, is developed according to the algorithm presented in step 1(d) in section 9.2.3.3.2. There are 45 initial states from which the firewater deluge system is able to start.

Step 2: The probabilities of all the initial states are then calculated at the time when the system is activated. The algorithm underlying the calculation is given in equation 9.1. In the calculation, the state code of the basic event should be considered to account for the implied condition of any other basic events related to this component. For example, when the state code of basic event
'E1D' is '0', it means the pump E1 is ready to start, i.e. the pump has not failed in standby, which implies the state code of 'E1F' and basic events which represent the corresponding supporting components, such as FB1, IV3, IV4 and CV1 should be '0' too. The conditional probability in equation 9.1 is calculated by breaking down the conditions into a hierarchical structure, as is shown in figure 9.8, according to the modularized structure of the static-phase fault tree shown in figure 9.6.

![Hierarchical structure](image)

Figure 9.8 Hierarchical structure of conditions regarding the system static failure (referring to figure 9.6)

In terms of the calculation of the probability of system failure, conditional on the initial state (equation 9.1), the basic events which feature in a module assessed using a BDD are determined individually, as the joint probability can be calculated by multiplying independent probabilities. These events include DS, IV11, MPF, APF, INAF, AEF, ADF, M7007 and M7010. Basic events contained in a Markov model, such as E1F, E2F, D1F, D2F and M7001, have to be looked into as a group, the joint probability of several component conditions is derived by adding up the probability of the states in the Markov model in which the basic events are featured in the required state as in the condition list.

Step 3: Establish the appropriate quantification model. Since standby dependency exists in the system dynamic phase, the dynamic failure of the firewater deluge system has to be tackled with the Markov method.

a) A review is conducted on the dynamic-phase fault tree to see if any basic events should be added to the Markov model. For the firewater deluge system, basic events PRV1 – PRV6 are
only relevant to the system dynamic failure, therefore they are not considered with regard to the initial states. They need to be taken into account during the process of quantification of the system dynamic failure probability.

b) The state codes of the basic events defining each initial state are reviewed to ensure that they correctly account for any standby dependency involved in the system dynamic phase. For example, consider the initial state in which the state code of 'E1D', 'E2D', 'D1D' and 'D2D' are all '0', which means the pumps are all ready to start. Since the two diesel pumps are set as standby, their state codes will be changed to '-I', which represents their standby state. Consequently, during the process of developing the Markov model, only three component state codes are used: '0' - working, '1' - failed and '-I' - standby.

c) The Markov model is re-modularized. As has been discussed in step a), basic events PRV1 - PRV6 need to be considered during the quantification process for the system dynamic phase. If they are directly included in the Markov model for module M7020, the size of the resulting model will increase dramatically by a factor of $2^6$ in terms of the system states in the model. An alternative to overcome this is to carry out the re-modularization within the top module M7020. The re-modularization is performed revolving around the basic events which are involved in functional or switching dependency relationships as the dependent element. Take for example the firewater electric pump E1 represented by basic event 'E1D', basic events PRV1, FB1, IV3, IV4 and CV1 are the functionally-controlling elements exclusive to the basic event E1D as indicated in table 9.10. If pump E1 successfully activates when the system is called upon to function, it will contribute to the system unreliability by losing its functionality during operation. The loss of the functionality of pump E1 can be interpreted from two aspects: one is the failure of the pump itself; and the other is the failure of its functionally-controlling/supporting components which renders the pump to be unusable or inaccessible. According to figure 9.5, components which are involved in both general and limited functional dependency relationships as the controlling/supporting element have an influence on the corresponding dependent component during its active operation. According to table 9.10, it can therefore be determined that among the functionally-controlling/supporting events which are exclusive to pump E1, basic event PRV1 has an influence on the state of E1D when pump E1 is
functioning. Correspondingly, a new module can be formed within module M7020 which consists of basic events PRV1 and E1D through the OR logic to model the loss of the functionality of pump E1 during its operation due to the failure of the pump itself and the failure of the associated pressure relief valve. The same algorithm is followed to model the loss of the functionality of other pumps, represented by basic events E2D, D1D, D2D, AED and ADD, during their active operation. Accordingly, six modules are established, shown in figure 9.9, to model the loss of the functionality of each pump during their operation respectively.

Another situation where pump failures may contribute to the system unreliability is that the standby pump fails in standby following the system activation, and thus cannot respond to maintain the system function when the duty pump fails. Take the AFFF pump system for example, if the system starts with the successful activation of AFFF electric pump, the failure of AFFF diesel pump in standby, represented by basic event ADF, will contribute to the system unreliability given that the AFFF electric pump fails during the required functioning period of time. The pump failure in standby can also be ascribed to two causes: one is the standby failure of the pump itself or the failure of components required to enable the activation of the pump. According to figure 9.5, the state of components which are involved in both general functional and switching dependency relationships will determine if a successful activation is possible. Therefore, according to table 9.10, the loss of
functionality of the AFFF diesel pump during standby can be represented through a module which consists of basic events $ADF$, $ADS$, $IV17$, $IV18$, $CV7$ and $IV14$. The same algorithm is followed to model the loss of functionality of pumps D1 and D2 during standby. Three modules are established as shown in figure 9.10.

![Diagram](image)

Figure 9.10 Modules established to model pump failures in standby

d) The 'No Further Influence' technique, presented in section 9.2.3.3.2 step 3(b) is applied to facilitate the generation of a smallest Markov model. This technique applies not only to basic events which are explicitly included in the dynamic-phase fault tree, but also to basic events which need to be taken into account in the dynamic-phase analysis, but are not included in the dynamic-phase fault tree. Such basic events are $MPF$, $APF$, $IV11$, module $M7007$ and $M7010$. Take the basic event $MPF$ for example, after the system has activated, the function of the Main Fire and Gas panel (represented by $MPF$) is to activate standby firewater pumps in the event that a working pump fails. When standby pumps have failed in standby prior to the failure of the main panel, the consequent failure of the Main Fire and Gas panel will not matter given the 'No Repair during Functioning' policy. That is, when no standby pumps are able to start, the failure of the Main Fire and Gas panel does not need to be considered. The same principle can be applied to basic events $APF$, $IV11$, module $M7007$ and module $M7010$. Due to the re-modularization and the implementation of the ‘No Further Influence’ technique, the size of the resulting Markov model for the system dynamic-phase analysis has been dramatically reduced from more than 30,000 states to 500 states despite there being nearly 20 events considered in this Markov model.

Step 4: Quantification is implemented on the Markov model developed to assess the unreliability of the firewater deluge system over the required period $T_r$. In terms of the modules established to
model the loss of functionality of pumps, they can be addressed with the BDD method. Figure 9.11 illustrates how pump failure rates are determined through the quantification of modules, shown in figures 9.9 and 9.10, depending on the current state of the pumps.

Figure 9.11 Modules modelling the failure of functionally dependent components
As repair is not conducted during the system dynamic phase, the quantification of the modules shown in figure 9.11 is achieved using the failure probability of each basic event prior to time $t$. This is equal to its unreliability during the period of time $(0, t]$, which is obtained by $(1-e^{-\lambda t})$, where $t$ is measured from the time of system activation.

### 9.3.2.3.3 Calculation of Importance Measures

- **Calculation of the criticality function $G_i(q(t))$ during the system static phase**

Special consideration needs to be given to the calculation of the criticality function within the Markov model which is reduced through the 'State-Lumping' technique. In modules M7007, M7008, M7009 and M7010, each state is identified by indicating the number of failed components, rather than a list of the states of basic events. Therefore, the algorithm presented in section 7.1.6 for calculating the criticality function within the Markov model is no longer directly applicable. In this case, a new algorithm is developed.

Assume the reduced Markov model includes $n$ components, of which $m$ failures will result in the module failure (when $m=1$, this represents an OR logic; when $m=n$, this represents an AND gate; and when $1<m<n$, this represents the $m$-out-of-$n$ voting logic). Therefore $(n+1)$ states are contained in the Markov model as are shown in figure 9.12.

![Figure 9.12 The general reduced Markov model](image)

The critical transition is the transition from state $m$ to state $m+1$, which brings the system from working to failed. Assume that component $i$ is one of the $n$ components included in the model, the criticality function of component $i$ can be expressed as:

$$G_i(q(t)) = Q_m(t).P\{\text{component } i \text{ belongs to the working components given state } m\} + Q_{m+1}(t).P\{\text{component } i \text{ belongs to the failed components given state } m+1\}$$
where $Q_m(t)$ and $Q_{m+1}(t)$ stand for the probability of state $m$ and $m+1$ at time $t$ respectively.

Therefore, the criticality function of component $i$ included in a reduced Markov model is obtained through:

$$G_i(q(t)) = Q_m(t) \cdot \frac{C_{n-m}}{C_n} + Q_{m+1}(t) \cdot \frac{C_{n-1}}{C_n}$$

$$= Q_m(t) \cdot \frac{n-m+1}{n} + Q_{m+1}(t) \cdot \frac{m}{n}$$  \hspace{1cm} (9.8)

In this way, the criticality function of each specific component included in modules M7007, M7008, M7009 and M7010 can be obtained.

- **Calculation of importance measures during the system dynamic phase**

The calculation of the criticality function of each component which contributes to the system unreliability is based on the algorithm proposed in section 7.1.6. The Barlow-Proschan measure is also determined according to equation 9.4.

### 9.3.2.4 Output

Three output files are produced containing the results of the reliability analysis for the firewater deluge system. Tables B.4, B.5 and B.6 in appendix B contain the system failure predictions and importance measures for each component during the system static and dynamic phases respectively.

In table B.4, the ‘Unreliability’ represents the likelihood that the system fails once activated at some point during the required operational time period of 6 hours. The system unreliability is obtained directly from the Markov analysis for the system dynamic-phase. The calculation of the initial state probabilities in the Markov model for the system dynamic-phase analysis accounts for the fact that the system has successfully activated at time $t$. 

248
The 'Static Failure Intensity', denoted by $w_{acc_s}$, refers to the frequency of the unmitigated fire incident due to the system being unavailable to respond at time $t$. That is, $w_{acc_s}(t) = w_{fire} \times Q_{sys}(t)$.

The 'Dynamic Failure Intensity', denoted by $w_{acc_d}$, refers to the frequency that the fire incident is unmitigated due to the system failing following the activation within the required period of 6 hours. That is, $w_{acc_d}(t) = w_{fire} \times F_{sys}(T_r)$, where $T_r$ is measured from the system activation time $t$.

From table B.4, it can be seen that $w_{acc_d}$ is significantly smaller than $w_{acc_s}$. By integrating $w_{acc_s}$ and $w_{acc_d}$ respectively over the whole assessment period, we can get the expected number of fire accidents due to the system unavailability ($\int w_{acc_1}(t)dt$) and the expected number of fire accidents due to the system unreliability ($\int w_{acc_2}(t)dt$). For the firewater deluge system, over the period of 5 years, the expected number of fire accidents due to the system being unavailable is $1.355263 \times 10^{-3}$, while the expected number of fire accidents due to the system being unreliable is $2.969246 \times 10^{-6}$, which altogether mean that the total expected number of fire accidents over 5 years is $1.358232 \times 10^{-3}$. The expected number of fire accidents due to the system being unreliable makes a negligible contribution.

Figure 9.13 illustrates how the system unavailability (non-conditional upon previous occurrences of a system failure) varies as the time elapses.

From figure 9.13, it can be noticed that the system unavailability, when it passes the transient period, takes an obvious pattern featured by a time cycle of 6 months. There is an abrupt drop in the system unavailability at the turn of cycles. In each cycle, there is also a decrease in the
system unavailability after 3 months. All these patterns can be attributed to the fixed test intervals assigned to each dormant-failure component during the system static phase. For example, all filters and check valves are tested every 2 months. Therefore, after a short time into the 3\textsuperscript{rd} and 5\textsuperscript{th} month, all revealed failures will be eliminated which will reduce the system unavailability. Then the system unavailability will continue to rise till another round of tests are conducted. Since components which have a test interval of 2 months only account for a relatively small portion of all the components included in the system, the effect of their repair is not as significant as the repair of the components which are tested every 3 months.

From table B.5, it can be noticed that basic events MPF, APF, INAF, INB, SB, NB, WDV, WVS, ADV, IV1, IV2, IV12, CV5 and IV13 all feature a very high and similar value in the criticality function in the system static-phase analysis. This is consistent with their significant contributions to the system static failure as are reflected in the system static-phase fault tree in figure 9.3. According to the system static-failure logic represent in the fault tree, the occurrence of any single event among them can result in the system static failure. It can also be noticed that basic events E1F, E2F, D1F, D2F, AEF and ADF feature a much smaller value of the criticality function in comparison with basic events discussed above, as is illustrated in figure 9.14. The difference can be explained by their less significant contributions to the system static failure as they are included in the system static-phase failure logic in minimal cut sets of lower order. Among these events, the criticality function values of E1F and D1F are smaller in comparison with other basic events. This can be ascribed to the 3-out-of-4 voting logic which governs the firewater pump failures with regard to their effect on the system failure.
As the Barlow-Proschan measure of each basic event during the system dynamic phase features a
time cycle of 6 months in terms of the time the system activates, only the first two time cycles
are displayed in table B.6 for illustration. Basic events which are relevant during the system
dynamic phase but not included in the table, such as WPS1F - WPS4F, APS1F - APS3F, FB1 -
FB4, IV3 - IV11, IV14 - IV18, CV1 - CV4, CV6 - CV7, MPF and APF, all feature a Barlow-
Proschan measure of '0' since they are enabling events. For example, the firewater pressure
sensors, the Main Fire and Gas control panel, the AFFF pressure sensors and the AFFF panel all
contribute the function to call upon the standby pumps when the duty pump fails. After the
activation is implemented, the failure of these components does not have any effect on the
functioning system. The filters, isolation valves and check valves can get blocked on the standby
pump streams after the system activates, thus disabling the standby pump to activate. But due to
the assumption that they won't get blocked after the standby pump activates, they also act as
enabling events. Therefore, they make no direct contributions to the system dynamic failure,
which means their Barlow-Proscan measure is zero.

For those basic events listed in table B.6, their Barlow-Proscan measures at each starting time
provides an indication of their direct contribution to the system dynamic failure. These are
consistent with their contribution to the system dynamic-phase failure logic represented by the
dynamic-phase fault tree. For example, in general, the basic event 'INAF' has a relatively high
Barlow-Proscan importance measure as it is directly linked to the top event in the dynamic-
phase fault tree through OR gate, which means it forms a single order minimal cut set in the system dynamic-phase failure logic. Although the four firewater pumps and the two AFFF pumps assume the parallel functions during the system dynamic phase, the Barlow-Proshchan measure of the former group is much lower than that of the latter. This is again due to the voting logic between the firewater pumps. In other words, the use of pooled spares (the case for firewater pumps), other than the single spare (the case for AFFF pumps), reduces the contribution that each component failure makes to the system failure.

Also in table B.6, it can be noticed that the change in the Barlow-Proshchan measure of basic event 'INAF' is the reverse reflection of the change in the system unavailability, while for other basic events included in table B.6, their B-P measures reflect the same pattern in system unavailability. This feature is illustrated in figure 9.15. This is related to the inspections carried out on the system during the system static phase since the calculation of the Barlow-Proshchan measure accounts for the probability of a successful activation. Take basic event INAF for instance, for 'INAF' to be the direct cause of the system unreliability, both the firewater system and the AFFF system must be available at the starting time as well as reliable during the dynamic phase. Therefore, the Barlow-Proshchan measure of 'INAF' will take the same pattern as the system availability, i.e. opposite to the system unavailability.

![Comparison between B-P importance measures](image)

Figure 9.15 Illustration of opposite patterns in the Barlow-Proshchan importance measure
9.3.3 Review of Static-dynamic Two-phase Approach

As is illustrated in the analysis of the firewater deluge system, in the static-dynamic two-phase approach, the system unavailability and unreliability are investigated separately through two fault trees which represent the system static and dynamic failure logic respectively. With this approach, the analyst can get a better understanding of which component failures are relevant in which system phase and how each component failure mode contributes to the system failure. It saves a lot of manual effort which would be required for the dynamic fault tree analysis as in the dynamic fault tree method, both system unavailability and unreliability are evaluated in the same fault tree structure, thus a lot of manual input is required to identify relevant component failures in terms of the system unavailability or unreliability.

The establishment of the dependency file provides a systematic and flexible way to represent different types of dependency relationship existing in the system static and dynamic phases. The identification of switching dependency relationships enables the analysis to take into account the failure of functionally controlling/supporting components during the system dynamic phase when it is difficult to explicitly include them in the dynamic-phase fault tree structure as the system may enter the dynamic phase in numerous states. Combined with the pre-processing of the system static-phase and dynamic-phase fault trees, the static-dynamic two-phase approach delivers an efficient reliability analysis of active-on-demand systems as dependent component failures are investigated in the smallest possible fault tree sections resulting from the re-modularization process.

9.3.4 Comparison with Results Assuming Independence

The quantification is carried out to obtain the system unavailability based on the assumption that all component failure events occur independently. That is, there exists no dependency relationship during the static phase for the firewater deluge system.

9.3.4.1 Comparison of System Unavailability for Static Phase

The comparison with the actual system unavailability is illustrated in figure 9.16.
In figure 9.16, it can be seen that the system unavailability under the independence assumption follows the same pattern as the system unavailability calculated accounting for the dependency. In these two situations, the value of the system unavailability at each discrete point of time is almost exactly the same as each other. This can be explained by investigating the effect that the maintenance dependency and the test dependency have on the system evaluation respectively. As is illustrated in section 8.4.5.2, the test dependency improves the system availability as it shortens the down time of relevant components by introducing extra common inspections. Alternatively, the maintenance dependency reduces the system availability as its existence means that the actual down time of relevant components is prolonged and thus the component failure probability increases. In the firewater deluge system, both the maintenance and test dependency exist during the system static phase. It happens that the effect of the test dependency is cancelled by that of the maintenance dependency so that the system unavailability by assuming independence does not show a significant difference from the accurate value where both maintenance and test dependency have been accounted for. An investigation is carried out to support the above argument by illustrating separately the effect of the maintenance dependency and test dependency on the evaluation of the firewater deluge system.
Figure 9.17 demonstrates the effect that the test dependency has on the evaluation of the system availability for the firewater deluge system.

In figure 9.17, it can be seen that the existence of the test dependency causes a difference in the value of the system unavailability during the last three months in each cycle of 6 months from the situation where no dependency is considered. Such a difference lies in the common inspections which brings about the test dependency. For example, without the common inspections, filters and check valves are tested every 2 months, and isolation valves are tested every 6 months. With the test dependency indicated in table 9.7, these components are now tested every 3 months as a group in addition to their individual inspections. It means that at the end of the first 3 months in each cycle the system unavailability will be reduced to a lower level by the common inspections conducted on relevant components. This explains why there is a gap between the two situations at the end of the forth, fifth and sixth month in each cycle. The impact of the test dependency is also reflected in the value of the expected number of unmitigated fire accidents due to the unavailability of the firewater deluge system. With the test dependency taken into account in the analysis, the expected number of fire accidents is obtained as 1.227442e-003, compared with 1.354592e-003 with the independence assumption.
In contrast, figure 9.18 illustrates the impact of the maintenance dependency on the system unavailability through the comparison with the situation where no dependency is accounted for.

![Comparison of System Unavailability](image)

Figure 9.18 Illustration of the effect of the maintenance dependency

In figure 9.18, it can be seen that the maintenance dependency results in a larger value of the system unavailability during the system static phase over the 5-year period of time. It also results in a larger value of the expected number of fire accidents as 1.478633e-003.

### 9.3.4.2 Comparison of System Unreliability for Dynamic Phase

Since the dependency is not considered, the BDD approach is adopted for the dynamic phase analysis. The fault tree, as shown in figure 9.19, is constructed to model the system dynamic failure assuming independence. To allow the fault tree structure to represent all possible situations, standby failures of firewater pump streams 3 and 4 and AFFF pump stream 2 are considered.

Simplification and modularization processes are conducted on the dynamic-phase fault tree in figure 9.19. The corresponding BDD is then established based on the modularized fault tree structure. Quantification is carried out on the BDD by applying equation 9.2. All different starting states in which the system can start the dynamic phase are identified in the static-phase analysis. The system unreliability is obtained for each different starting state. By taking into
account the probability of each starting state as in equation 9.2, the final system unreliability can then be obtained regardless in which state the system starts functioning.

Figure 9.19a Dynamic-phase fault tree for the failure of the firewater pump system
Firewater pump stream 3 unable to activate

Firewater pump stream 3 fails during functioning

Control system fails to activate firewater pump stream 3

Firewater pressure sensing system fails to detect the falling pressure

Main fire and gas panel fails to send signal to pump 3

Firewater pump stream 3 fails in standby

Diesel firewater pump 1 fails

Pressure relief valve 3 fails

Firewater pump stream 4 unable to activate

Firewater pump stream 4 fails during functioning

Figure 9.19b Failure of firewater pump stream 3 during the system dynamic phase

Figure 9.19c Failure of firewater pump stream 4 during the system dynamic phase

Main fire and gas panel fails to send signal to pump 3

Firewater pressure sensing system fails to detect the falling pressure
The AFFF electric pump stream fails during functioning

The AFFF diesel pump stream fails during the system dynamic phase

Pressure relief valve 5 fails

The AFFF diesel pump stream fails in standby

The AFFF control system fails to activate the AFFF diesel pump

The AFFF control panel fails to send signal to AFFF diesel pump

The AFFF pressure sensing system fails to detect the falling pressure

AFFF supply fails

(See figure 9.3f)

Figure 9.19d Failure of AFFF pump system during the system dynamic phase
Figure 9.20 displays the comparison in terms of the system unreliability between situations where dependency is taken into account and not.

It can be seen in figure 9.20 that although the system unreliability follows the similar pattern in the two different situations, there is a significant gap in the value of the system unreliability from the same starting point between the two situations. The value of system unreliability assuming independence is nearly twice the value with the dependency taken into consideration during the analysis.

As the standby dependency is not accounted for, the alternation of the state of standby pumps cannot be represented. This results in that the standby failure probabilities of firewater diesel pumps and AFFF diesel pump are quantified over the whole system functioning period, such as $q_{D1F}$, $q_{D2F}$, and $q_{ADF}$, when actually they are only relevant prior to the time they get activated. The same problem exists with regard to the dynamic failure probabilities of these standby pumps, such as $q_{D1D}$, $q_{D2D}$, and $q_{ADD}$. These basic events are also investigated over the whole system functioning period, when in fact they only need to be considered from the point they activate.

What’s more, as the functional dependency and switching dependency are ignored, the influence of some functionally controlling/supporting components are misinterpreted in the quantification process. For example, the pressure relief valve can only fail open when the pump stream is
functioning, which means its failure probability can only be calculated from the time the corresponding pump starts functioning. However, for the pressure relief valves fitted on standby firewater and AFFF diesel pump streams, this characteristic cannot be represented in the quantification process. Same as the dynamic failures of the standby pumps, the failure probability of these pressure relief valves are considered over the whole system functioning period. Similarly, when the switching dependency existing on each standby pump stream between isolation valves, check valves and the pump is ignored, the failure of these valves are concerned over the whole system functioning period, when they should only be considered as the contribution to the system unreliability during the time before the pump activates.

All the above factors explain the gap existing between the two lines in figure 9.20. With the greater value of system unreliability based on the ‘Independence’ assumption, the expected number of fire accidents unmitigated due to the system being unreliable over the period of 5 years is 5.8791E-06.

Another notable point with figure 9.20 is that the system unreliability has strictly taken a cycle of 3 months over the 5-year period. As the system unreliability is partly dependent on the system availability at the time of activation, components which are required during both system static and dynamic phases would have influence on the system unreliability through their availability at the time of activation. Since most of these components feature an inspection interval of either 2 or 3 months, this explains how the 3-month cycle comes into existence. Thus, with the identification of these components, the 3-month cycle can be determined in advance, which means that the calculation only needs to be carried out over the first 3 months.
Chapter 10. Review of Phased-mission System Analysis

In previous chapters, techniques have been discussed and illustrated which are applied to the reliability assessment of continuously-running systems and active-on-demand systems. In this chapter, a different type of system, phased-mission system, is considered. Following a review of the currently available methods to assess this type of system in this chapter, a new technique is presented in the following chapter to perform the reliability assessment on phased-mission systems when dependency relationships between components exist.

10.1 Introduction

A system may be referred to as a Phased Mission, if its success depends on a sequential set of objectives operating over different time intervals, each of which forms the different phase. During the execution of the phases in a mission, the system configuration, the system failure logic model or system failure characteristic changes to accomplish a different objective. For the mission to be a success, the system must operate successfully through each of the phases. The phases in a mission may be identified by: phase number, time interval, system configuration, tasks to be undertaken, performance measures of interest, or maintenance policy.

The operation of many systems feature the phased-mission characteristic. Examples of these systems include space vehicle operations, aircraft flights and ship operations.

The reliability analysis of phased-mission systems involves complexity not encountered in that of single-phased systems, because the system failure logic changes from phase to phase, and component failures may be common to more than one of the phases. That is, whilst component failures can occur at any point during the mission, they may only contribute to system failure for some of the phases. A component may have failed in a phase in which its failure has no direct impact, but its failure will make a contribution to the system failure during a later phase. In this case, it may be that the transition from one phase to another is the critical event leading to mission failure.

It is inappropriate to obtain the reliability of a mission by simply multiplying the individual phase reliabilities. This is due to the fact that the reliability of a particular phase is not only decided by the specific system failure logic model for this phase but is also dependent on the state in which the system resided at the end of the preceding phase. As a result, the phases of the mission are statistically dependent. It is also not appropriate to assume that each component is
functioning at the beginning of each phase. Therefore, an important problem in phased-mission analysis is to develop an appropriate approach to calculating, as efficiently as possible, the accurate value for the Mission Unreliability. Mission unreliability is defined as the probability that the system fails to function successfully through at least one phase of the mission. So far, a large amount of research has been carried out into phased-mission analysis, and methods have been proposed to address the mission unreliability and other related problems. These methods can be categorized as those appropriate for systems which are non-repairable or repairable over the mission duration, and are discussed separately in the following sections. For the illustrative purpose, a general phased-mission is introduced which contains \( m \) phases. For each phase \( j \) in the mission, it has a time duration of \( [t_{j-1}, t_j) \).

10.2 Existing Methods for Phased-mission Systems

10.2.1 Non-repairable Phased-mission Systems

In a non-repairable phased-mission system, a component failure is permanent, i.e. the component will remain in the failed state once it fails during the mission.

10.2.1.1 Transformation of a Multi-Phased Mission to an Equivalent Single-Phase Mission

The earliest consideration of a phased-mission analysis was made by Esary and Ziehms [43] using fault tree analysis. One of the important outcomes of their research was to transform a multi-phased mission into an equivalent single-phase mission, which would allow the existing techniques to be applied to calculate the mission reliability.

In a multi-phased mission, the performance of a component in a particular phase is dependent on its behaviour through previous phases. It will only be working in a phase if it has performed successfully through all preceding phases. Also, in a multi-phased mission where the repair is not possible, if a component is residing in the failed state in a phase, the component failure may have occurred in any of the phases up to and including the phase in question. Therefore, in the fault tree analysis for the phased-mission, the single event input representing the failure of component \( c \) in phase \( j \) will be replaced by an OR combination of the failure of component \( c \) in any phase up to and including phase \( j \), shown in figure 10.1.
Component $c$ is failed in phase $j$

c_1  c_2  ......  c_3

Figure 10.1 Component failure in fault tree for phase $j$ in a multi-phased mission

The individual phase configurations can then be joined in series to form a single system. The overall Mission Unreliability can then be represented in a single fault tree structure by combining the fault tree structures which model individual phase failures through an OR gate. A simple 3-phase-mission system is used, shown in figure 10.2, to illustrate such a transformation.

Figure 10.2 Reliability network of a 3-phase-mission system

The corresponding fault tree structures for each phase are shown in figure 10.3.

Figure 10.3 Fault tree structure for individual phase failure

By expanding each component failure in terms of its phase failures and combining the phase failure conditions, an equivalent single-phase fault tree structure can be established which represents the overall mission failure, as shown in figure 10.4.
10.2.1.1 Cut Set Cancellation

Cut set cancellation is a technique which can be applied to phased-mission systems to simplify the system phase configurations prior to the transformation of the multi-phased mission to an equivalent single-phase mission.

The rule for cut set cancellation is:

A minimal cut set in a phase can be cancelled, i.e. omitted from the list of minimal cut sets for that phase, if it contains a minimal cut set of a later phase.

The cut set cancellation won’t affect the mission reliability/unreliability. If a minimal cut set exists in the system failure logic model for both phase \( j \) and \( k \) \((j<k)\), its occurrence in phase \( j \) doesn’t need to be considered since it will be accounted for in phase \( k \) with regard to the mission failure.

The minimal cut sets for each phase of the system in figure 10.4 are:

Phase 1: \( \{A, B, C\} \)

Phase 2: \( \{A\}, \{B, C\} \)

Phase 3: \( \{A\}, \{B\}, \{C\} \)

Minimal cut set \( \{A, B, C\} \) in phase 1 can be eliminated since it contains minimal cut sets \( \{A\} \) and \( \{B, C\} \) from phase 2. In the same way, minimal cut sets for phase 2 can be both removed since they both contain minimal cut sets from phase 3. Therefore, only phase 3 needs to be considered with respect to the mission reliability. Correspondingly, the transformation based on
the simplified system configurations will produce the equivalent single-phase fault tree structure shown in figure 10.5.

![Fault Tree Diagram](image)

**Figure 10.5 Single-phase fault tree structure after the cut set cancellation**

### 10.2.1.2 Approximation Techniques for Mission Unreliability

An important problem with phased-mission analysis is to calculate as efficiently as possible either the exact value or bounds for mission unreliability. For complex phased-mission systems, the calculation of the exact unreliability of a mission can be prohibitively expensive largely due to the number of minimal cut sets being added to the analysis by the basic event transformation illustrated in figure 10.1. In this case, approximation techniques may be necessary, especially for systems containing a large number of components or a large number of phases.

Approximation methods have been developed to estimate the system unreliability without the application of basic event transformation. Four such methods were given in Burdick et al [44] based on a review of the developments by Esary and Ziehms. These methods are:

- **Inclusion-Exclusion Expansion of Phase Unreliabilities (INEX)**
  
  In this method, the minimal cut sets are obtained for each phase from the corresponding logic model. The unreliability of phase \( j \), \( Q_j \), is calculated using the inclusion-exclusion expansion of the phase \( j \) minimal cut sets (equation 2.5) using unconditional basic event unreliabilities. The unconditional basic event reliability during phase \( j \), \( P_{c_j} \), is defined as the probability that the corresponding component has functioning through the phases up to and including phase \( j \), i.e. the component is working at the end of phase \( j \). \( P_{c_j} \) can be derived through equation 10.1.

\[
\bar{P}_{c_j} = P[x_{c}(t_j) = 0] = \prod_{i=1}^{j} \rho_{c_i} \quad \text{for } j = 1, 2, ..., m
\]  

10.1
where \( x_c(t) \) represents the state of component \( c \) at time \( t \) with '1' indicating a failed state and '0' indicating a working state; \( \rho_{c_i} \) is the conditional basic event reliability on the condition that the component is working at the start of phase \( i \).

An approximation for mission reliability, \( R_{IN-EX} \), can be expressed by the product of the individual phase reliabilities \( R_j \) in equation 10.2.

\[
R_{IN-EX} = \prod_{j=1}^{m} R_j
\]

An approximation of mission unreliability, \( \overline{R}_{IN-EX} \), can also be obtained by the sum of the individual phase unreliabilities \( \overline{R}_j \) using equation 10.3.

\[
\overline{R}_{IN-EX} \leq \sum_{j=1}^{m} \overline{R}_j
\]

• **Inclusion-Exclusion Expansion with Cut Set Cancellation (INEX-CC)**
This method is similar to the INEX method. The difference is that in this method mission cut set cancellation is carried out before \( \overline{R}_j \) is calculated for each phase. In this case, the \( \overline{R}_j \) obtained for each phase will in general be less than the \( \overline{R}_j \) calculated in INEX method because there are fewer cut sets included in the logic model for each phase.

• **Minimal Cut Set Bound (MCB)**
In this method, minimal cut sets are obtained for each phase. The probability that minimal cut set \( C_i \) occurs during phase \( j \), \( q_{c_{i(j)}} \), is calculated using equation 10.4.

\[
q_{c_{i(j)}} = \prod_{c=1}^{N_{c_{i(j)}}} P\{c\}
\]

where \( P\{c\} \): the probability of the occurrence of basic event \( c \) in cut set \( C_i \) of phase \( j \)

\( N_{c_{i(j)}} \): the number of basic events included in minimal cut set \( C_i \) of phase \( j \)

The reliability of phase \( j \) can then be estimated using the minimal cut set bound in equation 10.5.

\[
R_j = \prod_{i=1}^{N_{CS_j}} q_{c_{i(j)}}
\]

where \( N_{CS_j} \): the number of minimal cut sets in phase \( j \)
\( q_{C_{ij}} \): the probability of success of cut set \( C_i \) through phase \( j \)

The approximation for the mission reliability using the minimal cut set bound, \( R_{MCB} \), and mission unreliability \( \overline{R}_{MCB} \) can then be obtained in the same way as in equations 10.2 and 10.3.

- **Minimal Cut Set Bound with Cut Set Cancellation (MCB-CC)**
  The MCB-CC method is stepwise identical to the MCB method except that mission cut set cancellation is implemented prior to any calculation.

The following ordering, shown in equation 10.6, exists among the bounds obtained from the above approximations:

\[
\overline{R}_{MISS} \leq \overline{R}_{INEX-CC} \leq \left\{ \overline{R}_{MCB-CC} \right\} \leq \overline{R}_{MCB}
\]

10.6

No general comparison can be made between \( \overline{R}_{INEX} \) and \( \overline{R}_{MCB-CC} \) as the ordering of these values depends on the particular problem being solved.

### 10.2.1.3 Laws of Boolean Phase Algebra

Boolean Algebra has formed the foundation underlying fault tree analysis. A set of additional Boolean algebraic laws have been developed by Xue and Wang [45] to enable the dependency between different phases due to common component failures to be taken into account in the phased-mission analysis. In the Boolean phase algebra, a basic event \( A \) may be represented in the following way:

- \( A_j \) – Basic event \( A \) occurs during phase \( j \), i.e. the failure only occurs during this particular phase.
- \( A_{\overline{j}} \) – Basic event \( A \) exists in phase \( j \), i.e. the failure can occur in any of the previous phases \( 1, \ldots, j-1 \) or in phase \( j \).

Suppose phase \( k \) comes before phase \( j \) in a mission. The intersection and union concept rules can be applied to phased-missions by extending to events included in different phases, as are given in equation 10.7, 10.8 and 10.9.
In the same way, for the system to be in a failed state in phase $j$, $X(j)$, system failure could have occurred in any phase up to and including phase $j$. This can be expressed in equation 10.10.

$$X(j) = X_1 \cup X_2 \cup ... \cup X_j$$  

where $X(j)$ is the event that the system is failed in phase $j$;

$X_i$ is the event that the system fails first during phase $i$

$$X_j = \bigcup_{i=1}^{Nc(j)} C_{i}(j), \text{ where } C_{i}(j) \text{ represents the existence of cut set } C_i \text{ in phase } j.$$

The mission unreliability can then be expressed in equation 10.11.

$$R_{MISS} = P[X(m)]$$

$$= P \left( \bigcup_{j=1}^{m} X_j \right) = P \left[ \bigcup_{j=1}^{m} \left( \bigcup_{i=1}^{Nc(j)} C_{i}(j) \right) \right]$$

10.11

By applying the rules presented in equations 10.7 – 10.9, equation 10.11 automatically implements the cut set cancellation presented in section 10.2.1.1.1.

Somani and Trivedi [46] present further methods for phased mission system reliability analysis based on Boolean algebraic methods and fault trees. Instead of transforming a multi-phased mission into an equivalent single-phase mission, the fault trees for different phases are solved individually. The probabilities of all possible combinations contributing to the mission failure
during individual phases are computed. However, this requires that information must be carried from phase to phase since phases are not independent and leads to lengthy calculations for situations where there are numerous phases or cut sets in each phase. This work is extended by Ma and Trivedi [47] in which the mission unreliability is obtained in the form of the sum of disjoint products using a computational algorithm and the algorithm is implemented using the SHARPE software package.

10.2.1.4 Binary Decision Diagrams

The methods which have been described in previous sections are all based on the identification of minimal cut sets of the phased-mission system (PMS) and need to obtain the sum of disjoint products explicitly, which can be computationally intensive. Zang et al [48] present a BDD-based algorithm (PMS-BDD) for reliability analysis of phased-mission systems. This algorithm uses phase algebra to deal with the dependence across the phases, and a new BDD operation to incorporate the phase algebra. With this algorithm, the cancellation of common components among the phases can be combined within the process of BDD generation without additional operations, and the sum of disjoint products can be implicitly represented by the final BDD. The basic event transformation is carried out in PMS-BDD algorithm before the PMS-BDD is established.

Let \( F_{C_j} \) denote the failure function for component \( C \) up to phase \( j \); \( q_{C_j} \) the failure function for component \( C \) in phase \( j \) given that component \( C \) is working at the start of phase \( j \). Then the failure function for component \( C \) up to phase \( j \) can be expressed in equation 10.12.

\[
F_{C_j} = \left[ 1 - \prod_{i=1}^{j-1} (1-q_{C_i}) \right] + \left[ \prod_{i=1}^{j-1} (1-q_{C_i}) \right] \cdot q_{C_j} \tag{10.12}
\]

The first term of equation 10.12 represents the probability that the component has already failed during the previous phases 1, 2, ..., \( j-1 \). The second term represents the probability that the component has functioned through all the previous phases and fails in phase \( j \).

In the same way as for a single-phase mission, an ordering sequence is required to enable the construction of the BDD. In terms of the basic event transformation for each variable, Zang et al present two possible ordering schemes:

- **Forward Phase-Dependent Operation (PDO):** The variables are ordered in the same pattern as the phase order, \( C_1, C_2, ..., C_m \).
• Backward Phase-Dependent Operation (PDO): The variables are ordered in the reverse pattern of the phase order, $C_m, C_{m-1}, \ldots, C_1$.

Let $i < j$, and assume component $C$ is relevant in both phase $i$ and $j$. The $ite$ structure for component $C$ in phase $i$ and $j$ can then be respectively represented by $E_i$ and $E_j$.

$$E_i = ite(C_i, G1, G2)$$
$$E_j = ite(C_j, H1, H2)$$

The logic operation ($\oplus$: AND or OR) between $E_i$ and $E_j$ can be represented by BDD manipulations as:

Forward PDO: $ite(C_i, G1, G2) \oplus ite(C_j, H1, H2) = ite(C_i, G1 \oplus H1, G2 \oplus E_j)$

Backward PDO: $ite(C_i, G1, G2) \oplus ite(C_j, H1, H2) = ite(C_j, E_i \oplus H1, G2 \oplus H2)$

The ordering of variables is very important to the size of a BDD. Methods may be implemented to select the most appropriate or efficient ordering sequence of variables in BDD. Generally speaking, backward PDO will produce a smaller BDD and enable the cancellation of common components without requiring additional operations.

An algorithm is presented to construct a BDD for a phased-mission system:

1. Obtain the failure function for each variable using equation 10.12.
2. Decide the ordering for the mission components using a heuristic method.
3. Generate the BDD for each phase.
4. Use phase algebra and the Backward PDO to combine the BDD for each phase using OR logic to obtain the final mission BDD
5. Calculate the mission unreliability from the mission BDD

The limitations of Zang et al’s approach are identified by Dugan and Xing [49]. The developed Phase-Dependent Operation (PDO) will be able to generate the correct PSM-BDD only given that the ordering scheme abides by the following rules:

• Rule 1: Ordering adopted in the generation of each single-phase BDD are consistent or the same for all the phases

• Rule 2: Orderings of variables that belong to the same component but to different phases stay together, which is achieved by replacing each component indicator-variable with a set of variables which represent the component in each phase after ordering components using heuristics
10.2.1.5 Non-coherent Fault Tree Analysis

For some phased-mission systems, failure in different phases will result in different consequences. It may then be necessary to establish the phase as well as mission failure probability. When not only the mission reliability parameters but also the individual phase reliability parameters are of concern, most methods which have been discussed in previous sections are limited by their inability to calculate the individual phase failure probabilities. If cut set cancellation is used, it makes it impossible to account for individual phases. To overcome this problem, a method is proposed by R. A. La Band and J. D. Andrews [50] which enables the failure probability of each phase to be determined in addition to the whole mission unreliability by combining the causes of success of previous phases with the causes of failure for the phase in question.

The basic event transformation is carried out to reflect that the component being failed in phase $j$ can be due to the failure which occurs in any phase up to and including phase $j$. The event that the system fails first in phase $j$ is represented by the success of all the phases 1 to $j-1$, and the failure during phase $j$. In the fault tree analysis, the success of phase $i$ is represented by a non-coherent fault tree structure using NOT logic shown in figure 10.6.

