

This item was submitted to Loughborough's Institutional Repository (<https://dspace.lboro.ac.uk/>) by the author and is made available under the following Creative Commons Licence conditions.



For the full text of this licence, please go to:  
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

# **A Fast Reliability Analysis for an Unmanned Aerial Vehicle Performing a Phased Mission**

by

James Poole

A Doctoral thesis

Submitted in partial fulfilment of the requirement for the award of  
Doctor of Philosophy of Loughborough University

January 2011

© by James Poole, 2011

## **Abstract**

It is becoming more common for Unmanned Aerial Vehicle (UAV) to perform phased mission where the phase's causes of failure may be different. The reliabilities of the phases are required throughout the mission in order to make future decisions for the mission. However, previous research of phased mission analysis has shown it to be very complex and take significantly long amounts of time. Also the analysis cannot be performed before the mission because information that is only available when the mission is active is required for the analysis. The aim of this research is develop new methods for a phased mission analysis which can obtain the phases reliabilities on a real structure UAV mission, where all the components are non-repairable, in the fastest time as possible.

The present methods are explored and the outcome is that the methods based on Binary Decision Diagram (BDD) analysis are the most efficient. Therefore the BDD analysis is use as a starting point for the new method. The phase mission BDD based methods are improved by altering the procedure of the analysis. Also modules that can appear in many phases can be taken out to simplify the analysis.

Search methods that lookup computations that have already been done before are investigated to determine how much impact it has on the speed of the analysis.

A method that restructures the phase's mission fault trees to optimize the number of modules that can be taken out is developed. It is tested on a real UAV mission and it is shown to significantly simplify the analysis. This method is extended by situation where a mission is being reconfigured several times throughout a mission and the analysis also has to be done several times. Additional changes are made by using part of the analysis of the original mission for the new one to speed up the analysis.

A method is developed which identifies parts of the analysis referred to as groups which can treated as a mini phase missions. Each group can be performed on separate processer in parallel that reduces the online analysis.

**I dedicate my Ph.D to  
My Mum**

## **Acknowledgements**

I would like to thank my supervisors Professor John Andrews and Dr Wen-Hua Chen for their guidance, advice and encouragement throughout the course of my Ph.D. I would like to say a huge thanks to my family for their support and endless enthusiasm. I also like to say thanks to the friends that I have made throughout my time at university for their friendship and laughter.

## Contents

### 1 Introduction

1.1	Risk and Reliability Assessment	1
1.2	System Failure Quantification	3
1.3	Fault Tree Analysis	6
1.4	Binary Decision diagrams	7
1.5	Phased Mission System	7
1.6	Research Objective	9

### 2 Reliability Tools

2.1	Introduction	11
2.2	Fault Tree Analysis	11
2.2.1	Introduction	11
2.2.2	Construction of a fault tree	11
2.3	Qualitative Analysis	13
2.4	Quantitative Analysis	14
2.4.1	Top Event Probability	14
2.4.2	Upper and lower Bounds for the existence of system failure	15
2.4.3	Minimal cut Upper Bound	16
2.5	Expected number of system failures	17
2.6	Structure Functions	19
2.7	Binary Decision Diagram	21
2.7.1	Introduction	21
2.7.2	Properties of the BDD	22
2.7.3	Formation of a BDD using If-Then-Else Structure	23
2.7.4	BDD Minimisation	27
2.7.5	Top event probability	27

### **3 Phased Mission Systems**

3.1	Introduction	29
3.2	Non-Repairable System	30
3.2.1	Introduction	30
3.2.2	The Cut Set Cancellation method	34
3.2.3	PMS Unreliability Approximation	36
3.2.4	Individual Phase Unreliability	39
3.2.4.2	Fault Tree Restructuring Techniques	40
3.2.4.3	Obtain the Prime Implicants PMS	42
3.2.5	Laws of Boolean Phase Algebra	43
3.2.6	Binary Decision Diagrams for Phased Mission Systems	52
3.2.7	Imperfect Fault Coverage	59
3.2.8	Cause-Consequence-Analysis	60

### **4 Development of new method for PMS Analysis using BDD**

4.1	Introduction	61
4.2	Efficient BDD method for PMS Analysis	61
4.3	Concise representation of mission phase failure (Method 1)	68
4.4	Alternative approach for Phased Mission Analysis using based on the Trivedi method ( Method 2)	74
4.5	Fault Tree Modularisation	85
4.6	Applying Modularisation to phase mission analysis (Method 3)	90
4.7	Results	107

### **5 Investigating the distribution of analysis time spent on recursive function**

5.1	Introduction	118
5.2	Storing the BDD in the code	118

5.3 Recursive function for the qualitative analysis	119
5.4 Modifying the recursive function for the qualitative analysis for PMS node	122
5.5 Investigating the analysis time taken for the nodes search function	124
5.6 Investigating the analysis time taken for the computation loop up function	136
 <b>6 Modularization by repeated gates and events method</b>	
6.1 Introduction	142
6.2 Restructuring technique	143
6.2.1 Push-up	143
6.2.2 Common-input Push-up	144
6.2.3 Elimination	147
6.2.4 Factorization	148
6.3 Worked example	150
6.3.1 Inputting the fault tree to the program	153
6.3.2 Restructuring the fault tree	156
6.4 Results	185
6.4.1 The test mission example	185
6.4.2 The effects of restructuring and taking out modules on the trees	186
6.4.3 The times of the mission calculation	190
6.4.5 Conclusion and further work	192
 <b>7 Updating the Phased Mission Analysis with mission reconfiguration</b>	
7.1 Introduction	193
7.2 Reconfiguration	193
7.3 The phase tasks	194
7.4 The method	195