![Figure 10.6 Generalized phase fault tree structure](image)

Mission unreliability, $\overline{R_{MISS}}$, can then be obtained by summing up the probabilities of mission failure in individual phases as shown in equation 10.13.
\[ \bar{R}_{\text{MISS}} = \sum_{i=1}^{m} \bar{R}_i \]  

where \( \bar{R}_i \) is the probability that the mission fails first during phase \( i \); \( m \) is the total number of phases included in the mission.

Since for mission failure in any particular phase, the possibility that the mission has already failed during previous phases has been ruled out by explicitly accounting for the mission success up to the current phase, equation 10.13 provides the correct and exact value of mission unreliability.

This method allows for the evaluation of individual phase failures, and also accounts for the condition where components are known to have functioned to enable the system to function through previous phases. However, owing to the fact that minimal cut sets are not removed from earlier phases in the analysis, the fault tree can be complex and require significant effort to solve. The fault tree simplification techniques illustrated in section 5.3.2 can be applied to reduce the size of the fault tree structure. The Factorization operation however needs extra attention. Since NOT logic is included in the phase fault tree structure, for the primary basic events to be identified as factors, they must always occur together in one gate type, and their complements, if there are any, should always occur together in the opposite gate type by De Morgans' laws. For example, for basic events \( A \) and \( B \) to be qualified as factors, they should always be found as either \( A+B \), \( \bar{A} \cdot \bar{B} \) or \( A \cdot B \), \( \bar{A} + \bar{B} \).

Owing to the non-coherent nature of the fault trees, the combinations of basic events that lead to the individual phase failure are expressed as prime implicants. In addition to event \( C_i \) which represents the component failure during phase \( i \), \( C_{ij} \) is introduced to represent the event that the component fails at some time from the start of phase \( i \) to the end of phase \( j \). This makes it possible to define a new algebra over the phases to manipulate the logic equations. The algebraic laws can be summarized as follows (\( i < j \)):

\[
\begin{align*}
A_i \cdot A_i &= A_i \\
A_i \cdot A_j &= 0 \\
A_i \cdot A_{ij} &= A_i \\
\bar{A}_i \cdot \bar{A}_i &= 0 \\
\bar{A}_i \cdot A_{ij} &= A_{i+1,j} \\
\bar{A}_i \cdot \bar{A}_{i+1} \ldots \bar{A}_j &= \bar{A}_{ij} \\
A_i + A_{i+1} + \ldots + A_j &= A_{ij}
\end{align*}
\]
Therefore, if two implicant sets contain exactly the same components where all but one occur over the same time and the other represents the component failure in continuous phases, the two implicant sets may be combined in the way illustrated as follows:

\[ A_1B_1 \quad A_2B_2 \Rightarrow A_1B_{12} \]

Having established the prime implicants for each phase, they may now be used to quantify the probability of phase and mission failure. The unreliability for each individual phase \( j \), \( R_j \), can be obtained using a simple inclusion-exclusion expansion for the prime implicants \( C_i \) in the phase.

The phase unreliability can also be obtained through the binary decision diagrams approach which stands as a better alternative to the quantification technique based on minimal cut sets/prime implicants. With the BDD approach, the fault tree structure for each phase in the mission is converted to a binary decision diagram. By summing up the probability of each disjoint path leading to terminal node '1' in the BDD, it will produce the exact value of phase unreliability, thus the exact value of mission unreliability through equation 10.13. This avoids the need to determine all phase prime implicants as an intermediate stage.

### 10.2.1.6 The Markov Method

The Markov method serves as an alternative to combinatorial techniques. There are two general approaches to the solution of multi-phased missions using Markov methods. One is to treat each phase individually, where the Markov model for each individual phase is solved separately and linked by a state probability vector. The other approach is to analyze the entire mission within a single Markov model. The latter approach is considered by Dugan [51] who presents a method to construct a single continuous-time discrete-space Markov model for phased mission systems where the state space is the size of the union of the components in each phase model.

To combine the Markov model for each individual phase into a single mission Markov model, a multiplicative factor \( \Phi_i \) is appended to each transition in phase \( i \), as is shown in figure 10.7. The combined Markov chain has a state space defined by the union of the individual phase Markov models, and the state transitions in the combined Markov model are defined by the sum of corresponding phase transitions.
The combined Markov model may be solved using a standard numerical technique. In the solution, when the mission time has progressed into phase $i$ ($t_{i-1} \leq t \leq t_i$), $\Phi_i$ is set to one, and all other $\Phi_j$ ($j \neq i$) are set to zero, thus removing any transitions that do not belong to the current phase. The state space does not change and rather than transforming the state probabilities, the state transitions change as the phase change. The initial conditions for the first phase are known, and the failure and success probability of each phase can be obtained using the Markov state probabilities at the end of each phase. The final state probability vector of each phase is passed directly to the following phase for further analysis.

In the case that the components are not the same in each phase, a full Markov state listing is formed by including all components that contribute to the mission failure at some point. For each source state in the combined Markov model, the destination state corresponding to the failure of a component can be different in different phases, and so each component must be considered several times during the mission. A state in one phase that causes the system to fail is not necessarily a failure stated of the previous phases. However, if a system failure state is reached in phase $i$, it becomes absorbing for all later phases. The system states are then defined as 'operational for all phases' or 'failed in phase $i$', where phase $i$ is the first phase in which the system fails. Dugan also considers this method for systems with imperfect coverage.

The approach provides an efficient representation of the multi-phased mission. The construction of a single model eliminates the problems faced across a phase boundary if the state-space of the phases are not the same. However, the single model has a state space defined by all components required in the mission regardless of their relevance in each individual phase. In some cases, a mission may require a large number of components that are not necessarily required through all phases. The resulting state space of the single model will become large and the set of differential equations will also increase in many cases, making the problem complex to solve or even intractable.

Figure 10.7 Combination of individual phase Markov models
10.2.2 Repairable Phased-mission Systems

The methods presented so far have only accounted for non-repairable phased-mission systems. In some practical situations, it may be possible for maintenance to be performed on a system. In this case, assessment techniques are required which are able to take into consideration the repair process occurring during the mission phases.

10.2.2.1 Combinatorial Approaches

Somani [52] extended the work of Trivedi and Somani [46] and presented combinatorial approaches for the solution of repairable components in a multi-phased mission. The approach they presented considers the situation where a component can only be repaired if it is not required in a particular phase. When a component is required for the successful operation of a phase, repair cannot be initiated. Four possible cases are considered for the component in any phase: the component may begin in the working or failed state at the start of the phase, and may end in either the working or failed state at the end of the phase. Probabilities for each case are obtained based on the assumption that the component failure and repair times follow an exponential distribution.

Another combinatorial approach is presented by Vaurio [53] who considers calculations for the system unavailability and failure intensity for each phase of the mission separately. This method does not model the dependencies that arise in the situation of repairable systems. The phase unavailability and failure intensity will be approximations. Also the phase calculations involved do not account for the outcome of previous phases.

10.2.2.2 Markov Methods

Phase algebra is no longer applicable when components can be repaired during a multi-phased mission. In such circumstances, combinatorial methods cannot adequately deal with problem solution as they do not account for when the repairs are made, which may be in a later phase than when the failure occurred. The Markov approach can provide a solution for repairable phased-mission systems, and has been the focus of a substantial amount of research.

Early investigations into the use of Markov methods to solve phased mission problems were carried out by Clarotti et al [54] who used a homogeneous model. This method establishes new initial conditions for the start of each phase, and requires the entire mission to be solved using phase Markov models with the same state space.
Homogeneous Markov models are also used to solve phased-mission systems where the phase durations are random variables [55, 56, 57]. Mura and Bondavalli [58] investigated situations where phases have a pre-determined time duration, but the next phase to be performed is chosen depending on the system state at the end of the current phase. A two-level analysis method of state-dependent phase sequences is presented by Mura and Bondavalli, in which the higher level method models the structure of the mission with regard to only the pattern of phases, and the lower level method models the configuration of the individual phases. A probabilistic phase transition model is constructed to represent the state dependencies between phases. Each lower-level model is solved in the order of the phase sequences in the upper-level model, where the initial state probability vector for each phase is obtained by application of the appropriate transition model to the state probabilities at the end of the preceding phase. The upper-level model can then be solved to evaluate parameters of interest.

A non-homogeneous Markov method is presented by Smotherman and Zemoudeh [59] to overcome the limitations of homogeneous Markov models. The approach is based on a single non-homogeneous Markov model in which the concept of state transition is extended to include globally-time-dependent phase changes. Phase change times are specified using non-overlapping distributions with phase duration functions that are zero outside assigned time intervals ordered according to the phase sequence. The generalized non-homogeneous Markov model is also able to model globally-time-dependent failure and repair rates.

10.2.2.3 Minimal Markov Method

Applying the Markov method to assess repairable phased-mission systems may produce an explosion in the model state space which is usually based on all components in the system. However, during some phases in the mission, only some of the components may be relevant to the system failure. This feature provides an option to reduce the model size by considering the phases individually and has resulted in the minimal Markov method [60].

For the minimal Markov method, the smallest possible Markov models are formed for each individual phase. The analysis carried out over each phase duration utilises a Markov model formed by considering only components which are functionally required in the particular phase \(i\).

To facilitate the modelling process, initially all components are considered and a full set of states for the total mission is produced. For each phase the model is reduced by considering only components relevant to that phase. Appropriate initial conditions are determined. Once the phase
model is analyzed, the results are expanded out again in terms of the full model to enable the
calculation of the initial conditions for the immediately succeeding phases. The expansion of the
reduced phase component state probabilities into the full state probabilities are realised by
multiplying the reduced state probabilities with other excluded component availabilities and
unavailabilities at the end of the phase. The probability that any component \( c \) which is not
required in phase \( j \) being in failed state at the end of phase \( j \) is calculated through equation 10.14
and 10.15 depending on whether it features a revealed or dormant failure.

\[
q_c(t_j) = \frac{\lambda_c}{\lambda_c + \nu_c} \left[ 1 - e^{- (\lambda_c + \nu_c) t_j} \right] \tag{10.14}
\]

\[
q_c(t_j) = 1 - e^{-\lambda_c (t_j - k\theta_c)} \tag{10.15}
\]

where in equation 10.15, \( k\theta_c \) is the time when the last inspection is carried out on
component \( c \) prior to the end of phase \( i \), i.e. \( t_i \).

Then the reduction process is carried out on the full Markov model at the beginning of the
analysis for the succeeding phase \( j \) by removing components which are not required in the phase.
Initial state probabilities in the minimal Markov model for phase \( j \) are obtained through the
summation of the probabilities of all full states that contribute to each of the reduced states. This
method leads to a sequence whereby the full model is reduced for the analysis of each phase and
then expanded to give a full set of state probabilities following the phase analysis. Figure 10.8
displays the algorithm underlying the minimal Markov method.

![Algorithm for the minimal Markov method](image)

Figure 10.8 Algorithm for the minimal Markov method
An example phased-mission system is used to illustrate how the minimal Markov method applies. The system is composed of three components A, B and C, which all feature a revealed failure and are maintained throughout the mission. The system failure logic in each individual phase is shown in figure 10.9.

The reliability analysis of this phased-mission system is carried out as follows:

A full Markov model is constructed based on all components as shown in figure 10.10.

Assuming all components working at the start of the mission, the initial state probabilities for the full model are \( P[0] = [1, 0, 0, ..., 0] \).

The analysis for each phase is carried out in turn.

**Phase 1**

Only components A and B are required in phase 1. A reduced Markov model is constructed for phase 1 as shown in figure 10.11 by combining the states 1 and 4, states 2 and 6, states 3 and 7 and states 5 and 8 in the full model shown in figure 10.10. To distinguish from the states in the full model, the states in phase 1 model is attached with the sub-script ‘1’.
In the reduced model shown in figure 10.11, state 4 is the failed system state and therefore made absorbing. The initial state probabilities for phase 1 are also updated as $P[0] = [1, 0, 0, 0]$.

Then the quantification is carried out on the reduced Markov model for phase 1. The probability of each state at the end of phase 1, i.e. $t_1$, is obtained. Accordingly, the system failure probability during phase 1 is $Q_4(t_1)$.

**Phase 1 → Phase 2**

In preparation for the analysis of phase 2, the phase 1 model is expanded back to the full states by the inclusion of component C as state 1 is expanded to states 1 and 4 in the full model, state 2 is expanded to states 2 and 6 in the full model, state 3 is expanded to states 3 and 7 in the full model, and state 4 is expanded to states 5 and 8 in the full model. The probability of each state in the full model is the multiple of the original state probability in the phase 1 model and the availability/unavailability of component C at the end of phase 1. That is,

$$
\begin{align*}
Q_1(t_1) &= Q_1(t_1) \cdot [1-q_C(t_1)]; \\
Q_2(t_1) &= Q_2(t_1) \cdot [1-q_C(t_1)]; \\
Q_3(t_1) &= Q_3(t_1) \cdot [1-q_C(t_1)]; \\
Q_4(t_1) &= Q_4(t_1) \cdot q_C(t_1); \\
Q_5(t_1) &= Q_5(t_1) \cdot [1-q_C(t_1)]; \\
Q_6(t_1) &= Q_6(t_1) \cdot q_C(t_1); \\
Q_7(t_1) &= Q_7(t_1) \cdot q_C(t_1); \\
Q_8(t_1) &= Q_8(t_1) \cdot q_C(t_1);
\end{align*}
$$

where: $q_C(t_1) = \frac{\lambda_c}{\lambda_c + v_c} \left[1 - e^{-(\lambda_c + v_c) t_1}\right]$

Among the 8 system states in the full model, the system fails if it arrives in states 5 and 8 during phase 1 (absorbing) and consequently as the mission will fail they should not be considered for phase 2 analysis. The system works in state 6 during phase 1 but fails in phase 2. Accordingly, the phase 1 to phase 2 transition failure probability is $Q_6(t_1)$. Therefore, in forming the initial
state probabilities for phase 2, the probabilities of states 5, 6 and 8 at the end of phase 1 are set as '0'.

**Phase 2**

Only components A and C are relevant during phase 2, so component B needs to be removed from the full model to form the minimal Markov model for phase 2. Accordingly, states 1 and 3, 2 and 5, 4 and 7 and states 6 and 8 in the full model are combined. The minimal Markov model for phase 2 is then constructed as shown in figure 10.12. State 42 is the failed system state and therefore made absorbing.

The initial state probabilities for phase 2 are then determined as $P[t_1] = [Q_1(t_1)+Q_3(t_1), Q_2(t_1), Q_4(t_1)+Q_7(t_1), 0]$. The quantification on the Markov model in figure 10.12 produces the probability of each state at the end of phase 2, i.e. $t_2$. System failure probability during phase 2 is obtained as $Q_{42}(t_2)$.

**Phase 2 → Phase 3:**

In preparation for the analysis of phase 3, the reduced Markov model for phase 2 is expanded back to full states by including component B. State 12 is expanded to states 1 and 3 in the full model, state 22 is expanded to states 2 and 5 in the full model, state 32 is expanded to states 4 and 7 in the full model, and state 43 is expanded to states 6 and 8 in the full model. The probability of each state in the full model at the end of phase 2 is:

$$Q_1[t_2] = Q_{11}(t_2).[1-q_B(t_2)]; Q_2[t_2] = Q_{21}(t_2).[1-q_B(t_2)];$$

$$Q_3[t_2] = Q_{31}(t_2).q_B(t_2); Q_4[t_2] = Q_{41}(t_2).[1-q_B(t_2)]$$

$$Q_5[t_2] = Q_{51}(t_2).q_B(t_1); Q_6[t_2] = Q_{61}(t_2).[1-q_B(t_2)]$$

$$Q_7[t_2] = Q_{71}(t_2).q_B(t_2); Q_8[t_2] = Q_{81}(t_2).q_B(t_2)$$

where: $q_B(t_2) = \frac{\lambda_B}{\lambda_B + \nu_B} \left[ 1 - e^{-(\lambda_B + \nu_B)t_2} \right]$
Among the 8 states in the full model, the system fails in states 6 and 8 during phase 2, and works in state 7 in phase 2 but fails in phase 3. Phase 2 to phase 3 transition failure probability is then obtained as $Q_7(t_2)$. Consequently, the system cannot reside in states 6, 7 and 8 at the start of phase 3 analysis and their state probabilities at the end of phase 2 are set as ‘0’ in preparation for analysis of phase 3.

**Phase 3**

In phase 3, only components B and C are relevant. Component A is removed from the full model to form the minimal Markov model for phase 3. States 1 and 2, 3 and 5, 4 and 6 and states 7 and 8 in the full model are combined. The minimal Markov model for phase 3 is shown in figure 10.13.

![Minimal Markov model for phase 3 in example phased-mission](image)

The initial state probabilities for phase 3 are determined according to the full state probabilities at the end of phase 2 as $P(t_2) = P(Q_1(t_2) + Q_2(t_2), Q_3(t_2) + Q_5(t_2), Q_4(t_2), 0)$. The quantification on the reduced Markov model for phase 3 produces the state probabilities at the end of phase 3, i.e. $t_3$. As phase 3 is the last phase in the mission, the mission success probability is then obtained as the sum of the success state probabilities at the end of phase 3, i.e. $R_{\text{MISS}} = Q_1(t_3) + Q_2(t_3) + Q_3(t_3)$.

**10.2.3 Review of Existing Solutions to Phased-mission Analysis**

Much research has been carried out to investigate the reliability of phased-mission systems. For non-repairable systems, significant advances include the basic event transformation, the transformation of the multi-phased mission into a single-phase mission, the cut set cancellation technique, approximation techniques, the development of phase algebra, the use of binary decision diagrams, non-coherent fault tree analysis methods which are able to produce the phase failure parameters and the Markov method. For phased-mission systems where component maintenance and repair is possible, combinatorial approaches are available which can provide the approximate value of mission unreliability. However, the main techniques used for the
solution of this type of problem are homogeneous and non-homogeneous Markov methods. These methods are aimed at capturing the repair process during the mission, the system state-dependent behaviour, the random phase durations and mission time-dependent component failure and repair rates. The minimal Markov method provides an efficient alternative to the full Markov method. It is able to establish the smallest possible Markov model for each phase according to the specific phase failure logic and therefore avoids the state space explosion problem by which the applicability of the Markov method is significantly weakened.

Little research has been undertaken into the solution of phased mission systems where component inter-dependence exists during phases in the mission. Dependencies could exist, for example, due to the use of standby components, sequential failures, the existence of initiating and enabling events and the maintenance dependency existing during the repair process. Therefore, a general method is required to account for the dependency relationships existing between components during the mission phases in order to produce the accurate mission unreliability parameters. This will be discussed in Chapter 11.
Chapter 11. Dependency Modelling in Phased-mission Systems

11.1 Introduction

In many multi-phased missions, the system may feature complex relationships which give rise to dependencies between individual components. To obtain an accurate prediction of the mission reliability, it is necessary to take the dependencies into account in the modelling. So far little research has been carried out on this aspect. In this chapter, a generic methodology is presented to perform the reliability assessment of phased-mission systems with inter-dependencies between components. The technique is also extended to consider phased-mission systems where the maintenance of components is possible during the mission.

Assumptions

Before the modelling approach is described, assumptions are given for the basic characteristics of the phased-mission system under investigation:

a) The mission fails if the system fails during any phase.

b) Phase transitions are instantaneous.

c) Phase durations are pre-determined.

d) Repair of components is possible during the mission whether the component failure contributes to the system failure or not in a specific phase.

e) Components have constant failure rates.

f) Both phase reliability and mission reliability predictions are of interest.

11.2 Phased-mission Systems Involving Dependencies

In this section, issues concerned with phased-mission systems featuring dependencies are highlighted to indicate the necessity of developing a new methodology and what features the new methodology should achieve. These issues are:

a). The existing techniques assume independence between components in the modelling of phased-mission systems. The new methodology must overcome this deficiency and incorporate the ability to model dependency relationships.

b). Some phased-mission systems are non-repairable, some are fully repairable and other systems are partially-repairable where only some of the components can be maintained during the mission. For non-repairable systems, the 'basic event transformation' [43] is carried out to identify the phase in which components fail. However, for repairable or partially-repairable
systems, this technique is no longer appropriate. The new methodology must be able to account for the influence of component behaviour during previous phases and flexible enough to analyse different types of phased-mission systems.

c). Modelling dependency relationships together with the repair process determines that the Markov method needs to be used in the analysis. This has the potential to result in state space explosion if the full Markov model is used. The minimal Markov method [60] is no longer applicable to overcome this as the expansion process assumes the independence between components not required in the phase. The new methodology should be able to deal with the dependency problem and at the same time solve the explosion problem.

d). A phased-mission system can feature different maintenance characteristics during different phases. It might be the case that during some phases no maintenance can be carried out and the independence assumption can also hold. In this case, the Markov method would be an inefficient quantification tool, and other tools which are based on the independence assumption will be a better alternative. The new methodology should be able to utilise the most efficient analysis tool depending on the characteristics of each phase.

e). In some phased-mission systems, dependency relationships are self-contained in independent subsystems. The analysis efficiency can be improved if the new methodology can identify these subsystems as independent modules through all the phases.

f). When systems are non-repairable, some analysis techniques provide the causes of system failure for the mission, while other techniques can provide implicants for each individual phase failure. However, for repairable or partially-repairable systems, the phase algebra developed to produce the phase failure modes for non-repairable systems is no longer applicable. In the new methodology, it is necessary to obtain implicants for each phase failure which can take into account component behaviour during previous phases.

g). The causes of system failure will change from phase to phase in a phased-mission. This can result in failure on transition to a phase. The phase transition failure is a different mechanism from the system failure which occurs during the phase at the time of a component failure. It is of interest to identify the conditions for phase transition failures and obtain the phase transition failure probabilities in addition to those which occur during the phase.
11.3 Generic Modified Phase Algebra Method

Considering the issues discussed above, a generic modified phase algebra method has been developed. This will enable the dependency modelling of phased-mission systems to acquire system reliability measures. Before the method is described in detail, an overview of the algorithm underlying the modified phase algebra is provided in the following section.

11.3.1 Repair Process and Phase Algebra

One important problem in phased-mission system analysis is how to account for the component behaviour during previous phases in the current phase analysis. Several existing methods for non-repairable systems use a phase algebra [45, 46, 47, 50]. It assesses the current phase taking into account component failures during previous phases which once failed do not change state. The algebra is based on the ‘basic event transformation’ [43] which expresses that the existence of a component C in the failed state in phase \( j \) can be caused by the failure of component C during any phases from phase 1 to phase \( j \). At the system level for each phase, the system failure is investigated considering component failures which occurred in the previous phases and the current phase using the basic event transformation.

When maintenance is possible during the mission on some or all the components, the representation of phase algebra must be modified. The influence of the repair process is examined at two levels: basic event level and system level.

**At basic event level**

If a component C is repairable during the mission, then when investigating phase \( j \), what really matters is the state of component C at the point that the mission proceeds into phase \( j \) and the behaviour of component C during phase \( j \). As the phase transition is assumed to be instantaneous, the state of component C at the point of transition into the phase \( j \) is the state of component C at the end of phase \( j-1 \). The notation \( C_{(i)} \) is used to represent that component C is in a failed state at the end of phase \( i \). It is not necessary to consider the history of failure and repair of component C during the phase.

**At system level**

The system failure in a phase \( j \) is investigated in a different way depending on whether maintenance is available for components whose functionality is relevant to the success of the phase. It is considered as follows:
a) No maintenance on components: in this case, once the component fails, it remains in the failed state throughout the mission. The system failure during the phase is investigated in the same way as for non-repairable phased-mission systems. The non-repairable basic event transformation is carried out on components whose functionality is relevant to the phase and accounts for the effect of component failures in previous phases. That is, that the component $C$ is failed in phase $j$ is expressed as $C_1 + C_2 + ... + C_j$.

b) Maintenance is carried out on some or all of the phase-relevant components. The possibility of maintenance means that the state of the component can change one or more times during phase $j$. In this case, the non-repairable basic event transformation is no longer appropriate. A new way to account for component failures in previous phases needs to be established. This is achieved by investigating the system failure in each phase, $j$, from two perspectives. The first is that the system failure logic for phase $j$ has been satisfied in previous phases, i.e. at the end of phase $j-1$ the system is residing in a state in which the system failure logic for phase $j$ is met. The second perspective is that the system successfully enters phase $j$ and fails during phase $j$. With regard to the second situation, it must account for all the required phase failure conditions to exist at the same time where the component states can be continually changing. Since the mission terminates once the system fails during the phase, repair is not carried out and so the failed system state will remain till the end of the phase. Therefore, when considering if the system has failed during a certain phase, it is required to investigate the system state at the end of the phase, i.e. whether the combination of component states at the end of the phase satisfies the system failure logic for the phase. To illustrate this, a simple example is used. Assume that during a certain phase the system fails when both components $A$ and $B$ fail. Figure 11.1 displays the behaviour of components $A$ and $B$ during the phase.
In figure 11.1, mode 1 illustrates that although both components A and B have failed during the phase, the system does not fail as the components are not in the failed state at the same time, i.e. the system failure logic has never been met during the phase. However, in mode 2, the failure of component B following the failure of component A satisfies the system failure logic. Consequently, the mission terminates and since the components are not repaired the system remains in the failed state with components A and B both failed till the end of the phase. That is, in the example, the minimal cut set used to express the system failure during the phase \( j \) is \( A_jB \). In the example, it is demonstrated that when components are repairable during the phase, a way to judge whether the system fails during the phase is to investigate the system state at the end of the phase.

A special situation arises which means that maintenance on the component is not always performed in all the phases. When the failure of a single component \( C \) is sufficient to bring about the system failure during phase \( j \), the component \( C \) will not get repaired as once it fails, the mission terminates. In this case, component \( C \) is regarded as 'non-repairable' during phase \( j \), and in this case the basic event transformation is carried out in the form: \( C(j-1) + C \).

### 11.3.2 General Process of Modified Phase Algebra Method

This section describes the modified phase algebra method. For illustrative purposes, it is assumed that for each phase \( j \) in the mission, its time duration is defined as \([t_{j-1}, t_j)\) \((j = 1, 2, \ldots)\). The method is carried out through the following steps.

**Step 1: Acquire Comprehensive System Information.**

a). Establish the fault tree which represents the system failure for each individual phase. At this stage, component failures are included in each phase fault tree structure without the specification...
of the phase number in which they fail. During this process, gates which appear in more than one phase fault tree are assigned the same label so that they can be identified as an independent module in the later modularization process.

b). Establish the dependency information for the phased-mission system. As different dependency relationships may exist among the same group of components during different phases, information is specified for each dependency group to indicate the phase(s) during which this dependency relationship exists.

c). Establish the basic event information. For each basic event which represents a component failure, information is required as to whether the maintenance can be carried out during the mission. Also information is required on the component failure characteristics with regard to each phase from phase 1 to the last phase in which the component influences the system performance.

Step 2: Identify Modules.
Independent modules are identified from the phase fault trees across all the phases. Sections identified as modules must be modules for each phase fault tree in which any of the components that the module contains appear. This is achieved in the following way:

a). Construct a combined mission fault tree by grouping the phase fault trees as inputs to an OR logic gate. With this structure the factorization and modularization process can be carried out using one fault tree and one coherent process which avoids having to cross-reference between different phase fault trees.

b). Identify modules and factors in the combined fault tree by following the pre-processing procedures described and illustrated in section 5.2. During the pre-processing, during the fault tree simplification stage, the contraction of the top event of any phase fault tree, i.e. the level immediately below the top event in the created combined fault tree, is not permitted in order to keep the failure logic for each phase intact.

c). Where appropriate, replace the basic events in the combined fault tree with the identified modules; and extract each individual phase fault tree from the combined fault tree structure by eliminating the top OR gate.
With this procedure, modules can then be identified across all the phases, and the smallest possible modules then identified for each dependency relationship in the system. This solves issue e) in section 11.2.

Step 3: Phase Fault Tree Transformation.

For each phase \( j \), each basic event is expanded to account for the phases in which it can have failed (as discussed in section 11.3.1). Prior to the transformation, it must be determined if, for the phase under investigation, maintenance can be performed on each component. If during phase \( j \), a single failure of a component occurs which on its own will bring about the system failure, maintenance is not carried out on that component during phase \( j \). Different approaches are taken for the phase fault tree transformation depending on whether the maintenance is carried out during the phase or not as it influences how the system failure during the phase is expressed through combination of individual component failures. These two situations are described below:

a). No maintenance on relevant components during phase \( j \): this includes situations where components are non-repairable during the whole mission or have restricted maintenance resulting from their single order contribution to the phase failure. For components which are non-repairable over the mission, the phase fault tree transformation is carried out by transforming each relevant basic event to account for component failures which occur in all previous phases up to phase \( j \). This is expressed in equation 11.1:

\[
C = C_1 + \ldots + C_{j-1} + C_j
\]

11.1

where \( C_i \) represents that component \( C \) fails during phase \( i \).

For components which feature the restricted maintenance, the transformation only has to consider the state in which component \( C \) enters phase \( j \), i.e. the state of component \( C \) at the end of phase \( j-1 \) and the component behaviour during phase \( j \). This is expressed in equation 11.2:

\[
C = C_{(j-1)} + C_j
\]

11.2

In some cases, the component may feature restricted maintenance in successive phases from phase \( k \) to phase \( j \). To represent this, the transformation is carried out as in equation 11.3:

\[
C = C_{(k-1)} + C_k + \ldots + C_j
\]

11.3

b). Maintenance is carried out on relevant components during phase \( j \): in this circumstance, the two situations which result in system failure during a phase are investigated separately. The system either fails when the system failure logic for phase \( j \) has been satisfied at the end of phase
The left input to the top event in figure 11.2 represents the phase transition failure which occurs as soon as the mission enters phase $j$. This intermediate event is developed from the original phase fault tree structure established in Step 1 by investigating the state of each relevant component at the end of phase $(j-1)$, i.e. $C_{(j-1)}$.

When the component is non-repairable during the whole mission, the event $C_{(j-1)}$ can be further expanded into a series of basic events as:

$$C_{(j-1)} = C_1 + C_2 + \ldots + C_{j-1}$$  \hspace{1cm} 11.4

Or when the component features restricted maintenance during phase $j-1$, the expansion is performed as:

$$C_{(j-1)} = C_{(j-2)} + C_{j-1}$$  \hspace{1cm} 11.5

The right input to the top event in figure 11.2 represents an in-phase failure. It is also developed based on the original phase fault tree structure by investigating the component states at the end of phase $j$, as the contribution that the failure of these components during phase $j$ makes to the system unreliability during the phase is reflected by the state of the component at the end of phase $j$, i.e. $C_{(j)}$. When there is partial maintenance during phase $j$, i.e. only some of the required components are maintained, for those non-repairable, the basic event $C_{(j)}$ is further expanded in the same way as in equation 11.4.

When a phase fault tree contains a module, it has to be established whether the module is repairable or not during the phase to determine how the phase fault tree transformation will be
performed. This does not simply depend on whether components included in the module are repairable. The module is treated as non-repairable during the phase when it satisfies one of the following conditions:

- All components included in the module are non-repairable during the mission; OR
- The failure of the module is sufficient to bring about the system failure in the phase. This is a similar case to the restricted maintenance of components from first-order minimal cut sets.

The phase fault tree transformation process establishes a method which accounts for component failures which occurred in previous phases for both repairable and non-repairable components. Phase transition failures are clearly identified and integrated into the phase analysis.

Step 4. Simplification of Transformed Phase Fault Tree
The phase fault tree transformation conducted in the previous step replaces the single component failure with a series of basic events adding complexity to the phase fault tree structure. Therefore, the contraction, extraction and elimination processes are carried out on each transformed phase fault tree to take advantage of any simplification that can be achieved on the phase fault tree structure.

Step 5: Establish the Mission BDD
The mission BDD represents causes of system unreliability over the whole mission period by considering the system failure in each phase in turn. In the mission BDD the terminal node which represents the system failure also indicates in which phase the system failure occurs. The terminal node is labelled $1[i]$, which means that the system fails during phase $i$. Consequently, by following the path which leads to a terminal node $1[j]$ in a minimal mission BDD, the prime implicants for the system failure during each phase $j$ can be obtained. The construction of the mission BDD is carried out through the following procedures:

a). Decide the basic event ordering for each phase fault tree: for a general approach, the modified left-right top-down method [16] can be used, where basic events are considered in the order from the top to the bottom level in the fault tree and from left to right at the same level with the priority given to those with higher repeated appearances. An extra condition applies in that basic events which represent component failures in earlier phases are placed prior to basic events related to later phases in the ordering.
b). Produce the overall basic event ordering for the whole mission: this can be achieved by taking the orderings obtained for each phase fault tree and listing them one after the other in the order of the phases. For basic events which appear in the ordering more than once, their position is fixed by their first occurrence and later occurrences are deleted from the list.

c). Construct the mission BDD according to the established overall basic event ordering. The process progresses through each phase in turn. The terminal node 1[i] is added when it is determined that the system fails in phase i as a result of component failures included in the path. Figure 11.3 displays the algorithm for constructing the mission BDD given the pre-determined ordering.

An example phased-mission system is used to illustrate the process of constructing the mission BDD. The system failure logic for each phase is listed as follows:

Phase 1: A.B (no maintenance)
Phase 2: A.C (no maintenance)

With phase fault tree transformation:
Phase 1: $A_1 B_1$
Phase 2: $(A_1 + A_2). (C_1 + C_2)$

The basic event ordering for each phase is obtained as:

Basic event ordering for phase 1: $A_1, B_1$
Basic event ordering for phase 2: $A_1, C_1, A_2, C_2$

Obtain the overall mission ordering as: $A_1, B_1, C_1, A_2, C_2$

The mission BDD is then established as shown in figure 11.4 according to the algorithm displayed in figure 11.3.

[Diagram of the mission BDD for the example system]

Step 6: Obtain Disjoint Phase Failure Modes.
The Boolean algebra representation of the system failure during phase $i$ can be obtained by tracking paths in the mission BDD leading to the terminal node $1[i]$.

Take for example the mission BDD in figure 11.4, the Boolean algebra representation for system failure in phase 1 is obtained by following paths leading to terminal node $1[1]$ in the BDD, which results in: $A_1 B_1$. In the same way, four paths can be identified leading to the terminal node $1[2]$, which produce the Boolean algebra representation for system failure in phase 2 as $A_1 B_1 . C_1 + A_1 B_1 . C_1 . C_2 + A_1 . C_1 . A_2 + A_1 . C_1 . A_2 . C_2$. As the BDD in figure 11.4 is in its minimal form, the disjoint paths provide the prime implicants for system failure in each phase. The mission BDD provides an efficient way to obtain implicants for system failure in each individual phase without having to construct the non-coherent fault tree for each phase.

294
Step 7: Determine the Quantification Technique for Each Phase

As has been discussed in issue d) in section 11.2, the Markov method may not be the most appropriate quantification technique for each phase. In this step, a procedure is established to determine the appropriate quantification method according to the specific characteristics of each phase:

a). Identify component failures relevant to the phase under investigation from the corresponding phase fault tree. If a module is contained in the phase fault tree, it needs to be determined whether the module can be quantified separately in the phase analysis. In general, a module can be quantified separately if:

- The module is irrelevant to the system failure during phase $j$; OR
- The module is regarded as non-repairable during phase $j$ as it is a single-order failure event (as determined in Step 3);

If a module cannot be quantified separately, the module has to be replaced with its basic event descendants during the quantification.

b). Determine the appropriate quantification model for the phase.

- when no statistical dependency exists between relevant component failures AND no repair is carried out on relevant components during the phase: the Markov method is not necessary and the disjoint paths established from the mission BDD can be used.
- Otherwise, the Markov method is used.

Step 8: Simplify the Boolean Algebra Representations

Reductions are performed on the Boolean algebra for system failure in each individual phase obtained from Step 6. To achieve this, the following procedures are implemented in turn:

a). Delete events which do not need to be included in the analysis of the current phase $j$: assume phase $m$ ($m < j$) is the most recent phase for which the Markov method is used as the quantification tool, then any events which are related to component failures prior to phase $m$ are deleted as the probabilities of these events will have been taken into account in determining the initial state probabilities for phase $m$ and should not be accounted for in phase $j$. 
b). Apply the following rules to the Boolean algebra representation for the current phase $j$:

$$
\begin{align*}
\overline{A_i.A_{i+1}} \cdots \overline{A_k} &= \overline{A_{i,k}} \\
A_i + A_{i+1} + \cdots + A_k &= A_{i,k} \\
\overline{A_i.A_{i+1}} \cdots \overline{A_{k-1}.A_k} &= A_k \\
A_{i,k}A_i &= A_i \\
A_{i,k}A_k &= 0 \\
A_{i,k}A_{i,k} &= A_i \\
\overline{A_i.A_i} &= 0 \\
\overline{A_i.A_{i,k}} &= A_{i+1,k} \\
\overline{A_{(k-1)}.A_k} &= (A_k | \overline{A_{(k-1)}}) \overline{A_{(k-1)}} \\
\overline{A_{(k-1)} \cdot A_k} &= (A_k | \overline{A_{(k-1)}}) \overline{A_{(k-1)}} \\
A_{(k-1)}.A_{(k)} &= (A_{(k)} | \overline{A_{(k-1)}}) \overline{A_{(k-1)}} \\
\overline{A_{(k-1)} \cdot A_{(k)}} &= (A_{(k)} | \overline{A_{(k-1)}}) \overline{A_{(k-1)}} \\
\overline{A_{(k-1)} \cdot \overline{A_{(k)}}} &= (A_{(k)} | \overline{A_{(k-1)}}) \overline{A_{(k-1)}} \\
\end{align*}
$$

where the event $A_{i,k}$ represents that component $A$ fails during any phase between phase $i$ and phase $k$ inclusive ($i < k \leq j$).

All the equations contained in the equation set 11.5 correspond to the situation where component $A$ is non-repairable during the mission. The two equations included in equation set 11.6 are applicable when component $A$ is repairable during the whole mission but features a restricted maintenance period during phase $k$. And equations contained in equation set 11.7 apply when component $A$ is repairable during the whole mission.

In terms of the equation set 11.6, the simplification of the algebra $\overline{A_{(k-1)} \cdot A_k}$ can be carried out in another way as: $\overline{A_{(k-1)} \cdot A_k} \Rightarrow A_k$ as the event $A_k$ represents that component $A$ fails during phase $k$, implying that component $A$ enters phase $k$ in a working state. This may lead to a problem during
the quantification process. For components which are non-repairable during the whole mission, a consistent probability density function$^1$ can be defined for the component over the whole mission, and thus the probability of event $A_k$ can be easily obtained as $\int_{-\infty}^{t} f(u)du$. However, in the case where the component is maintained during the whole mission except phase $k$, it is very difficult to define a consistent probability density function for the component over the whole mission period. Consequently, to obtain the probability of $\overline{A_{(k-1)} \cdot A_k}$, the following transformation is carried out:

$$\overline{A_{(k-1)} \cdot A_k} \Rightarrow A_k \Rightarrow (A_k \mid \overline{A_{(k-1)}}) \cdot \overline{A_{(k-1)}}$$

where $(A_k \mid \overline{A_{(k-1)}})$ represents that component A fails during phase $k$ with component A entering phase $k$ in a working state.

The probability density function for component A during phase $k$ given that the component is working at the start of phase $k$ can be defined separately. In the quantification process at the later stage, the probability of $\overline{A_{(k-1)} \cdot A_k}$ can then be obtained as:

$$P[\overline{A_{(k-1)} \cdot A_k}] = \int_{-\infty}^{t} f_{A_k}(u)du \cdot P[\overline{A_{(k-1)}}] \quad 11.8$$

where $f_{A_k}(t)$ is the probability density function of component A for phase $k$ given that component A is working at the start of phase $k$.

Accordingly, the probability of $\overline{A_{(k-1)} \cdot A_k}$ is then obtained as in equation 11.9.

$$P[\overline{A_{(k-1)} \cdot A_k}] = \{1 - \int_{0}^{t} f_{A_k}(u)du\} \cdot P[\overline{A_{(k-1)}}] \quad 11.9$$

In terms of the equation set 11.7, the transformation accounts for the initial conditions of component A in phase $k$ when component A is maintained during the phase. To illustrate the influence, consider a single component A with two states: working and failed. The probability of the event $A_{(k-1)} \cdot A_k$, $P(A_{(k-1)} \cdot A_k)$, is obtained by quantifying the Markov model over the time duration $[t_{k-1}, t_k]$ with an initial state probability vector $P(working, failed \ (AND \ component \ A \ entered \ phase \ k \ failed)) = [0, P(A_{(k-1)})]$, while $P(\overline{A_{(k-1)} \cdot A_k})$ is calculated over the same time.

$^1$ It is assumed in this research that all component failures feature the exponential distribution.
duration with an initial state probability vector \( P[\text{working, failed} \ (\text{AND component A entered phase } k \text{ working})] = [1-P(A_{k-1}), 0]. \)

c). Eliminate repeated Boolean algebra terms which can result from the reduction procedures.

Take for example the mission BDD in figure 11.4, the simplification of the phase algebra representation for phase 2 results in the simplified phase algebra as

\[
A_1 \cdot B_1 \cdot C_1 + A_1 \cdot B_1 \cdot C_2 + A_2 \cdot C_1 + A_2 \cdot C_2.
\]

Step 9: Phase and Mission Unreliability Quantification
The quantification process is carried out for each phase in turn to obtain the phase unreliability depending on which quantification method is adopted.

a). when the BDD quantification is applied, the phase unreliability can be directly obtained by calculating the probability of the corresponding phase failure modes. The phase unreliability is then equal to the sum of the probabilities of all the phase failure modes represented by the Boolean algebra.

Take for example the example phased-mission system used in step 5,

\[
Q[\text{phase 1}] = P[A_1 \cdot B_1] = \int_0^f f_A(u)du \cdot \int_0^f f_B(u)du
\]

\[
Q[\text{phase 2}] = P[A_1 \cdot B_1 \cdot C_1 + A_1 \cdot B_1 \cdot C_2 + A_2 \cdot C_1 + A_2 \cdot C_2]
= \int_0^f f_A(u)du \cdot [1- \int_0^f f_B(u)du] \cdot \int_0^f f_C(u)du + \int_0^f f_A(u)du \cdot [1- \int_0^f f_B(u)du] \cdot \int_0^f f_C(u)du
+ \int_0^f f_A(u)du \cdot \int_0^f f_C(u)du + \int_0^f f_A(u)du \cdot \int_0^f f_C(u)du
\]

b). when the Markov method is used for the quantification of the phase, the following procedures are implemented to enable the development of a minimal Markov model which can avoid the state-space explosion problem as well as correctly model the system behaviour during the phase.

1). Determine which components need to be included in the Markov model to be constructed: components which are contained in the simplified phase algebra representation need to be included in the model for the purpose of determining the initial state probabilities. If a module is involved in the phase and cannot be quantified separately, all the basic event
descendants contained in the module also have to be explicitly included in the Markov model.

2). Determine initial states and initial state probabilities for the Markov model. The initial states are states of the component in which it is possible for the system to start the current phase, and are used to generate the complete Markov model. These initial states are determined by the states in which the system was residing at the end of the preceding phase. With the simplified phase algebra representation derived from previous steps, the analyst can identify possible system states at the end of the preceding phase by investigating events contained in each phase failure mode which are related to the preceding phase. Assume that component C is relevant during phase \( j \), i.e. the current phase under investigation, and the \( k \)th term of the phase algebra expression for phase \( j \) failure contains the event \( C_{(j-1)} \). This means that in the initial state developed from the \( k \)th algebra term, the state of component C is failed as indicated by \( C_{(j-1)} \). In some cases, some phase failure modes do not contain any event which relates component C to the preceding phase. This means that component C can enter phase \( j \) in either the working or the failed state. As such, two initial states can be developed from this failure mode where the states of other components are as specified in the failure mode, while component C is working in one and failed in the other. Thus, by looking into each term in the phase algebra, an exhaustive list of initial states for phase \( j \) can be developed.

Initial state probabilities can then be determined according to the specific combination of component states. A general rule applies as follows in calculating the initial state probabilities:

- When the preceding phase \( j-1 \) is quantified using the failure mode generated from the mission BDD, the probability of the initial state developed from the \( k \)th failure mode for phase \( j \) is obtained as:

\[
Q = \prod_{n} \left[ \frac{1}{n} \int_{0}^{1} f_{C_{n}}(u)du \cdot \int_{1}^{\frac{1}{n}} 1 - \int_{0}^{1} f_{C_{n}}(u)du \right]
\]

where, as indicated in the \( k \)th failure mode, component \( C_{n} \) is failed in phase \( j-1 \), represented by event \( C_{n} \), and component \( C_{i} \) has worked through phase \( j-1 \), represented by event \( \overline{C_{i}} \).
When phase \( j-1 \) is quantified using the Markov method, the probability for the initial state developed from the \( k \)th failure mode is calculated as:

\[
Q_m = \sum_m Q_m(t_{j-1}) \cdot \prod_n P[C_n] \cdot \prod_j \{1 - P[C_s]\}
\]

where \( Q_m(t_{j-1}) \) represents the probability of state \( m \) at the end of phase \( j-1 \), in which the combination of component states is consistent with that indicated in the \( k \)th failure mode for phase \( j \); components \( C_n \) and \( C_s \) are required in phase \( j \) but not in phase \( j-1 \) (thus will not be included in the Markov model for phase \( j-1 \)), and as indicated in the \( k \)th failure mode, component \( C_n \) is failed at the end of phase \( j-1 \), and component \( C_s \) is functional at the end of phase \( j-1 \).

In the process of determining the initial state probabilities, attention needs to be given to 'Phase Transition Failure'. This concept is due to the change in the system configuration across different phases, and may result in situations where the system fails as soon as the mission progresses into phase \( j \). In this case, the component failures have occurred during earlier phases, but the system does not show its impact until a certain phase \( j \) is reached. Such a mission failure is defined as the 'Transition Failure' for phase \( j \).

The transition failure for phase \( j \) (denoted by \( TRF_j \)) can be identified from the phase algebra representation for phase \( j \). A failure mode term in the phase algebra represents a latent mission failure for phase \( j \) if the component failure events contained in the term are all related to previous phases. For example, consider the failure modes for phase 2 in the example phased-mission system in step 5, among the four phase algebra terms \( A_1 \cdot \bar{B}_1 \cdot C_1, A_1 \cdot \bar{B}_1 \cdot C_2, A_2 \cdot C_1 \) and \( A_2 \cdot C_2 \), the first term represents a transition failure for phase 2 as component failure events contained in this term, \( A_1 \) and \( C_1 \), are both related to phase 1. The phase algebra term which represents a phase transition failure is not considered when establishing initial states for the Markov model for phase \( j \) as the system will not start phase \( j \) in such states.

The calculation of the initial state probabilities becomes complex when some basic events contained in the phase algebra relate to component failures which are only relevant during some of the phases in the mission. For example if component \( C \) is not required until phase \( j \), then the probability of component \( C \) being working or failed at the end of phase \( j-1 \) can be determined from relevant component failure models dependent upon whether component \( C \) is
repairable or not. In the case that component $C$ is non-repairable during the mission, the probability that it is failed at the end of phase $j-1$ can be easily obtained as:

$$Q_c(t_{j-1}) = \int_0^{t_{j-1}} f_c(u) du$$  \hspace{1cm} (11.14)

If component $C$ is repairable during the mission, and assume its failure times are exponentially distributed, then the probability of component $C$ being failed at the end of phase $j-1$ can be obtained as in equation 11.15.

$$Q_c(t_{j-1}) = \frac{\lambda_c}{\lambda_c + \nu_c} \left[ 1 - e^{-(\lambda_c + \nu_c) t_{j-1}} \right] \text{ (revealed failure)}$$

$$Q_c(t_{j-1}) = 1 - e^{-\lambda_c(t_{j-1} - t_s)} \text{ (dormant failure)}$$  \hspace{1cm} (11.15)

where $t_s$ is the time when the last inspection prior to $t_{j-1}$ is carried out on component $C$.

3). Delete irrelevant basic events from the established initial states. Basic events which relate to components not required during phase $j$ are deleted from the initial states so that a minimal Markov model can be constructed for phase $j$ at a later stage. The reduction may need special attention during the quantification process conducted in a later phase. If component $C$ is last relevant in phase $j-1$ and now phase $m$ ($m > j$) is considered, then component $C$ would be excluded from the analysis of phase $j$, and will not have been included in the phase model from phase $j$ to phase $m-1$. This means that for the analysis of phase $m$, the probability of component $C$ being working or failed at the beginning of phase $m$ has to be determined separately. Two situations are considered:

First, component $C$ is independent from other component failures throughout the whole mission: in this case, equation 11.15 can be applied to calculate the probability of component $C$ being failed at the end of phase $m-1$.

Second, component $C$ is involved in dependency relationships during phases prior to phase $j$: this means that equation 11.15 is no longer applicable. Its failure probability can be derived as shown in equation 11.16.

$$Q_c(t_{m-1}) = Q[C(m-1)|C^{(0)}] \cdot Q[C^{(0)}] + Q[C(m-1)|\overline{C}^{(0)}] \cdot Q[\overline{C}^{(0)}]$$  \hspace{1cm} (11.16)

where the event $C^{(0)}$ represents that component $C$ is failed at the start of phase $j$, i.e. $C^{(j)} = C_{j-1}$.
Equation 11.16 expresses the failure probability of component C at the end of phase \( m-1 \) in terms of its failure probability at the start of phase \( j \). This means that the influence of dependency relationships involving component C prior to phase \( j \) can be taken into account.

In order to calculate the two conditional probabilities in equation 11.16, two cases are considered depending on whether component C is repairable or not during the mission:

- If component C is non-repairable:
  \[
  Q[C_{(m-1)} | C_{(j)}] = 1; \\
  Q[C_{(m-1)} | \bar{C}_{(j)}] = \int_{j}^{m-1} f_C(u) \, du 
  \]
  where \( f_C(u) \, du \) is the probability density function defined for component C from the start of phase \( j \) to the end of phase \( m-1 \) with component C working at the start of phase \( j \) assuming its failure times are exponentially distributed.

- If component C is repairable:
  \[
  Q[C_{(m-1)} | C_{(j)}] = \frac{\lambda_C}{\lambda_C + \nu_C} \left[ 1 - e^{-\lambda_C (t_m - t_j)} \right] \quad \text{(revealed failure)} \\
  Q[C_{(m-1)} | \bar{C}_{(j)}] = 1 - e^{-\lambda_C (t_m - t_j)} \quad \text{(dormant failure)}
  \]
  where \( t_j \) is the time when the last inspection is carried out on component C prior to \( t_{m-1} \).

\( Q[C_{(m-1)} | C_{(j)}] \) can be obtained by quantifying a Markov model established for component C during the time duration \([t_{j-1}, t_m]\) with an initial state probability \( q_C \text{ working} = 0, q_C \text{ failed} = 1 \).

\( Q[C_{(0)}] \) and \( Q[\bar{C}_{(j)}] \) in equation 11.16 respectively stand for the probability that component C is failed or working at the start of phase \( j \) given that the system is able to start phase \( j \). The sum of all initial state probabilities for phase \( j \) gives the probability that the system is able to start phase \( j \). As such, \( Q[C_{(0)}] \) and \( Q[\bar{C}_{(j)}] \) are obtained as:
\[ Q[C^{(j)}] = \frac{P[C \text{ failed at } t_{j-1}] \cdot P[\text{system able to start phase } j]}{P[C \text{ working at } t_{j-1}] \cdot P[\text{system able to start phase } j]} \]

\[ = \frac{\sum_{i=1}^{n} Q_i(t_{j-1})}{\sum_{i=1}^{s} Q_i(t_{j-1})} \]

where \( n \) is the total number of initial states for phase \( j \); and \( s \) the number of initial states for phase \( j \) in which component \( C \) is failed;

\[ Q[C^{-j}] = \frac{P[C \text{ working at } t_{j-1}] \cdot P[\text{system able to start phase } j]}{P[C \text{ working at } t_{j-1}] \cdot P[\text{system able to start phase } j]} \]

\[ = \frac{\sum_{i=1}^{p} Q_i(t_{j-1})}{\sum_{i=1}^{n} Q_i(t_{j-1})} \]

where \( n \) is the total number of initial states for phase \( j \); and \( p \) the number of initial states for phase \( j \) in which component \( C \) is working;

Therefore, when a component \( C \) is deleted from the analysis of phase \( j \), \( Q[C^{(j)}] \) and \( Q[C^{-j}] \) are calculated and stored for later reference. Then when the analysis proceeds into phase \( m \) and component \( C \) has to be re-included in the analysis, \( Q[C^{(j)} | C^{(0)}] \) and \( Q[C^{-j} | C^{(0)}] \) can be obtained according to equations 11.17 and 11.18. Together according to equation 11.16, the probability of component \( C \) working or failed at the start of phase \( m \) can then be derived without having to keep component \( C \) in the analysis of phases during which its functionality is not required. This enables the development of a minimal Markov model for each individual phase.

4). Trim the list of established initial states. In the preceding procedure, components which are not required in the current phase are deleted from the initial states. When the only difference between two initial states is due to the state of deleted components, the deletion will result in same states. In this procedure, these same initial states will be combined into one, and the probability of the combined initial state is equal to the sum of the individual ones.
5. Develop the minimal Markov model from the established initial states. When dependency relationships are involved in the current phase, the algorithm for the development of the Markov model for each type of dependency presented in Chapter 6 are referred to. Failed system states are absorbing in the development of the model.

6. Carry out the quantification on the established Markov model. The phase unreliability is then obtained as:

\[ Q_{[\text{phase } j]} = Q_{\text{LMF} j} + \sum_k Q_k(t_j^-) \]  \hspace{1cm} (11.21)

where \( Q_{\text{LMF} j} \) represents the probability of transition failure for phase \( j \), which can be obtained by calculating the probability of the corresponding phase algebra terms; and \( Q_k(t_j^-) \) represents the probability of being in state \( k \) at the end of phase \( j \) which is a system failed state.

Step 10. obtain the mission unreliability by summing up the phase failure probabilities. That is:

\[ \bar{R}_{\text{MSS}} = \sum_i Q_{[\text{phase } i]} \]  \hspace{1cm} (11.22)

11.3.3 Importance Measures in Phased-mission Systems

Importance measures provide an indicator of the contribution that a component failure makes to the mission failure in each individual phase.

11.3.3.1 Phase Criticality Function \( G_{i,j}(q(t)) \)

Assume phase \( j \) is the current phase under investigation; and component \( i \) is required during the phase. The criticality function for component \( i \) during phase \( j \), denoted by \( G_{i,j}(q(t)) \), is obtained by different means depending on which type of model is used for the quantification of phase \( j \).

- when the BDD quantification is applied for phase \( j \): the criticality function for component \( C \) during phase \( j \) is calculated directly referring to its the definition \( G_{i}(q(t)) = Q_{\text{sys}(1_i)} - Q_{\text{sys}(0_i)} \) (equation 2.19). That is:

\[ G_{i,j}(q(t)) = P\{\text{system fails during phase } j \mid q_{ij} = 1\} - P\{\text{system fails during phase } j \mid q_{ij} = 0\} \]  \hspace{1cm} (11.23)
Since the criticality function $G_{i,j}(q(t))$ is aimed at measuring the contribution of the component failure during phase $j$ (which implies that the component enters phase $j$ in a working state), the phase algebra terms are ignored which contain the component having failed during previous phases. Take for example the phased-mission system in figure 11.4:

**Phase 1:** required components $A$ and $B$ with phase algebra $A_1.B_1$:

- $G_{A,1}(q(t)) = \int f_a(u)du \ (0 \leq t < t_1)$
- $G_{B,1}(q(t)) = \int f_b(u)du \ (0 \leq t < t_1)$

**Phase 2:** required components $A$ and $C$ with phase algebra $A_1.\overline{B_1}.C_1 + A_1.B_1.C_2 + A_2.C_1 + A_2.C_2$:

- $G_{A,2}(q(t)) = \int f_a(u)du + \int f_c(u)du \ (t_1 \leq t < t_2)$

(where algebra terms $A_1.\overline{B_1}.C_1$ and $A_1.B_1.C_2$ are ignored as they both contain the basic event $A_1$ which means that component $A$ has already failed during phase 1 and thus does not need to be considered with regard to its contribution to the system failure during phase 2).

- $G_{C,2}(q(t)) = \int f_a(u)du \cdot [1 - \int f_a(u)du] + \int f_a(u)du \ (t_1 \leq t < t_2)$

(where algebra terms $A_1.\overline{B_1}.C_1$ and $A_2.C_1$ are ignored as they both contain the basic event $C_1$ and do not need to be considered regarding the contribution of component $C$ to the system failure during phase 2).

- when the Markov method is used for the quantification of phase $j$, the criticality function for component $C$ is obtained in the same way as defined in equation 7.12. That is:

$$G_{i,j}(q(t)) = \sum_m Q_m(t) + \sum_k Q_k(t) \quad (t_{j-1} \leq t < t_j)$$

where in the Markov model for phase $j$, both states $m$ and $k$ represent the critical state for component $i$. In state $m$, component $i$ is working, whilst in state $k$, component $i$ is failed.

### 11.3.3.2 Phase Criticality Measures

The phase criticality function is an importance measure in its own right and indicates the susceptibility of the system to the failure of each component in each phase. For a system to fail, it needs to be in a critical condition for the $i$th component and also the component fails. If this contribution is then divided by the phase failure probability, it produces the proportion of times
that the failure of component $i$ causes the phase failure, i.e. a relative measure of the contribution
that component $i$ makes to the phase failure. Two scenarios are considered [61]:

1). For any phase the system can be in a critical state for a component $i$ in phase $j$ and
then component $i$ fails during the phase to bring about the system failure (In-phase
Importance). OR
2). The failure conditions for phase $j$ occurred prior to phase $j$ and the system failure
occurs on transition to phase $j$ (Transition Importance). For a component $i$ to contribute to
transition failure, the system is in a critical state for component $i$ with respect to phase $j$
in a phase prior to phase $j$ and component $i$ also fails prior to phase $j$.

These two scenarios give rise to two measures of component importance for each phase.

11.3.3.2.1 In-phase Criticality Measure $I_{i,j}^p$

The in-phase criticality measure for component $i$ in phase $j$, denoted by $I_{i,j}^p$, is defined as in
equation 11.25:

$$I_{i,j}^p = \frac{G_{i,j} q_{ij}}{Q[phase j]}$$

Equation 11.25 gives the general definition of in-phase criticality measure and is applicable in
phases where the BDD quantification is applied. When the analysis of phase $j$ is performed using
the Markov method, the in-phase criticality measure for component $i$ can be calculated as
expressed in equation 11.26:

$$I_{i,j}^p = \frac{\sum_{i=1}^{n} Q_i (u) \lambda_i du}{Q[phase j]}$$

where $n$ is the number of critical states for component $i$ in the Markov model for phase $j$;
and $\lambda_i$ is the conditional failure rate of component $i$.

All parameters in equations 11.25 and 11.26 will have been obtained in the previous
quantification process.
11.3.3.2 Transition Criticality Measure \( I_{i-j}^T \)

The transition criticality measure, denoted by \( I_{i-j}^T \), represents the contribution that the failure of component \( i \) during phases prior to phase \( j \) makes to the transition failure on entering phase \( j \). By examining each failure mode for phase \( j \), derived from the mission BDD, which represents a transition failure for phase \( j \), information can be obtained on which component failures contribute to the transition failure on phase \( j \) and which components do not need to be considered with respect to the transition criticality measure.

The transition criticality measure of component \( i \) is equal to zero if the failure of component \( i \) in phase \( k \) \((k = 1, 2, ..., j-1)\) makes no contribution to the transition failure for phase \( j \). Only component failure events, which are contained in the phase transition failure modes for phase \( j \), need to be considered for the transition criticality measure in phase \( j \).

The transition criticality function, denoted by \( G_{i-j,k}^T \), is introduced to facilitate the calculation of the transition criticality measure for component \( i \). The transition criticality function provides the importance of failure of component \( i \) during phase \( k \) \((k < j)\) with respect to the transition failure on entering phase \( j \). It is defined as in equation 11.27.

\[
G_{i-j,k}^T = \frac{\partial Q_{TRF_j}}{\partial q_{ik}}
= Q_{TRF_j} \mid \text{component } i \text{ fails in phase } k - Q_{TRF_j} \mid \text{component } i \text{ functions through phase } k
\]

Correspondingly, the transition criticality measure for component \( i \) with respect to phase \( j \) is obtained as in equation 11.28.

\[
I_{i-j}^T = \frac{\sum_{k=1}^{j-1} G_{i-j,k}^T q_{ik}}{Q[\text{phase } j]} = \frac{\sum_{k=1}^{j-1} \frac{\partial Q_{TRF_j}}{\partial q_{ik}} q_{ik}}{Q[\text{phase } j]}  
\]

With the in-phase criticality measure and the transition criticality measure, the total importance contribution of failure of component \( i \) with respect to phase \( j \) is obtained as in equation 11.29.

\[
I_{ij} = I_{i-j}^T + I_{i-j}^I
\]
Take for example the mission BDD in figure 11.4, among the four failure modes for phase 2, \(A_1 \overline{B}_1 . C_1 + A_1 . \overline{B}_1 . C_2 + A_2 . C_1 + A_2 . C_2\), the failure mode \(A_1 \overline{B}_1 . C_1\) represents a transition failure for phase 2. Phase 1 is the only phase prior to phase 2. Accordingly, the transition criticality function for components A, B and C are calculated as follows:

\[
G^T_{A-2,1} = Q_{TRF} \mid A_1 - Q_{TRF} \mid \overline{A}_1 \\
= (1-q_{B_1}) \cdot q_{C_1} \\
G^T_{B-2,1} = 0 \\
G^T_{C-2,1} = Q_{TRF} \mid C_1 - Q_{TRF} \mid \overline{C}_1 \\
= q_A \cdot (1-q_{B_1})
\]

The transition criticality measure for each component can then be obtained according to equation 11.28.

### 11.3.3.3 Mission Importance Contribution

The importance measures discussed so far are all about component failures with respect to a particular phase. A measure is introduced here to indicate the total contribution that the failure of component \(i\) makes to the whole mission failure [7]. Denoted by \(I_i\), this importance measure is defined in equation 11.30.

\[
I_i = \frac{\sum_{all \ phase \ j} \left( I_{i-j}^T + I_{i-j}^T \right) \cdot Q[phase \ j]}{Q_{MISS}} \\
= \frac{\sum_{all \ phase \ j} \left( G_{i-j} \cdot q_{i_j} + \sum_{k=1}^{i-1} G^T_{i-j,k} \cdot q_{i_k} \right)}{Q_{MISS}} \\
= \frac{\sum_{all \ phase \ j} \left( \frac{\partial Q[phase \ j]}{\partial q_{i_j}} q_{i_j} + \sum_{k=1}^{i-1} \frac{\partial Q_{TRF_i}}{\partial q_{i_k}} q_{i_k} \right)}{Q_{MISS}}
\]

11.3.3.4 Fussell-Veseley Measure of Importance

This measure is defined as the probability of the union of the minimal cut sets containing the failure of component \(i\) given that the system has failed. Putting it into the context of a phased-mission, this measure can be obtained for each component \(i\) failure during each individual phase
$j$, denoted by $I_{i-j}^{F-V}$, the measure provides the contribution made by the prime implicants containing the failure of component $i$ during phase $j$ to the system failure during phase $j$. It is defined as equation 11.31:

$$I_{i-j}^{F-V} = \frac{P\left(\bigcup_{k \in e_k} PI_{k-j}\right)}{Q[\text{phase } j]}$$

where $PI_{k-j}$ are prime implicants for phase $j$ failure which contain the failure of component $i$.

When phase $j$ is quantified using the BDD quantification method, $PI_{k-j}$ are equivalent to the phase failure modes which contain the failure of component $i$ in phase $j$ if the mission BDD is constructed in its minimal form. When phase $j$ is quantified using the Markov method, the Fussell-Vesely measure of importance is obtained as in equation 11.32:

$$I_{i-j}^{F-V} = \frac{\sum Q_k}{Q[\text{phase } j]}$$

where state $k$ represents states in the Markov model for phase $j$ in which the system is failed and component $i$ fails in phase $j$.

11.3.4 Summary of the Modified Phase Algebra Method

With the process described above, the modified phase algebra method is able to account for the dependency relationships existing in phased-mission systems. Due to the construction of the mission BDD, all possible failure modes can be clearly identified for each phase failure (prime implicants can be extracted if the mission BDD is in its minimal form). Causes of phase transition failures can also be identified and their probabilities can be calculated separately. BDD quantification and the Markov method are applied to the phase analysis depending on whether maintenance is carried out during the phase or any dependency is involved. This improves the analysis efficiency compared with the conventional Markov method applied to repairable phased-mission systems. For each phase to which the Markov method applies, the minimal Markov model is established by removing non-required components. This also contributes to improving the analysis efficiency. Importance measures can also be obtained to determine the criticality and contribution of each component failure to the phase and mission failure. An example phased-mission system is given in Chapter 12 to demonstrate how the modified phase algebra method is applied.
Chapter 12. Application of Modified Phase Algebra Approach

12.1 Example Phased-mission System
The 4-phase mission system is composed of components A, B, C, D, E, F, G and H. Sub-system 1 consists of components B, C and D. Component C serves as the standby component for component B, component D activates C when B fails. Sub-system 2 consists of components F, G and H, and only when components F, G and H fail in this specific order, will the sub-system 2 fail. Components A and E are required during phase 1; components A and sub-system 1 are required during phase 2; sub-system 1 is required during phase 3; and components A and sub-system 2 are required during phase 4. All components are repairable during the mission. Among them, each of sub-systems 1 and 2 is maintained by a single engineer.

The system fails in each phase as illustrated in figure 12.1. For the mission to complete successfully, the system must function through each of the 4 phases over the time duration \([0, t_1), [t_1, t_2), [t_2, t_3)\) and \([t_3, t_4)\) respectively.

12.2 Application Process

12.2.1 Phase and Mission Unreliability
Step 1: Acquire System Information
a). The fault trees representing the phase failure logic are shown in figure 12.1.

Note: basic event CS represents the standby failure of component C; while basic event CF represents the active failure of component C.

Figure 12.1 Phase fault trees for example system
b). Establish the dependency information for the system as displayed in table 12.1. In this phased-mission system, maintenance dependency exists between components B, C and D through phases 1, 2 and 3, and also between components F, G and H through the whole mission. The sequential dependency exists between components F, G and H through the whole mission. During phases 2 and phase 3 in which the subsystem 1 is required, the standby dependency exists between basic events B and CF; and initiator-enabler dependency exists between basic events B and CS and D.

<table>
<thead>
<tr>
<th>Dep. group no.</th>
<th>Dep. Type</th>
<th>Number 1</th>
<th>Number 2</th>
<th>List 1</th>
<th>List 2</th>
<th>Involved phases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sdbyl</td>
<td>1</td>
<td>1</td>
<td>B</td>
<td>CF</td>
<td>2, 3</td>
</tr>
<tr>
<td>2</td>
<td>i-e</td>
<td>-</td>
<td>2</td>
<td>B</td>
<td>CS</td>
<td>2, 3</td>
</tr>
<tr>
<td>3</td>
<td>i-e</td>
<td>-</td>
<td>2</td>
<td>B</td>
<td>D</td>
<td>2, 3</td>
</tr>
<tr>
<td>4</td>
<td>sq</td>
<td>-</td>
<td>4</td>
<td>-</td>
<td>F, G, H</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>5</td>
<td>mnc</td>
<td>3</td>
<td>1</td>
<td>B, CF, CS, D</td>
<td>-</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>6</td>
<td>mnc</td>
<td>3</td>
<td>1</td>
<td>F, G, H</td>
<td>-</td>
<td>1, 2, 3, 4</td>
</tr>
</tbody>
</table>

Table 12.1 Dependency information for example phased-mission system

c). Establish the basic event data as shown in table 12.2:

<table>
<thead>
<tr>
<th>Basic event name</th>
<th>Relevant to which phases</th>
<th>Maintenance (Repairable or non-repairable)</th>
<th>Failure model</th>
<th>Failure parameters</th>
<th>Enabler</th>
<th>Dependency groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1, 2, 4</td>
<td>Repairable</td>
<td>Phase 1</td>
<td>( \lambda_A )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 2</td>
<td>( \lambda_A, \nu_A )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 3</td>
<td>( \lambda_A, \tau_A, \theta_A )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 4</td>
<td>( \lambda_A )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>2, 3</td>
<td>Repairable</td>
<td>Phase 1</td>
<td>( \lambda_B', \tau_B, \theta_B )</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 2</td>
<td>( \lambda_B, \nu_B )</td>
<td>0</td>
<td>1, 2, 3, 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 3</td>
<td>( \lambda_B, \nu_B )</td>
<td>0</td>
<td>1, 2, 3, 5</td>
</tr>
<tr>
<td>CS</td>
<td>2, 3</td>
<td>Repairable</td>
<td>Phase 1</td>
<td>( \lambda_{CS}, \tau_{CS}, \theta_{CS} )</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 2</td>
<td>( \lambda_{CS}, \tau_{CS}, \theta_{CS} )</td>
<td>1</td>
<td>2, 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 3</td>
<td>( \lambda_{CS}, \tau_{CS}, \theta_{CS} )</td>
<td>1</td>
<td>2, 5</td>
</tr>
<tr>
<td>CF</td>
<td>2, 3</td>
<td>Repairable</td>
<td>Phase 2</td>
<td>( \lambda_{CF}, \nu_{CF} )</td>
<td>0</td>
<td>1, 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 3</td>
<td>( \lambda_{CF}, \nu_{CF} )</td>
<td>0</td>
<td>1, 5</td>
</tr>
<tr>
<td>D</td>
<td>2, 3</td>
<td>Repairable</td>
<td>Phase 1</td>
<td>( \lambda_D, \tau_D, \theta_D )</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 2</td>
<td>( \lambda_D, \tau_D, \theta_D )</td>
<td>2</td>
<td>3, 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Phase 3</td>
<td>( \lambda_D, \tau_D, \theta_D )</td>
<td>2</td>
<td>3, 5</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>Repairable</td>
<td>Phase 1</td>
<td>( \lambda_E )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

311
Step 2: Identify Modules

a). Construct a combined mission fault tree as shown in figure 12.2 by including individual phase fault trees as inputs to an OR gate.

Figure 12.2 Combined fault tree for example phased-mission system
b). Identify modules by carrying out the simplification and modularization processes on the combined mission fault tree structure. The contraction is not carried out on gates G1 and G6 in figure 12.2 in order to reserve the structure of individual phase fault tree. Module M1 led by gate G3 and module M2 led by gate G7 are identified.

c). Restore the phase fault tree structure, as shown in figure 12.3, where basic events are replaced with corresponding modules.

![Figure 12.3 Modularized phase fault tree for example phased-mission system](image)

Step 3: Carry out the Phase Fault Tree Transformation

**Phase 1**

No maintenance is carried out on components A and E during phase 1 as the single failure of either of the two components is able to bring about the system failure. In this case, the phase fault tree transformation takes place through the basic event transformation:

\[
A \rightarrow A_1 \\
E \rightarrow E_1
\]

**Phase 2**

During this phase, components A, B, C and D are all repairable. Module M1 is also repairable as its single failure will not bring about the system failure. With the repair process, the system failure in phase 2 must be considered from two perspectives: ‘failure conditions satisfied at the end of phase 1’ OR ‘system fails during phase 2’.

In terms of the first perspective, it means that both component A and module M1 are failed at the end of phase 1, i.e. \(A_{(1)}\) AND \(M1_{(1)}\) respectively. Since component A is non-repairable during phase 1, \(A_{(1)}\) is replaced by \(A_1\). For the second perspective, since maintenance is carried out on both component A and module M1 during the phase, whether the system fails during phase 2 is judged by investigating the states of component A and module M1 at the end of the phase, i.e. \(A_{(2)}\) AND \(M1_{(2)}\). The fault tree transformation for phase 2 is shown as follows:

\[
A.M1 \rightarrow A_1.M1_{(1)} + A_{(2)}.M1_{(2)}
\]
**Phase 3**

As the failure of module M1 is sufficient to bring about the system failure during phase 3, it is considered as non-repairable as a whole during the phase. The system fails in phase 3 either when module M1 is failed at the end of phase 2 or module M1 fails during phase 3. Accordingly, the phase transformation is carried out as:

\[ M1 \rightarrow M1(2) + M1_3. \]

**Phase 4**

During the phase, either the failure of component A or module M2 is sufficient to bring about the system failure. Consequently, maintenance is not carried out on either component A or module M2 as a whole. The fault tree transformation is implemented through the basic event transformation as follows:

\[ A \rightarrow A(3) + A_4, \]
\[ M2 \rightarrow M2(3) + M2_4 \]

The transformed phase fault trees are shown in figure 12.4:

![Fault Tree Diagram](image)

Figure 12.4 Transformed phase fault trees for example phased-mission system

**Step 4. Simplify Transformed Phase Fault Trees.**

In figure 12.4, in the fault tree for phase 4, basic events \( A(3), A_4, M2(3) \) and \( M2_4 \) are contracted up to the level immediately below the top event.

**Step 5. Construct the Mission BDD**

a). Decide the basic event ordering for each phase

Phase 1: \( A_1, E_1 \)

Phase 2: \( A_1, M1(1), A_2, M1_2 \)

Phase 3: \( M1_2, M1_3 \)

Phase 4: \( A(3), M2(3), A_4, M2_4 \)
b). Establish the combined ordering: $A_1 > E_1 > M_{1(1)} > A_{(2)} > M_{1(2)} > M_{13} > A_{(3)} > M_{2(3)} > A_4 > M_{24}$

c). Establish the mission BDD as shown in figure 12.5.

![Mission BDD for example phased-mission system](image)

Figure 12.5 Mission BDD for example phased-mission system

Step 6: Obtain Disjoint Failure Modes for Each Phase

By tracing the paths leading to the terminal nodes representing the system failure in a particular phase, the disjoint failure modes for each phase are obtained as follows in the form of Boolean algebra.

**Phase 1:** $A_1 + \overline{A_1}E_1$

**Phase 2:** $\overline{A_1} \cdot \overline{E_1} \cdot A_{(2)} \cdot M_{1(2)}$

**Phase 3:** $\overline{A_1} \cdot E_1 \cdot A_{(2)} \cdot M_{1(2)} \cdot M_{13} + \overline{A_1} \cdot E_1 \cdot A_{(2)} \cdot M_{1(2)} + \overline{A_1} \cdot E_1 \cdot A_{(2)} \cdot M_{1(2)} \cdot M_{13} \cdot A_{(3)} \cdot M_{2(3)}$

**Phase 4:**

$\overline{A_1} \cdot E_1 \cdot A_{(2)} \cdot M_{1(2)} \cdot M_{13} \cdot A_{(3)} + \overline{A_1} \cdot E_1 \cdot A_{(2)} \cdot M_{1(2)} \cdot M_{13} \cdot A_{(3)} \cdot M_{2(3)} \cdot A_4 \cdot M_{24}$

$\overline{A_1} \cdot E_1 \cdot A_{(2)} \cdot M_{1(2)} \cdot M_{13} \cdot A_{(3)} + \overline{A_1} \cdot E_1 \cdot A_{(2)} \cdot M_{1(2)} \cdot M_{13} \cdot A_{(3)} \cdot M_{2(3)} \cdot A_4 \cdot M_{24}$

$\overline{A_1} \cdot E_1 \cdot A_{(2)} \cdot M_{1(2)} \cdot M_{13} \cdot A_{(3)} + \overline{A_1} \cdot E_1 \cdot A_{(2)} \cdot M_{1(2)} \cdot M_{13} \cdot A_{(3)} \cdot M_{2(3)} \cdot A_4 \cdot M_{24}$

315
Step 7: Determine the Appropriate Phase Quantification Model

Phase 1
The BDD quantification method is adopted as maintenance is not carried out and events are independent.

Phase 2
The Markov method is employed as the maintenance is carried out during the phase and in addition statistical dependency exists between components B, C and D.

Phase 3
The Markov method is again employed for the quantification of this phase due to the statistical dependency between events and since maintenance is conducted on components B, C and D during the phase.

Phase 4
no maintenance is carried out during phase 4. As module M2 can be quantified separately and no statistical dependency exists between component A and module M2, the direct numerical solution is adopted to address the system unreliability during phase 4.

Step 8: Simplify the Boolean Algebra Representations for Each Phase

Phase 1: $A_1 + \overline{A_1}E_1$

Phase 2: $\overline{E_1} \cdot (A_{(2)} | \overline{A_1}) \cdot \overline{A_1} \cdot M_{1(2)}$

Phase 3:
a). as the Markov method is determined as the quantification model for phase 2, any events contained in the phase failure modes which relate to phase 1 are deleted, such as events $\overline{A_1}$ and $\overline{E_1}$.
b). Equation 11.6 applies:

$$M_{1(2)} \cdot M_{1_3} = (M_{1_3} | M_{1(2)}) \cdot M_{1(2)}$$

The simplified phase failure modes for phase 3 is displayed as follows:

$$A_{(2)} \cdot (M_{1_3} | M_{1(2)}) \cdot M_{1(2)} + \overline{A_{(2)}} \cdot (M_{1_3} | M_{1(2)}) \cdot M_{1(2)} + \overline{A_{(2)}} \cdot M_{1(2)}$$
Phase 4:

a). as the Markov method is employed as the quantification method for phase 3, any events which relate to phase 1 or phase 2, such as $A_1$, $A_2$, $A_3$, $E_1$ and $M_1(2)$, are deleted from the phase failure modes.

b). Equation 11.6 applies:

$$A_3 ^{M_2(3)} A_4 = (A_4 | A_3 ^{M_2(3)}) A_3$$

$$A_3 ^{M_2(3)} A_4 = (A_4 | A_3 ^{M_2(3)}) A_3$$

$$M_2(3) ^{M_2(3)} M_2 = (M_2 | M_2 ^{M_2(3)}) M_2$$

The simplified phase failure modes for phase 4 is displayed as follows:

$$M_1 ^{M_2(3)} A_3 + M_1 ^{M_2(3)} A_3 ^{M_2(3)} + M_1 ^{M_2(3)} M_2 ^{M_2(3)} (A_4 | A_3 ^{M_2(3)}) A_3$$

Step 9: Perform the Phase and Mission Quantification

Phase 1

It is determined in step 7 that the BDD is the appropriate quantification method for phase 1. The phase failure probability is obtained through the sum of the probability of each phase failure mode.

$$Q_{\text{phase 1}} = \int_0^t f_A(u)du + [1 - \int_0^t f_A(u)du \cdot \int_0^t f_E(u)du]$$

where $f_A(u)$ and $f_E(u)$ are probability density functions defined respectively for components A and E over the time duration $[0, t_1]$ given that both components A and E working at the start of the phase, i.e. $t=0$.

As both components A and E feature an exponential failure distribution during phase 1, then

$$Q_{\text{phase 1}} = \left(1 - e^{-\lambda_A t}\right) + e^{-\lambda_A t} \left(1 - e^{-\lambda_E t}\right)$$

Phase 2

The Markov method is employed as the appropriate quantification method. The following procedures are taken to obtain the phase failure probability.
1). Determine basic events which need to be considered to establish initial states: by referring to the phase failure modes (Step 6) for phase 2, it can be determined that basic events A, E and module M1 need to be included. Since module M1 cannot be quantified separately during phase 2, it is replaced by its basic event descendants. Therefore, basic events which need to be considered are: A, E, B, CS, D, CF

2). Determine initial states and initial state probabilities: as is indicated in the phase failure mode for phase 2, as far as components A and E are concerned, the events \( \overline{A_i} \) and \( \overline{E_i} \) indicate that the system enters phase 2 with both components A and E working. In terms of module M1, the phase failure mode does not contain any event which represents its state at the end of phase 1. This means that module M1 can reside in any possible state when the system enters phase 2. Therefore, to obtain an exhaustive list of initial states for phase 2, module M1 has to be modelled over phase 1 to determine what states it may reside at the end of phase 1 and corresponding state probabilities.

Since the maintenance dependency exists within module M1 during phase 1, a Markov model is established to model module M1 over the time duration \([0, t_1]\). The quantification is carried out on the established Markov model to determine the state probabilities at the end of phase 1. Tables 12.3 and 12.4 display the Markov model for module M1 over phase 1. As basic event CF is not relevant during phase 1, it is not considered.

<table>
<thead>
<tr>
<th>state no.</th>
<th>B</th>
<th>CS</th>
<th>D</th>
<th>CF</th>
<th>M1</th>
<th>State no.</th>
<th>B</th>
<th>CS</th>
<th>D</th>
<th>CF</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>19</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>21</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>0</td>
<td>22</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>23</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>24</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>1</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>26</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>-</td>
<td>0</td>
<td>27</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>28</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>29</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>1</td>
<td>30</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>31</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>-</td>
<td>1</td>
<td>32</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>-</td>
<td>1</td>
<td>33</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>34</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>0</td>
<td>35</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>0</td>
<td>36</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 12.3 states included in the Markov model of module M1 in phase 1

\(^1\) State code '0' – working; code '1' – failed and revealed; code '2' – failed, revealed and queuing for repair; code '3' – failed and unrevealed.
Note: the rate represented by the name of a basic event refers to its conditional failure rate; the rate represented by '-' followed with the name of a basic event refers to its conditional repair rate.

Table 12.4 State transitions in the Markov model for module M1 for phase 1

Each state in table 12.3 is passed to phase 2 as the state of module M1 at the start of phase 2. In states where the component B is failed and both components C and D are functional, component C is switched on to replace component B as the system enters phase 2. Accordingly, the initial states for phase 2 are determined and displayed in table 12.5.

<table>
<thead>
<tr>
<th>State No.</th>
<th>A</th>
<th>E</th>
<th>B</th>
<th>CS</th>
<th>D</th>
<th>CF</th>
<th>System No.</th>
<th>A</th>
<th>E</th>
<th>B</th>
<th>CS</th>
<th>D</th>
<th>CF</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Notes:
- when B is failed at the end of phase 1 and component C and D are both functional, component C will be activated by D as soon as the mission enters phase 2.
- If a component is repairable during phase j, inspection is always carried out on the component when the system enters phase j.