7.4.1 Analysis of Original Mission	196
7.4.2 Analysis of Reconfigured Mission	200
7.5 Results and discussion	204
<b>8 Parallelization of a PMS Analysis</b>	
8.1 Introduction	208
8.2 The method	208
8.3 Example	212
8.4 Fault trees structure of practical UAV mission	218
8.5 Results and testing	219
8.6 Summary	222
<b>9 Conclusions and Future Work</b>	
9.1 Summary of Work	223
9.2 Conclusions	227
9.3 Further Work	228
9.3.1 Factorization	228
9.3.2 Optimum BDD Ordering schemes for PMS	229
9.3.3 Multi Platform Missions	229
9.3.4 Dependency	229
9.3.5 Multi Component states	229
9.3.6 Important Measure	230
<b>Reference</b>	231
<b>Appendices</b>	
Appendix A Mission data of random generated phase fault trees	234
Appendix B Mission data of UAV generated phase fault trees	245

Appendix C	Results for search lookup technique on Method 3	274
Appendix D	Results for search lookup technique on Method 2	280
Appendix E	Fault trees data for mission task phases ASW, ASUW and SAR	285
Appendix F	Mission Fault tree structure	290
	F.1 Introduction	290
	F.2 Phases of the mission	290
	F.3 Sub-Systems	294
	F.3.1 Hydraulic system	294
	F.3.2 Flight Control system	297
	F.3.3 Avionic System	299
	F.3.4 Fuel system	302
	F.3.5 Landing gear system	304
	F.3.6 Braking system	307
	F.3.7 Engine Turbo-fan	309
	F.3.8 Reverse Thrust	311
	F.3.9 Electrical system	313
	F.4 Method of building fault trees	315

## Nomenclature

$A_{(0,t_i]}$	Component A in the failed state at some point in phase i
$A_{(t_i,\infty)}$	Component A in the working state throughout phase i
$A_{[i,j]}$	Component A fail at some point in the between phases i and j
$A(t)$	Availability function
$C$	Consequence of an event
$C_i$	Existence of minimal cut set i
$C_{j,k}$	The event that the kth cut set exist in phase j
$E_j$	Boolean expression that represents the failure combinations of phase j.
$F(t)$	Unreliability function
$F_{xj}(t)$	Failure cumulative distribution probability function for component X up to time t in phase j
$g_{i,j}$	Boolean logic of group i in phase j
$G_i(q)$	Criticality function for event i (Birnbbaum's measure of importance)
$k_j$	Total number of minimal cut sets in phase j.
$MCS_{ij}$	Minimal cut-set i in phase j.
$n_{ij}$	Total number of basic event in the ith minimal cut-set of the jth phase
$N_{c_i}$	Number of basic events in the minimal cut set $C_i$
$N_{mcs}$	Total number of minimal cut sets.

$N_i$	BDD node i
$N_{mscj}$	Total number of cut sets in phase j.
P	Probability
$p_i$	Failure logic from the fault tree of phase i
$P_j$	The event that the system is in the failed state in phase j
$PFC_j$	Phase failure combination for phase j
$PFC_{j,k}$	Phase failure combination for phase j where K is the last phase
$\Pr(C_i)$	Probability of existence of minimal cut set i
$q_X$	The probability that basic event X exists
$q_X(t)$	The probability that basic event X exists at time t
$q_{xj}(t)$	Probability that the component X is in the failed state at t given that it was in the working state at the beginning of the phase j.
$Q_j$	Probability that mission failure occurs in phase j
$Q_{k / j}$	Probability that mission failure occurs in phases k given that the mission has be successful up to phase j.
$Q(t)$	Unavailability function
$Q_{MCSU}$	Minimal cut set upper bound
$Q_{MISS}$	The exact mission unreliability.
$Q_{MISSINEX}$	Unreliability mission bound using the inclusion-exclusion expansion

$Q_{MISSINEX-CC}$	$Q_{MISSINEX}$ with cut set cancellation technique
$Q_{MISSMCB}$	Minimal unreliability bound using the minimal cut set bound to estimate phase unreliability
$Q_{MISSMCB-CC}$	$Q_{MISSMCB}$ with cut set cancellation technique
$Q_{sys}(t)$	System, or top event, unavailability function (failure probability)
$R$	Risk
$R(t)$	Reliability function
$T_i$	Duration of phase i and t
$U_i$	System unreliability in phase i
$w_i$	Unconditional failure intensity for component i
$w_{sys}(t)$	System unconditional failure intensity
$W_{sys}(t_0, t_1)$	Expected number of system failure (top event occurrences) in $(t_0, t_1)$
$x_i$	Component $X$ failure occurs in phase i
$x_{i,j}$	Component $X$ failure occurs in phases i to j
$\bar{x}_{i,j}$	Component $X$ failure does not occur throughout phases i to j
$X_i$	The event that the component $X$ is in the failed state at the end of the phase i
$\bar{X}_i$	The event that the component $X$ is in the working state at the end of the phase i

$\lambda$	Component hazed rate
$\phi_{c_i}$	Structure function for minimal cut set $c_i$
$\phi_{sys}(\underline{x})$	System structure function, where $\underline{x}$ is vector of components binary indicator variable
$\phi_x$	Structure function for component x

# Chapter 1: Introduction

## 1.1 Risk and Reliability Assessment

There are systems in industries, such as nuclear, aeronautical, offshore and transport that when they fail can cause catastrophic consequences. Examples of such catastrophes are the explosion on the Piper Alpha oil platform in 1988 and the fire at the Chernobyl nuclear power plant in 1986, both of these examples resulted in multiple fatalities. To decrease the likelihood of such events occurring and to increase the safety of such systems, safety assessments are regularly conducted.

Reliability and risk methods have been around for a number of years with significant advances been made since Second World War. These methods calculate the probability or frequency of system failure along with the consequences and from this a decision can be made if the risk is acceptable.

The risk or 'expected loss',  $R$ , of any hazardous event is defined as the product of its consequence,  $C$ , and the probability or frequency of its occurrence,  $P$ , shown in equation 1.1.

$$R = C \times P \quad (1.1)$$

The risk can be decreased by reducing the consequence of the event, or by reducing the probability or frequency of the event.

The procedure to quantify the risk for a specific system hazard goes through the following four stages:

1. Identification of the potential safety hazards.
2. Estimation of the consequences of each hazard.
3. Estimation of the probability of occurrence of each hazard

4. Calculate the risk and compare the results of the analysis against the acceptability criteria.

The Consequence of a system hazard is usually obtained by predicting the number of fatalities. Modelling the consequence depends upon the industry since the system failure mode will be different from one industry to another. The reliability methods that calculate the probability and frequency of system hazard are: Event Tree Analysis (ETA), Markov Analysis and Fault Tree Analysis [1] which is commonly used and will be discussed in more detail later in this chapter.

Once for a system hazard the consequence and probability or frequency have been obtained, the risk can be calculated by equation 1.1. Now it must be decided if the risk is acceptable or not. The HSE (Health and Safety Executive) provide a three-band approach called the ALARP [1] principle which is shown in figure 1.1.

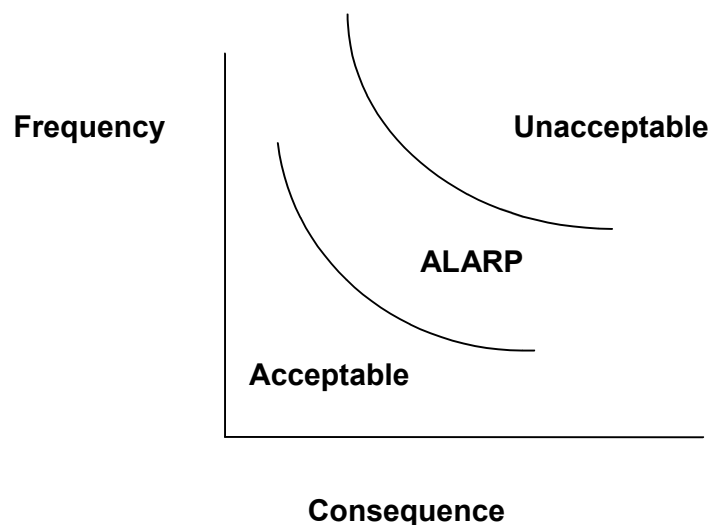


Figure 1.1: The ALARP principle



If the risk falls in the acceptable region then the risk is low enough to be permissible if current safeguards are maintained. If the risk falls in the unacceptable region then the risk is too high and must be reduced. The region between the two bands is called the ALARP region which is referred to as 'as low as reasonably practicable'. If the risk falls into this region then the risk must be shown to be as low as possible, whilst still being economically feasible.

## 1.2 System Failure Quantification

Reliability techniques are used to calculate the reliability performance of systems and components. System parameters such as reliability are generally computed in terms of the parameters of the components of the system. The main parameters that describe the system and component performance are defined below:

The unreliability of a system or component is defined as:

- The probability that a system or component fails to function successfully over a specified time period  $[0, t)$  and is denoted by  $F(t)$ .

The complement of unreliability is reliability. The reliability of a system or component is defined as:

- The probability that a system or component function successfully over a specified time period  $[0, t)$ , is denoted by  $R(t)$ . Therefore:

$$R(t) + F(t) = 1 \quad (1.2)$$

This parameter is more relevant for a system or a component where failure can only occur once such as catastrophic failure (non-repairable), therefore for the system to be successful it must function continuously over the specified time period.

When considering a system or a component which is repairable. A more appropriate parameter is the unavailability, which is defined as:

- The probability that a system or component is in the failed state at time  $t$ ,  $Q(t)$

Unavailability can also be defined as:

- The fraction of the total time that the system or component is in the failed state.

The complement of unavailability is availability  $A(t)$ . The two definitions of availability which correspond to the previous two unavailability definitions are as follows:

- The probability that a system or component is working at time  $t$ .
- The fraction of the total time that the system or component is successfully operating. Therefore:

$$Q(t) + A(t) = 1 \quad (1.3)$$

If the system or component is non-repairable then the parameters unreliability and unavailability are equal.

The hazard rate, also referred to as the conditional failure rate, is a parameter which measures the rate at which the system or component failure occurs is defined as:

- The probability that a system or component failure occurs during the interval  $[t, t + dt)$ , where  $dt$  is small, given that it has functioned successfully in the interval  $[0, t)$

The hazard rate plotted against time is commonly assumed to follow a curve called the reliability bath-tub curve, shown in figure 1.2.

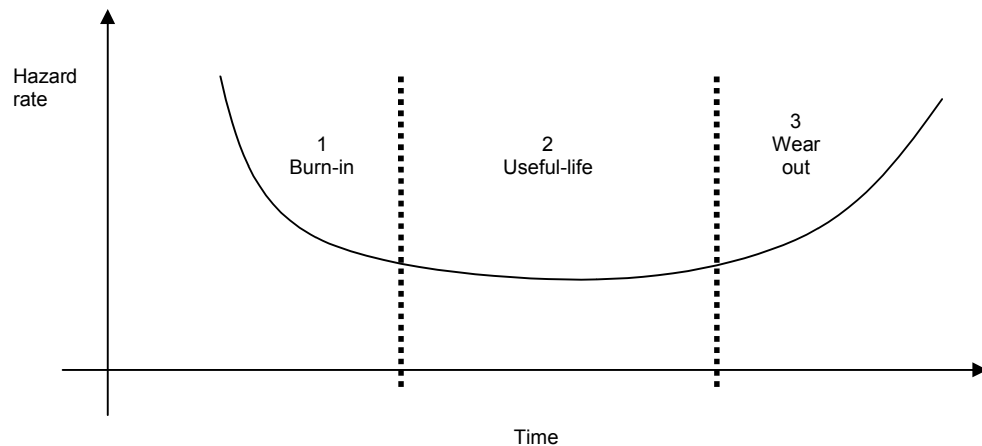


Figure 1.2: Bathtub Curve

The first phase of the curve shows the hazard rate decreasing. This is due to weak components that are eliminated. In the second phase the hazard rate stays approximately constant this is referred as the useful life phase. In the last phase the hazard rate increases due to the component wearing out. The reliability for a component that assumes that the hazard rate  $\lambda$  is constant (useful life phase) can be expressed as a function of time in equation (1.4).