Table 12.5 initial states for phase 2

The relationship between the states of module M1 at the end of phase 1 and its initial states at the start of phase 2 are displayed in table 12.6.
State at the end of phase 1 (table 11.5) | Initial state at the start of phase 2 (table 11.7)
---|---
1 | 1
2 + 5 | 2
3 + 8 | 3
4 + 10 | 4
6 + 11 + 13 | 5
7 + 12 + 15 | 6
9 + 17 + 18 | 7
14 + 21 + 22 + 23 + 24 + 31 | 8
16 + 25 | 9
19 + 28 | 10
20 + 30 | 11
26 + 27 + 32 + 33 | 12
29 + 34 + 35 + 36 | 13

Table 12.6 Relationship between states of module M1 at the end of phase 1 and the beginning of phase 2

Then the initial state probabilities for phase 2 can be determined as follows:

\[ Q_1(t_1) = Q_1(t_1-).[1- \int_0^t f_A(u)du ].[1- \int_0^t f_E(u)du ] \]
\[ Q_2(t_1) = [Q_2(t_1-)+Q_5(t_1-)].[1- \int_0^t f_A(u)du ].[1- \int_0^t f_E(u)du ] \]
\[ Q_3(t_1) = [Q_3(t_1-)+Q_8(t_1-)].[1- \int_0^t f_A(u)du ].[1- \int_0^t f_E(u)du ] \]
......
\[ Q_{10}(t_1) = [Q_{29}(t_1-)+Q_{34}(t_1-)+Q_{35}(t_1-)+Q_{50}(t_1-)].[1- \int_0^t f_A(u)du ].[1- \int_0^t f_E(u)du ] \]

where \( Q_i(t_1) \) represents the probability of the initial state \( i \) for phase 2; \( Q_i(t_1-) \) represents the probability of state \( j \) at the end of phase 1;

3). Delete irrelevant components from established initial states: component E is not required during phase 2 and therefore deleted from the initial states. As component E is not required for the rest of the mission, there is no need to obtain \( q_{E21} \).

4). Trim the initial state list: no repeated states result from the deletion of the basic event E.

5). Develop the minimal Markov model for phase 2 from the refined initial states as shown in table 12.7 and 12.8.
Table 12.7 The Markov model for phase 2

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 A</td>
<td>14</td>
<td>-A</td>
<td>14 B</td>
<td>25</td>
<td>0</td>
<td>43</td>
<td>21</td>
<td>-A</td>
</tr>
<tr>
<td>1 B</td>
<td>15</td>
<td>-B</td>
<td>15 CS</td>
<td>26</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>1 CS</td>
<td>16</td>
<td>-CS</td>
<td>16 D</td>
<td>27</td>
<td>0</td>
<td>43</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>2 A</td>
<td>17</td>
<td>-A</td>
<td>17 B</td>
<td>28</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>2 B</td>
<td>18</td>
<td>-B</td>
<td>18 D</td>
<td>29</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>2 D</td>
<td>19</td>
<td>-D</td>
<td>19 CF</td>
<td>30</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>3 A</td>
<td>20</td>
<td>-A</td>
<td>20 B</td>
<td>31</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>3 B</td>
<td>21</td>
<td>-B</td>
<td>21 CS</td>
<td>32</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>3 CS</td>
<td>22</td>
<td>-CS</td>
<td>22 D</td>
<td>33</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>4 A</td>
<td>23</td>
<td>-A</td>
<td>23 B</td>
<td>34</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>4 B</td>
<td>24</td>
<td>-B</td>
<td>24 CS</td>
<td>35</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>4 CS</td>
<td>25</td>
<td>-CS</td>
<td>25 D</td>
<td>36</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>5 A</td>
<td>26</td>
<td>-A</td>
<td>26 B</td>
<td>37</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>5 B</td>
<td>27</td>
<td>-B</td>
<td>27 CS</td>
<td>38</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>5 CS</td>
<td>28</td>
<td>-CS</td>
<td>28 D</td>
<td>39</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>6 A</td>
<td>29</td>
<td>-A</td>
<td>29 B</td>
<td>40</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>6 B</td>
<td>30</td>
<td>-B</td>
<td>30 CS</td>
<td>41</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>6 CS</td>
<td>31</td>
<td>-CS</td>
<td>31 D</td>
<td>42</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>7 A</td>
<td>32</td>
<td>-A</td>
<td>32 B</td>
<td>43</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>7 B</td>
<td>33</td>
<td>-B</td>
<td>33 CS</td>
<td>44</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>7 CS</td>
<td>34</td>
<td>-CS</td>
<td>34 D</td>
<td>45</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>8 A</td>
<td>35</td>
<td>-A</td>
<td>35 B</td>
<td>46</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>8 B</td>
<td>36</td>
<td>-B</td>
<td>36 CS</td>
<td>47</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>8 CS</td>
<td>37</td>
<td>-CS</td>
<td>37 D</td>
<td>48</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>9 A</td>
<td>38</td>
<td>-A</td>
<td>38 B</td>
<td>49</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>9 B</td>
<td>39</td>
<td>-B</td>
<td>39 CS</td>
<td>40</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>9 CS</td>
<td>41</td>
<td>-CS</td>
<td>41 D</td>
<td>42</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>10 A</td>
<td>43</td>
<td>-A</td>
<td>43 B</td>
<td>44</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>10 B</td>
<td>45</td>
<td>-B</td>
<td>45 CS</td>
<td>46</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
<tr>
<td>10 CS</td>
<td>47</td>
<td>-CS</td>
<td>47 D</td>
<td>48</td>
<td>0</td>
<td>43</td>
<td>34</td>
<td>B</td>
</tr>
</tbody>
</table>

321
6). Carry out quantification on the established Markov model over the time duration \([t_1, t_2]\): as no latent mission failure exists for phase 2, the failure probability for phase 2 is equal to the sum of the probability of failed states at the end of phase 2, i.e.

\[
Q_{\text{phase 2}} = \sum_{m} Q_m(t_2-)
\]

where state \(m\) includes states 24, 29, 30, 34, 35, 40, 45, 47, 49, 56 and 58 in table 12.7.

**Phase 3**

The Markov method is employed for the phase quantification.

1). Component A and module M1 are contained in the failure modes for phase 3 and therefore are considered in establishing the initial states.

2). Determine initial states and probabilities for phase 3.

The first failure mode for phase 3, \(A_{(2)} \cdot (M1_3 | M1_{(2)}) \cdot M1_{(2)}\), indicates that component A is failed and module M1 is working at the start of phase 3. States 14, 17, 20, 22, 26, 28, 32, 33, 36, 37, 39, 43, 44, 48, 50, 52, 54 and 56 in the Markov model for phase 2 (table 12.7) satisfy this condition and therefore are passed directly to phase 3 as the initial states.

The second failure mode, \(\overline{A_{(2)}} \cdot (M1_3 | M1_{(2)}) \cdot M1_{(2)}\), indicates that both component A and module M1 are working at the start of phase 3. States 1, 2, 3, 4, 6, 7, 10, 11, 15, 16, 18, 21, 23, 38, 42, 46, and 51 in the Markov model for phase 2 (table 12.7) are those which meet the condition and therefore are passed directly to phase 3 as the initial states.

The third phase failure mode, \(\overline{A_{(2)}} \cdot M1_{(2)}\), represents a transition failure for phase 3, since it does not contain any component failure event related to phase 3. States 5, 8, 9, 12, 13, 19, 25, 27, 31,
41, 53, 55 and 58 in the Markov model for phase 2 (table 12.7) are states which are consistent with this failure mode. These states are not considered with respect to the establishment of initial states.

Finally, the initial states for phase 3 are identified as shown in table 12.9.

<table>
<thead>
<tr>
<th>State No.</th>
<th>A</th>
<th>B</th>
<th>CS</th>
<th>D</th>
<th>CF</th>
<th>Mission</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: If during phase 2, component C fails while running and remains in the failed state till the end of phase 2, i.e. the state code of basic event ‘CF’ is ‘1’ at the end of phase 2, then at the start of phase 3, it is regarded as the equivalent of the states in which component C has already failed in standby.

Table 12.9 Initial states for phase 3

Table 12.10 indicates the corresponding relationship between the initial states for phase 3 and the system states at the end of phase 2.

<table>
<thead>
<tr>
<th>States at the end of phase 2 (table 12.7)</th>
<th>Initial states for phase 3 (table 12.9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3, 15, 42</td>
<td>3</td>
</tr>
<tr>
<td>4, 16</td>
<td>4</td>
</tr>
<tr>
<td>6, 18</td>
<td>5</td>
</tr>
<tr>
<td>7, 21, 38, 46, 50</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>11, 23</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>20, 36, 51</td>
<td>11</td>
</tr>
<tr>
<td>22, 37</td>
<td>12</td>
</tr>
<tr>
<td>26, 39</td>
<td>13</td>
</tr>
<tr>
<td>28, 43, 48, 53, 55</td>
<td>14</td>
</tr>
<tr>
<td>32</td>
<td>15</td>
</tr>
<tr>
<td>33, 44</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 12.10 Relationship between states at the end of phase 2 and initial states for phase 3

Initial state probabilities for phase 3 are then determined as follows:

\[ Q_1(t_2) = Q_1(t_2^-) \]
\[ Q_2(t_2) = Q_2(t_2^-) \]

......
\[ Q_{20}(t_2) = Q_{53}(t_2) + Q_{55}(t_2) \]

where \( Q_i(t_2) \) represents the initial state probability at the beginning of phase 3; and \( Q_j(t_2) \) represents the state probability at the end of phase 2.

3). Delete irrelevant components: component A is not required during phase 3, and therefore deleted from the Markov model. Since component A is required during phase 4, it is necessary to obtain the probability of component A being failed or working at the start of phase 4. This is achieved by following equation 11.16.

According to equations 11.19 and 11.20, the probability of component A being failed or working at the start of phase 3 given that the system is able to start phase 3, i.e. \( Q[A^{(3)}] \) and \( Q[A^{(3)}] \), are obtained as:

\[
q_{A^{(3)}} = \frac{\sum_{s=9}^{16} Q_s(t_2)}{\sum_{k=1}^{20} Q_k(t_2)} \quad \text{12.1}
\]

\[
P_{A^{(3)}} = \frac{\sum_{p=1}^{9} Q_p(t_2)}{\sum_{k=1}^{20} Q_k(t_2)} \quad \text{12.2}
\]

where states \( s, p, k \) correspond to states displayed in table 12.9

4). Trim the established initial states: the deletion of basic event A results in repeated states such as states 1 and 9, states 2 and 10, states 3 and 11, states 4 and 12, states 5 and 13, states 6 and 14, states 7 and 15 and states 8 and 16 in table 12.9. These repeated states are combined and the initial probability of these combined states are the sum of the original initial probability of repeated states. The refined initial states for phase 3 are displayed in table 12.11.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 12.11 Final list of initial states for phase 3

5). Develop the Markov model for phase 3 from the established initial states. See table 12.12 and 12.13.
Table 12.12 States in the Markov model for phase 3

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>17</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>19</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>20</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>-1</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>22</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 12.13 State transitions in the Markov model for phase 3

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>B</td>
<td>4</td>
<td>15</td>
<td>CS</td>
<td>9</td>
<td>20</td>
<td>B</td>
<td>14</td>
<td>10</td>
<td>-CS</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>CS</td>
<td>4</td>
<td>1</td>
<td>-D</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>14</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>D</td>
<td>5</td>
<td>4</td>
<td>-B</td>
<td>9</td>
<td>21</td>
<td>D</td>
<td>15</td>
<td>19</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-B</td>
<td>5</td>
<td>16</td>
<td>CF</td>
<td>10</td>
<td>5</td>
<td>B</td>
<td>15</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>D</td>
<td>6</td>
<td>17</td>
<td>B</td>
<td>10</td>
<td>21</td>
<td>CS</td>
<td>15</td>
<td>9</td>
<td>-D</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>CF</td>
<td>6</td>
<td>4</td>
<td>-CS</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>21</td>
<td>22</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>B</td>
<td>7</td>
<td>2</td>
<td>-D</td>
<td>11</td>
<td>10</td>
<td>-B</td>
<td>21</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-CS</td>
<td>7</td>
<td>18</td>
<td>CF</td>
<td>11</td>
<td>5</td>
<td>0</td>
<td>21</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>D</td>
<td>8</td>
<td>19</td>
<td>B</td>
<td>11</td>
<td>16</td>
<td>CF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>B</td>
<td>8</td>
<td>3</td>
<td>-D</td>
<td>14</td>
<td>17</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6). Carry out quantification on the established minimal Markov model over the time duration \([t_2, t_3]\). The system failure probability in phase 3 is obtained as follows:

\[
Q_{\text{phase 3}} = Q_{LMF_3} + \sum_k Q_k(t_3 -)
\]

where \(Q_{LMF_3}\) = \(\sum_m Q_m(t_2 -)\), in which \(Q_m(t_2 -)\) is the probability of state \(m\) at the end of phase 2 in which component A is working and module M1 is failed; state \(k\) represents the failed system state in the Markov model for phase 3, i.e. states 12, 13, 16-20 and 22 in table 12.12.

**Phase 4**

The BDD is employed as the quantification method.

Among all the failure modes for phase 4, both \(\overline{M_1}.A_{(3)}\) and \(\overline{M_1}.\overline{A_{(3)}}.M_2_{(3)}\) represent transition failures for phase 4 as neither of them contains component failure event during phase 4. The probability of each of the two failure modes is obtained by the product of the probability of each individual event contained in the failure mode.
The probability that module M1 is working at the end of phase 3, i.e. \( Q(M_1) \), can be obtained from the quantification for phase 3:

\[
P_{M_1} = \sum_i Q_i(t_3-)
\]

where state \( i \) represents the working state of module M1 at the end of phase 3, i.e. states 1-11, 14, 15 and 21 in table 12.12.

The probability of component A being failed at the end of phase 3 is calculated according to equation 11.16:

\[
q_{A} = q_{A|t_3} \cdot q_{A|t_3} + q_{A|t_3} \cdot p_{A|t_3}
\]

\( q_{A|t_3} \) and \( p_{A|t_3} \) have been obtained through equations 12.1 and 12.2 during the quantification process for phase 3. Assume that failure times of component A features an exponential distribution, then with a dormant failure during phase 3, the probability that component A is failed at the end of phase 3 given that it is working at the start of phase 3 can be obtained as:

\[
q_{A|t} = 1 - e^{-\lambda_A(t_3-t)}
\]

where \( t \) is the time of the last inspection on component A in phase 3.

The probability of component A being failed at the end of phase 3 given that component A is failed at the start of phase 3 can be obtained by quantifying the Markov model established to model the behaviour of component A over the time duration \([t_2, t_3]\) (Figure 12.6) with an initial probability \([0, 0, 1]\). Thus, \( q_{A|t_3} = Q_2(t_3) + Q_3(t_3) \). In this way, \( q_{A|t_3} \) and \( p_{A|t_3} \) can be obtained.

![Figure 12.6 Markov model for component A during phase 3](image)

In terms of module M2, the probability of module M2 being failed at the end of phase 3 can be obtained by quantifying module M2 over the time duration \([0, t_3]\) as it is required in none of the previous phases. Due to the maintenance dependency and sequential dependency existing
between components F, G and H through the mission, the Markov method is employed. Tables 12.14 and 12.15 display the Markov model established for module M2.

<table>
<thead>
<tr>
<th>State No.</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>M2</th>
<th>State No.</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0-0</td>
<td>0-0</td>
<td>0-0</td>
<td>0</td>
<td>21</td>
<td>3-2</td>
<td>3-1</td>
<td>3-3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3-1</td>
<td>0-0</td>
<td>0-0</td>
<td>0</td>
<td>22</td>
<td>3-2</td>
<td>1-1</td>
<td>0-0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0-0</td>
<td>3-1</td>
<td>0-0</td>
<td>0</td>
<td>23</td>
<td>0-0</td>
<td>1-1</td>
<td>3-2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0-0</td>
<td>0-0</td>
<td>3-1</td>
<td>0</td>
<td>24</td>
<td>3-3</td>
<td>3-1</td>
<td>3-2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1-1</td>
<td>0-0</td>
<td>0-0</td>
<td>0</td>
<td>25</td>
<td>0-0</td>
<td>1-1</td>
<td>2-2</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>3-1</td>
<td>3-2</td>
<td>0-0</td>
<td>0</td>
<td>26</td>
<td>1-2</td>
<td>0-0</td>
<td>2-1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3-1</td>
<td>0-0</td>
<td>3-2</td>
<td>0</td>
<td>27</td>
<td>3-2</td>
<td>3-3</td>
<td>3-1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>3-2</td>
<td>3-1</td>
<td>0-0</td>
<td>0</td>
<td>28</td>
<td>3-3</td>
<td>3-2</td>
<td>3-1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0-0</td>
<td>1-1</td>
<td>0-0</td>
<td>0</td>
<td>29</td>
<td>0-0</td>
<td>1-2</td>
<td>2-1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0-0</td>
<td>3-1</td>
<td>3-2</td>
<td>0</td>
<td>30</td>
<td>3-2</td>
<td>0-0</td>
<td>1-1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>3-2</td>
<td>0-0</td>
<td>3-1</td>
<td>0</td>
<td>31</td>
<td>0-0</td>
<td>3-2</td>
<td>1-1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0-0</td>
<td>3-2</td>
<td>3-1</td>
<td>0</td>
<td>32</td>
<td>1-1</td>
<td>3-2</td>
<td>3-3</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>0-0</td>
<td>0-0</td>
<td>1-1</td>
<td>0</td>
<td>33</td>
<td>1-1</td>
<td>3-3</td>
<td>3-2</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>1-1</td>
<td>3-2</td>
<td>0-0</td>
<td>0</td>
<td>34</td>
<td>1-1</td>
<td>2-2</td>
<td>3-3</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1-1</td>
<td>0-0</td>
<td>3-2</td>
<td>0</td>
<td>35</td>
<td>1-1</td>
<td>2-2</td>
<td>2-3</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1-1</td>
<td>2-2</td>
<td>0-0</td>
<td>0</td>
<td>36</td>
<td>1-1</td>
<td>3-3</td>
<td>2-2</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>3-1</td>
<td>3-2</td>
<td>3-3</td>
<td>1</td>
<td>37</td>
<td>1-1</td>
<td>2-3</td>
<td>2-2</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>1-1</td>
<td>0-0</td>
<td>2-2</td>
<td>0</td>
<td>38</td>
<td>1-2</td>
<td>2-1</td>
<td>3-3</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>3-1</td>
<td>3-3</td>
<td>3-2</td>
<td>0</td>
<td>39</td>
<td>1-2</td>
<td>2-1</td>
<td>2-3</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>1-2</td>
<td>2-1</td>
<td>0-0</td>
<td>0</td>
<td>40</td>
<td>2-2</td>
<td>1-1</td>
<td>0-0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 12.14 States in the Markov model for module M2 for first three phases

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>F</td>
<td>12</td>
<td>29</td>
<td>0</td>
<td>26</td>
<td>45</td>
<td>G</td>
<td>42</td>
<td>11</td>
<td>-G</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>G</td>
<td>13</td>
<td>30</td>
<td>F</td>
<td>27</td>
<td>46</td>
<td>0</td>
<td>43</td>
<td>25</td>
<td>-F</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>H</td>
<td>13</td>
<td>31</td>
<td>G</td>
<td>28</td>
<td>47</td>
<td>0</td>
<td>44</td>
<td>55</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>13</td>
<td>1</td>
<td>-H</td>
<td>29</td>
<td>48</td>
<td>F</td>
<td>44</td>
<td>30</td>
<td>-G</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>G</td>
<td>14</td>
<td>3</td>
<td>-F</td>
<td>29</td>
<td>13</td>
<td>-G</td>
<td>45</td>
<td>31</td>
<td>-F</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>H</td>
<td>14</td>
<td>16</td>
<td>0</td>
<td>30</td>
<td>49</td>
<td>0</td>
<td>45</td>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>F</td>
<td>14</td>
<td>32</td>
<td>H</td>
<td>30</td>
<td>50</td>
<td>G</td>
<td>46</td>
<td>29</td>
<td>-F</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>0</td>
<td>15</td>
<td>4</td>
<td>-F</td>
<td>30</td>
<td>2</td>
<td>-H</td>
<td>47</td>
<td>29</td>
<td>-F</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>H</td>
<td>15</td>
<td>33</td>
<td>G</td>
<td>31</td>
<td>51</td>
<td>F</td>
<td>48</td>
<td>56</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>F</td>
<td>15</td>
<td>18</td>
<td>0</td>
<td>31</td>
<td>52</td>
<td>0</td>
<td>48</td>
<td>30</td>
<td>-G</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>G</td>
<td>16</td>
<td>9</td>
<td>-F</td>
<td>31</td>
<td>3</td>
<td>-H</td>
<td>49</td>
<td>57</td>
<td>G</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>0</td>
<td>16</td>
<td>34</td>
<td>H</td>
<td>32</td>
<td>10</td>
<td>-F</td>
<td>49</td>
<td>5</td>
<td>-H</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-F</td>
<td>17</td>
<td>35</td>
<td>0</td>
<td>32</td>
<td>35</td>
<td>0</td>
<td>50</td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>G</td>
<td>18</td>
<td>13</td>
<td>-F</td>
<td>33</td>
<td>12</td>
<td>-F</td>
<td>50</td>
<td>6</td>
<td>-H</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>H</td>
<td>18</td>
<td>36</td>
<td>G</td>
<td>33</td>
<td>37</td>
<td>0</td>
<td>51</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>0</td>
<td>19</td>
<td>37</td>
<td>0</td>
<td>34</td>
<td>23</td>
<td>-F</td>
<td>51</td>
<td>8</td>
<td>-H</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>H</td>
<td>20</td>
<td>9</td>
<td>-F</td>
<td>34</td>
<td>35</td>
<td>0</td>
<td>52</td>
<td>60</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>18</td>
<td>0</td>
<td>20</td>
<td>38</td>
<td>H</td>
<td>35</td>
<td>25</td>
<td>-F</td>
<td>52</td>
<td>9</td>
<td>-H</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>G</td>
<td>21</td>
<td>39</td>
<td>0</td>
<td>36</td>
<td>31</td>
<td>-F</td>
<td>53</td>
<td>15</td>
<td>-G</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>0</td>
<td>22</td>
<td>40</td>
<td>0</td>
<td>36</td>
<td>37</td>
<td>0</td>
<td>53</td>
<td>54</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>H</td>
<td>22</td>
<td>2</td>
<td>-G</td>
<td>37</td>
<td>29</td>
<td>-F</td>
<td>54</td>
<td>18</td>
<td>-G</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
<td>F</td>
<td>22</td>
<td>41</td>
<td>H</td>
<td>38</td>
<td>23</td>
<td>-F</td>
<td>55</td>
<td>26</td>
<td>-G</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>-G</td>
<td>23</td>
<td>42</td>
<td>F</td>
<td>38</td>
<td>39</td>
<td>0</td>
<td>56</td>
<td>26</td>
<td>-G</td>
</tr>
<tr>
<td>9</td>
<td>23</td>
<td>H</td>
<td>23</td>
<td>4</td>
<td>-G</td>
<td>39</td>
<td>25</td>
<td>-F</td>
<td>57</td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td>F</td>
<td>23</td>
<td>25</td>
<td>0</td>
<td>40</td>
<td>5</td>
<td>-G</td>
<td>57</td>
<td>14</td>
<td>-H</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>0</td>
<td>24</td>
<td>43</td>
<td>0</td>
<td>40</td>
<td>53</td>
<td>H</td>
<td>58</td>
<td>16</td>
<td>-H</td>
</tr>
<tr>
<td>11</td>
<td>26</td>
<td>0</td>
<td>25</td>
<td>44</td>
<td>F</td>
<td>41</td>
<td>54</td>
<td>0</td>
<td>59</td>
<td>20</td>
<td>-H</td>
</tr>
<tr>
<td>11</td>
<td>27</td>
<td>G</td>
<td>25</td>
<td>13</td>
<td>-G</td>
<td>41</td>
<td>7</td>
<td>-G</td>
<td>60</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>28</td>
<td>F</td>
<td>26</td>
<td>13</td>
<td>-F</td>
<td>42</td>
<td>55</td>
<td>0</td>
<td>60</td>
<td>22</td>
<td>-H</td>
</tr>
</tbody>
</table>

Table 12.15 State transitions in the Markov model for module M2 for the first three phases
Then by quantifying the Markov model displayed above over the time duration \([0, t_3]\), the probability of module M2 being failed at the end of phase 3 can be obtained as:

\[
q_{M2(t_3)} = q_{M2(t_3-)} + \sum_i Q_i(t_3-)
\]

where state \(i\) represents the failed state of module M2, i.e. states 17, 32, 34 and 35 in table 12.14.

Thus the probabilities of the two failure modes which represent transition failure for phase 4 are obtained.

The probability of the other two failure modes, \(M_{13}, M_{32}(3)\) and \(M_{34}(3)\), is obtained in a similar way by the product of the probability of each event contained in these two terms.

For the failure mode \(M_{13}, M_{32}(3)\), the only unknown probability is \(q_{M_{13}}\). This can be obtained by quantifying module M2 independently over the time duration \([t_3, t_4]\). The working states of module M2 at the end of phase 3 form the initial states for module M2 at the start of phase 4. Tables 12.16 and 12.17 display the initial states of module M2 at the start of phase 4 and its relationship with the states of module M2 at the end of phase 3.

<table>
<thead>
<tr>
<th>State No.</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>M2</th>
<th>State No.</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0-0</td>
<td>0-0</td>
<td>0-0</td>
<td>0</td>
<td>9</td>
<td>1-2</td>
<td>0-0</td>
<td>2-1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1-1</td>
<td>0-0</td>
<td>0-0</td>
<td>0</td>
<td>10</td>
<td>0-0</td>
<td>1-2</td>
<td>2-1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0-0</td>
<td>1-1</td>
<td>0-0</td>
<td>0</td>
<td>11</td>
<td>1-1</td>
<td>2-3</td>
<td>2-2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0-0</td>
<td>0-0</td>
<td>1-1</td>
<td>0</td>
<td>12</td>
<td>1-2</td>
<td>2-1</td>
<td>2-3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1-1</td>
<td>2-2</td>
<td>0-0</td>
<td>0</td>
<td>13</td>
<td>2-2</td>
<td>1-1</td>
<td>0-0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1-1</td>
<td>0-0</td>
<td>2-2</td>
<td>0</td>
<td>14</td>
<td>1-3</td>
<td>2-1</td>
<td>2-2</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1-1</td>
<td>2-2</td>
<td>0-0</td>
<td>0</td>
<td>15</td>
<td>2-3</td>
<td>2-1</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>0-0</td>
<td>1-1</td>
<td>2-2</td>
<td>0</td>
<td>16</td>
<td>1-3</td>
<td>2-2</td>
<td>2-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 12.16 Initial states of module M2 at the start of phase 4
Initial states of module M2 for phase 4 | Working states of module M2 at the end of phase 3 (table 12.14)
--- | ---
1 | 1
2 | 2, 5
3 | 3, 9
4 | 4, 13
5 | 6, 14, 16
6 | 7, 15, 16
7 | 8, 20
8 | 10, 23, 25
9 | 11, 26
10 | 12, 29
11 | 19, 33, 36, 37
12 | 21, 38, 39
13 | 22, 40
14 | 24, 43
15 | 27, 45, 46
16 | 28, 47
17 | 30, 49
18 | 31, 52
19 | 41, 53, 54
20 | 42, 44, 55
21 | 48, 56
22 | 50, 57, 58
23 | 51, 59, 60

Table 12.17 Corresponding relationship between initial states of module M2 for phase 4 and its states at the end of phase 3

The Markov model is then established for module M2 over phase 4 as displayed in tables 12.18 and 12.19.

<table>
<thead>
<tr>
<th>State No.</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>M2</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0-0</td>
<td>0-0</td>
<td>0-0</td>
<td>0</td>
<td>9</td>
<td>1-2</td>
<td>0-0</td>
</tr>
<tr>
<td>2</td>
<td>1-1</td>
<td>0-0</td>
<td>0-0</td>
<td>0</td>
<td>10</td>
<td>0-0</td>
<td>1-2</td>
</tr>
<tr>
<td>3</td>
<td>0-0</td>
<td>1-1</td>
<td>0-0</td>
<td>0</td>
<td>11</td>
<td>1-1</td>
<td>2-3</td>
</tr>
<tr>
<td>4</td>
<td>0-0</td>
<td>0-0</td>
<td>1-1</td>
<td>0</td>
<td>12</td>
<td>1-2</td>
<td>2-1</td>
</tr>
<tr>
<td>5</td>
<td>1-1</td>
<td>0-0</td>
<td>0-0</td>
<td>0</td>
<td>13</td>
<td>1-2</td>
<td>2-1</td>
</tr>
<tr>
<td>6</td>
<td>1-1</td>
<td>0-0</td>
<td>2-2</td>
<td>0</td>
<td>14</td>
<td>1-3</td>
<td>2-1</td>
</tr>
<tr>
<td>7</td>
<td>1-2</td>
<td>0-0</td>
<td>0-0</td>
<td>0</td>
<td>15</td>
<td>1-2</td>
<td>2-3</td>
</tr>
<tr>
<td>8</td>
<td>0-0</td>
<td>1-1</td>
<td>2-2</td>
<td>0</td>
<td>16</td>
<td>1-3</td>
<td>2-2</td>
</tr>
</tbody>
</table>

Table 12.18 States in the Markov model for module M2 during phase 4

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
<th>From</th>
<th>To</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>F</td>
<td>4</td>
<td>18</td>
<td>G</td>
<td>9</td>
<td>4</td>
<td>-F</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>G</td>
<td>4</td>
<td>1</td>
<td>-H</td>
<td>9</td>
<td>15</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>H</td>
<td>5</td>
<td>3</td>
<td>-F</td>
<td>10</td>
<td>21</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-F</td>
<td>5</td>
<td>24</td>
<td>H</td>
<td>10</td>
<td>4</td>
<td>-G</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>G</td>
<td>6</td>
<td>4</td>
<td>-F</td>
<td>11</td>
<td>10</td>
<td>-F</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>H</td>
<td>6</td>
<td>11</td>
<td>G</td>
<td>12</td>
<td>8</td>
<td>-F</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>F</td>
<td>7</td>
<td>3</td>
<td>-F</td>
<td>13</td>
<td>2</td>
<td>-G</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>G</td>
<td>7</td>
<td>12</td>
<td>H</td>
<td>13</td>
<td>19</td>
<td>H</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>H</td>
<td>8</td>
<td>20</td>
<td>F</td>
<td>14</td>
<td>8</td>
<td>-F</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>F</td>
<td>8</td>
<td>4</td>
<td>-G</td>
<td>15</td>
<td>10</td>
<td>-F</td>
</tr>
</tbody>
</table>

Table 12.19 State transitions in the Markov model for module M2 during phase 4
By accounting for the event $M_2(t_3)$, the state probabilities in module M2 at the end of phase 3 are passed to corresponding initial states in module M2 at the start of phase 4. That is, referring to table 12.17:

$$Q_1(t_3) = Q_1(t_3-)$$
$$Q_2(t_3) = Q_2(t_3-) + Q_5(t_3-)$$

......

$$Q_{23}(t_3) = Q_{51}(t_3-) + Q_{59}(t_3-) + Q_{60}(t_3-)$$

where $Q_i(t_3)$ is the initial state probability of module M2 at the start of phase 4; $Q_i(t_3-)$ is the state of probability of module M2 at the end of phase 3.

Then the quantification of module M2 over phase 4 based on the pre-determined initial state probabilities will produce the probability $Q[\{ M_2 | M_2(t_3) \}, M_2(t_3)]$.

With the probability of each failure mode obtained, the system failure probability in phase 4 is then obtained as:

$$Q[\text{phase 4}] = Q[\overline{M_1}, A(3)] + Q[\overline{M_1}, \overline{M_2}(3)] +$$
$$Q[\overline{M_1}, \overline{M_2}(3), \{ (A_4 | \overline{A}(3)), \overline{A}(3) \}] +$$
$$Q[\overline{M_1}, \{ (A_4 | \overline{A}(3)), \overline{A}(3), (M_2 | \overline{M_2}(3)) \}, \overline{M_2}(3)]$$

$$= (1-q_{M_1}), q_{A_1} + (1-q_{M_1})(1-q_{A_3}), q_{M_23} + (1-q_{M_1})(1-q_{M_23})$$
$$q_{A_4 | \overline{A}(3)} + (1-q_{M_1})(1-q_{A_3}), q_{M_24 | \overline{M_2}(3)}, (1-q_{M_23})$$

Step 10. Calculate the mission failure probability by summing up the system failure probability in each phase. That is,

$$\overline{R_{\text{mass}}} = \sum_{i=1}^{4} Q[\text{phase } i]$$

### 12.2.2 Importance Measures

**Phase 1**

Components A and E are required during phase 1.
Phase 1

Criticality Function $G_{i-1}(q(t))$

\[
G_{A_{i-1}}(q(t)) = (Q_{\text{phase I}} | q_{A_{i}} = 1) - (Q_{\text{phase I}} | q_{A_{i}} = 0)
\]

\[
= 1 - \int_0^t f_E(u)du \quad (0 \leq t < t_1)
\]

\[
= e^{-\lambda t} \quad (0 \leq t < t_1)
\]

\[
G_{B_{i-1}}(q(t)) = G_{C_{i-1}}(q(t)) = G_{D_{i-1}}(q(t)) = 0
\]

\[
G_{E_{i-1}}(q(t)) = Q_{\text{phase I}} | q_{E_{i}} = 1) - (Q_{\text{phase I}} | q_{E_{i}} = 0)
\]

\[
= 1 - \int_0^t f_A(u)du \quad (0 \leq t < t_1)
\]

\[
= e^{-\lambda t} \quad (0 \leq t < t_1)
\]

\[
G_{F_{i-1}}(q(t)) = G_{G_{i-1}}(q(t)) = G_{H_{i-1}}(q(t)) = 0
\]

Criticality Measure $I_{i-1}$

\[
I_{A_{i-1}} = I_{D_{i-1}} = \frac{G_{A_{i-1}} q_{A_{i}}}{Q[\text{phase I}]} = \frac{[1 - \int_0^t f_E(u)du] \cdot \int_0^t f_A(u)du}{Q[\text{phase I}]} = \frac{e^{-\lambda t} \cdot (1 - e^{-\lambda t})}{Q[\text{phase I}]}
\]

\[
I_{B_{i-1}} = I_{C_{i-1}} = I_{D_{i-1}} = 0
\]

\[
I_{E_{i-1}} = I_{G_{i-1}} = \frac{G_{E_{i-1}} q_{E_{i}}}{Q[\text{phase I}]} = \frac{[1 - \int_0^t f_A(u)du] \cdot \int_0^t f_E(u)du}{Q[\text{phase I}]} = \frac{e^{-\lambda t} \cdot (1 - e^{-\lambda t})}{Q[\text{phase I}]}
\]

\[
I_{F_{i-1}} = I_{G_{i-1}} = I_{H_{i-1}} = 0
\]

Phase 2

Components A, B, C and D are required during phase 2.

Phase Criticality Function $G_{i-2}(q(t))$

\[
G_{A_{i-2}}(q(t)) = \sum_m Q_m(t) + \sum_k Q_k(t) \quad (t_1 \leq t < t_2)
\]

where state $m$ represents states 5, 8, 9, 12, 13, 19, 25, 27, 31, 41, 52, 54 and 57 in table 12.7.

state $k$ represents states 24, 29, 30, 34, 35, 40, 45, 56 and 58 in table 12.7.

\[
G_{B_{i-2}}(q(t)) = \sum_m Q_m(t) + \sum_k Q_k(t) \quad (t_1 \leq t < t_2)
\]

where state $m$ represents states 20, 28, 33, 36, 37, 38, 43, 44, 48, 51, 53 and 55 in table 12.7.

state $k$ represents states 24, 29, 30, 34, 35, 49, 56 and 58 in table 12.7.
\[ G_{CS-2}(q(t)) = G_{D-2}(q(t)) = 0, \] as basic events CS and D are both enabling events.

\[ G_{CF-2}(q(t)) = \sum_m Q_m(t) + \sum_k Q_k(t) \quad (t_1 \leq t < t_2) \]

where state \( m \) represents states 17, 26, 32 and 39 in table 12.7; state \( k \) represents states 40, 45 and 47 in table 12.7.

\[ G_{E-2}(q(t)) = G_{F-2}(q(t)) = G_{G-2}(q(t)) = G_{H-2}(q(t)) = 0 \]

**Phase Criticality Measure** \( I_{p-2} \)

\[ I_{A-2} = I_{A-2}^p = \frac{\int_0^t \sum_m Q_m(u) \lambda_A du}{Q[\text{phase 2}]} \quad (m \text{ as same for } G_{A-2}(q(t))) \]

\[ I_{B-2} = I_{B-2}^p = \frac{\int_0^t \sum_m Q_m(u) \lambda_B du}{Q[\text{phase 2}]} \quad (m \text{ as same for } G_{B-2}(q(t))) \]

\[ I_{C-2} = I_{C-2}^p = \frac{\int_0^t \sum_m Q_m(u) \lambda_{CF} du}{Q[\text{phase 2}]} \quad (m \text{ as same for } G_{CF-2}(q(t))) \]

\[ I_{E-2} = I_{F-2} = I_{G-2} = I_{H-2} = 0 \]

**Phase 3**

Components B, C and D are required during phase 3.

**Phase Criticality Function** \( G_{i-3}(q(t)) \)

\[ G_{A-3}(q(t)) = 0 \]

\[ G_{B-3}(q(t)) = \sum_m Q_m(t) + \sum_k Q_k(t) \quad (t_2 \leq t < t_3), \] where state \( m \) represents states 3, 6, 8, 9, 14, 15 and 21, state \( k \) represents states 13, 17, 19, 20 and 22 in table 12.12.

\[ G_{CS-3}(q(t)) = G_{D-3}(q(t)) = 0, \] as basic events CS and D are both enabling events during phase 3.

\[ G_{CF-3}(q(t)) = \sum_m Q_m(t) + \sum_k Q_k(t) \quad (t_2 \leq t < t_3), \] where state \( m \) represents states 2, 5, 7 and 11, state \( k \) represents states 12, 16 and 18 in table 12.12.

\[ G_{E-3}(q(t)) = G_{F-3}(q(t)) = G_{G-3}(q(t)) = G_{H-3}(q(t)) = 0 \]
• Phase Criticality Measure $I_{i,j}$

$$I_{a,3} = 0$$

$$I_{b,3} = I_{b,3}^{p} + I_{b,3}^{T}$$

where:

$$I_{b,3}^{p} = \frac{\int \sum \frac{Q_m(u, \beta_B)}{Q[phase 3]} \, du}{m \text{ as same for } G_{b,3}(q(t))}$$

$$I_{b,3}^{T} = \frac{\sum Q_m(t_2, \beta_B)}{Q[phase 3]}$$, where state $m$ represents the states in the Markov model for phase 2 where component A is working and the failure of component B will bring about the failure of module M1, i.e. states 3, 7, 11, 15, 21, 23, 38, 42, 46 and 50 in table 12.7

$$I_{c,3} = I_{c,3}^{p} + I_{c,3}^{T}$$

where:

$$I_{c,3}^{p} = \frac{\int \sum \frac{Q_m(u, \beta_{cF})}{Q[phase 3]} \, du}{m \text{ as same for } G_{cF,3}(q(t))}$$

$$I_{c,3}^{T} = \frac{\sum Q_m(t_2, \beta_{cF})}{Q[phase 3]}$$, where state $m$ represents states in the Markov model for phase 2 where component A is working and the active failure of component C will bring about the failure of module M1, i.e. states 2, 6, 10 and 18 in table 12.7

$$I_{e,3} = I_{f,3} = I_{g,3} = I_{h,3} = 0$$

**Phase 4**

Components A, F, G and H are required during phase 4.