$$R(t) = e^{-\lambda t} \quad (1.4)$$

More parameters can be expression as mathematical functions which are presented in [1]. The parameters of a system are generally expressed in terms of the components parameters. There are several methods which can be used to calculate system reliability parameters. The most popular one which is used a lot in system safety assessment is Fault Tree analysis. This is discussed in the next section.

### 1.3 Fault Tree Analysis

The Fault Tree Analysis was developed by H.A.Watson in the 1960s [1]. This is a deductive analysis method that provides a visual, symbolic diagram that logically represents the cause of a particular system failure mode by the concept of a 'what can cause this' approach. From this diagram the probability and frequency of system failure can be calculated.

The failure mode which is considered is referred to as the top event. The fault tree is constructed by starting with the top event and working downwards, building the fault tree beneath. Therefore branches are coming off from below the top event and logically describe the combination of events that cause this. This process continuous, until the basic events are encountered, which are the lowest form of causes. This is an example of 'top-down' technique, an alternative method is a 'bottom-top' approach such as FMEA which starts with a set of component failure conditions and considers the possible cause from these by a 'what happens if' approach.

Kinetic tree theory developed by Vesely in 1970's [2] is used to calculate reliability parameters such as the probability and frequency of the top event,

this information is used to determine if the risk is acceptable by the required safety standards. The disadvantage of using kinetic tree theory for the quantitative analysis is that for a large fault tree it becomes computationally intensive. This results in approximations being made which lead to inaccuracies in the calculation. However, a new method is developed which helps overcome this problem that is referred to as the Binary Decision Diagram method and is discussed in the next section.

#### **1.4 Binary Decision diagrams**

The Binary Decision Diagrams (BDD) technique for Fault tree Analysis was developed by Rauzy [3]. The BDD is construed by converting a Fault Tree, which encodes the system failure logic. Qualitative and Quantitative analysis are performed on the BDD, which is significantly more efficient than performing it on the original fault tree. The solution is exact and therefore there is no need for an approximation.

The BDD construction process requires the basic events from the fault tree to be placed in order. The size of the resulting BDD will depend on the order. Sizes can vary considerably depending on different orderings. Therefore the choice of order is important to have an efficient analysis. Previous research has never found a method of ordering which all ways results in the smallest size BDD.

#### **1.5 Phased Mission System**

Many systems operate what are known as phased mission. If a system must operate successfully over multiple, consecutive and non-overlapping periods (phases), where each phase performs a different function, then it is known as a Phased Mission System (PMSs). Many practical systems are PMSs, for example an aircraft flight mission that is divided into: taxiing to the runway,

take-off, climbing to the correct altitude, cruising, descending, landing and taxiing back to the terminal as shown in figure 1.3.

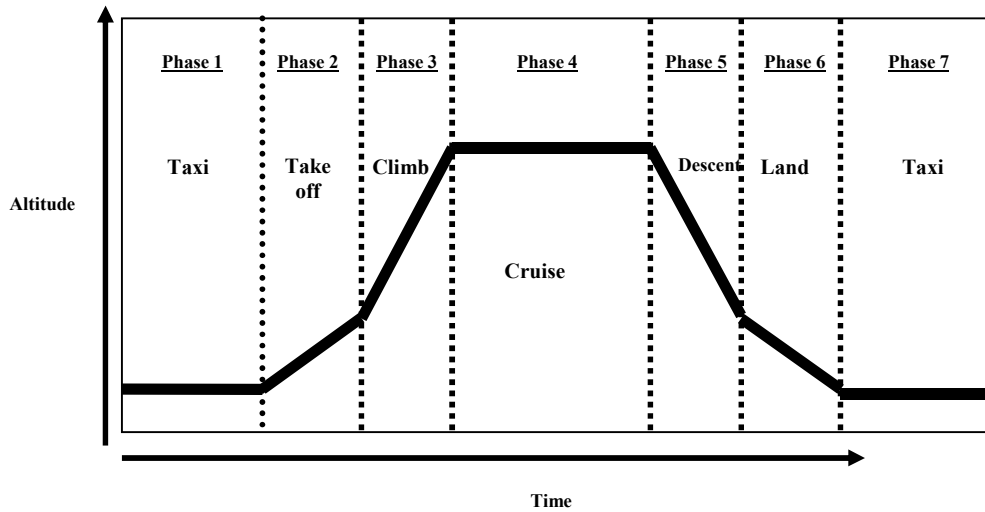


Figure 1.3 example of Phased Mission

Throughout the mission the configuration, success criteria and component behaviour may change between the different phases, since the phases accomplish different objectives. Each phase is identified by a phase index, phase length, success criteria, and failure parameters. A component may fail in any phase in the mission but may or may not contribute to the failure of that phase.

The PMS reliability is defined as the probability that the system operates successfully in all of its phases. The PMS unreliability is defined as the probability that at least one phase will not operate successfully. The reliability of the mission may not in general be obtained by the simple multiplication of the individual phase reliabilities since the phases are statistically dependent. The PMS analysis is significantly more complex than a single phase mission.