• Phase Criticality Function $G_{i,4}(q(t))$

$$G_{A,4}(q(t)) = Q[phase 4|q_{A|A_{3}} = 1] - Q[phase 4|q_{A|A_{3}} = 0]$$

$$= (1-q_{M_{H,1}}) \cdot (1-q_{M_{2,3}}) \cdot (1-q_{A_{3}}) \cdot [1-q_{M_{2,3}|M_{2,3}}](t)$$

$$G_{B,4}(q(t)) = G_{c,4}(q(t)) = G_{D,4}(q(t)) = G_{E,4}(q(t)) = 0$$

$$G_{F,4}(q(t)) = G_{G,4}(q(t)) = 0$$, as basic events F and G are both sequential failure events during phase 4 which have to occur prior to H.
\[ G_{H-4}(q(t)) = G_{H/M2-4}(q(t)) \cdot G_{M2-4}(q(t)) \]

where:

\[ G_{H/M2-4}(q(t)) = \sum_m Q_m(t) + \sum_k Q_k(t) \quad (t_3 \leq t < t_4) \]

where state \( m \) represents state 5, and state \( k \) represents state 24 in table 12.18.

\[ G_{M2-4}(q(t)) = Q[\text{phase 4} \mid q_{M2,1}|q_{M2,3}] = 1 - Q[\text{phase 4} \mid q_{M2,1}|q_{M2,3}] = 0 \]

\[ = (1-q_{M1})(1-q_{M2,3})(1-q_{A3})(1-q_{M2,3}) \]

- Phase Criticality Measure \( I_{A-4} \)

\[ I_{A-4} = I_{A-4}^p + I_{A-4}^r \]

where:

\[ I_{A-4}^p = \frac{G_{A-4} \cdot q_{A_3}}{Q[\text{phase 4}]} \]

\[ = \frac{(1-q_{M1})(1-q_{M2,3})(1-q_{A3})(1-q_{M2,3})}{Q[\text{phase 4}]} \]

\[ I_{A-4}^r = \frac{Q[\text{phase 4} \mid q_{A3} = 1] - Q[\text{phase 4} \mid q_{A3} = 0]}{Q[\text{phase 4}]} \cdot q_{A3} \]

\[ I_{B-4} = I_{C-4} = I_{D-4} = I_{E-4} = 0 \]

\[ I_{F-4} = I_{G-4} = 0 \]

\[ I_{H-4} = I_{H-4}^p + I_{H-4}^r \]

where:

\[ I_{H-4}^p = I_{H/M2-4}^p \cdot I_{M2-4}^p \]

\[ = \int_0^t Q_m(u) \lambda_{M2}du \cdot \frac{G_{M2-4} \cdot q_{M2}}{Q[\text{phase 4}]} \quad (m \text{ same for } G_{H/M2-4}(q(t))) \]

\[ = \int_0^t Q_m(u) \lambda_{M2}du \cdot \frac{(1-q_{M1})(1-q_{A3})(1-q_{A3})(1-q_{M2,3})}{Q[\text{phase 4}]} \]

\[ = \frac{(1-q_{M1})(1-q_{A3})(1-q_{A3})(1-q_{M2,3})}{Q[\text{phase 4}]} \cdot \int_0^t Q_m(u) \lambda_{M2}du \]
\[ I_{H-4}^T = I_{H/M2-4}^T + I_{M2-4}^T \]

\[
= \frac{Q_m(t_3) \lambda_H}{Q_{M2(3)}} \left\{ Q \left[ \text{LMF}_{q_{M2(3)} = 1} - Q \left[ \text{LMF}_{q_{M2(3)} = 0} \right] \right] q_{M2(3)} \right\} 
\]

\[
= \frac{(1-q_{M1}) (1-q_{\lambda H}) Q_m(t_3) \lambda_H}{Q[\text{phase 4}]} \quad (m \text{ as same for } G_{H/M2-4}(q(t)))
\]
Chapter 13. Dependency Modelling Using Binary Decision Diagrams

13.1 Introduction
The previous chapters provide an in-depth probe into how the Markov method can be applied to the reliability evaluation of different types of system which feature dependency relationships. To overcome the weakness of the Markov method resulting from the model size, an attempt is made in this chapter to explore the applicability of the BDD method in dependency modelling. A method is presented which integrates a Bayesian, conditional probability, approach in the binary decision diagram approach [62]. With the Bayesian analysis, some dependency relationships between components can be taken into account. In the traditional BDD analysis, the independence between component failures has always been assumed.

With regard to the system dependency modelling using binary decision diagrams, the research has been focused on non-repairable systems.

13.2 Dependency Modelling Using BDD for Non-repairable Systems
Four types of dependency are relevant to non-repairable systems. These include standby dependency, stress-related dependency, initiator-enabler dependency and sequential failure dependency. Among these four types of dependency, the first two both represent the situation where the failure of some component is influenced by the functionality of another component; and the latter two types of dependency both represent the situation where the order of the occurrence of component failures has to be taken into account in the analysis.

With the standby dependency, either cold-standby or warm-standby mode, the change in the failure rate of the standby component is initiated by the failure of the duty component. In terms of the stress-related dependency, the failure of a component will result in the increase in the working load of remaining components which in turn leads to a higher failure rate.

For the initiator-enabler dependency and the sequential failure dependency, the failure process of each component is independent, although only when the component failures occur in a certain order, will the system fail.

For non-repairable systems, the system performance is assessed by the prediction of the system unreliability over a certain period of time, \([0, T]\). T is usually the period of time required for the system to achieve the completion of a specific task. For systems which contain no dependency,
the system failure probability, which is a function of individual component unreliability \( \int_0^T f_c(u) du \), can be obtained through the quantification of the corresponding BDD.

When \( \int f(u) du \) has no closed form, an appropriate numerical integration technique [63] can be applied to obtain the approximated value of the integral at discrete time point.

When there is no dependency relationship involved in the system, the numerical integration for the failure probability of each basic event is carried out separately over the same time duration \([0, T)\). However, with the dependency relationships involved in the system, the quantification will not be as straightforward.

The following sections describe how conditional probabilities can be used in the BDD approach to model each of the above dependency types to obtain the system unreliability.

### 13.2.1 Dependency Modelling for Initiator-enabler Dependency

The typical fault tree structure in which the initiator-enabler dependency lies is an output event with two input events linked to each other through an AND gate, as shown in figure 13.1.

![Figure 13.1 Typical fault tree structure representing initiator-enabler dependency](image)

In figure 13.1, both the initiating event (I) and the enabling event (E) can be a single basic event or an intermediate event which can be further developed into combinations of basic events.

Consider when both the initiating event and the enabling event are basic events, the corresponding BDD is established as in figure 13.2:

![Figure 13.2 BDD for i-e dependency](image)
The probability density functions \( f_E(t) \) and \( f_I(t) \) can be established for the enabling event and initiating event respectively. These two variables are independent from each other, given that no repair is carried out. The failure probability of the enabling event \( E \) or initiating event \( I \) over any period of time \([t_1, t_2]\) can be obtained as \( \int_{t_1}^{t_2} f_E(u)du \) and \( \int_{t_1}^{t_2} f_I(u)du \).

The system will only fail when the enabling event \( E \) occurs prior to the initiating event \( I \). If the enabling event occurs at time \( s \), only if the initiating event occurs during \([s, T)\), will the system fail. In the integral form, the system failure probability over the time duration \([0, T]\) can then be expressed as:

\[
F_{sys}(T) = \int f_E(s) \int f_I(u)du ds
\]

which means that the enabling event occurs at any time \( s \) during \([0, T]\) followed by the occurrence of the initiating event \( I \) during \([s, T]\).

If the occurrence times of the initiating and enabling events are both governed by the exponential distribution, the integral in equation 13.1 will have a closed form. In other cases, a numerical integration technique may be necessary to evaluate the integral.

When either the initiating event \( I \) or the enabling event \( E \) is an intermediate event connecting a series of basic events through a set of logic gates, they can be quantified separately as modules and then substituted into an equation of the same form as equation 13.1. Let \( M_E \) and \( M_I \) denote modules representing causes of the enabling event and the initiating event respectively. Then the system failure probability \( F_{sys}(T) \) is obtained as:

\[
F_{sys}(T) = \int f_{M_E}(s) \int f_{M_I}(u)du ds
\]

This requires a process to obtain \( f_{M_E}(t) \) and \( f_{M_I}(t) \).

In module \( M_E \), assume it contains basic events \( E_1, E_2, \ldots, E_n \), then:

\[
f_{M_E}(t) . dt = P[\text{module } M_E \text{ fails in time period } [t, t+dt)]
\]

\[
= \sum_{i=1}^{n} P[\text{module } M_E \text{ has not failed prior to time } t \text{ and fails in } [t, t+dt) \text{ due to the failure of basic event } E_i]
\]
For module $M_E$ to fail due to the failure of the basic event $E_i$ in $[t, t+dt)$, the module must be residing in a critical state for $E_i$ at time $t$. Therefore,

$$f_{M_E}(t).dt = \sum_{i=1}^{n} G_{E_i}(t).f_{E_i}(t).dt.$$  

That is:

$$f_{M_E}(t) = \sum_{i=1}^{n} G_{E_i}(t).f_{E_i}(t)$$ \hspace{1cm} (13.3)

where $G_{E_i}(t)$ is the criticality function for basic event $E_i$ at time $t$, which can be obtained in the BDD for module $M_E$ as described in section 2.2.4.4. In the same way, $f_{M_I}(t)$ can be obtained.

When causes of the initiating event or the enabling event are developed as a module, the quantification is carried out on the BDD for the corresponding module to obtain $f_E(s)$ and $f(t)$. These are then substituted into equation 13.2 using a numerical integration scheme if the forms of the failure time distributions do not produce a closed form solution to the integral.

An example system which features the initiator-enabler dependency is used to illustrate how the BDD approach addresses this type of dependency.

In the simplified vaporizer system shown in figure 13.3, liquid is pumped from a tank into a butane vaporizer where it is heated to form a gas. In the event of a pump surge the pressure in the vaporizer exceeds the rating of the vaporizer tubes. To prevent the tubes from rupturing, safety systems have been incorporated at several locations on the inlet to the vaporizer, which will shut down the process on the pump surge. In total, three protective systems have been used: two trip loops which close a valve halting the butane flow and a vent valve which opens allowing the butane to return to the tank if the pressure exceeds the preset limit. There is no maintenance to be carried out during the time the system is working.
The fault tree for the top event ‘Vaporizer coil ruptures under high pressure’ is shown in figure 13.4.

![Fault Tree](image)

**Figure 13.4 Fault tree for vaporizer system**

In figure 13.4, the basic event ‘PS’ represents the initiating event and the intermediate event represented by G1 is the overall enabling event. The pre-processing of the fault tree structure in figure 13.4 will identify module M1 led by gate G1. BDDs are established for the system and for module M1 respectively as in figure 13.5.

![BDD Diagram](image)

**Figure 13.5 BDD for the system and module M1**

Independent probability density functions exist for basic events PS, T1, V, and T2, \( f_{PS}(t), f_{T1}(t), f_{V}(t) \) and \( f_{T2}(t) \) respectively.

Considering module M1. From equation 13.3:

\[
f_{M1}(t) = G_{T1}(t).f_{T1}(t) + G_{V}(t).f_{V}(t) + G_{T2}(t).f_{T2}(t)
\]

where the criticality function for each basic event in module M1 is obtained as follows:

\[
G_{T1}(t) = P[F2] = F_{T2}(t).F_{V}(t) = \int f_{T2}(u)du . \int f_{V}(u)du
\]

\[
G_{V}(t) = F_{T1}(t).P[F3] = F_{T1}(t).F_{T2}(t) = \int f_{T1}(u)du . \int f_{T2}(u)du
\]
\[ G_{T2}(t) = F_{T1}(t) \cdot F_v(t) = \int_{0}^{\infty} F_{T1}(u) du \cdot \int_{0}^{\infty} f_v(u) du \]

### 13.2.2 Dependency Modelling for Sequential Dependency

The typical fault tree structure representing the sequential dependency is shown in figure 13.6, where there are \( n \) inputs which must occur in order and these inputs can be either basic events or gates. (In this case, basic events are illustrated in the figure).

**Figure 13.6 Typical fault tree structure for sequential dependency**

The sequential failure event \( C_i \) can be either a single basic event or an intermediate event which it is assumed can be identified as an independent module. The BDD established for the fault tree in figure 13.6 is shown in figure 13.7.

**Figure 13.7 BDD for sequential dependency**

For the output event to occur, the events must occur in the specified order. The system failure probability is expressed as in equation 13.4.

\[ F_{sys}(T) = \int_{0}^{\infty} f_{C1}(s_1) \int_{0}^{\infty} f_{C2}(s_2) ... \int_{0}^{\infty} f_{Cn}(s_n) ds_1 ds_2 ... ds_n \]

\[ 13.4 \]

It must be noted that in establishing the corresponding BDD, the variable ordering must be the same as the specified order of occurrence. Initiator-enabler dependency can also be tackled in the same way as for sequential dependency since it can be regarded as a sequential dependency with two input events. In this case, the enabling event must be considered prior to the initiating event in the variable ordering.
13.2.3 Dependency Modelling for Standby Dependency

It is in the warm-spare and cold-spare cases that a dependency arises between the duty component and the standby component. The dependency occurs since the failure rate of the standby component varies according to the state of the duty component. Take for example a simple system with redundancy in the design, in which component C1 is the duty component and C2 the standby activated when C1 fails. The system fails when both components fail. The system failure probability over the system functioning period \([0, T)\) is \(F_{\text{sys}}(T)\). The failure probability of the duty component C1 can be obtained from \(\int_0^t f_{c1}(u)du\). For system failure it is then necessary to consider the standby component C2 which, in the case of a warm-spare or cold-spare, has no consistent probability density function that can be defined over the whole duration \([0, T)\). Instead, two probability density functions are required to define the behaviour of C2 over \([0, T)\). The first is concerned with C2 in its standby state, denoted by \(f_{c2|\bar{x}}(t)\), and the other is concerned with C2 following its activation, denoted by \(f_{c2|x}(t)\), where \(x\) refers to the failure of duty component C1. Conditional probabilities are used to evaluate \(F_{\text{sys}}(T)\) in the form of integrals in the following form:

\[
F_{\text{sys}}(T) = \int_0^t f_{c1}(s) \int_0^s f_{c2|\bar{x}}(u)du ds + \int_0^t f_{c1}(s) \left[1 - \int_0^s f_{c2|\bar{x}}(u)du\right] \int_0^t f_{c2|x}(t-s)dt ds
\]

where the first term on the right-hand of the equation represents the situation that when C1 fails, component C2 has already failed in the standby state. The second term represents the situation that component C2 has not failed in its standby but fails after its activation triggered by the failure of component C1. When component C2 is characterized by a cold-standby, \(f_{c2|\bar{x}}(t)=0\).

The above example illustrates the basic algorithm of using conditional probabilities to evaluate the reliability of standby systems. The following section describes the generic process of extending the conditional probability approach using the binary decision diagram to the solution of standby systems.

13.2.3.1 Generic Process

Consider the general case: \(n - m\), where \(n\) duty components are required to keep the system functioning; and \(m\) standby components are assigned \((n>=1, m>=1)\).
Step 1: establish the fault tree structure for the failure of the standby system.

The system fails when \((n-1)\) or less components are working. That is, among all the \((n+m)\) components, when \((n+m-(n-1))\), i.e. \((m+1)\) components have failed, the system fails.

Two different failure modes are considered for the standby component: fails in standby and fails functioning. The general fault tree structure for the system failure (only considering the key components) is displayed in figure 13.8.

![Figure 13.8 The generic fault tree structure for standby dependency](image)

In the case of cold-standby, basic events \(SCiS\) which represent the standby failures do not need to be included in the fault tree. Different probability density functions can be established for each failure mode: fails functioning and fails in standby.

In most standby systems, a controlling component is required to activate the standby component in the event that the duty component fails. Other supporting components are also required to enable the key component to achieve its functionality, such as a power supply and perhaps isolation or pressure relief valves depending on the system type. The failure of the controlling component and the supporting components are taken into consideration in the generic fault tree structure as shown in figure 13.9.
When $n = 1$, i.e. only one duty component is required, the voting gate in the fault tree structure in figure 12.10 turns into an AND gate.

Step 2: establish the required component dependency information and failure data.

The dependency relationships are evident from the fault tree structure in figure 13.9. The basic event data required are the form and parameters of probability density functions for each component failure. $f_{DCi}(t)$, $f_{SCiF}(t)$, $f_{SUi}(t)$, $f_C(t)$ and $f_{SCiS}(t)$ are established to model the failure times of duty component DCi, the standby component (in active failure mode) SCiF, the supporting component SUi, the controlling component C and the standby component (in standby mode) SCiS. For the active failure of the standby component, i.e. SCiF, the time ‘$t$’ is measured from the time the component gets activated.

Step 3: carry out the simplification and modularization process on the fault tree structure.

In this process, the active failure of duty component or standby component (DC1, ..., DCn, SC1F, ..., SCmF) can be combined with its own functionally supporting component (SU1, ..., SUn+m) to form a module.
Step 4: construct the BDD based on the modularized fault tree structure, and the BDD for each identified module.

In the process of constructing the BDD, an appropriate variable ordering is needed. An ordering is used which follows the time sequence of events. Although in the BDD the variable ordering does not need to follow the actual sequence of component failures, in this case it provides a structured approach by considering the active failure of the standby components after the duty component, controlling component and standby component failures.

Another factor which may be considered in the process of establishing the variable ordering is the failure to activate the standby components (either due to the controlling component failure or the failure of the components in standby) and the failure of duty components. Two ways of ordering these failure events have been investigated. The first ordering considers the controlling component and standby component failures prior to the duty component failures. The second ordering considers these elements in the reverse order. Based on these two orderings, different binary decision diagrams will be generated, which will produce different collections of paths leading to system failure. To illustrate this, a simple standby system is introduced in figure 13.10.

![Figure 13.10 Example standby system](image)

In this system, four pumps P1, P2, P3 and P4 form a parallel system to provide water supply. Pumps P1 and P2 are duty pumps, and pumps P3 and P4 are standby pumps which activate when pump 1 or pump 2 fails. The system fails when 3-out-of-4 supply streams are unavailable. The corresponding fault tree structure for the system failure is shown in figure 13.11.
By adopting the first variable ordering, i.e. P3S>P4S>P1>P2>P3F>P4F, the resulting BDD is shown in figure 13.12.

By following the second variable ordering, i.e., P1>P2>P3S>P4S>P3F>P4F, the resulting BDD is shown in figure 13.13.
Table 13.1 and figure 13.14 provides a comparison between the failure paths obtained from the two BDDs.

![BDD Diagram](image)

**Figure 13.13 BDD for the example system on variable ordering 2**

<table>
<thead>
<tr>
<th>Path number</th>
<th>Ordering 1</th>
<th>Ordering 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P3S,P4S,P1</td>
<td>P1,P2,P3S</td>
</tr>
<tr>
<td>2</td>
<td>P3S,P4S,P1,P2</td>
<td>P1,P2,P3S,P4S</td>
</tr>
<tr>
<td>3</td>
<td>P3S,P4S,P1,P2</td>
<td>P1,P2,P3S,P4S,P3F</td>
</tr>
<tr>
<td>4</td>
<td>P3S,P4S,P1,P2,P4F</td>
<td>P1,P2,P3S,P4S,P3F,P4F</td>
</tr>
<tr>
<td>5</td>
<td>P3S,P4S,P1,P2,P4F</td>
<td>P1,P2,P3S,P4S</td>
</tr>
<tr>
<td>6</td>
<td>P3S,P4S,P1,P2</td>
<td>P1,P2,P3S,P4S,P3F</td>
</tr>
<tr>
<td>7</td>
<td>P3S,P4S,P1,P2,P3F</td>
<td>P1,P2,P3S,P4S,P3F</td>
</tr>
<tr>
<td>8</td>
<td>P3S,P4S,P1,P2,P3F</td>
<td>P1,P2,P3S,P4S,P3F,P4F</td>
</tr>
<tr>
<td>9</td>
<td>P3S,P4S,P1,P2,P3F</td>
<td>P1,P2,P3S,P4S</td>
</tr>
<tr>
<td>10</td>
<td>P3S,P4S,P1,P2,P3F,P4F</td>
<td>P1,P2,P3S,P4S,P4F</td>
</tr>
<tr>
<td>11</td>
<td>P3S,P4S,P1,P2,P3F,P4F</td>
<td>P1,P2,P3S,P4S,P3F</td>
</tr>
<tr>
<td>12</td>
<td>P3S,P4S,P1,P2,P3F,P4F</td>
<td>P1,P2,P3S,P4S,P3F,P4F</td>
</tr>
</tbody>
</table>

Table 13.1 Lists of paths in different BDDs for example system
Linked paths in figure 13.14 are the same in each ordering. The underlined paths do not appear in both orderings.

The system failure probability would, of course, be the same whichever of the two different variable orderings was used. However, to maintain the consistency and enable a structured approach, variable ordering 1 is selected as it makes the mathematical logic model easier to interpret from the viewpoint of the engineering application. For example, when both pump 3 and pump 4 have failed in standby (P3S.P4S), the failure of pump 1 is sufficient to bring about the system failure without having to consider the state of pump 2. That is, P3S.P4S.P1 is better than P1.P2.P3S.P4S in the sense of actual representation. Also, when P1 fails, pump 3 and pump 4 will be investigated in turn to determine if either of them can be activated to replace pump 1. If neither of them is available (P3S.P4S), the failure of pump 1 will bring about the system failure (P3S.P4S.P1). If pump 3 is not available (P3S) but pump 4 is available ($\overline{P4S}$) and thus activated, the failure of pump 1 is not sufficient to bring about the system failure and state of pump 2 would be relevant (P3S.$\overline{P4S}$.P1.P2). In contrast, the path P1.P2.P3S produced from the ordering 2 cannot reflect this process as in the real situation.
By referring to the general fault tree structure for typical standby systems shown in figure 13.10, component failures can be categorized into four groups as illustrated in table 13.2.

<table>
<thead>
<tr>
<th>Categories of component failures in standby systems</th>
<th>Component failures included in the category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Failures of duty components and corresponding supporting components (DCi, SUi)</td>
</tr>
<tr>
<td>2</td>
<td>Active failures of standby components and corresponding supporting component (SCiF, SUi+F)</td>
</tr>
<tr>
<td>3</td>
<td>Failures of components required to activate the standby components (C)</td>
</tr>
<tr>
<td>4</td>
<td>Standby failure of standby components (SCiS)</td>
</tr>
</tbody>
</table>

Table 13.2 Categories of component failures in standby systems

The preferred variable ordering on which the BDD construction is based is category 3 > category 4 > category 1 > category 2. In each category, the order between individual component failures is determined by the number of the appearances of the events in the fault tree.

Step 5: obtain disjoint paths leading to the system failure (terminal node ‘1’) in the BDD, and divide the events contained in each path into 8 categories. Categories 1 – 4 represent the occurrence of a component failure as in categories 1 – 4 in table 13.2. Categories 5 – 8 are defined to represent the non-occurrence of the component failures of categories 1 – 4 respectively.

Step 6: calculate the probability of each path.

In the calculation of the probability for each path, it is necessary to evaluate the correct form of the integral. Among the 8 categories of events contained in each path, the likelihood of events which belong to categories 5 and 7 can be quantified separately as they are both determined from a probability density function which is consistent over the whole time of system operation [0, T], and as they represent the success of a component over [0, T), they are not dependent on the condition of other components.

Among the 4 categories of component failures, categories 1 and 2 (duty component and active standby component failures) are key failures. They feature similar characteristics to initiating events (they place a demand on other components to respond) as the system will never fail if these events do not occur. An integral is then established centring around these key component failures. Some paths in the BDD contain more than one key component failure. This means that more than one integral may be formed depending on the order of the occurrence of these key component failures. Take for example the system in figure 13.11. One failure path obtained from the BDD in figure 13.12 is $\overline{P3S} \cdot P4S \cdot P1 \cdot P2 \cdot P3F$ as shown in table 13.1. As can be seen in this
path, three key component failures have occurred: P1, P2 and P3F. Between the three key component failures, four sequences can be identified as illustrated in figure 13.15.

For each sequence, an integral is established:

Sequence 1: \( \int_0^s f_{P1}(s) \left[ 1 - \int_0^s f_{P3S}(x) dx \right] \int_0^u f_{P2}(u) \left[ 1 - \int_0^u f_{P4S}(x) dx \right] \int_0^t f_{P3}(t-s) dt du ds \)

Sequence 2: \( \int_0^s f_{P1}(s) \left[ 1 - \int_0^s f_{P3S}(x) dx \right] \int_0^u f_{P3}(u-s) \left[ 1 - \int_0^u f_{P4S}(x) dx \right] \int_0^u f_{P2}(u) du ds \)
Sequence 3:
\[ \int_{0}^{t} f_{p3}(s) \left[ 1 - \int_{0}^{s} f_{p35}(x)dx \right] \int_{s}^{t} f_{p1}(u) \left[ 1 - \int_{u}^{t} f_{p45}(x)dx \right] \int_{0}^{s} f_{p3}(t-s)dtduds \]

Sequence 4:
\[ \int_{0}^{t} f_{p3}(s) \left[ 1 - \int_{0}^{s} f_{p35}(x)dx \right] \int_{s}^{t} f_{p5}(u-s) \left[ 1 - \int_{u-s}^{t} f_{p45}(x)dx \right] \int_{0}^{s} f_{p1}(t)dtduds \]

The path probability is equal to the sum of the probabilities of each sequence. It can be seen from the above example that to obtain the correct integral the key component failures must be investigated in association with the standby components to enable the different failure distributions of the standby components to be accounted for separately. To achieve this, a general algorithm is presented to identify all the possible sequences of key component failures contained in the path with respect to standby and controlling component failures. It also establishes the correct integral for each sequence. This is achieved in three stages described in procedures a), b) and c) below.

Procedure a): a list structure is established indicating how the standby components and the controlling component react to each of the key component failures included in the path. Each node \( i \) in the list represents the combination of states of the controlling components and the standby components at the time when the \( i^{th} \) key failure occurs. It also indicates the response of the standby component following activation triggered by this key failure. The underlying principle is whenever any key component failure occurs, the standby components are examined one by one to determine their availability to activate to replace the failed component. The process continues until a standby component is identified as available or all standby components have been investigated. To determine if a standby component is able to activate, it is required to establish the functionality or failure of the controlling/supporting components. Relevant events include the functionality or failure of the controlling component (categories 3 and 7), and the functionality or failure of components in standby (categories 4 and 8). Accordingly, the structure of each node in the list structure is defined as in figure 13.16.
For $i^{th}$ key component failure

<table>
<thead>
<tr>
<th>Events $e_a$ representing the failure to activate standby component (from categories 3 and 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events $e_c$ representing the success of the controlling component (from category 7)</td>
</tr>
<tr>
<td>Event $e_s$ representing the non-occurrence of standby failures (from category 8)</td>
</tr>
<tr>
<td>Event $e_{sa}$ representing the success of standby component following the activation (from category 6)</td>
</tr>
</tbody>
</table>

Figure 13.16 Node structure in the list

The reason that event $e_{sa}$ is included in the node is that it cannot be quantified separately as its failure probability is dependent on the time since activation. By linking it with the key failure which results in the activation of the corresponding standby component, its likelihood can be obtained by evaluating the integral over the correct time duration. The algorithm for establishing the list is presented in figure 13.17.

```c
Path j with events categorized into 8 groups
Establish_list(path j)
{
    define integer num_key, count;
    define pointer *head; /* pointing at the head node in the list */
    num_key = total of number of events in categories 1 & 2 in path j;
    initialize head;
    head->i=0;
    for(count = 1; count <= num_key; count++)
    {
        define pointer *new_node;
        initialize new_node;
        new_node->i = count;
        while(1st standby to last standby component)
        {
            if(has been activated in previous failures)
                continue;
            else
            {
                while(each corresponding functionally controlling/supporting component)
                {
                    if(failed)
                        add to events $e_a$; break;
                    else
                        add to event $e_c$ or $e_s$;
                }
                if(jumped out of the loop) /* unable to activate */
                    continue;
                else
                {
                    if(the standby component does not fail after the activation as indicated in the path)
                        add to event $e_{sa}$;
                    break;
                }
            }
        }
    }
}

Figure 13.17 The algorithm for establishing the list structure for each path
Procedure b): in this procedure, all the possible sequences of key component failures are identified. A general structure is defined to store all the sequences contained in the path. Assume that \( n \) key component failures are included in the path. Between the \( i^{th} \) and \((i+1)^{th}\) key failures, the sequence is identified in the structure shown in figure 13.18.

![Figure 13.18 Sequence structure](image)

In the structure illustrated in figure 13.18, each node represents a key component failure, i.e. the event belonging to categories 1 or 2 in the path. Node \( j_k \) represents that the failure of component \( C_j \) occurs as the \( k^{th} \) key failure. Each node has 3 pointers with one pointing at the next key component failure (solid vertical arrow), one pointing at the immediate preceding key failure (dotted vertical arrow), and the other pointing at the next possible component failure as this \( i^{th} \) key failure (horizontal arrow). Thus, between the \( i^{th} \) and \((i+1)^{th}\) key failures, all possible sequences can be established as

- \( Node_{11} \rightarrow Node_{11+i} \), \( Node_{12} \rightarrow Node_{12+i} \), \( \ldots \), \( Node_{1i} \rightarrow Node_{1i+i} \),
- \( Node_{21} \rightarrow Node_{21+i} \), \( Node_{22} \rightarrow Node_{22+i} \), \( \ldots \), \( Node_{2i} \rightarrow Node_{2i+i} \),
- \( \ldots \),
- \( \ldots \),
- \( Node_{mi} \rightarrow Node_{mi+i} \), \( Node_{m2} \rightarrow Node_{m2+i} \), \( \ldots \), \( Node_{mi} \rightarrow Node_{mi+i} \).

By applying this sequence structure to account for all \( n \) key failures contained in the path, all possible sequences of all the key failures can be identified.

In the process of establishing the sequence structure, all \( n \) key failures in the path are investigated one by one in the order from the first to the last. For the \( i^{th} \) key failure (\( i = 1, 2, \ldots, n \)), it needs first to identify which component failure can occur as this \( i^{th} \) key failure. Component failures which can occur as the key failure either come from duty component failures (from category 1) which have not been included in the sequence as previous key failures, or active failures of standby components (category 2) which have been activated by the first \((i-1)\) failures in the sequence. Therefore, when \( i = 1 \), i.e. the first key failure is considered, the possible component failures can only be duty component failures as no standby component will have activated. To determine whether a standby component \( SC_k \) has been activated and therefore is likely to fail as the \( i^{th} \) key component failure (\( i > 1 \)), one has to refer to the list structure established in procedure a) to see if this standby component has activated in the previous \((i-1)\)
key failures. For example, if in the list structure, the node corresponding to the $j^{th}$ key failure ($j < i$) contains event $\overline{SC_k}$ and $SC_k$ is the first available standby component to be activated, then the active failure of the standby component ($SC_kF$) can then be accepted as the $i^{th}$ key failure.

The algorithm for constructing the sequence structure for each path is shown in figure 13.19.

```c
Path j with events categorized into 8 groups
#define structure sequence_node
{ define variable e;
  sequence_node *p1, *p2, *p3;
} NODE;
/* p1: pointing at next key failure;
p2: pointing at other events for the failure;
p3: pointing at the last key failure; */

Establish_sequence(pathj, list *h for pathj)
{ define variable count;
  define node structure NODE;
  NODE *root;
  Initialize root;
  Root->p1=Root->p2=Root->p3=0;
  Count = 0;
  Identify_failsures(root, count, pathj, *h);
}

Identify_failsures(parent, count, pathj, list *h)
{ NODE *compare;
  while(each event k belonging to categories 1 or 2)
  { Compare=parent;
    While(compare!=NULL)
    { if(compare->e_k == event k)
      break;
      compare=compare->p3;
    }
  }
  if(compare==0)
  { /* event k is not included in the sequence */
    define variable add;
    add=0;
    if (event k belongs to category 1)
      add=1;
  }

Figure 13.19 Algorithm for constructing the sequence structure for each path

Procedure c): is to calculate the probability of each sequence based on the list structure and sequence structure established in procedures a) and b).

To illustrate how the integral is established for each sequence, the most simple situation is considered first, where $n$ independent component failures occur. Of the $n!$ sequences in which these $n$ component failures can occur, one sequence is identified as $C1>C2>...>Cn$. This sequence probability over the time period $[0, T)$ is then given in equation 13.6:
\[ P[\text{Sequence}] = \int_0^T \int_s f_{C_1}(s_1) \int_{s_1} f_{C_2}(s_2) \ldots \int_{s_{n-1}} f_{C_n}(s_n) ds_n ds_{n-1} \ldots ds_1 \]

where \( f_{C_k}(t) \) is the probability density function for component failure \( C_k \) in the sequence.

However, equation 13.6 is inappropriate when standby dependency is involved in the sequence. In this case, the probabilities of events which are included in the node (Figure 13.16) associated with each key component failure in the sequence, as well as the key component failure, need to be considered in relation to each other. Consider component failure \( e_k \) as the \( k \)th key failure in the sequence following the \((k-1)\)th failure occurring at \( t=S_{k-1} \), the probability that the \( k \)th failure occurs during the time period \([S_{k-1}, T]\), \( P_k(S_{k-1}, T) \), with other component states as indicated in the node corresponding to the \( k \)th failure, can be expressed in the integral form as:

\[ P_k(S_{k-1}, T) = \int_{S_{k-1}}^T f_{e_k}(s_k) \left[ 1 - \int_s^T f_{e_a(k)}(u) du \right] \left[ 1 - \int_s^T f_{e_n(k)}(u - s_k) du \right] \prod_a \int_s^T f_{e_a(k)}(u) du \ ds_k \]

where:

- \( f_{e_k}(t) \) is the probability density function for the \( k \)th key component failure \( e_k \);
- \( e_a(k) \) represents the standby failure event whose complement \( \overline{e_a} \) is included in the node (Figure 13.16) corresponding to the \( k \)th key failure; and \( \left[ 1 - \int_s^T f_{e_a(k)}(u) du \right] \) gives the probability that the event \( e_a(k) \) has not occurred by the time the \( k \)th failure \( e_k \) occurs;
- \( e_n(k) \) represents the active failure of the standby component following the activation with the complement \( \overline{e_n} \) included in the node corresponding to the \( k \)th key failure; and \( \left[ 1 - \int_s^T f_{e_n(k)}(u - s_k) du \right] \) gives the probability that the standby component has not failed since its activation at \( t=S_{k-1} \);
- \( e_a(k) \) represents the failure to activate the standby component as the event \( e_a \) included in the node corresponding to the \( k \)th key failure, and \( \prod_a \int_s^T f_{e_a(k)}(u) du \) gives the probability that these failure events have occurred by the time the \( k \)th failure \( e_k \) occurs at \( t=S_{k-1} \);

In equation 13.7, the probability density function \( f_{e_n(k)}(t) \) for the active failure of the standby component is only applicable from the time it is activated, hence the time is adjusted to \((u-s_k)\). In the same way, when the \( k \)th failure event \( e_k \) is an active failure of a standby component, its probability density function also needs adjusting to \( f_{e_n}(t_a - s_k) \), where \( t_a \) is the time that this standby component is activated.
The event $e_d(k)$ in equation 13.7 represents either the failure of the controlling component or the failure in standby of standby components. As the failure of the controlling component is always considered prior to the standby failures in the BDD construction (as dictated by the predetermined variable ordering), these two failure events will not be included in the same failure path, because when the controlling component is known to be failed, there is no need to investigate the availability of standby components.

Event $e_c$, which is also included in the node structure, is not considered in equation 13.7 as it represents the success of the controlling component through the system operation and thus can be quantified separately.

As a whole, equation 13.7 gives the probability that the $k^{th}$ key failure occurs during the time period $[s_k, t)$, and by the time the $k^{th}$ failure occurs, the controlling component or all standby components have failed (as in $\prod_c \int_{s_c}^{t} f_{e_c}(u) du$); or on its failure a standby component activates (i.e. the standby component has not failed during $[0, s_k)$ as in $\Big[1 - \int_{0}^{s_k} f_{e_c}(u) du\Big]$), and this standby component has not failed since its activation at $s_k$ (as in $\Big[1 - \int_{s_k}^{t} f_{e_s}(u- s_k) du\Big]$).

When the event contained in the path is a module, the probability density function for the module is defined as:

$$f_M(t) = \sum_{i=1}^{n} G_{q_i} (F(t)) f_{e_i}(t)$$  \hspace{1cm} (13.8)

where $n$ is the number of components included in the module; $f_{e_i}(t)$ is the probability density function for basic event $e_i$ contained in the module; and $G_{q_i} (F(t))$ is the criticality function of basic event $e_i$ regarding the module at time $t$.

Then the failure probability of a certain sequence of failures in the sequence structure can be obtained by substituting equation 13.7 into equation 13.6, with first key failure $e_{k1}$ occurring at $s_1$ during $[0, T)$, $2^{nd}$ $e_{k2}$ occurring at $s_2$ during $[s_1, T)$, ..., $n^{th}$ $e_{kn}$ occurring at $s_n$ during $[s_{n-1}, T)$. 

356
The path probability is then obtained as in equation 13.10:

\[
P[\text{Path } i] = \sum_j P[\text{Sequence } j] \prod_m \left[ 1 - \int_0^1 f_e(t) \, dt \right] \prod_k \left[ 1 - \int_0^1 f_c(t) \, dt \right]
\]

where \(m\) and \(k\) respectively represent the number of events \(e_m\) and \(e_c\) contained in the path which belong to categories 5 and 7 respectively.

Step 7: obtain the system failure probability by summing up all the path probabilities. That is,

\[
F_{\text{sys}}(T) = \sum_i P[\text{Path } i]
\]

13.2.3.2 Application

A safety system which features redundancy in its design is used to illustrate how the process described in the previous section is applied to evaluate the system reliability. The system is shown in figure 13.20.

![Example safety system](image)

Figure 13.20 Example safety system

The system is composed of four pumps (P1 – P4) as the key components to provide a water supply on demand. Pumps P1 and P2 are pre-set duty pumps, and pumps P3 and P4 serve as the
warm-spare standby pumps. All four pumps are electric pumps, of which P1 and P2 are powered by the electric supply 1, and P3 and P4 by the electric supply 2. On each pump stream, a pressure relief valve is fitted to protect the pumps in the event that the line blocks. The pressure relief valve may fail open under normal pressure when the pump is operating, thus reducing the water supply. A controller is fitted in the system to detect the pressure drop in the outward distribution channel either due to the failure of the pump or the failure of the pressure relief valve in which case it activates the standby pumps P3 or P4. There is no maintenance carried out when the system is functioning.

Step 1: establish the fault tree structure.

This is a 2-out-of-4 to function standby system which means that when 3-out-of-4 pumps fail, the system fails. The fault tree structure is constructed as shown in figure 13.21.
Step 2: establish the component dependency information and failure data. As can be seen from the fault tree in figure 13.21, the standby dependency exists between basic events 'P1', 'P2' representing the failure of duty pumps and 'P3F', 'P4F' representing the active failure of standby pumps. $f_{P1}(t), f_{P2}(t), f_{P3F}(t), f_{P4F}(t), f_{P3S}(t), f_{P4S}(t), f_{V1}(t) - f_{V4}(t), f_{c}(t), f_{E1}(t)$ and $f_{E2}(t)$ are established to define the failure times of relevant components.

Step 3: simplification and modularization is carried out on the fault tree structure in figure 13.22. The following modules are obtained:

- $M1 = P1 \text{ OR } V1$
- $M2 = P2 \text{ OR } V2$
- $M3 = P3 \text{ OR } V3$
- $M4 = P4 \text{ OR } V4$

The modularized fault tree structure is shown in figure 13.22.
Step 4: Construct BDDs for the modularized fault tree and identified modules.

For the system BDD, the variable ordering is $C > E2 > P3S > P4S > E1 > M1 > M2 > M3 > M4$. As the electric supply 2 is required for pump 3 or pump 4 to function and to activate, $E2$ is included in category 3 with regard to the variable ordering. This is because in terms of the standby components, the first consideration is always about whether they can activate in the first place. According to the variable ordering, the BDD for the system failure is shown in figure 13.23.

![Figure 13.23 BDD for the example safety system](image)

Step 5: Obtain the paths leading to system failure in the system BDD, and divide the events contained in each path into the 8 categories as defined in table 13.2. Table 13.3 shows all BDD failure paths, and the number included in the bracket following each event represents the category number to which the event belongs.

<table>
<thead>
<tr>
<th>Path number</th>
<th>Paths expressed in categorized events</th>
<th>Path number</th>
<th>Paths expressed in categorized events</th>
</tr>
</thead>
</table>

360
Step 6: calculate the probability of each path.

Procedure a): establish the list structure for each path. Take for example path 20 in table 13.3, its list structure is shown in figure 13.24.

As can be seen, the total number of key failures in this path is 3 (M1, M2, M3). When the first key failure occurs, the system examines each standby component in turn. As the controller C and the electric power supply E2 are both working and pump 3 has not failed in standby, pump 3 is activated. Then in the second key failure, as pump 3 has activated in response to previous failures, the availability of pump 4 is investigated. Again, as the controller C and the electric power supply E2 are both working and pump 4 has not failed in standby, pump 4 is activated. For the third failure, as both standby pumps have activated previously, no standby pump is available.

Procedure b): Establish the sequence structure for each path by following the algorithm in figure 13.18. The sequence structure for this path is shown in figure 13.25.
Four sequences of key component failures can be identified from the sequence structure in figure 13.26 for path 20 as M1→M2→M3, M1→M3→M2, M2→M1→M3, M2→M3→M1.