Qualitative analysis takes care of the dependencies between the phases. It identifies all possible causes of the phase and mission failure in terms of the component failures (basic events), which are also identified by which phase the failure occurs in. Once the qualitative analysis has been done and the components' failure probabilities are known the quantitative analysis of the system can be done. It calculates the probability that the mission fails and hence what is the reliability of the system.

Previous research has solved the Phased Mission Problem by fault tree analysis, Markov analysis and simulation. Fault tree analysis is commonly used in industry for calculating the probability of system failure. This can be extended to solve systems consisting of more than one phase, where the failure logic will be different in the phases. Therefore the PMS problem is more complex than the single phase system. In the case where the assumption has been made that the components are not independent from each other, for example that the components can be repaired, then the Markov method is used. In some situations the system is too complex to model using deterministic analysis. In such circumstances, simulation is applied. The advantage of simulation is that it has a good representation of the system. However, the disadvantage is that it is computationally expensive.

## **1.6 Research Objective**

The aim of this project is to develop an efficient way of calculating the reliability of phased mission analysis. The application focus for this work is a UAV (unmanned aerial vehicle) mission and to perform analysis in the fastest time as possible in order for the results to be used in the decision making. The importance for this is that UAV have decision making algorithms that need to run in real time to decide whether to continue the mission or to change it. A reliability phased mission analysis of the rest of the mission or an alternative mission is important information in the decision-making algorithm. Therefore a phased mission reliability analysis must be performed in real-time. Mission reliability analysis is important for UAV's as opposed to manned aircraft since

they have a human doing the decisions making. Only the case of non-repairable components is considered throughout this research since the UAV is non-repairable during its mission. The objectives of this research are followed:

- Review all existing Phased mission analysis methods.
- Identify aspects of the analysis which could be performed faster. Since the objective is for the work to be applicable to a UAV mission advantage can be taken of features which are specific to this problem. The means identified to speed-up calculations will not necessarily generalise to other system.
- Improve the more efficient existing methods.
- Investigate into the efficiency of lookup search techniques for computations that have already been done before.
- Explore the effectiveness for taking out modules common to all phases in the mission.
- Develop a method for restructuring the phase fault trees in the aim of taking out an optimum number of modules.
- Apply the developed method to an aircraft system where mission is expressed in different phases.
- Apply the fastest PMS analysis method for a UAV obtained from this research to a situation where the mission is been reconfigured throughout.
- Explore ways of parallelizing the analysis so the computations can be shared between several processors. Therefore reducing the overall time of time of the analysis.



## **Chapter 2: Reliability Tools**

### **2.1 Introduction**

There are many methods that can be used to obtain the reliability performance of a system in term of reliability performance of the components of that system. This chapter discusses these methods and among the more common method is fault tree analysis.

### **2.2 Fault Tree Analysis**

#### **2.2.1 Introduction**

A Fault tree is a visual symbolic diagram that logically represents the causes of a particular system failure mode by the concept of a 'what can cause this' approach. Therefore a fault tree illustrates the events that cause the occurrence of the system failure and the causes of these events until the basic events are encountered.

#### **2.2.2 Construction of a fault tree**

Once a particular system failure mode is identified then the fault tree construction process starts with this failure mode as the top event. Branches then come off from below the top event which logically describes the combination of events that cause this. This development process is continues, until the basic events are encountered. The basic events are the lowest form of causes, failure probability data is required for these events for the fault tree to be analysed.

A fault tree is a diagram that contains two types of symbols, events and gates. There are two types of events, intermediate or basic events, which symbols are shown in table (2.1). The intermediate events can be expressed by other

events and the basic events cannot. The gate symbols have one branch coming off the top to the output event, and one or more coming off the bottom representing the cause. There are three main types of logic gates `AND`, `OR` and `NOT` which correspond to the Boolean operators `intersection`, `union` and `complementation`. The gates symbols are shown in table (2.2).

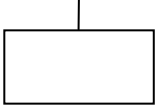

Event symbol	Meaning of symbol
	Top event or intermediate event description box. These events are further developed by a logic gate
	Basic event

Table 2.1: Event Symbols

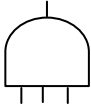
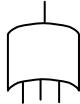
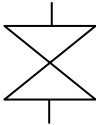
Gate symbol	Gate name	Relationship
	AND gate	Output event occurs if all input events occur simultaneously
	OR gate	Output event occurs if at least one input event occurs.
	NOT gate	Output event occurs if input event does not

Table 2.2: Gate Symbols

If the fault tree just contains `AND` and `OR` gates then it is known as a coherent system. On the other hand if the fault tree also contains `NOT` gates then it is known as a non-coherent system.

Once the fault tree is constructed then qualitative and quantitative analysis is performed. This is discussed in the next two sections.

### **2.3 Qualitative Analysis**

The goal of qualitative analysis is to obtain the combinations of the basic events that will cause the failure of the system. Each combination that causes system failure is known as a cut set, which is defined as follows:

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

However, a cut set may cause system failure without all of its basic events occurring. For example, A, B and C are basic events that make up a cut set, but the occurrence of A and B alone will cause system failure. Therefore the occurrence of C is irrelevant. It is only necessary to obtain the cut sets which do not contain any irrelevant basic events, these are referred to as minimal cut sets and are defined as:

- A minimal cut set is the smallest combination of basic events, such that if any basic event is removed from the set, the top event will not occur.

The order of minimal cut set is the number of basic event that it contains. Generally lower orders contribute more to system failure.

Cut sets and minimal cut sets are relevant for coherent systems. However, if a system is non-coherent then the failure cause may include working component state and the equivalent to the cut sets are known as implicants and minimal cut sets as prime implicants.