Procedure c): Calculate the probability of each sequence in the path according to equation 13.8. Take for example path 20, the probability of each sequence expressed in the form of integral is obtained as follows:

**Sequence 1: M1, M2, M3**

\[
P[\text{sequence 1}] = \int \int \int f_{M1}(s_1) \left[1 - \int f_{P3S}(x)dx \right] \int \int f_{M2}(s_2) \left[1 - \int f_{P4S}(x)dx \right] \int f_{M3}(s_3 - s_1)ds_3ds_2ds_1
\]

**Sequence 2: M1, M3, M2**

\[
P[\text{sequence 2}] = \int \int \int f_{M1}(s_1) \left[1 - \int f_{P3S}(x)dx \right] \int f_{M3}(s_2 - s_1) \left[1 - \int f_{P4S}(x)dx \right] \int f_{M2}(s_3)ds_3ds_2ds_1
\]

**Sequence 3: M2, M1, M3**

\[
P[\text{sequence 3}] = \int \int \int f_{M2}(s_1) \left[1 - \int f_{P3S}(x)dx \right] \int f_{M1}(s_2) \left[1 - \int f_{P4S}(x)dx \right] \int f_{M3}(s_3 - s_1)ds_3ds_2ds_1
\]

**Sequence 4: M2, M3, M1**

\[
P[\text{sequence 4}] = \int \int \int f_{M2}(s_1) \left[1 - \int f_{P3S}(x)dx \right] \int f_{M3}(s_2 - s_1) \left[1 - \int f_{P4S}(x)dx \right] \int f_{M1}(s_3)ds_3ds_2ds_1
\]

where:

\[
f_{M1}(t) = G_{P1}(F(t)).f_{P1}(t) + G_{V1}(F(t)).f_{V1}(t)
\]

\[
= [1-F_{V1}(t)].f_{P1}(t) + [1-F_{P1}(t)].f_{V1}(t)
\]
\[ f_{M1}(t) = [1 - \int_0^t f_{V1}(u) du] \cdot f_{P1}(t) + [1 - \int_0^t f_{P1}(u) du] \cdot f_{V1}(t) \]
\[ f_{M2}(t) = [1 - \int_0^t f_{V2}(u) du] \cdot f_{P2}(t) + [1 - \int_0^t f_{P2}(u) du] \cdot f_{V2}(t) \]
\[ f_{M3}(t) = [1 - \int_0^t f_{V3}(u) du] \cdot f_{P3}(t) + [1 - \int_0^t f_{P3}(u) du] \cdot f_{V3}(t) \]
\[ f_{M4}(t) = [1 - \int_0^t f_{V4}(u) du] \cdot f_{P4}(t) + [1 - \int_0^t f_{P4}(u) du] \cdot f_{V4}(t) \]

Then calculate the path probability according to equation 13.10.

In path 20, events \( \overline{E1} \), \( \overline{C} \) and \( \overline{E2} \) belong to categories 5 and 7 and are quantified separately from events of other categories. According to equation 13.10:

\[
P[\text{Path 20}] = \sum_{i=1}^{4} P[\text{sequence } i] \cdot \left[ 1 - \int f_{E1}(u) du \right] \cdot \left[ 1 - \int f_{C}(u) du \right] \cdot \left[ 1 - \int f_{E2}(u) du \right] \]

Step 7: calculate the system failure probability according to equation 13.11.

\[
F_{sys}(T) = \sum_{i=1}^{23} P[\text{Path } i]
\]

13.2.4 Dependency Modelling for Stress-related Dependency

Systems which involve, what is here termed, the stress-related dependency feature a parallel system structure with \( n \) sub-systems operating at the same time. It is assumed that the output from the system required to satisfy the specific demand is at least \( m \)-out-of-\( n \) parallel subsystems functioning normally. The system therefore fails when \( k \)-out-of-\( n \) subsystems fail (where \( k = n - m + 1 \)). The typical configuration of each subsystem is composed of one key component, which provides the essential system function, and passive supporting components, such as a power supply, pipework and valves etc., which enable the key component to function. Some of the supporting components can be exclusive to the specific subsystem, while other supporting components may be common to more than one subsystem.

The stress-related dependency arises when only \( p \) (\( m \leq p < n \)) subsystems are functioning. The failure of the \( (n-p) \) subsystems means that the working load of the functioning part of the system will have to increase to meet the required output. This consequently puts extra loading on the working elements and increases their failure rates. The variable failure rates experienced by the key components mean that the failure probability of the key components cannot be modelled over the whole system functioning period \([0, T]\) since its probability density function is
dependent upon the performance of other components. For a $k$-out-of-$n$ to fail parallel system, $k$ probability density functions have to be defined for each key component corresponding to situations where $0, 1, \ldots, k-1$ subsystems have failed. Then with respect to the system failure, i.e. $k$ subsystem failures, the failure probability of the key components is investigated over $k$ separate periods of time $[0, t_1), [t_1, t_2), \ldots, [t_{k-1}, T)$ using the appropriate probability density functions. The start of each of the periods of time is marked with the failure of another subsystem.

The following section describes the general process of using a BDD to analyse parallel systems featuring the stress-related dependency.

### 13.2.4.1 Generic Process

A general parallel system is used to illustrate how the analysis proceeds using the BDD method. The system consists of $n$ parallel subsystems and fails when $k$ subsystems fail.

Step 1: Represent the System Failure Logic.

Establish the fault tree structure to represent the system failure. Figure 13.26 displays the fault tree representation for the failure of a parallel system with the general features described above for the stress-related dependency.

![General fault tree structure for parallel system](image)

where: the basic event $SS_j$, represents the failure of the $i$th supporting component exclusive to the subsystem $j$; the basic event $CS_{j-i}$ represents the failure of the supporting component which is common to subsystems $j, j+1, \ldots, i$.

Figure 13.26 General fault tree structure for parallel system
Step 2: Establish Component Failure Data.

Establish the dependency and basic event data for the developed fault tree. In the system, the stress-related dependency existing between the $n$ key components gives rise to a statistical dependency. In terms of the stress-related dependency, information is required such as the total number of components involved, the name of the relevant basic events and the number of situations which need to be considered with respect to the probability density functions for each key component. In the general $k$-out-of-$n$ system, $k$ situations are considered where the number of failed subsystems increase from 0 to $k-1$.

The basic event data is also required to account for the influence of the stress-related dependency. For the basic event $KCi (1 \leq i \leq n)$ in figure 13.26, which represents the component failure involved in the stress-related dependency, $k$ probability density functions $f_{KC1}^0(u)$, $f_{KC1}^1(u)$, ..., $f_{KC1}^{k-1}(u)$ are defined which correspond to situations where 0, 1, ..., $k-1$ subsystems have failed respectively.

Step 3: Fault Tree Modularisation

Carry out the simplification and modularization process on the fault tree structure. In this process, the failure of key components ($KC1$, ..., $KCn$) can be combined with its own functionally supporting component ($SS1$, ..., $SSn$) to form a module.

Step 4: System BDD Construction

Construct the system BDD based on the modularized fault tree structure, and the BDDs for each identified module. The modified left-right top-down method is adopted to decide the variable ordering in the process of the BDD construction.

Step 5: System Failure Mode Identification

Identify all disjoint paths for the system BDD leading to the system failure (i.e. terminal node '1'), and categorize the events contained in each path into 4 groups: category 1 includes events which represent key component failures or failure of modules which contain key components, i.e. failures involved in stress-related dependency; category 2 includes events which represent the failure of supporting components; category 3 and category 4 are similar to categories 1 and 2,
but contain the success of a component/module through the system functioning period, i.e. the non-occurrence of the corresponding events contained in categories 1 and 2 respectively.

Step 6: Failure Mode Quantification

System quantification requires the calculation of the probability of each path to a terminal ‘1’ state on the BDD. In this process, the sequence of the component failures will need to be taken into account when events belonging to category 1 are included in the path. This is illustrated through a simple system which contains two parallel components P1 and P2. The system fails when both of the components fail. The stress-related dependency exists between P1 and P2 as the failure rate of either P1 or P2 increases when the other component fails. The corresponding path identified from the BDD constructed for the system is P1.P2. Both of the two events belong to category 1, and two sequences can be identified with respect to the occurrence of the two events as illustrated in figure 13.27.

![Diagram of two sequences of component failures in the simple parallel system](image)

The probability that the system fails with components P1 and P2 failing in sequence 1 is given in equation 13.12:

\[
P[\text{sequence 1}] = \int f_{P_1}^2(s) \left[ 1 - \int f_{P_2}^0(u) du \right] \left[ \int f_{P_2}^1(t-s) dt \right] ds
\]

In equation 13.12, the event that component P2 has not failed prior to P1 has to be investigated separately from the event that component P2 fails after P1 does. This is because no continuous probability density function can be defined for component P2 over the whole system operating duration [0, T). Instead, two probability density functions, \( f_{P_2}^0(u) \) and \( f_{P_2}^1(u) \), are defined for component P2, of which \( f_{P_2}^0(u) \) applies when both components are working and
\[ 1 - \int f_{P2}(u) du \] gives the probability that component \( P2 \) does not fail during \([0, s)\), and \( f_{P2}^1(u) \) applies when component \( P1 \) has failed and \( \int f_{P2}^1(t-s) dt \) gives the conditional probability that component \( P2 \) fails following the failure of \( P1 \) during \([s, T)\). Since the probability density function \( f_{P2}^1(u) \) is not applicable until component \( P1 \) has failed, the time parameter is adjusted to \((t-s)\).

In the same way, the probability that the system fails with \( P1 \) and \( P2 \) failing in sequence 2 is obtained in equation 13.13.

\[
P[\text{sequence 2}] = \int f_{P1}^0(s) \left[ 1 - \int f_{P2}^0(u) du \right] \left[ \int f_{P2}^1(t-s) dt \right] ds
\]

Finally, the path probability can be obtained as the sum of the two sequence probabilities.

To determine the system failure probability, it is also necessary to account for the sequence of failure events in category 1 and category 2. Considering the general fault tree in figure 13.27, when events \( KC_i \) and \( CS_{kj} \) \((i \neq [k, j])\) are both included in a BDD path, consideration needs to be given to the sequence of the occurrence of these two events. When the event \( CS_{kj} \) occurs prior to the event \( KC_i \), the probability density function which is used to obtain the failure probability of \( KC_i \) would be different from when the sequence of occurrence is reversed. This is because the occurrence of the event \( CS_{kj} \) would result in the failure of \((j-k+1)\) subsystems and consequently the moment when the event \( CS_{kj} \) occurs in relation to the event \( KC_i \) will determine which probability density function for \( KC_i \) would apply.

When events belonging to category 3 are included in the path together with events of categories 1 or 2, the probability of the path must be calculated considering all of the events at the same time to account for the dependency. Take for example the general fault tree in figure 13.27, assume that one path which occurs in the BDD is \( KC_1.KC_2.....KC_i.....KC(k+1) \) \((1 \leq i \leq k)\), which features the event \( KCi \) from category 3, and the other \( k \) events belonging to category 1. Considering the event \( \overline{KC_i} \), whichever sequence is under investigation, the probability \( P[\overline{KC_i}] \) cannot be obtained separately as the probability density function for \( \overline{KC_i} \) varies as the number of failed subsystems increases. The correct way to account for the event \( \overline{KC_i} \) in the sequence probability is to consider separate periods of time over which the number of subsystem failures is known. Consider the situation where the events occur in a sequence \( KC1>KC2>...>KC(i-1)>KC(i+1)>...>KC(k+1) \) as illustrated in figure 13.28:
In this sequence, the probability that the key component KCi does not fail during the system operation is evaluated over separate periods of time \([s_{j-1}, s_j)\) \((j = 1, 2, ..., k\) as \(k\) component failures are required for the system to fail), which are determined by the moment when each key component failure KC\(j\) occurs. That is, the event \(\overline{KCi}\) is considered over periods of time which represent that the component KC\(i\) does not fail prior to the first subsystem failure, KC\(i\) does not fail between the first and second subsystem failure, ..., does not fail between the \((k-1)\)th and the \(k\)th subsystem failure. Accordingly, the sequence probability expressed in the integral form can be obtained in equation 13.14.

\[
\text{P[sequence]} = \left[ k \int_{s_{j-1}}^{s_j} f_{KC1}^0(s_{j-1}) \prod_{j=2}^{k+1} \left( 1 - \int_{s_{j-2}}^{s_j} f_{KCj}^0(u)du \right) \int_{s_{j-1}}^{s_j} f_{KC2}^1(s_j - s_{j-1}) \prod_{j=3}^{k+1} \left( 1 - \int_{s_{j-3}}^{s_j} f_{KCj}^0(u - s_{j-1})du \right) \prod_{j=4}^{k+1} \left( 1 - \int_{s_{j-4}}^{s_j} f_{KCj}^0(u - s_{j-3})du \right) \right] ds_k ds_{k-1} ... ds_1
\]

13.14

In calculating the path probability, whichever sequence is under investigation, the events which belong to category 4 can always be quantified independently because a continuous probability density function can be defined for each of these events over the whole system operation duration \([0, T)\).

When the key component failure event is included in a module along with its supporting component failure events, the probability density function for the resulting module also varies as the number of failed subsystems increases. If a module M\(j\) consists of key component failure event KC\(j\) and the event representing the failure of its supporting component SS\(j\), then the probability density function for module M\(j\) with \(n_s\) subsystem having already failed \((0 \leq n_s \leq k-1)\) is obtained as shown in equation 13.15.

\[
f_{M_j}^{n_s}(t) = G_{KCj}(F(t)).f_{KCj}(t)_{|n_s \text{ subsystems have failed}} + G_{SSj}(F(t)).f_{SSj}(t)_{|n_s \text{ subsystems have failed}}
\]

\[= G_{KCj}(F(t)).f_{KCj}(t) + G_{SSj}(F(t)).f_{SSj}(t)\]
\[ \begin{align*}
&= \left(1 - \int f_{SS_j}^{m_j}(u)du \right) f_{K_j}^{m_j}(t) + \left(1 - \int f_{K_j}^{m_j}(u)du \right) f_{SS_j}(t) \\
&= \left(1 - \int f_{SS_j}^{m_j}(u)du \right) f_{K_j}^{m_j}(t) + \left(1 - \int f_{K_j}^{m_j}(u)du \right) f_{SS_j}(t)
\end{align*} \]  

13.15

where \( G(F(t)) \) is the criticality function for each component with respect to the failure of module \( M_j \).

Consequently, to obtain the path probability, it requires an exhaustive list of all possible sequences of the events in the path. The probability for each sequence can then be calculated. The following procedures are used for the probability evaluation:

**Procedure (a):** this procedure will judge whether the sequence of events needs to be considered. When events which belong to categories 1 and 3 are contained in the path, different sequences of event occurrence will need to be identified.

**Procedure (b):** determines the path probability based on the outcome of procedure (a). If procedure (a) has identified that the sequence of failures is not important: the path only contains events representing supporting component failures in the system, such as \( SS_j \) and \( CS_{j-l} \) in figure 13.27, and none of the events is involved in the stress-related dependency. In this case, the path probability is given by the product of the probability of each individual event in the path as shown in equation 13.16.

\[ P[\text{Path}] = \prod_i Q_{E_i} \prod_j \left[1 - Q_{E_j}\right] \]  

13.16

where \( i \) and \( j \) represent the number of events which belong to categories 2 and 4 respectively; and \( Q_{E_i} = \int f_{E_i}(u)du \).

If procedure (a) has identified the necessity to account for the event sequence: in this case, the path probability is equal to the sum of all sequence probabilities. To identify all the possible sequences contained in the path and obtain the correct probability for each sequence, the process continues with procedures (c) and (d).

**Procedure (c):** A structure is defined to generate all the possible sequences contained in a specific path. The structure is constructed by considering events which belong to categories 1 and 2. The algorithm underlying the construction of the structure is similar to that represented in figure 13.18. Each node represents an event included in the path which belongs to category 1 or
2. An additional piece of information is also included in each node which is the number of subsystem failures resulting from the specific component failure represented by the node. Each node has 3 pointers, one points at the next component failure in the sequence, one points at the immediately preceding component failure in the sequence, and the last one points at the component failure which can be an alternative to the current component failure and appear in the same place in the sequence. Assume that events C1, C2, ..., Ck are events included in a path which belong to categories 1 and 2. Figure 13.29 displays the sequence structure for this series of events. n1, n2, ..., ni, ..., nk contained in each node in the figure represent the number of subsystem failures resulting from each corresponding component failure event C1, C2, ..., Ci, Ck respectively.

where the solid-line vertical pointer points at the next failure event in the sequence; the dotted-line vertical pointer points at the immediately preceding event in the sequence; and the horizontal pointer points at an alternative failure in the same place in the sequence.

Figure 13.29 Illustration of the sequence structure

A situation may arise where the same subsystem failures can result from different component failures. For example, a component failure event Cj results in the failure of subsystems Sm, Sn and S_p, and another component failure Ci results in the failure of subsystems S_p, Sr and Sy. In this case, when events Cj and Ci are both included in a path for which the sequence structure needs to be constructed, the number nj or ni included in nodes Cj or Ci would need to be adjusted appropriately dependent upon which event appears later in the sequence. If the event Ci occurs
after $C_j$ in the sequence, the value of $n_i$ will be set as '2' instead of '3' as the failure of subsystem $S_p$ has already been accounted for when the event $C_j$ occurs.

Figure 13.30 shows the algorithm for constructing the sequence structure.

```
Path $P$, including events $E1$, $E2$, ..., $E_j$ which belong to categories 1 or 2
Node structure
{
   Event_name $N_i$; No. of subsystem failures $n_i$;
   Pointer $p_1, p_2, p_3$/* $p_1$ pointing at the next event in the sequence; $p_2$ pointing at the alternative event in the sequence; $p_3$ pointing at the preceding event in the sequence */
}

Construct_sequences($P_i$)
{
   Establish root node $Root$;
   $Root$->$N_E$ = Null;
   $Root$->$n_i$ = 0;
   $Root$->$p_1$ = $Root$->$p_2$ = $Root$->$p_3$ = 0;
   Sub_construct($Root$, $P_i$);
   Return($Root$);
}

Sub_construct(root, $P_i$)
{
   Define pointer $p_{wk}$;
   Define pointer head = 0, end = 0;
   while(...$E_k$...)/* 1 $k$ $j$ */
   {
      $p_{wk}$ = root;
      while($p_{wk}$!=0)
      {
         If($p_{wk}$->$N_E$ = $E_k$)
            Break;
         $p_{wk}$= $p_{wk}$->$p_3$;
      }
      If($p_{wk}$ = = 0)/* has not appeared in the sequence */
         {Establish new_node;
         
         new_node->$N_E$ = $E_k$;
         new_node->$n_i$ = judging from fault tree;
         new_node->$p_1$ = new_node->$p_2$=0;
         new_node->$p_3$ = root;
         If(head = = 0)
            head = new_node;
         root->$p_1$ = new_node;
         end = new_node;
         else
            end->$p_2$ = new_node;
            end = new_node;
      }
   }
   If(head != 0)
      Sub_construct(head, $P_i$);
   $p_{wk}$ = root->$p_2$;
   while($p_{wk}$ != 0)
      { Sub_construct($p_{wk}$, $P_i$);
      $p_{wk}$ = $p_{wk}$->$p_2$;
      }
}

Figure 13.30 Algorithm for construct the sequence structure for stress-related dependency

Procedure (d): evaluates the probability of each sequence using the sequence structure. The sequence probability is expressed in integral form using the appropriate probability density functions for the failure times of events contained in the sequence. In figure 13.30, assume that the event $C_i$ contained in the first sequence (left-hand vertical branch) represents a component failure which is involved in the stress-related dependency, i.e. belonging to category 1. It therefore features a varying probability density function dependent upon the number of subsystem failures. In the specific sequence, by tracing the upward pointer from node $C_i$ to identify the preceding event in the sequence, , it can be seen that by the time the component $C_i$ fails, $(n_1+n_2+...+n_{i-1})$ subsystems have failed due to the occurrence of the failure events $C_1$, $C_2$, ..., $C_{i-1}$, and $f^r_{C_i}^{n_1+n_2+...+n_{i-1}}(u)$ is the correct probability density function for component $C_i$ to be used for this specific sequence. Also as the $i$th failure in the sequence, it means that component $C_i$ has not failed when the previous $(i-1)$ component failures occur. The non-occurrence of the
failure event needs to be accounted for at the time of the other failures as shown in equation 13.14.

The approach suggested above produces a sequence which contains only events which belong to categories 1 and 2. The process also needs to be able to account for events in category 3. Equation 12.14 displays how events belonging to category 3 are integrated with events of category 1 or 2. Assume that a path includes events $C_1, C_2, \ldots, C_k$ which belong to categories 1 or 2, events $E_1, E_2, \ldots, E_j$ which belong to category 3 and $D_1, D_2, \ldots, D_m$ which belong to category 4. For the specific sequence $C_1 \rightarrow C_2 \rightarrow \ldots \rightarrow C_k$, its probability can then be expressed in the integral form as in equation 13.17.

$$P[\text{sequence}] = \prod_{p=1}^{m} \left[ 1 - \int_0^1 f_{D_p}(u)\,du \right] \cdot \prod_{i=1}^{k} \left[ 1 - \int_0^1 f_{C_p}(u)\,du \right] \cdot \prod_{p=1}^{j} \left[ 1 - \int_0^1 f_{E_p}(u)\,du \right]$$

The term $f_{C_p}(u)$ represents the cumulative distribution function of the stress-related dependency for event $C_p$. For the specific path $C_1 \rightarrow C_2 \rightarrow \ldots \rightarrow C_k$, its probability can be expressed as

$$\prod_{p=1}^{m} \left[ 1 - \int_0^1 f_{D_p}(u)\,du \right] \cdot \prod_{i=1}^{k} \left[ 1 - \int_0^1 f_{C_p}(u)\,du \right] \cdot \prod_{p=1}^{j} \left[ 1 - \int_0^1 f_{E_p}(u)\,du \right] \cdot \int_{s_1}^{s_2} \cdots \int_{s_{k-1}}^{s_k} f_{C_k}(s_k - s_{k-1}) \, ds_k \, ds_{k-1} \, \cdots \, ds_2 \, ds_1$$

All sequence probabilities can then be obtained according to the algorithm underlying equation 13.17. Finally, the probability for a specific path is obtained by adding up all sequence probabilities.

Step 7: calculate the system failure probability according to equation 13.11.

13.2.4.2 Application

A parallel safety system which involves the stress-related dependency is used to illustrate how the generic process described above is applied to evaluate the system reliability. The system is shown in figure 13.31.
The system is composed of four pumps (P1 – P4) as the key components to provide a water supply on demand. Pumps P1 and P2 are electric powered and pumps P3 and P4 are powered by a diesel supply. On each pump stream, a pressure relief valve is fitted to protect the pumps in the event of line blockage. The pressure relief valve may fail open under normal pressure when the pump is operating, thus reducing the water supply. The system starts with all four pumps operating. When one or two streams fail, the work load on the pumps on the functioning streams will increase to ensure that the water supplied should still satisfy the demand. This results in higher failure rates of the pumps. The system fails when 3-out-of-4 streams fail. There is no maintenance carried out when the system is functioning.

Step 1: Develop the fault tree structure for the system shown in figure 13.31. Figure 13.32 displays the fault tree representation of the system failure logic.

---

Figure 13.31 Example parallel safety system

Figure 13.32 Fault tree structure for the parallel system
Step 2: Establish the dependency and basic event information required.

Since the example system is a 3-out-of-4 to fail redundant system, 3 probability density functions, $f_{P_i}(t)$, $f_{V_i}(t)$ and $f_{P_i^2}(t)$ ($1 \leq i \leq 4$), have to be defined for pumps $P_i$ ($i = 1 \ldots 4$) which are involved in the stress-related dependency. Each of the probability density functions corresponds to the three situations where all pumps are functioning (i.e. no pump has failed), one pump has failed and two pumps have failed.

Step 3: carry out the simplification and modularization process on the fault tree structure. In this process, four modules are identified as:

$$M_1 = P_1 + V_1; \quad M_2 = P_2 + V_2; \quad M_3 = P_3 + V_3; \quad M_4 = P_4 + V_4.$$ 

Step 4: construct the system BDD based on the modularized fault tree structure. Figure 13.33 shows the BDD constructed on the variable ordering $ES > DS > M_1 > M_2 > M_3 > M_4$. The left and right branch of each node correspond to ‘1’ and ‘0’ respectively, representing the occurrence and non-occurrence of the event.

![BDD constructed for the parallel system](image)

Step 5: identify disjoint paths leading to the system failure from the BDD shown in figure 13.33. Categorization of the events contained in each path is also carried out in this step. Table 13.4 displays all the paths of the BDD which cause the fault tree top event. The number in the bracket following each event is the category to which the event belongs.
Step 6: calculate the probability of each path shown in table 13.4. Of the 9 paths in table 13.4, only path 1 does not include any event which belong to categories 1 or 3. This means that for path 1 the sequence of events does not need to be taken into consideration and its probability can be obtained according to equation 13.16 as:

\[ P[\text{path } 1] = \int_0^1 f_{ES}(u) \, du \cdot \int_0^1 f_{DS}(u) \, du \]

All other 8 paths contain events which belong to categories 1 or 3, and as such procedures (c) and (d) will be followed to identify all the possible sequences for each path and calculate the sequence probabilities. This is demonstrated considering paths 3 and 7 in table 13.4.

In path 3, the event \( \overline{DS} \) belongs to category 4 and therefore can be quantified separately. A sequence structure is constructed based on events ES and M4 as shown in figure 13.34.

Figure 13.34 Sequence structure for path 3 in table 13.4

The event \( \overline{M3} \) will be considered with respect to each component failure in both of the sequences shown in figure 13.34. The probability of the two sequences ES\( \rightarrow \)M4 and M4\( \rightarrow \)ES are:

\[
P[\text{sequence 1}] = \int_0^1 f_{ES}(s_1) \left[ 1 - \int_0^1 f_{M3}^o(u) \, du \right] \left[ 1 - \int_0^1 f_{M4}^o(u) \, du \right] \int_0^{s_1} f_{M4}^o(u) \, du \int_0^{s_2-s_1} f_{DS}(u) \, du \cdot \left[ 1 - \int_0^1 f_{DS}(u) \, du \right]
\]
The probability of path 3 can then be obtained as:

\[ P[\text{path 3}] = P[\text{sequence 1}] + P[\text{sequence 2}] \]

For path 7, events \( ES \) and \( DS \) are quantified separately. The sequence structure based on events M1, M2 and M4 is shown in figure 13.35.

Figure 13.35 Sequence structure for path 7 in table 13.4

Six sequences are contained in the sequence structure in figure 13.35. Take for example sequence 1, \( M1 \rightarrow M2 \rightarrow M4 \), its probability is:

\[ P[\text{sequence 1}] = \]

\[ \int_0^\infty f_{M4}^0(s_1) \left[ \int_0^\infty f_{M3}^0(u)du \right] \int_0^\infty f_{ES}(s_2) \left[ \int_0^\infty f_{M3}^1(u-s_1)du \right] ds_2ds_1 \left[ \int_0^\infty f_{DS}(u)du \right] \]

where:

\[ f_{M3}^{\text{0/1/2}}(t) = G_p^2(F(t))f_{M3}^{\text{0/1/2}}(t) + G_v^2(F(t))f_{V3}(t) \]

\[ = \left[ 1 - \int_0^\infty f_{V3}(u)du \right] f_{M3}^{\text{0/1/2}}(t) + \left[ 1 - \int_0^\infty f_{V3}^0(u)du \right] f_{V3}(t) \]

\[ f_{M4}^{\text{0/1/2}}(t) = G_p^2(F(t))f_{M4}^{\text{0/1/2}}(t) + G_v^2(F(t))f_{V4}(t) \]

\[ = \left[ 1 - \int_0^\infty f_{V4}(u)du \right] f_{M4}^{\text{0/1/2}}(t) + \left[ 1 - \int_0^\infty f_{V4}^0(u)du \right] f_{V4}(t) \]
The probability for other sequences can be obtained according to the same algorithm, and the path probability can then be obtained.

Step 7: obtain the system failure probability according to equation 13.11.

\[ F_{sys}(T) = \sum_{i=1}^{9} P[Path \ i] \]
Chapter 14. Conclusions and Further Work

14.1 Summary
Dependency relationships between components in a system render the fault tree analysis (FTA) and its efficient implementation, the BDD approach, inappropriate in predicting the system failure probability. Consequently, the Markov method is used to assess systems which involve dependencies.

The applicability of the Markov analysis is restricted by the state-space explosion problem which may arise for large systems. To solve this problem, a process has been established where a fault tree is used to represent the system failure logic and the smallest independent sections (modules) which contain each dependency are identified and analysed by the best method. Thus, BDD and the Markov analysis are applied in a combined way to improve the analysis efficiency. The BDD method is applied to modules which contain no dependency, and the Markov analysis applied to modules in which dependencies exist.

Different types of dependency which can arise in an engineering system assessment were identified in this research as below. Different algorithms for establishing a Markov model were also developed for each type of dependency.

- Maintenance dependency
- Standby dependency
- Sequential dependency
- Sequence-enforcing dependency
- Secondary-failure dependency
- Initiator-enabler dependency
- Revealing dependency
- Test dependency
- Functional dependency
- Switching dependency

Three types of system were investigated in this research in the context of dependency modelling, the continuously-operating system, the active-on-demand system and the phased-mission system. For the analysis of continuously-operating systems, quantification is carried out over a specified period of time assuming that the system starts with all components operational. System
availability and failure intensity at specific time points can be obtained as well as the expected number of system failures.

The analysis of active-on-demand systems, such as safety/protection systems, requires both the system availability and reliability to be evaluated. This type of system resides in a standby state during which time some component failures may still occur, and is required to react when a particular hazardous event happens. The static-dynamic two-phase approach has been developed for the analysis of active-on-demand systems. This approach identifies the causes of system unavailability and unreliability separately by developing fault trees for each type of failure. The system unavailability is obtained by quantifying the static-phase fault tree. To obtain the system unreliability, all possible states are identified from which the system is able to activate, and the quantification is carried out over the dynamic phase using the Markov method.

For phased-mission systems, a modified phase algebra method was developed to enable the analysis of both non-repairable and repairable systems which involve dependencies. A mission BDD is constructed and failure modes for each phase can be derived. Depending on whether dependency exists or the repair is carried out during the phase, it is determined whether the BDD approach or the Markov method applies to the phase analysis. The system failure probability in each phase as well as the mission unreliability are obtained.

A further attempt to improve the efficiency of dependency modelling was carried out in this research by investigating the use of BDD in assessing non-repairable systems involving dependencies. Four types of dependency were studied: the initiator-enabler dependency, the sequential dependency, the standby dependency, and the stress-related dependency. General processes have been established for each type of dependency to carry out the quantification.

Figure 14.1 provides a complete picture of the thesis by highlighting the connection between each chapter as well as the contribution in each chapter.
14.2 Conclusions

- The identification of smallest dependency modules proves to be effective in improving the efficiency as well as applicability of the Markov analysis for large systems.
- Typical types of dependency are identified and their characteristics are captured in the process of constructing the Markov model.
For the active-on-demand systems, the static-dynamic two-phase approach distinguishes between different causes to the system failure in standby and in operation, and enables the calculation of both system unavailability and unreliability.

For phased-mission systems, the modified phase algebra method identifies each possible failure mode for each individual phase throughout the mission BDD. The Markov analysis applies when dependencies or repair are involved in the phase, otherwise, the BDD approach applies. This improves the analysis efficiency.

For non-repairable systems involving dependencies, the BDD approach offers an alternative to the Markov method. It is able to account for dependency relationships through the integral representation of system failure probability.

14.3 Further Work

- To look into how the BDD approach can be applied to repairable systems involving dependencies
- To write program codes to enable the quantification of non-repairable systems with dependencies using the BDD approach; and compare the efficiency of the approach with the Markov method
- To carry out the comparison between the Markov method and the Bayesian Belief Network (BBN) with respect to dealing with dependencies
References:


382


385


Appendix A Markov Model for Module in Chapter 8

<table>
<thead>
<tr>
<th>State no.</th>
<th>M6005</th>
<th>e8</th>
<th>M6003</th>
<th>e5</th>
<th>e6</th>
<th>e11</th>
<th>Module State</th>
<th>State no.</th>
<th>M6005</th>
<th>e8</th>
<th>M6003</th>
<th>e5</th>
<th>e6</th>
<th>e11</th>
<th>Module State</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 91</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 92</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STATE 3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 93</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 94</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STATE 5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 95</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>STATE 96</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>STATE 97</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 98</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>STATE 9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 99</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>STATE 10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>STATE 11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>STATE 101</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>STATE 12</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>STATE 102</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STATE 13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 103</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 14</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 104</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STATE 15</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>STATE 105</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>STATE 16</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>STATE 106</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>STATE 17</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 107</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>STATE 18</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>STATE 108</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STATE 19</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>STATE 109</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 20</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 110</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STATE 21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>STATE 111</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>STATE 112</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STATE 23</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 113</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STATE 24</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>STATE 114</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>STATE 115</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STATE 26</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 116</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>STATE 27</td>
<td>1 1 0 2 0 0 0 0</td>
<td>STATE 117</td>
<td>1 1 0 0 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 28</td>
<td>1 1 0 0 3 0 1</td>
<td>STATE 118</td>
<td>1 1 0 0 3 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 29</td>
<td>1 1 0 0 0 3 1</td>
<td>STATE 119</td>
<td>1 2 1 1 3 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 30</td>
<td>1 0 1 1 0 0 0</td>
<td>STATE 120</td>
<td>1 2 1 1 0 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 31</td>
<td>1 0 1 0 3 0 1</td>
<td>STATE 121</td>
<td>1 0 1 1 1 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 32</td>
<td>1 0 1 0 0 3 1</td>
<td>STATE 122</td>
<td>1 0 1 1 3 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 33</td>
<td>1 2 0 1 0 0 0</td>
<td>STATE 123</td>
<td>1 0 1 1 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 34</td>
<td>1 0 0 1 3 0 1</td>
<td>STATE 124</td>
<td>1 0 1 0 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 35</td>
<td>1 0 0 1 0 3 1</td>
<td>STATE 125</td>
<td>1 0 1 0 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 36</td>
<td>1 0 0 0 1 0 1</td>
<td>STATE 126</td>
<td>1 0 1 0 3 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 37</td>
<td>1 0 0 0 0 3 3 1</td>
<td>STATE 127</td>
<td>1 2 0 1 1 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 38</td>
<td>1 0 0 0 0 1 1</td>
<td>STATE 128</td>
<td>1 2 0 1 3 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 39</td>
<td>0 1 1 2 0 0 0</td>
<td>STATE 129</td>
<td>1 2 0 1 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 40</td>
<td>0 1 1 0 3 0 1</td>
<td>STATE 130</td>
<td>1 0 0 1 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 41</td>
<td>0 1 1 0 0 3 1</td>
<td>STATE 131</td>
<td>1 0 0 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 42</td>
<td>0 1 0 2 3 0 1</td>
<td>STATE 132</td>
<td>1 0 0 3 3 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 43</td>
<td>0 1 0 2 0 3 1</td>
<td>STATE 133</td>
<td>0 1 1 2 1 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 44</td>
<td>0 1 0 0 1 0 1</td>
<td>STATE 134</td>
<td>0 1 1 2 3 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 45</td>
<td>0 1 0 0 3 3 1</td>
<td>STATE 135</td>
<td>0 1 1 2 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 46</td>
<td>0 1 0 0 0 1 1</td>
<td>STATE 136</td>
<td>0 1 1 0 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 47</td>
<td>0 2 1 1 0 0 0</td>
<td>STATE 137</td>
<td>0 1 1 0 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 48</td>
<td>0 0 1 1 3 0 1</td>
<td>STATE 138</td>
<td>0 1 1 0 3 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 49</td>
<td>0 0 1 1 0 3 0</td>
<td>STATE 139</td>
<td>0 1 0 2 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 50</td>
<td>0 0 1 0 1 0 0</td>
<td>STATE 140</td>
<td>0 1 0 2 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 51</td>
<td>0 0 1 0 3 3 0</td>
<td>STATE 141</td>
<td>0 1 0 2 3 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 52</td>
<td>0 0 1 0 0 1 0</td>
<td>STATE 142</td>
<td>0 2 1 1 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 53</td>
<td>0 2 0 1 3 0 1</td>
<td>STATE 143</td>
<td>0 2 1 1 3 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 54</td>
<td>0 2 0 1 0 3 1</td>
<td>STATE 144</td>
<td>0 2 1 1 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 55</td>
<td>0 0 0 1 1 0 0</td>
<td>STATE 145</td>
<td>0 0 1 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 56</td>
<td>0 0 0 1 3 3 0</td>
<td>STATE 146</td>
<td>0 0 1 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 57</td>
<td>0 0 0 1 0 1 0</td>
<td>STATE 147</td>
<td>0 0 1 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------------</td>
<td>------------</td>
<td>----------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 58</td>
<td>0 0 0 1 3 0</td>
<td>STATE 148</td>
<td>0 2 0 1 1 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 59</td>
<td>0 0 0 1 1 0</td>
<td>STATE 149</td>
<td>0 2 0 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 60</td>
<td>0 0 0 3 1 0</td>
<td>STATE 150</td>
<td>0 2 0 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 61</td>
<td>1 1 1 2 0 0 0</td>
<td>STATE 151</td>
<td>1 1 1 2 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 62</td>
<td>1 1 1 0 3 0 1</td>
<td>STATE 152</td>
<td>1 1 1 2 3 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 63</td>
<td>1 1 1 0 0 3 1</td>
<td>STATE 153</td>
<td>1 1 1 2 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 64</td>
<td>1 1 0 2 3 0 1</td>
<td>STATE 154</td>
<td>1 1 1 0 1 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 65</td>
<td>1 1 0 2 0 3 1</td>
<td>STATE 155</td>
<td>1 1 1 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 66</td>
<td>1 1 0 0 1 0 1</td>
<td>STATE 156</td>
<td>1 1 1 0 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 67</td>
<td>1 1 0 0 3 3 1</td>
<td>STATE 157</td>
<td>1 1 0 2 1 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 68</td>
<td>1 1 0 0 0 1 1</td>
<td>STATE 158</td>
<td>1 1 0 2 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 69</td>
<td>1 2 1 1 0 0 0</td>
<td>STATE 159</td>
<td>1 1 0 2 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 70</td>
<td>1 0 1 1 3 0 1</td>
<td>STATE 160</td>
<td>1 2 1 1 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 71</td>
<td>1 0 1 1 0 3 1</td>
<td>STATE 161</td>
<td>1 2 1 1 3 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 72</td>
<td>1 0 1 0 1 0 1</td>
<td>STATE 162</td>
<td>1 2 1 1 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 73</td>
<td>1 0 1 0 3 3 1</td>
<td>STATE 163</td>
<td>1 0 1 1 1 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 74</td>
<td>1 0 1 0 0 1 1</td>
<td>STATE 164</td>
<td>1 0 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 75</td>
<td>1 2 0 1 0 3 0</td>
<td>STATE 165</td>
<td>1 0 1 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 76</td>
<td>1 2 0 1 0 3 1</td>
<td>STATE 166</td>
<td>1 2 0 1 1 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 77</td>
<td>1 0 0 1 1 0 1</td>
<td>STATE 167</td>
<td>1 2 0 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 78</td>
<td>1 0 0 1 3 3 1</td>
<td>STATE 168</td>
<td>1 2 0 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 79</td>
<td>1 0 0 1 0 1 1</td>
<td>STATE 169</td>
<td>0 1 1 2 1 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 80</td>
<td>1 0 0 0 1 3 1</td>
<td>STATE 170</td>
<td>0 1 1 2 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 81</td>
<td>1 0 0 0 1 1 1</td>
<td>STATE 171</td>
<td>0 1 1 2 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 82</td>
<td>1 0 0 0 3 1 1</td>
<td>STATE 172</td>
<td>0 2 1 1 1 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 83</td>
<td>0 1 1 2 3 0 1</td>
<td>STATE 173</td>
<td>0 2 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 84</td>
<td>0 1 1 2 0 3 1</td>
<td>STATE 174</td>
<td>0 2 1 1 3 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 85</td>
<td>0 1 1 0 1 0 1</td>
<td>STATE 175</td>
<td>1 1 1 2 1 3 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATE 86</td>
<td>0 1 1 0 3 3 1</td>
<td>STATE 176</td>
<td>1 1 1 2 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table A.1 States in the Markov model of module 6008

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>State 2</td>
<td>6005</td>
<td>State 35</td>
<td>State 71</td>
<td>6003</td>
<td>State 70</td>
<td>State 121</td>
<td>0</td>
<td>State 106</td>
<td>State 146</td>
<td>6003</td>
</tr>
<tr>
<td>State 1</td>
<td>State 3</td>
<td>8</td>
<td>State 35</td>
<td>State 12</td>
<td>-5</td>
<td>State 70</td>
<td>State 122</td>
<td>11</td>
<td>State 106</td>
<td>State 59</td>
<td>-5</td>
</tr>
<tr>
<td>State 1</td>
<td>State 4</td>
<td>6003</td>
<td>State 35</td>
<td>State 78</td>
<td>6</td>
<td>State 71</td>
<td>State 49</td>
<td>-6005</td>
<td>State 106</td>
<td>State 57</td>
<td>-6</td>
</tr>
<tr>
<td>State 1</td>
<td>State 5</td>
<td>5</td>
<td>State 35</td>
<td>State 79</td>
<td>0</td>
<td>State 71</td>
<td>State 120</td>
<td>8</td>
<td>State 106</td>
<td>State 55</td>
<td>-11</td>
</tr>
<tr>
<td>State 1</td>
<td>State 6</td>
<td>6</td>
<td>State 36</td>
<td>State 23</td>
<td>-6005</td>
<td>State 71</td>
<td>State 35</td>
<td>-6003</td>
<td>State 107</td>
<td>State 132</td>
<td>6005</td>
</tr>
<tr>
<td>State 1</td>
<td>State 7</td>
<td>11</td>
<td>State 36</td>
<td>State 66</td>
<td>8</td>
<td>State 71</td>
<td>State 32</td>
<td>-5</td>
<td>State 107</td>
<td>State 150</td>
<td>8</td>
</tr>
<tr>
<td>State 2</td>
<td>State 1</td>
<td>-6005</td>
<td>State 36</td>
<td>State 72</td>
<td>6003</td>
<td>State 71</td>
<td>State 122</td>
<td>6</td>
<td>State 107</td>
<td>State 147</td>
<td>6003</td>
</tr>
<tr>
<td>State 2</td>
<td>State 8</td>
<td>8</td>
<td>State 36</td>
<td>State 77</td>
<td>5</td>
<td>State 71</td>
<td>State 123</td>
<td>0</td>
<td>State 107</td>
<td>State 60</td>
<td>-5</td>
</tr>
<tr>
<td>State 2</td>
<td>State 9</td>
<td>6003</td>
<td>State 36</td>
<td>State 2</td>
<td>-5</td>
<td>State 72</td>
<td>State 50</td>
<td>-6005</td>
<td>State 107</td>
<td>State 106</td>
<td>0</td>
</tr>
<tr>
<td>State 2</td>
<td>State 10</td>
<td>5</td>
<td>State 36</td>
<td>State 80</td>
<td>11</td>
<td>State 72</td>
<td>State 110</td>
<td>8</td>
<td>State 107</td>
<td>State 21</td>
<td>-11</td>
</tr>
<tr>
<td>State 2</td>
<td>State 11</td>
<td>6</td>
<td>State 37</td>
<td>State 24</td>
<td>-6005</td>
<td>State 72</td>
<td>State 36</td>
<td>-6003</td>
<td>State 108</td>
<td>State 83</td>
<td>-6005</td>
</tr>
<tr>
<td>State 2</td>
<td>State 12</td>
<td>11</td>
<td>State 37</td>
<td>State 67</td>
<td>8</td>
<td>State 72</td>
<td>State 121</td>
<td>5</td>
<td>State 108</td>
<td>State 70</td>
<td>-8</td>
</tr>
<tr>
<td>State 3</td>
<td>State 8</td>
<td>6005</td>
<td>State 37</td>
<td>State 73</td>
<td>6003</td>
<td>State 72</td>
<td>State 9</td>
<td>-6</td>
<td>State 108</td>
<td>State 64</td>
<td>-6003</td>
</tr>
<tr>
<td>State 3</td>
<td>State 1</td>
<td>-8</td>
<td>State 37</td>
<td>State 78</td>
<td>5</td>
<td>State 72</td>
<td>State 124</td>
<td>11</td>
<td>State 108</td>
<td>State 151</td>
<td>0</td>
</tr>
<tr>
<td>State 3</td>
<td>State 13</td>
<td>6003</td>
<td>State 37</td>
<td>State 81</td>
<td>0</td>
<td>State 73</td>
<td>State 51</td>
<td>-6005</td>
<td>State 108</td>
<td>State 152</td>
<td>11</td>
</tr>
<tr>
<td>State 3</td>
<td>State 14</td>
<td>5</td>
<td>State 37</td>
<td>State 82</td>
<td>0</td>
<td>State 73</td>
<td>State 111</td>
<td>8</td>
<td>State 109</td>
<td>State 84</td>
<td>-6005</td>
</tr>
<tr>
<td>State 3</td>
<td>State 15</td>
<td>6</td>
<td>State 38</td>
<td>State 25</td>
<td>-6005</td>
<td>State 73</td>
<td>State 37</td>
<td>-6003</td>
<td>State 109</td>
<td>State 71</td>
<td>-8</td>
</tr>
<tr>
<td>State 3</td>
<td>State 16</td>
<td>11</td>
<td>State 38</td>
<td>State 68</td>
<td>8</td>
<td>State 73</td>
<td>State 122</td>
<td>5</td>
<td>State 109</td>
<td>State 65</td>
<td>-6003</td>
</tr>
<tr>
<td>State 4</td>
<td>State 9</td>
<td>6005</td>
<td>State 38</td>
<td>State 74</td>
<td>6003</td>
<td>State 73</td>
<td>State 125</td>
<td>0</td>
<td>State 109</td>
<td>State 152</td>
<td>6</td>
</tr>
<tr>
<td>State 4</td>
<td>State 13</td>
<td>8</td>
<td>State 38</td>
<td>State 79</td>
<td>5</td>
<td>State 73</td>
<td>State 126</td>
<td>0</td>
<td>State 109</td>
<td>State 153</td>
<td>0</td>
</tr>
<tr>
<td>State 4</td>
<td>State 1</td>
<td>-6003</td>
<td>State 38</td>
<td>State 82</td>
<td>6</td>
<td>State 74</td>
<td>State 52</td>
<td>-6005</td>
<td>State 110</td>
<td>State 85</td>
<td>-6005</td>
</tr>
<tr>
<td>State 4</td>
<td>State 17</td>
<td>5</td>
<td>State 38</td>
<td>State 2</td>
<td>-11</td>
<td>State 74</td>
<td>State 112</td>
<td>8</td>
<td>State 110</td>
<td>State 72</td>
<td>-8</td>
</tr>
<tr>
<td>State 4</td>
<td>State 18</td>
<td>6</td>
<td>State 39</td>
<td>State 61</td>
<td>6005</td>
<td>State 74</td>
<td>State 38</td>
<td>-6003</td>
<td>State 110</td>
<td>State 66</td>
<td>-6003</td>
</tr>
<tr>
<td>State 4</td>
<td>State 19</td>
<td>11</td>
<td>State 39</td>
<td>State 17</td>
<td>-8</td>
<td>State 74</td>
<td>State 123</td>
<td>5</td>
<td>State 110</td>
<td>State 151</td>
<td>5</td>
</tr>
<tr>
<td>State 5</td>
<td>State 10</td>
<td>6005</td>
<td>State 39</td>
<td>State 14</td>
<td>-6003</td>
<td>State 74</td>
<td>State 126</td>
<td>6</td>
<td>State 110</td>
<td>State 26</td>
<td>-6</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>------</td>
<td>----------</td>
<td>---------</td>
<td>-------</td>
<td>----------</td>
<td>-----------</td>
<td>---</td>
<td>-----------</td>
<td>---------</td>
<td>----</td>
</tr>
<tr>
<td>State 5</td>
<td>State 20</td>
<td>8</td>
<td>State 39</td>
<td>State 83</td>
<td>6</td>
<td>State 74</td>
<td>State 9</td>
<td>-11</td>
<td>State 110</td>
<td>State 154</td>
<td>11</td>
</tr>
<tr>
<td>State 5</td>
<td>State 17</td>
<td>6003</td>
<td>State 39</td>
<td>State 84</td>
<td>11</td>
<td>State 75</td>
<td>State 53</td>
<td>-6005</td>
<td>State 111</td>
<td>State 86</td>
<td>-6005</td>
</tr>
<tr>
<td>State 5</td>
<td>State 1</td>
<td>-5</td>
<td>State 40</td>
<td>State 62</td>
<td>6005</td>
<td>State 75</td>
<td>State 119</td>
<td>6003</td>
<td>State 111</td>
<td>State 73</td>
<td>-8</td>
</tr>
<tr>
<td>State 5</td>
<td>State 21</td>
<td>6</td>
<td>State 40</td>
<td>State 18</td>
<td>-8</td>
<td>State 75</td>
<td>State 28</td>
<td>-5</td>
<td>State 111</td>
<td>State 67</td>
<td>-6003</td>
</tr>
<tr>
<td>State 5</td>
<td>State 22</td>
<td>11</td>
<td>State 40</td>
<td>State 15</td>
<td>-6003</td>
<td>State 75</td>
<td>State 127</td>
<td>0</td>
<td>State 111</td>
<td>State 152</td>
<td>5</td>
</tr>
<tr>
<td>State 6</td>
<td>State 11</td>
<td>6005</td>
<td>State 40</td>
<td>State 83</td>
<td>5</td>
<td>State 75</td>
<td>State 128</td>
<td>11</td>
<td>State 111</td>
<td>State 155</td>
<td>0</td>
</tr>
<tr>
<td>State 6</td>
<td>State 18</td>
<td>6003</td>
<td>State 40</td>
<td>State 86</td>
<td>11</td>
<td>State 76</td>
<td>State 120</td>
<td>6003</td>
<td>State 112</td>
<td>State 87</td>
<td>-6005</td>
</tr>
<tr>
<td>State 6</td>
<td>State 21</td>
<td>5</td>
<td>State 41</td>
<td>State 63</td>
<td>6005</td>
<td>State 76</td>
<td>State 29</td>
<td>-5</td>
<td>State 112</td>
<td>State 74</td>
<td>-8</td>
</tr>
<tr>
<td>State 6</td>
<td>State 23</td>
<td>0</td>
<td>State 41</td>
<td>State 19</td>
<td>-8</td>
<td>State 76</td>
<td>State 128</td>
<td>6</td>
<td>State 112</td>
<td>State 68</td>
<td>-6003</td>
</tr>
<tr>
<td>State 6</td>
<td>State 24</td>
<td>11</td>
<td>State 41</td>
<td>State 16</td>
<td>-6003</td>
<td>State 76</td>
<td>State 129</td>
<td>0</td>
<td>State 112</td>
<td>State 153</td>
<td>5</td>
</tr>
<tr>
<td>State 7</td>
<td>State 12</td>
<td>6005</td>
<td>State 41</td>
<td>State 84</td>
<td>5</td>
<td>State 77</td>
<td>State 55</td>
<td>-6005</td>
<td>State 112</td>
<td>State 156</td>
<td>6</td>
</tr>
<tr>
<td>State 7</td>
<td>State 16</td>
<td>8</td>
<td>State 41</td>
<td>State 86</td>
<td>6</td>
<td>State 77</td>
<td>State 127</td>
<td>8</td>
<td>State 112</td>
<td>State 26</td>
<td>-11</td>
</tr>
<tr>
<td>State 7</td>
<td>State 19</td>
<td>6003</td>
<td>State 41</td>
<td>State 87</td>
<td>0</td>
<td>State 77</td>
<td>State 121</td>
<td>6003</td>
<td>State 113</td>
<td>State 88</td>
<td>-6005</td>
</tr>
<tr>
<td>State 7</td>
<td>State 22</td>
<td>5</td>
<td>State 42</td>
<td>State 64</td>
<td>6005</td>
<td>State 77</td>
<td>State 36</td>
<td>-5</td>
<td>State 113</td>
<td>State 77</td>
<td>-8</td>
</tr>
<tr>
<td>State 7</td>
<td>State 24</td>
<td>6</td>
<td>State 42</td>
<td>State 21</td>
<td>-8</td>
<td>State 77</td>
<td>State 10</td>
<td>-6</td>
<td>State 113</td>
<td>State 151</td>
<td>6003</td>
</tr>
<tr>
<td>State 7</td>
<td>State 25</td>
<td>0</td>
<td>State 42</td>
<td>State 83</td>
<td>6003</td>
<td>State 77</td>
<td>State 130</td>
<td>11</td>
<td>State 113</td>
<td>State 27</td>
<td>-6</td>
</tr>
<tr>
<td>State 8</td>
<td>State 3</td>
<td>-6005</td>
<td>State 42</td>
<td>State 88</td>
<td>0</td>
<td>State 78</td>
<td>State 56</td>
<td>-6005</td>
<td>State 113</td>
<td>State 157</td>
<td>11</td>
</tr>
<tr>
<td>State 8</td>
<td>State 2</td>
<td>8</td>
<td>State 42</td>
<td>State 89</td>
<td>11</td>
<td>State 78</td>
<td>State 128</td>
<td>8</td>
<td>State 114</td>
<td>State 89</td>
<td>-6005</td>
</tr>
<tr>
<td>State 8</td>
<td>State 26</td>
<td>6003</td>
<td>State 43</td>
<td>State 65</td>
<td>6005</td>
<td>State 78</td>
<td>State 122</td>
<td>6003</td>
<td>State 114</td>
<td>State 78</td>
<td>-8</td>
</tr>
<tr>
<td>State 8</td>
<td>State 27</td>
<td>5</td>
<td>State 43</td>
<td>State 22</td>
<td>-8</td>
<td>State 78</td>
<td>State 37</td>
<td>-5</td>
<td>State 114</td>
<td>State 152</td>
<td>6003</td>
</tr>
<tr>
<td>State 8</td>
<td>State 28</td>
<td>6</td>
<td>State 43</td>
<td>State 84</td>
<td>6003</td>
<td>State 78</td>
<td>State 131</td>
<td>0</td>
<td>State 114</td>
<td>State 158</td>
<td>0</td>
</tr>
<tr>
<td>State 8</td>
<td>State 29</td>
<td>11</td>
<td>State 43</td>
<td>State 89</td>
<td>6</td>
<td>State 78</td>
<td>State 132</td>
<td>0</td>
<td>State 114</td>
<td>State 159</td>
<td>0</td>
</tr>
<tr>
<td>State 9</td>
<td>State 4</td>
<td>-6005</td>
<td>State 43</td>
<td>State 90</td>
<td>0</td>
<td>State 79</td>
<td>State 57</td>
<td>-6005</td>
<td>State 115</td>
<td>State 90</td>
<td>-6005</td>
</tr>
<tr>
<td>State 9</td>
<td>State 26</td>
<td>8</td>
<td>State 44</td>
<td>State 66</td>
<td>6005</td>
<td>State 79</td>
<td>State 129</td>
<td>8</td>
<td>State 115</td>
<td>State 79</td>
<td>-8</td>
</tr>
<tr>
<td>State 9</td>
<td>State 2 -6003</td>
<td>State 44</td>
<td>State 23</td>
<td>-8</td>
<td>State 79</td>
<td>State 123</td>
<td>6003</td>
<td>State 115</td>
<td>State 153</td>
<td>6003</td>
<td>State 152</td>
</tr>
<tr>
<td>State 9</td>
<td>State 30</td>
<td>5</td>
<td>State 44</td>
<td>State 85</td>
<td>6003</td>
<td>State 79</td>
<td>State 38</td>
<td>-5</td>
<td>State 115</td>
<td>State 159</td>
<td>6</td>
</tr>
<tr>
<td>State 9</td>
<td>State 31</td>
<td>6</td>
<td>State 44</td>
<td>State 88</td>
<td>5</td>
<td>State 79</td>
<td>State 132</td>
<td>6</td>
<td>State 115</td>
<td>State 27</td>
<td>-11</td>
</tr>
<tr>
<td>State 9</td>
<td>State 32</td>
<td>11</td>
<td>State 44</td>
<td>State 3</td>
<td>-6</td>
<td>State 79</td>
<td>State 10</td>
<td>-11</td>
<td>State 116</td>
<td>State 91</td>
<td>-6005</td>
</tr>
<tr>
<td>State 10</td>
<td>State 5</td>
<td>-6005</td>
<td>State 44</td>
<td>State 91</td>
<td>11</td>
<td>State 80</td>
<td>State 58</td>
<td>-6005</td>
<td>State 116</td>
<td>State 80</td>
<td>-8</td>
</tr>
<tr>
<td>State 10</td>
<td>State 33</td>
<td>8</td>
<td>State 45</td>
<td>State 67</td>
<td>6005</td>
<td>State 80</td>
<td>State 116</td>
<td>8</td>
<td>State 116</td>
<td>State 154</td>
<td>6003</td>
</tr>
<tr>
<td>State 10</td>
<td>State 30</td>
<td>6003</td>
<td>State 45</td>
<td>State 24</td>
<td>-8</td>
<td>State 80</td>
<td>State 124</td>
<td>6003</td>
<td>State 116</td>
<td>State 157</td>
<td>5</td>
</tr>
<tr>
<td>State 10</td>
<td>State 2</td>
<td>-5</td>
<td>State 45</td>
<td>State 86</td>
<td>6003</td>
<td>State 80</td>
<td>State 130</td>
<td>5</td>
<td>State 116</td>
<td>State 29</td>
<td>-6</td>
</tr>
<tr>
<td>State 10</td>
<td>State 34</td>
<td>6</td>
<td>State 45</td>
<td>State 89</td>
<td>5</td>
<td>State 80</td>
<td>State 12</td>
<td>-6</td>
<td>State 116</td>
<td>State 117</td>
<td>0</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>---</td>
<td>----------</td>
<td>----------</td>
<td>---</td>
<td>----------</td>
<td>----------</td>
<td>----</td>
<td>----------</td>
<td>----------</td>
<td>---</td>
</tr>
<tr>
<td>State 10</td>
<td>State 35</td>
<td>11</td>
<td>State 45</td>
<td>State 92</td>
<td>0</td>
<td>State 80</td>
<td>State 81</td>
<td>0</td>
<td>State 117</td>
<td>State 92</td>
<td>-6005</td>
</tr>
<tr>
<td>State 11</td>
<td>State 6</td>
<td>-6005</td>
<td>State 45</td>
<td>State 93</td>
<td>0</td>
<td>State 81</td>
<td>State 59</td>
<td>-6005</td>
<td>State 117</td>
<td>State 81</td>
<td>-8</td>
</tr>
<tr>
<td>State 11</td>
<td>State 28</td>
<td>8</td>
<td>State 46</td>
<td>State 68</td>
<td>6005</td>
<td>State 81</td>
<td>State 117</td>
<td>8</td>
<td>State 117</td>
<td>State 155</td>
<td>6003</td>
</tr>
<tr>
<td>State 11</td>
<td>State 31</td>
<td>6003</td>
<td>State 46</td>
<td>State 25</td>
<td>-8</td>
<td>State 81</td>
<td>State 125</td>
<td>6003</td>
<td>State 117</td>
<td>State 158</td>
<td>5</td>
</tr>
<tr>
<td>State 11</td>
<td>State 34</td>
<td>5</td>
<td>State 46</td>
<td>State 87</td>
<td>6003</td>
<td>State 81</td>
<td>State 131</td>
<td>5</td>
<td>State 117</td>
<td>State 68</td>
<td>-6</td>
</tr>
<tr>
<td>State 11</td>
<td>State 36</td>
<td>0</td>
<td>State 46</td>
<td>State 90</td>
<td>5</td>
<td>State 81</td>
<td>State 38</td>
<td>-6</td>
<td>State 117</td>
<td>State 66</td>
<td>-11</td>
</tr>
<tr>
<td>State 11</td>
<td>State 37</td>
<td>11</td>
<td>State 46</td>
<td>State 93</td>
<td>6</td>
<td>State 81</td>
<td>State 36</td>
<td>-11</td>
<td>State 118</td>
<td>State 93</td>
<td>-6005</td>
</tr>
<tr>
<td>State 12</td>
<td>State 7</td>
<td>-6005</td>
<td>State 46</td>
<td>State 3</td>
<td>-11</td>
<td>State 82</td>
<td>State 60</td>
<td>-6005</td>
<td>State 118</td>
<td>State 82</td>
<td>-8</td>
</tr>
<tr>
<td>State 12</td>
<td>State 29</td>
<td>8</td>
<td>State 47</td>
<td>State 69</td>
<td>6005</td>
<td>State 82</td>
<td>State 118</td>
<td>8</td>
<td>State 118</td>
<td>State 156</td>
<td>6003</td>
</tr>
<tr>
<td>State 12</td>
<td>State 32</td>
<td>6003</td>
<td>State 47</td>
<td>State 20</td>
<td>-6003</td>
<td>State 82</td>
<td>State 126</td>
<td>6003</td>
<td>State 118</td>
<td>State 159</td>
<td>5</td>
</tr>
<tr>
<td>State 12</td>
<td>State 35</td>
<td>5</td>
<td>State 47</td>
<td>State 13</td>
<td>-5</td>
<td>State 82</td>
<td>State 132</td>
<td>5</td>
<td>State 118</td>
<td>State 117</td>
<td>0</td>
</tr>
<tr>
<td>State 12</td>
<td>State 37</td>
<td>6</td>
<td>State 47</td>
<td>State 94</td>
<td>6</td>
<td>State 82</td>
<td>State 81</td>
<td>0</td>
<td>State 118</td>
<td>State 28</td>
<td>-11</td>
</tr>
<tr>
<td>State 12</td>
<td>State 38</td>
<td>0</td>
<td>State 47</td>
<td>State 95</td>
<td>11</td>
<td>State 82</td>
<td>State 11</td>
<td>-11</td>
<td>State 119</td>
<td>State 94</td>
<td>-6005</td>
</tr>
<tr>
<td>State 13</td>
<td>State 26</td>
<td>6005</td>
<td>State 48</td>
<td>State 70</td>
<td>6005</td>
<td>State 83</td>
<td>State 108</td>
<td>6005</td>
<td>State 119</td>
<td>State 75</td>
<td>-6003</td>
</tr>
<tr>
<td>State 13</td>
<td>State 4</td>
<td>-8</td>
<td>State 48</td>
<td>State 94</td>
<td>8</td>
<td>State 83</td>
<td>State 48</td>
<td>-8</td>
<td>State 119</td>
<td>State 62</td>
<td>-5</td>
</tr>
<tr>
<td>State 13</td>
<td>State 3</td>
<td>-6003</td>
<td>State 48</td>
<td>State 21</td>
<td>-6003</td>
<td>State 83</td>
<td>State 42</td>
<td>-6003</td>
<td>State 119</td>
<td>State 160</td>
<td>0</td>
</tr>
<tr>
<td>State 13</td>
<td>State 39</td>
<td>5</td>
<td>State 48</td>
<td>State 18</td>
<td>-5</td>
<td>State 83</td>
<td>State 133</td>
<td>0</td>
<td>State 119</td>
<td>State 161</td>
<td>11</td>
</tr>
<tr>
<td>State 13</td>
<td>State 40</td>
<td>6</td>
<td>State 48</td>
<td>State 96</td>
<td>0</td>
<td>State 83</td>
<td>State 134</td>
<td>11</td>
<td>State 120</td>
<td>State 95</td>
<td>-6005</td>
</tr>
<tr>
<td>State 13</td>
<td>State 41</td>
<td>11</td>
<td>State 48</td>
<td>State 97</td>
<td>11</td>
<td>State 84</td>
<td>State 109</td>
<td>6005</td>
<td>State 120</td>
<td>State 76</td>
<td>-6003</td>
</tr>
<tr>
<td>State 14</td>
<td>State 27</td>
<td>6005</td>
<td>State 49</td>
<td>State 71</td>
<td>6005</td>
<td>State 84</td>
<td>State 49</td>
<td>-8</td>
<td>State 120</td>
<td>State 63</td>
<td>-5</td>
</tr>
<tr>
<td>State 14</td>
<td>State 5</td>
<td>-8</td>
<td>State 49</td>
<td>State 95</td>
<td>8</td>
<td>State 84</td>
<td>State 43</td>
<td>-6003</td>
<td>State 120</td>
<td>State 161</td>
<td>6</td>
</tr>
<tr>
<td>State 14</td>
<td>State 39</td>
<td>6003</td>
<td>State 49</td>
<td>State 22</td>
<td>-6003</td>
<td>State 84</td>
<td>State 134</td>
<td>6</td>
<td>State 120</td>
<td>State 162</td>
<td>0</td>
</tr>
<tr>
<td>State 14</td>
<td>State 42</td>
<td>6</td>
<td>State 49</td>
<td>State 19</td>
<td>-5</td>
<td>State 84</td>
<td>State 135</td>
<td>0</td>
<td>State 121</td>
<td>State 96</td>
<td>-6005</td>
</tr>
<tr>
<td>State 14</td>
<td>State 43</td>
<td>11</td>
<td>State 49</td>
<td>State 97</td>
<td>6</td>
<td>State 85</td>
<td>State 110</td>
<td>6005</td>
<td>State 121</td>
<td>State 160</td>
<td>8</td>
</tr>
<tr>
<td>State 15</td>
<td>State 28</td>
<td>6005</td>
<td>State 49</td>
<td>State 98</td>
<td>0</td>
<td>State 85</td>
<td>State 50</td>
<td>-8</td>
<td>State 121</td>
<td>State 77</td>
<td>-6003</td>
</tr>
<tr>
<td>State 15</td>
<td>State 6</td>
<td>-8</td>
<td>State 50</td>
<td>State 72</td>
<td>6005</td>
<td>State 85</td>
<td>State 44</td>
<td>-6003</td>
<td>State 121</td>