The two main methods for obtaining the minimal cut sets from a fault tree are 'top-down' and 'bottom-up' analysis. The 'top-down' method starts off with

the top event and then substitutes a Boolean logic expression that represents its causes. The process repeats itself by substituting the logic expressions for the cause of these events. This process terminates when the top event expression just contains basic events. Alternatively the 'bottom-top' method starts at the bottom of the fault tree and works upwards. The notation used in this logic expression is by connection '•' for 'AND' logic and '+' for 'OR' logic. By manipulation the top event expression is expressed in a sum-of-product form from which the minimal cut set can be extracted. To just obtain the minimal cut sets Boolean algebra laws such as the absorption law can be applied to remove redundancy.

## 2.4 Quantitative Analysis

The quantitative analysis of a fault tree is the evaluation of the parameters of the top event such as: probability of occurrence, frequency and expected number of occurrences.

### 2.4.1 Top Event Probability

The probability of the top event can be obtained by applying the inclusion-exclusion expansion to the minimal cut sets and then substituting probabilities of the basic events into it.

The probability that a minimal cut set  $C_i$  exists at time  $t$  is calculated by the product of the probabilities of the basic events required to exist at time  $t$ , this is shown in equation (2.4).

$$\Pr(C_i) = \prod_{C=1}^{N_{C_i}} q_C(t) \quad (2.4)$$

Where  $N_{C_i}$  is the total number of basic events in the minimal cut set and

$q_X(t)$  is the probability that basic event  $X$  exists at time  $t$ .

The top event exists at time  $t$  if one or more minimal cut sets exist at time  $t$ , this is expressed by the union of the minimal cut sets and is shown in equation (2.5) and denoted by  $Q_{sys}(t)$ .

$$Q_{sys}(t) = \Pr\left(\bigcup_{i=1}^{N_{mcs}} C_i\right) \quad (2.5)$$

Where  $N_{mcs}$  is the total number of minimal cut sets.

Equation (2.5) can be expanded by the inclusion-exclusion explanation as follows:

$$Q_{SYS}(t) = \sum_{i=1}^{N_{mcs}} \Pr(C_i) - \sum_{i=2}^{N_{mcs}} \sum_{j=1}^{i-1} \Pr(C_i \cap C_j) + \dots + (-1)^{N_{mcs}-1} \Pr(C_1 \cap C_2 \cap \dots \cap C_{N_{mcs}}) \quad (2.6)$$

This expansion transforms the calculation from computing the probability of one complex event to many probabilities of simpler events. However, systems may contain a large number of minimal cut sets in which case the expansion becomes unpractical because of the many calculations that have to be performed. To overcome this an approximation is required.

#### 2.4.2 Upper and lower Bounds for the existence of system failure

The first term in the expansion (2.6) will give an upper bound for the top event this is called the rare approximation. The first two terms will give a lower bound, shown in the inequality (2.7).

$$\sum_{i=1}^{N_{mcs}} \Pr(C_i) - \sum_{i=2}^{N_{mcs}} \sum_{j=1}^{i-1} \Pr(C_i \cap C_j) \leq Q_{sys}(t) \leq \sum_{i=1}^{N_{mcs}} \Pr(C_i) \quad (2.7)$$

Furthermore truncating the expansion at any odd number of terms will result in an upper bound and truncation at an even number of terms will result in a lower bound. If more terms are taken this will give a better approximation.

### 2.4.3 Minimal cut Upper Bound

A more accurate upper bound for the top event probability is the Minimal Cut Upper Bound. The concept for this bound is as follows:

$$\begin{aligned}\Pr(\text{system failure}) &= \Pr(\text{at least one minimal cut set exists}) \\ &= 1 - \Pr(\text{no minimal cut sets exist})\end{aligned}$$

And also,

$$\Pr(\text{no minimal cut sets exist}) \geq \prod_{i=1}^{N_{mcs}} \Pr(\text{minimal cut set } i \text{ does not exist})$$

Note, in the case when there is no common basic events in the minimal cut sets then the inequality becomes equality.

Therefore the complement is.

$$\Pr(\text{System failure}) \leq 1 - \prod_{i=1}^{N_{mcs}} \Pr(\text{minimal cut set } i \text{ does not exist})$$

Therefore the upper bound  $Q_{MCSU}$  is:

$$Q_{MCSU} = 1 - \prod_{i=1}^{N_{msc}} [1 - \Pr(C_i)] \quad (2.8)$$

## 2.5 Expected number of system failures

The expected number of failures for a system over the interval  $[t_0, t_1]$  is denoted by  $W_{sys}(t_0, t_1)$  and defined as:

$$W_{sys}(t_0, t_1) = \int_{t_0}^{t_1} w_{sys}(t) dt \quad (2.9)$$

Where  $w_{sys}(t)$  is the system unconditional failure intensity function, which is defined as the rate that the system failure occurs at time  $t$ . If the system is represented by a fault tree then the occurrence of the top event means that the system has failed. Therefore  $w_{sys}(t)dt$  is defined as the probability of the occurrence of the top event that occurs in the interval  $[t, t + dt)$ . This can be expressed as:

$$w_{sys} = \sum_{i=1}^n G_i(\underline{q}) \cdot w_i \quad (2.10)$$

where  $n$  is the total number of components in the system,  $w_i$  is the unconditional failure intensity for component  $i$  and  $G_i(\underline{q})$  is the Criticality function [1].

The Criticality function for component  $i$  is defined as the probability that the system is in a critical state with respect to component  $i$  thus the failure of component  $i$  will then cause the system to go from the working state to the failed state. The 2 common ways for obtaining the Criticality function are:

1. The sum of the probabilities of all the critical system states for component  $i$ . For example a system has 3 components A, B and C. The minimal cut sets for system failure are  $\{A\}$  and  $\{BC\}$ . The procedure of calculating the criticality function for component A is

shown in table 2.3 where the first column shows all the different combination of the states of the other components. The second column is the probability of the state and third identity if the particular state is critical for A.