<td>State 72</td>
<td>-5</td>
</tr>
<tr>
<td>State 15</td>
<td>State 40</td>
<td>6003</td>
<td>State 50</td>
<td>State 85</td>
<td>8</td>
<td>State 85</td>
<td>State 133</td>
<td>5</td>
<td>State 121</td>
<td>State 30</td>
<td>-6</td>
</tr>
<tr>
<td>State 15</td>
<td>State 42</td>
<td>5</td>
<td>State 50</td>
<td>State 23</td>
<td>-6003</td>
<td>State 85</td>
<td>State 13</td>
<td>-6</td>
<td>State 121</td>
<td>State 163</td>
<td>11</td>
</tr>
<tr>
<td>State 15</td>
<td>State 44</td>
<td>0</td>
<td>State 50</td>
<td>State 96</td>
<td>5</td>
<td>State 85</td>
<td>State 136</td>
<td>11</td>
<td>State 122</td>
<td>State 97</td>
<td>-6005</td>
</tr>
<tr>
<td>State 15</td>
<td>State 45</td>
<td>11</td>
<td>State 50</td>
<td>State 4</td>
<td>-6</td>
<td>State 86</td>
<td>State 111</td>
<td>6005</td>
<td>State 122</td>
<td>State 161</td>
<td>8</td>
</tr>
<tr>
<td>State 16</td>
<td>State 29</td>
<td>6005</td>
<td>State 50</td>
<td>State 99</td>
<td>11</td>
<td>State 86</td>
<td>State 51</td>
<td>-8</td>
<td>State 122</td>
<td>State 78</td>
<td>-6003</td>
</tr>
<tr>
<td>State 16</td>
<td>State 7</td>
<td>-8</td>
<td>State 51</td>
<td>State 73</td>
<td>6005</td>
<td>State 86</td>
<td>State 45</td>
<td>-6003</td>
<td>State 122</td>
<td>State 73</td>
<td>-5</td>
</tr>
<tr>
<td>State 16</td>
<td>State 41</td>
<td>6003</td>
<td>State 51</td>
<td>State 86</td>
<td>8</td>
<td>State 86</td>
<td>State 134</td>
<td>5</td>
<td>State 122</td>
<td>State 164</td>
<td>0</td>
</tr>
<tr>
<td>State 16</td>
<td>State 43</td>
<td>5</td>
<td>State 51</td>
<td>State 24</td>
<td>-6003</td>
<td>State 86</td>
<td>State 137</td>
<td>0</td>
<td>State 122</td>
<td>State 165</td>
<td>0</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---</td>
<td>----------</td>
<td>---------</td>
<td>-------</td>
<td>---------</td>
<td>---------</td>
<td>---</td>
<td>----------</td>
<td>---------</td>
<td>---</td>
</tr>
<tr>
<td>State 16</td>
<td>State 45</td>
<td>6</td>
<td>State 51</td>
<td>State 97</td>
<td>5</td>
<td>State 86</td>
<td>State 138</td>
<td>0</td>
<td>State 123</td>
<td>State 98</td>
<td>-6005</td>
</tr>
<tr>
<td>State 16</td>
<td>State 46</td>
<td>0</td>
<td>State 51</td>
<td>State 100</td>
<td>0</td>
<td>State 87</td>
<td>State 112</td>
<td>6005</td>
<td>State 123</td>
<td>State 162</td>
<td>8</td>
</tr>
<tr>
<td>State 17</td>
<td>State 30</td>
<td>6005</td>
<td>State 51</td>
<td>State 101</td>
<td>0</td>
<td>State 87</td>
<td>State 52</td>
<td>-8</td>
<td>State 123</td>
<td>State 79</td>
<td>-6003</td>
</tr>
<tr>
<td>State 17</td>
<td>State 47</td>
<td>8</td>
<td>State 51</td>
<td>State 74</td>
<td>6005</td>
<td>State 87</td>
<td>State 46</td>
<td>-6003</td>
<td>State 123</td>
<td>State 74</td>
<td>-5</td>
</tr>
<tr>
<td>State 17</td>
<td>State 5</td>
<td>-6003</td>
<td>State 52</td>
<td>State 87</td>
<td>8</td>
<td>State 87</td>
<td>State 135</td>
<td>5</td>
<td>State 123</td>
<td>State 165</td>
<td>6</td>
</tr>
<tr>
<td>State 17</td>
<td>State 4</td>
<td>-5</td>
<td>State 52</td>
<td>State 25</td>
<td>-6003</td>
<td>State 87</td>
<td>State 138</td>
<td>6</td>
<td>State 123</td>
<td>State 30</td>
<td>-11</td>
</tr>
<tr>
<td>State 17</td>
<td>State 48</td>
<td>6</td>
<td>State 52</td>
<td>State 98</td>
<td>5</td>
<td>State 87</td>
<td>State 139</td>
<td>-11</td>
<td>State 124</td>
<td>State 99</td>
<td>-6005</td>
</tr>
<tr>
<td>State 17</td>
<td>State 49</td>
<td>11</td>
<td>State 52</td>
<td>State 101</td>
<td>6</td>
<td>State 88</td>
<td>State 113</td>
<td>6005</td>
<td>State 124</td>
<td>State 154</td>
<td>8</td>
</tr>
<tr>
<td>State 18</td>
<td>State 31</td>
<td>6005</td>
<td>State 52</td>
<td>State 4</td>
<td>-11</td>
<td>State 88</td>
<td>State 55</td>
<td>-8</td>
<td>State 124</td>
<td>State 80</td>
<td>-6003</td>
</tr>
<tr>
<td>State 18</td>
<td>State 40</td>
<td>8</td>
<td>State 53</td>
<td>State 75</td>
<td>6005</td>
<td>State 88</td>
<td>State 133</td>
<td>6003</td>
<td>State 124</td>
<td>State 163</td>
<td>5</td>
</tr>
<tr>
<td>State 18</td>
<td>State 6</td>
<td>-6003</td>
<td>State 53</td>
<td>State 94</td>
<td>6003</td>
<td>State 88</td>
<td>State 14</td>
<td>-6</td>
<td>State 124</td>
<td>State 32</td>
<td>-6</td>
</tr>
<tr>
<td>State 18</td>
<td>State 48</td>
<td>5</td>
<td>State 53</td>
<td>State 15</td>
<td>-5</td>
<td>State 88</td>
<td>State 139</td>
<td>11</td>
<td>State 124</td>
<td>State 125</td>
<td>0</td>
</tr>
<tr>
<td>State 18</td>
<td>State 50</td>
<td>0</td>
<td>State 53</td>
<td>State 102</td>
<td>0</td>
<td>State 89</td>
<td>State 114</td>
<td>6005</td>
<td>State 125</td>
<td>State 100</td>
<td>-6005</td>
</tr>
<tr>
<td>State 18</td>
<td>State 51</td>
<td>11</td>
<td>State 53</td>
<td>State 103</td>
<td>11</td>
<td>State 89</td>
<td>State 56</td>
<td>-8</td>
<td>State 125</td>
<td>State 155</td>
<td>8</td>
</tr>
<tr>
<td>State 19</td>
<td>State 32</td>
<td>6005</td>
<td>State 54</td>
<td>State 76</td>
<td>6005</td>
<td>State 89</td>
<td>State 134</td>
<td>6003</td>
<td>State 125</td>
<td>State 81</td>
<td>-6003</td>
</tr>
<tr>
<td>State 19</td>
<td>State 41</td>
<td>8</td>
<td>State 54</td>
<td>State 95</td>
<td>6003</td>
<td>State 89</td>
<td>State 140</td>
<td>0</td>
<td>State 125</td>
<td>State 164</td>
<td>5</td>
</tr>
<tr>
<td>State 19</td>
<td>State 7</td>
<td>-6003</td>
<td>State 54</td>
<td>State 54</td>
<td>State 16</td>
<td>-5</td>
<td>State 89</td>
<td>State 141</td>
<td>0</td>
<td>State 125</td>
<td>State 74</td>
</tr>
<tr>
<td>State 19</td>
<td>State 49</td>
<td>5</td>
<td>State 54</td>
<td>State 103</td>
<td>6</td>
<td>State 90</td>
<td>State 115</td>
<td>6005</td>
<td>State 125</td>
<td>State 72</td>
<td>-11</td>
</tr>
<tr>
<td>State 19</td>
<td>State 51</td>
<td>6</td>
<td>State 54</td>
<td>State 104</td>
<td>0</td>
<td>State 90</td>
<td>State 57</td>
<td>-8</td>
<td>State 126</td>
<td>State 101</td>
<td>-6005</td>
</tr>
<tr>
<td>State 19</td>
<td>State 52</td>
<td>0</td>
<td>State 55</td>
<td>State 77</td>
<td>6005</td>
<td>State 90</td>
<td>State 135</td>
<td>6003</td>
<td>State 126</td>
<td>State 156</td>
<td>8</td>
</tr>
<tr>
<td>State 20</td>
<td>State 33</td>
<td>6005</td>
<td>State 55</td>
<td>State 102</td>
<td>8</td>
<td>State 90</td>
<td>State 141</td>
<td>6</td>
<td>State 126</td>
<td>State 82</td>
<td>-6003</td>
</tr>
<tr>
<td>State 20</td>
<td>State 47</td>
<td>6003</td>
<td>State 55</td>
<td>State 96</td>
<td>6003</td>
<td>State 90</td>
<td>State 134</td>
<td>-11</td>
<td>State 126</td>
<td>State 165</td>
<td>5</td>
</tr>
<tr>
<td>State 20</td>
<td>State 3</td>
<td>-5</td>
<td>State 55</td>
<td>State 23</td>
<td>-5</td>
<td>State 91</td>
<td>State 116</td>
<td>6005</td>
<td>State 126</td>
<td>State 125</td>
<td>0</td>
</tr>
<tr>
<td>State 20</td>
<td>State 53</td>
<td>6</td>
<td>State 55</td>
<td>State 5</td>
<td>-6</td>
<td>State 91</td>
<td>State 58</td>
<td>-8</td>
<td>State 126</td>
<td>State 31</td>
<td>-11</td>
</tr>
<tr>
<td>State 20</td>
<td>State 54</td>
<td>11</td>
<td>State 55</td>
<td>State 105</td>
<td>11</td>
<td>State 91</td>
<td>State 136</td>
<td>6003</td>
<td>State 127</td>
<td>State 102</td>
<td>-6005</td>
</tr>
<tr>
<td>State 21</td>
<td>State 34</td>
<td>6005</td>
<td>State 56</td>
<td>State 78</td>
<td>6005</td>
<td>State 91</td>
<td>State 139</td>
<td>5</td>
<td>State 127</td>
<td>State 160</td>
<td>6003</td>
</tr>
<tr>
<td>State 21</td>
<td>State 53</td>
<td>8</td>
<td>State 56</td>
<td>State 103</td>
<td>8</td>
<td>State 91</td>
<td>State 16</td>
<td>-6</td>
<td>State 127</td>
<td>State 66</td>
<td>-5</td>
</tr>
<tr>
<td>State 21</td>
<td>State 48</td>
<td>6003</td>
<td>State 56</td>
<td>State 97</td>
<td>6003</td>
<td>State 91</td>
<td>State 92</td>
<td>0</td>
<td>State 127</td>
<td>State 33</td>
<td>-6</td>
</tr>
<tr>
<td>State 21</td>
<td>State 6</td>
<td>-5</td>
<td>State 56</td>
<td>State 24</td>
<td>-5</td>
<td>State 92</td>
<td>State 117</td>
<td>6005</td>
<td>State 127</td>
<td>State 166</td>
<td>11</td>
</tr>
<tr>
<td>State 21</td>
<td>State 55</td>
<td>0</td>
<td>State 56</td>
<td>State 106</td>
<td>0</td>
<td>State 92</td>
<td>State 59</td>
<td>-8</td>
<td>State 128</td>
<td>State 103</td>
<td>-6005</td>
</tr>
<tr>
<td>State 21</td>
<td>State 56</td>
<td>11</td>
<td>State 56</td>
<td>State 107</td>
<td>0</td>
<td>State 92</td>
<td>State 137</td>
<td>6003</td>
<td>State 128</td>
<td>State 161</td>
<td>6003</td>
</tr>
<tr>
<td>State 22</td>
<td>State 35</td>
<td>6005</td>
<td>State 57</td>
<td>State 79</td>
<td>6005</td>
<td>State 92</td>
<td>State 140</td>
<td>5</td>
<td>State 128</td>
<td>State 67</td>
<td>-5</td>
</tr>
<tr>
<td>State 22</td>
<td>State 54</td>
<td>8</td>
<td>State 57</td>
<td>State 104</td>
<td>8</td>
<td>State 92</td>
<td>State 46</td>
<td>-6</td>
<td>State 128</td>
<td>State 167</td>
<td>0</td>
</tr>
<tr>
<td>State 22</td>
<td>State 49</td>
<td>6003</td>
<td>State 57</td>
<td>State 98</td>
<td>6003</td>
<td>State 92</td>
<td>State 44</td>
<td>-11</td>
<td>State 128</td>
<td>State 168</td>
<td>0</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>------</td>
<td>---------</td>
<td>---------</td>
<td>------</td>
<td>---------</td>
<td>---------</td>
<td>------</td>
<td>------------</td>
<td>------------</td>
<td>---</td>
</tr>
<tr>
<td>State 22</td>
<td>State 7</td>
<td>-5</td>
<td>State 57</td>
<td>State 25</td>
<td>-5</td>
<td>State 93</td>
<td>State 118</td>
<td>6005</td>
<td>State 129</td>
<td>State 104</td>
<td>-6005</td>
</tr>
<tr>
<td>State 22</td>
<td>State 56</td>
<td>6</td>
<td>State 57</td>
<td>State 107</td>
<td>6</td>
<td>State 93</td>
<td>State 60</td>
<td>-8</td>
<td>State 129</td>
<td>State 162</td>
<td>6003</td>
</tr>
<tr>
<td>State 22</td>
<td>State 57</td>
<td>0</td>
<td>State 57</td>
<td>State 5</td>
<td>-11</td>
<td>State 93</td>
<td>State 138</td>
<td>6003</td>
<td>State 129</td>
<td>State 68</td>
<td>-5</td>
</tr>
<tr>
<td>State 23</td>
<td>State 36</td>
<td>6005</td>
<td>State 58</td>
<td>State 80</td>
<td>6005</td>
<td>State 93</td>
<td>State 141</td>
<td>5</td>
<td>State 129</td>
<td>State 168</td>
<td>6</td>
</tr>
<tr>
<td>State 23</td>
<td>State 44</td>
<td>8</td>
<td>State 58</td>
<td>State 91</td>
<td>8</td>
<td>State 93</td>
<td>State 92</td>
<td>0</td>
<td>State 129</td>
<td>State 33</td>
<td>-11</td>
</tr>
<tr>
<td>State 23</td>
<td>State 50</td>
<td>6003</td>
<td>State 58</td>
<td>State 99</td>
<td>6003</td>
<td>State 93</td>
<td>State 15</td>
<td>-11</td>
<td>State 130</td>
<td>State 105</td>
<td>-6005</td>
</tr>
<tr>
<td>State 23</td>
<td>State 55</td>
<td>5</td>
<td>State 58</td>
<td>State 105</td>
<td>5</td>
<td>State 94</td>
<td>State 119</td>
<td>6005</td>
<td>State 130</td>
<td>State 166</td>
<td>8</td>
</tr>
<tr>
<td>State 23</td>
<td>State 1</td>
<td>-6</td>
<td>State 58</td>
<td>State 7</td>
<td>-6</td>
<td>State 94</td>
<td>State 53</td>
<td>-6003</td>
<td>State 130</td>
<td>State 163</td>
<td>6003</td>
</tr>
<tr>
<td>State 23</td>
<td>State 58</td>
<td>11</td>
<td>State 58</td>
<td>State 59</td>
<td>0</td>
<td>State 94</td>
<td>State 40</td>
<td>-5</td>
<td>State 130</td>
<td>State 80</td>
<td>-5</td>
</tr>
<tr>
<td>State 24</td>
<td>State 37</td>
<td>6005</td>
<td>State 59</td>
<td>State 81</td>
<td>6005</td>
<td>State 94</td>
<td>State 142</td>
<td>0</td>
<td>State 130</td>
<td>State 35</td>
<td>-6</td>
</tr>
<tr>
<td>State 24</td>
<td>State 45</td>
<td>8</td>
<td>State 59</td>
<td>State 92</td>
<td>8</td>
<td>State 94</td>
<td>State 143</td>
<td>11</td>
<td>State 130</td>
<td>State 131</td>
<td>0</td>
</tr>
<tr>
<td>State 24</td>
<td>State 51</td>
<td>6003</td>
<td>State 59</td>
<td>State 100</td>
<td>6003</td>
<td>State 95</td>
<td>State 120</td>
<td>6005</td>
<td>State 131</td>
<td>State 106</td>
<td>-6005</td>
</tr>
<tr>
<td>State 24</td>
<td>State 56</td>
<td>5</td>
<td>State 59</td>
<td>State 106</td>
<td>5</td>
<td>State 95</td>
<td>State 54</td>
<td>-6003</td>
<td>State 131</td>
<td>State 167</td>
<td>8</td>
</tr>
<tr>
<td>State 24</td>
<td>State 59</td>
<td>0</td>
<td>State 59</td>
<td>State 25</td>
<td>-6</td>
<td>State 95</td>
<td>State 41</td>
<td>-5</td>
<td>State 131</td>
<td>State 164</td>
<td>6003</td>
</tr>
<tr>
<td>State 24</td>
<td>State 60</td>
<td>0</td>
<td>State 59</td>
<td>State 23</td>
<td>-11</td>
<td>State 95</td>
<td>State 143</td>
<td>6</td>
<td>State 131</td>
<td>State 81</td>
<td>-5</td>
</tr>
<tr>
<td>State 24</td>
<td>State 38</td>
<td>6005</td>
<td>State 60</td>
<td>State 82</td>
<td>6005</td>
<td>State 95</td>
<td>State 144</td>
<td>0</td>
<td>State 131</td>
<td>State 79</td>
<td>-6</td>
</tr>
<tr>
<td>State 25</td>
<td>State 46</td>
<td>8</td>
<td>State 60</td>
<td>State 93</td>
<td>8</td>
<td>State 96</td>
<td>State 121</td>
<td>6005</td>
<td>State 131</td>
<td>State 77</td>
<td>-11</td>
</tr>
<tr>
<td>State 25</td>
<td>State 52</td>
<td>6003</td>
<td>State 60</td>
<td>State 101</td>
<td>6003</td>
<td>State 96</td>
<td>State 142</td>
<td>8</td>
<td>State 132</td>
<td>State 107</td>
<td>-6005</td>
</tr>
<tr>
<td>State 25</td>
<td>State 57</td>
<td>5</td>
<td>State 60</td>
<td>State 107</td>
<td>5</td>
<td>State 96</td>
<td>State 55</td>
<td>-6003</td>
<td>State 132</td>
<td>State 168</td>
<td>8</td>
</tr>
<tr>
<td>State 25</td>
<td>State 60</td>
<td>6</td>
<td>State 60</td>
<td>State 59</td>
<td>0</td>
<td>State 96</td>
<td>State 50</td>
<td>-5</td>
<td>State 132</td>
<td>State 165</td>
<td>6003</td>
</tr>
<tr>
<td>State 25</td>
<td>State 21</td>
<td>-11</td>
<td>State 60</td>
<td>State 6</td>
<td>-11</td>
<td>State 96</td>
<td>State 17</td>
<td>-6</td>
<td>State 132</td>
<td>State 82</td>
<td>-5</td>
</tr>
<tr>
<td>State 26</td>
<td>State 9</td>
<td>-8</td>
<td>State 61</td>
<td>State 30</td>
<td>-8</td>
<td>State 97</td>
<td>State 122</td>
<td>6005</td>
<td>State 132</td>
<td>State 34</td>
<td>-11</td>
</tr>
<tr>
<td>State 26</td>
<td>State 8</td>
<td>-6003</td>
<td>State 61</td>
<td>State 27</td>
<td>-6003</td>
<td>State 97</td>
<td>State 143</td>
<td>8</td>
<td>State 133</td>
<td>State 151</td>
<td>6005</td>
</tr>
<tr>
<td>State 26</td>
<td>State 61</td>
<td>5</td>
<td>State 61</td>
<td>State 108</td>
<td>6</td>
<td>State 97</td>
<td>State 56</td>
<td>-6003</td>
<td>State 133</td>
<td>State 96</td>
<td>-8</td>
</tr>
<tr>
<td>State 26</td>
<td>State 63</td>
<td>11</td>
<td>State 62</td>
<td>State 40</td>
<td>-6005</td>
<td>State 97</td>
<td>State 146</td>
<td>0</td>
<td>State 133</td>
<td>State 39</td>
<td>-6</td>
</tr>
<tr>
<td>State 27</td>
<td>State 14</td>
<td>-6005</td>
<td>State 62</td>
<td>State 31</td>
<td>-8</td>
<td>State 97</td>
<td>State 147</td>
<td>0</td>
<td>State 133</td>
<td>State 169</td>
<td>11</td>
</tr>
<tr>
<td>State 27</td>
<td>State 10</td>
<td>-8</td>
<td>State 62</td>
<td>State 28</td>
<td>-6003</td>
<td>State 98</td>
<td>State 123</td>
<td>6005</td>
<td>State 134</td>
<td>State 152</td>
<td>6005</td>
</tr>
<tr>
<td>State 27</td>
<td>State 61</td>
<td>6003</td>
<td>State 62</td>
<td>State 108</td>
<td>5</td>
<td>State 98</td>
<td>State 144</td>
<td>8</td>
<td>State 134</td>
<td>State 97</td>
<td>-8</td>
</tr>
<tr>
<td>State 27</td>
<td>State 64</td>
<td>6</td>
<td>State 62</td>
<td>State 110</td>
<td>0</td>
<td>State 98</td>
<td>State 57</td>
<td>-6003</td>
<td>State 134</td>
<td>State 89</td>
<td>-6003</td>
</tr>
<tr>
<td>State 27</td>
<td>State 65</td>
<td>11</td>
<td>State 62</td>
<td>State 111</td>
<td>11</td>
<td>State 98</td>
<td>State 52</td>
<td>-5</td>
<td>State 134</td>
<td>State 170</td>
<td>0</td>
</tr>
<tr>
<td>State 28</td>
<td>State 15</td>
<td>-6005</td>
<td>State 63</td>
<td>State 41</td>
<td>-6005</td>
<td>State 98</td>
<td>State 147</td>
<td>6</td>
<td>State 134</td>
<td>State 171</td>
<td>0</td>
</tr>
<tr>
<td>State 28</td>
<td>State 11</td>
<td>State 63</td>
<td>State 32</td>
<td>State 98</td>
<td>State 17</td>
<td>State 135</td>
<td>State 153</td>
<td>6005</td>
<td>State 173</td>
<td>State 179</td>
<td>6005</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>------</td>
<td>---------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>State 28</td>
<td>State 62</td>
<td>6003</td>
<td>State 63</td>
<td>State 29</td>
<td>-6003</td>
<td>State 99</td>
<td>State 124</td>
<td>6005</td>
<td>State 135</td>
<td>State 98</td>
<td>-8</td>
</tr>
<tr>
<td>State 28</td>
<td>State 66</td>
<td>0</td>
<td>State 63</td>
<td>State 111</td>
<td>6</td>
<td>State 99</td>
<td>State 58</td>
<td>-6003</td>
<td>State 135</td>
<td>State 171</td>
<td>6</td>
</tr>
<tr>
<td>State 29</td>
<td>State 16</td>
<td>-6005</td>
<td>State 64</td>
<td>State 42</td>
<td>-6005</td>
<td>State 99</td>
<td>State 19</td>
<td>-6</td>
<td>State 136</td>
<td>State 154</td>
<td>6005</td>
</tr>
<tr>
<td>State 29</td>
<td>State 12</td>
<td>-8</td>
<td>State 64</td>
<td>State 34</td>
<td>-8</td>
<td>State 99</td>
<td>State 100</td>
<td>0</td>
<td>State 136</td>
<td>State 99</td>
<td>-8</td>
</tr>
<tr>
<td>State 29</td>
<td>State 63</td>
<td>6003</td>
<td>State 64</td>
<td>State 108</td>
<td>6003</td>
<td>State 100</td>
<td>State 125</td>
<td>6005</td>
<td>State 136</td>
<td>State 91</td>
<td>-6003</td>
</tr>
<tr>
<td>State 29</td>
<td>State 65</td>
<td>5</td>
<td>State 64</td>
<td>State 113</td>
<td>0</td>
<td>State 100</td>
<td>State 137</td>
<td>8</td>
<td>State 136</td>
<td>State 169</td>
<td>5</td>
</tr>
<tr>
<td>State 29</td>
<td>State 67</td>
<td>6</td>
<td>State 64</td>
<td>State 114</td>
<td>11</td>
<td>State 100</td>
<td>State 59</td>
<td>-6003</td>
<td>State 136</td>
<td>State 41</td>
<td>-6</td>
</tr>
<tr>
<td>State 29</td>
<td>State 68</td>
<td>0</td>
<td>State 65</td>
<td>State 43</td>
<td>-6005</td>
<td>State 100</td>
<td>State 146</td>
<td>5</td>
<td>State 136</td>
<td>State 137</td>
<td>0</td>
</tr>
<tr>
<td>State 30</td>
<td>State 17</td>
<td>-6005</td>
<td>State 65</td>
<td>State 35</td>
<td>-8</td>
<td>State 100</td>
<td>State 52</td>
<td>-6</td>
<td>State 137</td>
<td>State 155</td>
<td>6005</td>
</tr>
<tr>
<td>State 30</td>
<td>State 69</td>
<td>8</td>
<td>State 65</td>
<td>State 109</td>
<td>6003</td>
<td>State 100</td>
<td>State 50</td>
<td>-11</td>
<td>State 137</td>
<td>State 100</td>
<td>-8</td>
</tr>
<tr>
<td>State 30</td>
<td>State 10</td>
<td>-6003</td>
<td>State 65</td>
<td>State 114</td>
<td>6</td>
<td>State 101</td>
<td>State 126</td>
<td>6005</td>
<td>State 137</td>
<td>State 92</td>
<td>-6003</td>
</tr>
<tr>
<td>State 30</td>
<td>State 9</td>
<td>-5</td>
<td>State 65</td>
<td>State 115</td>
<td>0</td>
<td>State 101</td>
<td>State 138</td>
<td>8</td>
<td>State 136</td>
<td>State 170</td>
<td>5</td>
</tr>
<tr>
<td>State 30</td>
<td>State 70</td>
<td>6</td>
<td>State 66</td>
<td>State 44</td>
<td>-6005</td>
<td>State 101</td>
<td>State 60</td>
<td>-6003</td>
<td>State 137</td>
<td>State 87</td>
<td>-6</td>
</tr>
<tr>
<td>State 30</td>
<td>State 71</td>
<td>11</td>
<td>State 66</td>
<td>State 36</td>
<td>-8</td>
<td>State 101</td>
<td>State 147</td>
<td>5</td>
<td>State 137</td>
<td>State 85</td>
<td>-11</td>
</tr>
<tr>
<td>State 31</td>
<td>State 18</td>
<td>-6005</td>
<td>State 66</td>
<td>State 110</td>
<td>6003</td>
<td>State 101</td>
<td>State 100</td>
<td>0</td>
<td>State 138</td>
<td>State 156</td>
<td>6005</td>
</tr>
<tr>
<td>State 31</td>
<td>State 62</td>
<td>8</td>
<td>State 66</td>
<td>State 113</td>
<td>5</td>
<td>State 101</td>
<td>State 18</td>
<td>-11</td>
<td>State 138</td>
<td>State 101</td>
<td>-8</td>
</tr>
<tr>
<td>State 31</td>
<td>State 11</td>
<td>-6003</td>
<td>State 66</td>
<td>State 8</td>
<td>-6</td>
<td>State 102</td>
<td>State 127</td>
<td>6005</td>
<td>State 138</td>
<td>State 93</td>
<td>-6003</td>
</tr>
<tr>
<td>State 31</td>
<td>State 70</td>
<td>5</td>
<td>State 66</td>
<td>State 116</td>
<td>11</td>
<td>State 102</td>
<td>State 142</td>
<td>6003</td>
<td>State 138</td>
<td>State 171</td>
<td>5</td>
</tr>
<tr>
<td>State 31</td>
<td>State 72</td>
<td>0</td>
<td>State 67</td>
<td>State 45</td>
<td>-6005</td>
<td>State 102</td>
<td>State 44</td>
<td>-5</td>
<td>State 138</td>
<td>State 137</td>
<td>0</td>
</tr>
<tr>
<td>State 31</td>
<td>State 73</td>
<td>11</td>
<td>State 67</td>
<td>State 37</td>
<td>-8</td>
<td>State 102</td>
<td>State 20</td>
<td>-6</td>
<td>State 138</td>
<td>State 40</td>
<td>-11</td>
</tr>
<tr>
<td>State 32</td>
<td>State 19</td>
<td>-6005</td>
<td>State 67</td>
<td>State 111</td>
<td>6003</td>
<td>State 102</td>
<td>State 148</td>
<td>11</td>
<td>State 139</td>
<td>State 157</td>
<td>6005</td>
</tr>
<tr>
<td>State 32</td>
<td>State 63</td>
<td>8</td>
<td>State 67</td>
<td>State 114</td>
<td>5</td>
<td>State 103</td>
<td>State 128</td>
<td>6005</td>
<td>State 139</td>
<td>State 105</td>
<td>-8</td>
</tr>
<tr>
<td>State 32</td>
<td>State 12</td>
<td>-6003</td>
<td>State 67</td>
<td>State 117</td>
<td>0</td>
<td>State 103</td>
<td>State 143</td>
<td>6003</td>
<td>State 139</td>
<td>State 169</td>
<td>6003</td>
</tr>
<tr>
<td>State 32</td>
<td>State 71</td>
<td>5</td>
<td>State 67</td>
<td>State 118</td>
<td>0</td>
<td>State 103</td>
<td>State 45</td>
<td>-5</td>
<td>State 139</td>
<td>State 43</td>
<td>-6</td>
</tr>
<tr>
<td>State 32</td>
<td>State 73</td>
<td>6</td>
<td>State 68</td>
<td>State 46</td>
<td>-6005</td>
<td>State 103</td>
<td>State 149</td>
<td>0</td>
<td>State 139</td>
<td>State 140</td>
<td>0</td>
</tr>
<tr>
<td>State 32</td>
<td>State 74</td>
<td>0</td>
<td>State 68</td>
<td>State 38</td>
<td>-8</td>
<td>State 103</td>
<td>State 150</td>
<td>0</td>
<td>State 140</td>
<td>State 158</td>
<td>6005</td>
</tr>
<tr>
<td>State 33</td>
<td>State 20</td>
<td>-6005</td>
<td>State 68</td>
<td>State 112</td>
<td>6003</td>
<td>State 104</td>
<td>State 129</td>
<td>6005</td>
<td>State 140</td>
<td>State 106</td>
<td>-8</td>
</tr>
<tr>
<td>State 33</td>
<td>State 69</td>
<td>6003</td>
<td>State 68</td>
<td>State 115</td>
<td>5</td>
<td>State 104</td>
<td>State 144</td>
<td>6003</td>
<td>State 140</td>
<td>State 170</td>
<td>6003</td>
</tr>
<tr>
<td>State 33</td>
<td>State 8</td>
<td>-5</td>
<td>State 68</td>
<td>State 118</td>
<td>6</td>
<td>State 104</td>
<td>State 46</td>
<td>-5</td>
<td>State 140</td>
<td>State 90</td>
<td>-6</td>
</tr>
<tr>
<td>State 33</td>
<td>State 75</td>
<td>6</td>
<td>State 68</td>
<td>State 8</td>
<td>-11</td>
<td>State 104</td>
<td>State 150</td>
<td>6</td>
<td>State 140</td>
<td>State 88</td>
<td>-11</td>
</tr>
<tr>
<td>State 33</td>
<td>State 76</td>
<td>11</td>
<td>State 69</td>
<td>State 47</td>
<td>-6005</td>
<td>State 104</td>
<td>State 20</td>
<td>-11</td>
<td>State 141</td>
<td>State 159</td>
<td>6005</td>
</tr>
<tr>
<td>State 34</td>
<td>State 21</td>
<td>-6005</td>
<td>State 69</td>
<td>State 33</td>
<td>-6003</td>
<td>State 105</td>
<td>State 130</td>
<td>6005</td>
<td>State 141</td>
<td>State 107</td>
<td>-8</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-------</td>
<td>---------</td>
<td>---------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
<td>------</td>
<td>-----------</td>
<td>-----------</td>
<td>----</td>
</tr>
<tr>
<td>State 34</td>
<td>State 75</td>
<td>8</td>
<td>State 69</td>
<td>State 26</td>
<td>-5</td>
<td>State 105</td>
<td>State 148</td>
<td>8</td>
<td>State 141</td>
<td>State 171</td>
<td>6003</td>
</tr>
<tr>
<td>State 34</td>
<td>State 70</td>
<td>6003</td>
<td>State 69</td>
<td>State 119</td>
<td>6</td>
<td>State 105</td>
<td>State 145</td>
<td>6003</td>
<td>State 141</td>
<td>State 140</td>
<td>0</td>
</tr>
<tr>
<td>State 34</td>
<td>State 77</td>
<td>0</td>
<td>State 70</td>
<td>State 48</td>
<td>-6005</td>
<td>State 105</td>
<td>State 22</td>
<td>-6</td>
<td>State 142</td>
<td>State 160</td>
<td>6005</td>
</tr>
<tr>
<td>State 34</td>
<td>State 78</td>
<td>11</td>
<td>State 70</td>
<td>State 119</td>
<td>8</td>
<td>State 105</td>
<td>State 106</td>
<td>0</td>
<td>State 142</td>
<td>State 102</td>
<td>-6003</td>
</tr>
<tr>
<td>State 35</td>
<td>State 22</td>
<td>-6005</td>
<td>State 70</td>
<td>State 34</td>
<td>-6003</td>
<td>State 106</td>
<td>State 131</td>
<td>6005</td>
<td>State 142</td>
<td>State 85</td>
<td>-5</td>
</tr>
<tr>
<td>State 35</td>
<td>State 76</td>
<td>8</td>
<td>State 70</td>
<td>State 31</td>
<td>-5</td>
<td>State 106</td>
<td>State 149</td>
<td>8</td>
<td>State 142</td>
<td>State 47</td>
<td>-6</td>
</tr>
</tbody>
</table>

Note: the transition rate represented by the name of basic event stands for the corresponding conditional failure rate; the transition rate represented by '-' followed by the basic event name stands for the condition repair rate.

Table A.2 State transitions in the Markov model of module 6008
Appendix B Quantification Models for Modules in Chapter 9

Figure B.1 BDD for modules containing no dependency relationship (referring to figure 9.6)
The reduced Markov model for module 7001

The reduced Markov model for module 7007

The reduced Markov model for module 7008

The reduced Markov model for module 7009

The reduced Markov model for module 7001, 7007 – 7010
### Table B.1 States included in the Markov model for module 7011 (referring to figure 9.6)

<table>
<thead>
<tr>
<th>State no.</th>
<th>IV15</th>
<th>IV16</th>
<th>CV6</th>
<th>Module State</th>
<th>State no.</th>
<th>IV15</th>
<th>IV16</th>
<th>CV6</th>
<th>Module State</th>
<th>State no.</th>
<th>IV15</th>
<th>IV16</th>
<th>CV6</th>
<th>Module State</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>STATE 13</td>
<td>STATE 13</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>STATE 16</td>
<td>STATE 25</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>STATE 20</td>
</tr>
<tr>
<td>STATE 2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>STATE 14</td>
<td>STATE 14</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>STATE 27</td>
<td>STATE 26</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>STATE 30</td>
</tr>
<tr>
<td>STATE 3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>STATE 15</td>
<td>STATE 15</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>STATE 28</td>
<td>STATE 27</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>STATE 29</td>
</tr>
<tr>
<td>STATE 4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>STATE 16</td>
<td>STATE 16</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>STATE 29</td>
<td>STATE 28</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>STATE 19</td>
</tr>
<tr>
<td>STATE 5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>STATE 17</td>
<td>STATE 17</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>STATE 30</td>
<td>STATE 29</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>STATE 20</td>
</tr>
<tr>
<td>STATE 6</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>STATE 18</td>
<td>STATE 18</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>STATE 33</td>
<td>STATE 28</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>STATE 30</td>
</tr>
<tr>
<td>STATE 7</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>STATE 19</td>
<td>STATE 19</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>STATE 31</td>
<td>STATE 32</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>STATE 32</td>
</tr>
<tr>
<td>STATE 8</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>STATE 20</td>
<td>STATE 20</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>STATE 33</td>
<td>STATE 32</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>STATE 30</td>
</tr>
<tr>
<td>STATE 9</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>STATE 21</td>
<td>STATE 21</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>STATE 35</td>
<td>STATE 33</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>STATE 19</td>
</tr>
<tr>
<td>STATE 10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>STATE 22</td>
<td>STATE 22</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>STATE 34</td>
<td>STATE 34</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>STATE 32</td>
</tr>
<tr>
<td>STATE 11</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>STATE 23</td>
<td>STATE 23</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>STATE 35</td>
<td>STATE 34</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>STATE 32</td>
</tr>
<tr>
<td>STATE 12</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>STATE 24</td>
<td>STATE 24</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>STATE 35</td>
<td>STATE 34</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>STATE 32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
<th>From</th>
<th>To</th>
<th>Transition rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE 1</td>
<td>STATE 2</td>
<td>IV15</td>
<td>STATE 8</td>
<td>STATE 1</td>
<td>-IV16</td>
<td>STATE 16</td>
<td>STATE 24</td>
<td>IV16</td>
<td>STATE 25</td>
<td>STATE 20</td>
<td>-IV15</td>
<td>STATE 1</td>
<td>STATE 3</td>
<td>III6</td>
</tr>
<tr>
<td>STATE 1</td>
<td>STATE 3</td>
<td>IV16</td>
<td>STATE 8</td>
<td>STATE 18</td>
<td>CV6</td>
<td>STATE 16</td>
<td>STATE 2</td>
<td>-CV6</td>
<td>STATE 25</td>
<td>STATE 23</td>
<td>0</td>
<td>STATE 1</td>
<td>STATE 4</td>
<td>CV6</td>
</tr>
<tr>
<td>STATE 1</td>
<td>STATE 4</td>
<td>CV6</td>
<td>STATE 9</td>
<td>STATE 14</td>
<td>IV15</td>
<td>STATE 17</td>
<td>STATE 27</td>
<td>0</td>
<td>STATE 26</td>
<td>STATE 32</td>
<td>IV16</td>
<td>STATE 2</td>
<td>STATE 5</td>
<td>0</td>
</tr>
<tr>
<td>STATE 2</td>
<td>STATE 5</td>
<td>0</td>
<td>STATE 9</td>
<td>STATE 19</td>
<td>0</td>
<td>STATE 17</td>
<td>STATE 2</td>
<td>-IV16</td>
<td>STATE 26</td>
<td>STATE 5</td>
<td>-CV6</td>
<td>STATE 2</td>
<td>STATE 6</td>
<td>IV16</td>
</tr>
<tr>
<td>STATE 2</td>
<td>STATE 6</td>
<td>IV16</td>
<td>STATE 9</td>
<td>STATE 20</td>
<td>0</td>
<td>STATE 17</td>
<td>STATE 28</td>
<td>CV6</td>
<td>STATE 27</td>
<td>STATE 5</td>
<td>-IV16</td>
<td>STATE 2</td>
<td>STATE 7</td>
<td>CV6</td>
</tr>
<tr>
<td>STATE 2</td>
<td>STATE 7</td>
<td>CV6</td>
<td>STATE 10</td>
<td>STATE 16</td>
<td>IV15</td>
<td>STATE 18</td>
<td>STATE 28</td>
<td>CV6</td>
<td>STATE 27</td>
<td>STATE 5</td>
<td>-IV16</td>
<td>STATE 2</td>
<td>STATE 3</td>
<td>VI15</td>
</tr>
<tr>
<td>STATE 3</td>
<td>STATE 6</td>
<td>IV15</td>
<td>STATE 10</td>
<td>STATE 20</td>
<td>IV16</td>
<td>STATE 18</td>
<td>STATE 4</td>
<td>-IV16</td>
<td>STATE 28</td>
<td>STATE 34</td>
<td>0</td>
<td>STATE 3</td>
<td>STATE 8</td>
<td>0</td>
</tr>
<tr>
<td>STATE 3</td>
<td>STATE 8</td>
<td>0</td>
<td>STATE 10</td>
<td>STATE 1</td>
<td>-CV6</td>
<td>STATE 18</td>
<td>STATE 19</td>
<td>0</td>
<td>STATE 28</td>
<td>STATE 7</td>
<td>-IV16</td>
<td>STATE 3</td>
<td>STATE 9</td>
<td>CV6</td>
</tr>
<tr>
<td>STATE 3</td>
<td>STATE 9</td>
<td>CV6</td>
<td>STATE 11</td>
<td>STATE 3</td>
<td>-IV15</td>
<td>STATE 19</td>
<td>STATE 29</td>
<td>IV15</td>
<td>STATE 28</td>
<td>STATE 29</td>
<td>0</td>
<td>STATE 4</td>
<td>STATE 7</td>
<td>IV15</td>
</tr>
<tr>
<td>STATE 4</td>
<td>STATE 7</td>
<td>IV15</td>
<td>STATE 11</td>
<td>STATE 13</td>
<td>0</td>
<td>STATE 19</td>
<td>STATE 10</td>
<td>-IV16</td>
<td>STATE 29</td>
<td>STATE 34</td>
<td>0</td>
<td>STATE 4</td>
<td>STATE 9</td>
<td>IV16</td>
</tr>
<tr>
<td>STATE 4</td>
<td>STATE 9</td>
<td>IV16</td>
<td>STATE 11</td>
<td>STATE 21</td>
<td>CV6</td>
<td>STATE 20</td>
<td>STATE 24</td>
<td>IV15</td>
<td>STATE 29</td>
<td>STATE 16</td>
<td>-IV16</td>
<td>STATE 4</td>
<td>STATE 10</td>
<td>0</td>
</tr>
<tr>
<td>STATE 4</td>
<td>STATE 10</td>
<td>0</td>
<td>STATE 12</td>
<td>STATE 4</td>
<td>-IV15</td>
<td>STATE 20</td>
<td>STATE 30</td>
<td>0</td>
<td>STATE 30</td>
<td>STATE 35</td>
<td>IV15</td>
<td>STATE 5</td>
<td>STATE 1</td>
<td>-IV15</td>
</tr>
</tbody>
</table>
Table B.2 State transitions included in the Markov model for module 7011 (referring to figure 9.6)

The states included in the Markov model for module 7017 and 7018 are identical to the states included in the Markov model for module 7011. The state transitions included in the Markov model for these two modules are identical to each other since basic events included in the modules all have same inspection interval. State transitions included in the Markov model for these two modules are displayed in table B.8.
Note: basic events INB, SB, and NB in table B.8 correspond to basic events WDV, WVS and ADV respectively in module 7018.

Table B.3 State transitions included in the Markov model for module 7017 (referring to figure 9.6)

<table>
<thead>
<tr>
<th>TIME (in month)</th>
<th>UNAVAILABILITY $Q_{sys}(t)$</th>
<th>UNRELIABILITY $F_{sys}(T_r)$</th>
<th>CONDITIONAL UNRELIABILITY</th>
<th>STATIC FAILURE INTENSITY ($w_{acc, s}$)</th>
<th>DYNAMIC FAILURE INTENSITY ($w_{acc, d}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.9997E-04</td>
<td>1.8235E-05</td>
<td>1.8240E-05</td>
<td>2.9997E-10</td>
<td>1.8235E-11</td>
</tr>
<tr>
<td>1</td>
<td>1.5204E-02</td>
<td>4.1519E-05</td>
<td>4.2160E-05</td>
<td>1.5204E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>2</td>
<td>3.2424E-02</td>
<td>6.7737E-05</td>
<td>7.0007E-05</td>
<td>3.2424E-08</td>
<td>6.7737E-11</td>
</tr>
<tr>
<td>4</td>
<td>3.3363E-02</td>
<td>4.1519E-05</td>
<td>4.2952E-05</td>
<td>3.3363E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>5</td>
<td>4.6391E-02</td>
<td>6.7737E-05</td>
<td>7.1032E-05</td>
<td>4.6391E-08</td>
<td>6.7737E-11</td>
</tr>
<tr>
<td>7</td>
<td>1.5213E-02</td>
<td>4.1519E-05</td>
<td>4.2160E-05</td>
<td>1.5213E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>10</td>
<td>3.3405E-02</td>
<td>4.1519E-05</td>
<td>4.2954E-05</td>
<td>3.3405E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>13</td>
<td>1.5223E-02</td>
<td>4.1519E-05</td>
<td>4.2161E-05</td>
<td>1.5223E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>16</td>
<td>3.3449E-02</td>
<td>4.1519E-05</td>
<td>4.2956E-05</td>
<td>3.3449E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>17</td>
<td>4.6478E-02</td>
<td>6.7737E-05</td>
<td>7.1038E-05</td>
<td>4.6478E-08</td>
<td>6.7737E-11</td>
</tr>
<tr>
<td>19</td>
<td>1.5234E-02</td>
<td>4.1519E-05</td>
<td>4.2161E-05</td>
<td>1.5234E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>--------</td>
<td>--------</td>
<td>----------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>20</td>
<td>3.2597E-02</td>
<td>6.7737E-05</td>
<td>7.0019E-05</td>
<td>3.2597E-08</td>
<td>6.7737E-11</td>
</tr>
<tr>
<td>22</td>
<td>3.3496E-02</td>
<td>4.1519E-05</td>
<td>4.2958E-05</td>
<td>3.3496E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>23</td>
<td>4.6525E-02</td>
<td>6.7737E-05</td>
<td>7.1042E-05</td>
<td>4.6525E-08</td>
<td>6.7737E-11</td>
</tr>
<tr>
<td>25</td>
<td>1.5245E-02</td>
<td>4.1519E-05</td>
<td>4.2161E-05</td>
<td>1.5245E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>26</td>
<td>3.2661E-02</td>
<td>6.7737E-05</td>
<td>7.0024E-05</td>
<td>3.2661E-08</td>
<td>6.7737E-11</td>
</tr>
<tr>
<td>28</td>
<td>3.3545E-02</td>
<td>4.1519E-05</td>
<td>4.2960E-05</td>
<td>3.3545E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>31</td>
<td>1.5256E-02</td>
<td>4.1519E-05</td>
<td>4.2162E-05</td>
<td>1.5256E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>33</td>
<td>4.6676E-02</td>
<td>9.6031E-05</td>
<td>1.0073E-04</td>
<td>4.6676E-08</td>
<td>9.6031E-11</td>
</tr>
<tr>
<td>34</td>
<td>3.3598E-02</td>
<td>4.1519E-05</td>
<td>4.2962E-05</td>
<td>3.3598E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>35</td>
<td>4.6624E-02</td>
<td>6.7736E-05</td>
<td>7.1049E-05</td>
<td>4.6624E-08</td>
<td>6.7736E-11</td>
</tr>
<tr>
<td>37</td>
<td>1.5269E-02</td>
<td>4.1519E-05</td>
<td>4.2162E-05</td>
<td>1.5269E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>38</td>
<td>3.2800E-02</td>
<td>6.7736E-05</td>
<td>7.0034E-05</td>
<td>3.2800E-08</td>
<td>6.7736E-11</td>
</tr>
<tr>
<td>40</td>
<td>3.3653E-02</td>
<td>4.1519E-05</td>
<td>4.2965E-05</td>
<td>3.3653E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>41</td>
<td>4.6677E-02</td>
<td>6.7736E-05</td>
<td>7.1053E-05</td>
<td>4.6677E-08</td>
<td>6.7736E-11</td>
</tr>
<tr>
<td>43</td>
<td>1.5281E-02</td>
<td>4.1519E-05</td>
<td>4.2163E-05</td>
<td>1.5281E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>44</td>
<td>3.2875E-02</td>
<td>6.7736E-05</td>
<td>7.0039E-05</td>
<td>3.2875E-08</td>
<td>6.7736E-11</td>
</tr>
<tr>
<td>46</td>
<td>3.3713E-02</td>
<td>4.1519E-05</td>
<td>4.2967E-05</td>
<td>3.3713E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>47</td>
<td>4.6733E-02</td>
<td>6.7736E-05</td>
<td>7.1057E-05</td>
<td>4.6733E-08</td>
<td>6.7736E-11</td>
</tr>
<tr>
<td>49</td>
<td>1.5294E-02</td>
<td>4.1519E-05</td>
<td>4.2164E-05</td>
<td>1.5294E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>50</td>
<td>3.2954E-02</td>
<td>6.7736E-05</td>
<td>7.0045E-05</td>
<td>3.2954E-08</td>
<td>6.7736E-11</td>
</tr>
<tr>
<td>52</td>
<td>3.3776E-02</td>
<td>4.1519E-05</td>
<td>4.2970E-05</td>
<td>3.3776E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>55</td>
<td>1.5308E-02</td>
<td>4.1519E-05</td>
<td>4.2164E-05</td>
<td>1.5308E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>57</td>
<td>4.6927E-02</td>
<td>9.6030E-05</td>
<td>1.0076E-04</td>
<td>4.6927E-08</td>
<td>9.6030E-11</td>
</tr>
<tr>
<td>58</td>
<td>3.3843E-02</td>
<td>4.1519E-05</td>
<td>4.2973E-05</td>
<td>3.3843E-08</td>
<td>4.1519E-11</td>
</tr>
<tr>
<td>59</td>
<td>4.6852E-02</td>
<td>6.7736E-05</td>
<td>7.1066E-05</td>
<td>4.6852E-08</td>
<td>6.7736E-11</td>
</tr>
</tbody>
</table>

Table B.4 failure predictions for firewater deluge system

<table>
<thead>
<tr>
<th>Basic event</th>
<th>Time (months)</th>
<th>Criticality Function</th>
<th>Basic event</th>
<th>Time (months)</th>
<th>Criticality Function</th>
<th>Basic event</th>
<th>Time (months)</th>
<th>Criticality Function</th>
<th>Basic event</th>
<th>Time (months)</th>
<th>Criticality Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV11</td>
<td>0</td>
<td>0.0000E+00</td>
<td>DS</td>
<td>0</td>
<td>0.0000E+00</td>
<td>AES</td>
<td>0</td>
<td>1.9994E-04</td>
<td>AEF</td>
<td>0</td>
<td>1.9994E-04</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>8.2974E-02</td>
<td></td>
<td>10</td>
<td>8.2937E-02</td>
<td></td>
<td>-10</td>
<td>4.9601E-02</td>
<td></td>
<td>10</td>
<td>4.9923E-02</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>9.7992E-02</td>
<td></td>
<td>20</td>
<td>9.7885E-02</td>
<td></td>
<td>20</td>
<td>7.4374E-02</td>
<td></td>
<td>20</td>
<td>7.5344E-02</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1.2960E-01</td>
<td>DS</td>
<td>30</td>
<td>1.2938E-01</td>
<td>AES</td>
<td>30</td>
<td>1.1137E-01</td>
<td>AEF</td>
<td>30</td>
<td>1.1355E-01</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>9.3192E-02</td>
<td></td>
<td>40</td>
<td>9.3151E-02</td>
<td></td>
<td>40</td>
<td>4.9588E-02</td>
<td></td>
<td>40</td>
<td>4.9910E-02</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1.0713E-01</td>
<td></td>
<td>50</td>
<td>1.0702E-01</td>
<td></td>
<td>50</td>
<td>7.4346E-02</td>
<td></td>
<td>50</td>
<td>7.5316E-02</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>1.4074E-01</td>
<td>ADS</td>
<td>60</td>
<td>1.4050E-01</td>
<td></td>
<td>60</td>
<td>1.1126E-01</td>
<td></td>
<td>60</td>
<td>1.1345E-01</td>
</tr>
<tr>
<td>ADF</td>
<td>0</td>
<td>0.0000E+00</td>
<td>MPF</td>
<td>0</td>
<td>0.0000E+00</td>
<td>APF</td>
<td>0</td>
<td>9.9980E-01</td>
<td>INAF</td>
<td>0</td>
<td>0.0000E+00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2.9784E-02</td>
<td></td>
<td>10</td>
<td>2.8736E-02</td>
<td></td>
<td>10</td>
<td>9.6669E-01</td>
<td></td>
<td>10</td>
<td>6.2907E-04</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3.9550E-02</td>
<td></td>
<td>20</td>
<td>3.6810E-02</td>
<td></td>
<td>20</td>
<td>9.6750E-01</td>
<td></td>
<td>20</td>
<td>1.7863E-03</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>6.3122E-02</td>
<td></td>
<td>30</td>
<td>5.6672E-02</td>
<td></td>
<td>30</td>
<td>9.3336E-01</td>
<td></td>
<td>30</td>
<td>3.5500E-03</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>2.9776E-02</td>
<td></td>
<td>40</td>
<td>2.8729E-02</td>
<td></td>
<td>40</td>
<td>9.6644E-01</td>
<td></td>
<td>40</td>
<td>6.7605E-04</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>3.9535E-02</td>
<td></td>
<td>50</td>
<td>3.6796E-02</td>
<td></td>
<td>50</td>
<td>9.6714E-01</td>
<td></td>
<td>50</td>
<td>1.8637E-03</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>6.3064E-02</td>
<td></td>
<td>60</td>
<td>5.6619E-02</td>
<td></td>
<td>60</td>
<td>9.3249E-01</td>
<td></td>
<td>60</td>
<td>3.6813E-03</td>
</tr>
<tr>
<td></td>
<td>FB1</td>
<td>FB2</td>
<td>IV3</td>
<td>IV4</td>
<td>IV5</td>
<td>IV6</td>
<td>CV1</td>
<td>CV2</td>
<td>FB3</td>
<td>FB4</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WPS1F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WPS2F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WPS3F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WPS4F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SV1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SV2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FS1F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FS2F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FS3F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APS1F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APS2F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APS3F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IV16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Values:**
- **FB1:** 0.0000E+00
- **FB2:** 0.0000E+00
- **IV3:** 1.9994E-04
- **IV4:** 1.0360E-02
- **IV5:** 1.4296E-02
- **IV6:** 2.6230E-02
- **CV1:** 0.0000E+00
- **CV2:** 1.9994E-04
- **FB3:** 7.2262E-03
- **FB4:** 0.0000E+00
- **IV1:** 9.9970E-01
- **IV2:** 9.7076E-01
- **IV3:** 0.0000E+00
- **IV4:** 2.0488E-02
- **IV5:** 2.9009E-02
- **IV6:** 1.4550E-02
- **CV1:** 0.0000E+00
- **CV2:** 1.9994E-04
- **IV7:** 1.0543E-02
- **IV8:** 1.0360E-02
- **IV9:** 1.4296E-02
- **IV10:** 2.6230E-02
- **CV7:** 0.0000E+00
- **IV11:** 2.0488E-02
- **IV12:** 2.9009E-02
- **IV13:** 1.4550E-02
- **CV5:** 0.0000E+00

**Additional Values:**
- **IV1:** 2.0360E-02
- **IV2:** 3.8407E-02
- **IV3:** 1.4276E-02
- **IV4:** 5.0513E-02
- **IV5:** 4.9734E-02
- **IV6:** 4.9562E-02
- **CV5:** 0.0000E+00
- **IV7:** 7.1004E-03
- **IV8:** 7.0902E-03
- **IV9:** 9.9890E-03
- **IV10:** 7.0159E-03
- **CV5:** 0.0000E+00
- **IV11:** 1.1858E-02
- **IV12:** 1.1865E-02
- **CV5:** 0.0000E+00
- **IV13:** 1.0208E-02
- **IV14:** 1.0193E-02
- **CV5:** 0.0000E+00
- **IV15:** 1.0543E-02
- **IV16:** 1.0360E-02
- **CV5:** 0.0000E+00
- **IV17:** 2.0488E-02
- **IV18:** 2.9009E-02
- **CV5:** 0.0000E+00
<table>
<thead>
<tr>
<th>Time of activation</th>
<th>B-P measure</th>
<th>ES1</th>
<th>E1D</th>
<th>PRV1</th>
<th>PRV2</th>
<th>DS</th>
<th>D1D</th>
<th>PRV3</th>
<th>D2D</th>
<th>PRV4</th>
<th>AED</th>
<th>AES</th>
<th>PRV5</th>
<th>ADD</th>
<th>ADS</th>
<th>PRV6</th>
<th>INAF</th>
</tr>
</thead>
</table>

Table B.5 Criticality function of each component during the static-phase of the firewater deluge system in chapter 9.
Table B.6 Barlow-Proshan importance measure for dynamic-phase analysis of the firewater deluge system in chapter 9

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>-04</td>
<td>-02</td>
<td>-03</td>
<td>-03</td>
<td>-02</td>
<td>-04</td>
<td>-02</td>
<td>-04</td>
<td>-01</td>
<td>-02</td>
<td>-03</td>
<td>-01</td>
<td>-03</td>
<td>-03</td>
<td>-01</td>
</tr>
<tr>
<td>8</td>
<td>-04</td>
<td>-02</td>
<td>-03</td>
<td>-03</td>
<td>-02</td>
<td>-04</td>
<td>-02</td>
<td>-04</td>
<td>-01</td>
<td>-02</td>
<td>-03</td>
<td>-01</td>
<td>-03</td>
<td>-03</td>
<td>-01</td>
</tr>
<tr>
<td>10</td>
<td>-03</td>
<td>-02</td>
<td>-03</td>
<td>-03</td>
<td>-02</td>
<td>-04</td>
<td>-02</td>
<td>-04</td>
<td>-01</td>
<td>-02</td>
<td>-03</td>
<td>-01</td>
<td>-03</td>
<td>-03</td>
<td>-01</td>
</tr>
<tr>
<td>12</td>
<td>-04</td>
<td>-02</td>
<td>-03</td>
<td>-03</td>
<td>-02</td>
<td>-04</td>
<td>-02</td>
<td>-04</td>
<td>-01</td>
<td>-02</td>
<td>-03</td>
<td>-01</td>
<td>-03</td>
<td>-03</td>
<td>-01</td>
</tr>
</tbody>
</table>