State	Probability	Critical system state for component A
$\overline{B}\overline{C}$	$q_B(1 - q_C)$	Yes
$\overline{B}C$	$(1 - q_B)q_C$	Yes
$\overline{B}\overline{C}$	$(1 - q_B)(1 - q_C)$	Yes
$BC$	$q_Bq_C$	No

Table 2.3: Process for obtaining the probabilities of all the critical system states for component A.

Therefore the criticality function for component A is the sum of the Critical system state for component A, which is shown below:

$$\begin{aligned}
 G_A(\underline{q}) &= q_B(1 - q_C) + (1 - q_B)q_C + (1 - q_B)(1 - q_C) \\
 &= 1 - q_Bq_C
 \end{aligned}$$

2. The partial derivative of the top event probability with respect to the particular component failure probability which is given by:

$$G_i(\underline{q}) = \frac{\partial Q(\underline{q})}{\partial q_i} \quad (2.11)$$

For example considering the criticality function for component A as in the same system as the pervious example. First using the inclusion-exclusion explanation in equation (2.6) to obtain the top event probability which is:



$$Q = q_A + q_B q_C - q_A q_B q_C$$

The partial derivative with respect to  $q_A$  is

$$\frac{\partial Q}{\partial q_A} = 1 - q_B q_C$$

Alternative way of calculating partial derivative is by first principles, shown in equation (2.12).

$$\frac{\partial Q(\underline{q})}{\partial q_i} = \frac{Q(1_i, \underline{q}) - Q(0_i, \underline{q})}{1 - 0} \quad (2.12)$$

Therefore the criticality function for component i can be calculated by top event probability given that component i has definitely failure minus the top event probability given that component i definitely working. This expression is shown in equation (2.13).

$$G_i(\underline{q}) = Q(1_i, \underline{q}) - Q(0_i, \underline{q}) \quad (2.13)$$

## 2.6 Structure Functions

The System Structure Function represents the state of the system which is determined by the state of its components. A state of component X is represented by a binary indicator variable  $\phi_X$  as shown below:

$$\phi_X = \begin{cases} 1 & \text{if the component as failed} \\ 0 & \text{if the component is working} \end{cases} \quad (2.14)$$

The System Structure Function is also represented by a binary indicator variable shown be

$$\phi_{sys}(\underline{x}) = \begin{cases} 1 & \text{if the system as failed} \\ 0 & \text{if the system is working} \end{cases} \quad (2.15)$$

Where,

$$\underline{x} = (\phi_{X_1}, \phi_{X_2}, \dots, \phi_{X_n}) \quad (2.16)$$

The System Structure Function is expressed in terms of the component binary indicator variable as shown in equation (2.17).

$$\phi_{sys}(\underline{x}) = 1 - \prod_{i=1}^{N_{mcs}} (1 - \phi_{c_i}) \quad (2.17)$$

Where  $\phi_{c_i}$  is a binary indicator variable for the minimal cut set  $C_i$  and this is expressed as the product of the binary indicator variables for the components that are contained in the set, as shown in equation (2.18).

$$\phi_{c_i} = \prod_{X \in C_i}^{N_{c_i}} \phi_X \quad (2.18)$$

Where,

$$\phi_{c_i} = \begin{cases} 1 & \text{if the cut set exists} \\ 0 & \text{if the cut set does not exist} \end{cases} \quad (2.19)$$

The Expected value of System Structure Function is the probability of the top event as shown below.

$$\begin{aligned} E[\phi_{sys}(\underline{x})] &= \Pr(\phi_{sys}(\underline{x}) = 0) \cdot 0 + \Pr(\phi_{sys}(\underline{x}) = 1) \cdot 1 \\ &= \Pr(\phi_{sys}(\underline{x}) = 1) \\ &= Q_{sys} \end{aligned}$$

Therefore,

$$Q_{sys} = E[\phi_{sys}(\underline{x})] \quad (2.20)$$

If there are no common components between the minimal cut sets then this implies they are independent and the following is true:

$$E[\phi_{sys}(\underline{x})] = \phi_{sys}(E[\underline{x}]) \quad (2.21)$$

However in most system minimal cut sets will not be independent.

## 2.7 Binary Decision Diagrams

### 2.7.1 Introduction

A disadvantage with fault tree analysis is when the fault tree becomes large, especially when modelling PMSs, the qualitative analysis (minimal cut sets) and quantitative analysis (probability of the top event) performed on the fault

tree becomes very computationally intensive. A method to overcome this is to convert the fault tree into a Binary Decision Diagram (BDD) before the qualitative and quantitative analysis is performed. The BDD allows the efficient qualitative analysis and accurate quantitative analysis. Rauzy [3] was one of the first to consider using BDDs for reliability analysis and formalised the analysis process for fault trees.

### 2.7.2 Properties of the BDD

A BDD is a directed acyclic graph which consists of two types of nodes, non-terminal and terminal, which are linked by branches. The non-terminal nodes represent the basic events from the fault tree. The terminal nodes represent the state of the system, e.g. 1 the occurrence of the top event and 0 the non-occurrence of the top event. When considering reliability analysis the basic events will represent the components failures and the top event will represent system failure. The first node at the top of a BDD is a non-terminal node and is referred to as the root node. An example is shown in figure 2.2.

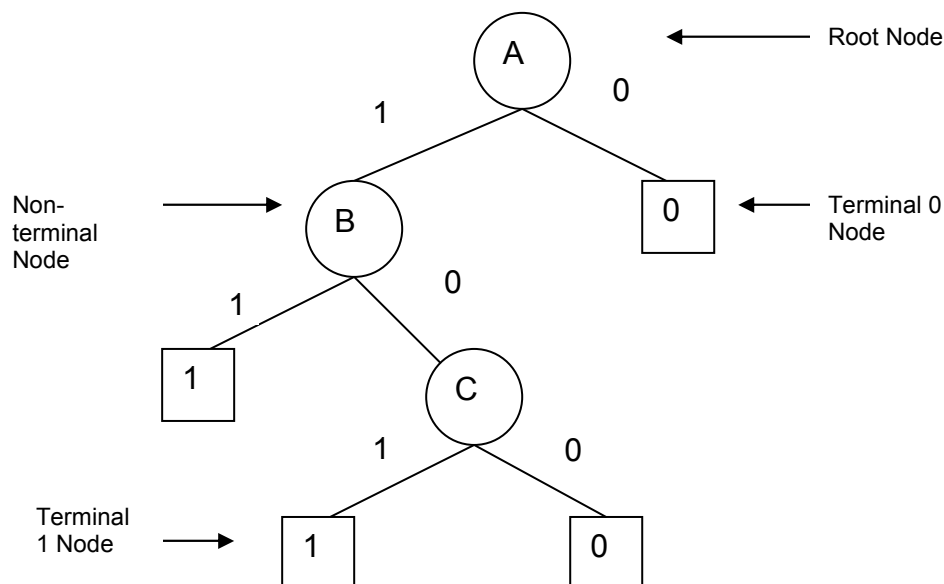


Figure 2.2: Example of BDD.

Every non-terminal node has two branches coming off the bottom of the node. The left branch with a one representing where the basic event in the node has occurred and the right branch with a zero representing that the basic event has not occurred see figure 2.2. The size of the BDD is defined by the number of non-terminal nodes.

The paths to a terminal 1 node can then be used to generate a cut set. For example the BDD in figure 1.2 contains 2 paths to a terminal 1,  $AB$  and  $A\bar{B}C$ . The cut sets can be obtain from these paths by ignoring the complements variables. The cut sets are therefore  $AB$  and  $AC$ .

There are many methods of constructing a BDD from a fault tree. The methods require an order to be placed on the basic events, this will determine the order of non-terminal nodes from top to bottom and affect the size of the BDD. Therefore this may be chosen to minimize the size of the BDD. This work was considered and extended by Bartlett and Andrews [4], [5].

### 2.7.3 Formation of a BDD using If-Then-Else Structure

Rauzy [3] developed a method for converting the fault tree into a BDD. The BDD construction is based on the concept of the if-then-else (ite) structure which is derived from Shannon's decomposition of a formula that can be written as shown in equation (2.22). Therefore the resulting BDD will encode this.

$$f(x) = Xf_1 + \bar{X}f_2 \quad (2.22)$$

where  $f(x)$  is a Boolean function that is being decomposed, this will be the top event of the fault tree. The Boolean function is being pivoted about variable  $X$ .  $f_1$  and  $f_2$  are the Boolean functions  $f(x)$  with  $X=1$  and  $X=0$  respectively.

Therefore the structure  $ite(X, f1, f2)$  represents a Boolean expression and is defined by “ if X occurs, then consider f1, else consider f2” where X is the pivoting basic event, and f1 and f2 are the Boolean expression with  $X=1$  (X has occurred) and  $X=0$  (X has not occurred) respectively.

In the BDD the variable X will be written inside the node, f1 and f2 will be below the 1 and 0 branches respectively, has shown in figure 2.3. f1 and f2 will also in turn have an ite structure and so will their sons and so on until the terminal 1 or 0 nodes are encountered.

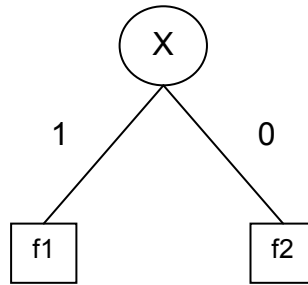


Figure 2.3: BDD node of  $ite(X, f1, f2)$

First step in the converting procedure is to assign every basic event to an ite form. In the second step all gates containing only basic events can be connected to an ite structure so if:

$$J = ite(X, f1, f2)$$

$$H = ite(Y, g1, g2)$$

are inputs to a gate of type \* (where \* can be either AND or OR ) then the output event is:

$$\text{If } X < Y \quad J * H = ite(X, f1 * H, f2 * H)$$

$$\text{If } X = Y \quad J * H = ite(X, f1 * g1, f2 * g2)$$

Where  $X < Y$  means that event  $X$  is considered before  $Y$  in the ordering.

The ite method is applied below to the fault tree in figure 2.4 with ordering  $A < B < C$ .

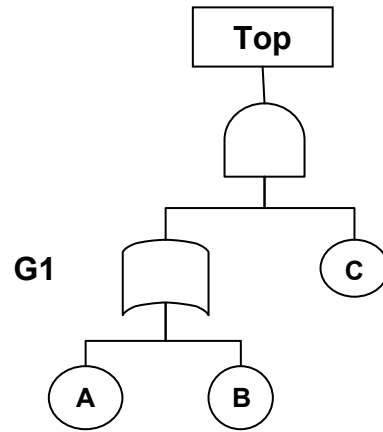


Figure 2.4: Example of a Fault tree

First all the basic events are given an ite structure.

$$A = ite(A, 1, 0)$$

$$B = ite(B, 1, 0)$$

$$C = ite(C, 1, 0)$$

Now choose a gate where all the inputs have already been given an ite structure, for the example G1. Now an ite structure is obtained for this gate by applying the operator of the gate type to all the gate inputs. Then the process is repeated until all gates are defined. The computation is as follows:

$$\begin{aligned} G1 &= A \cup B \\ &= ite(A, 1, 0) + ite(B, 1, 0) \\ &= ite(A, 1, ite(B, 1, 0)) \end{aligned}$$

Then moving up the tree structure to the top event:

$$\begin{aligned}
 Top &= G1 \cap C \\
 &= ite(A, 1, ite(B, 1, 0)) \cdot ite(C, 1, 0) \\
 &= ite(A, ite(C, 1, 0), ite(B, ite(C, 1, 0), 0))
 \end{aligned}$$

The top event ite structure holds all the information of the lay out of the resulting BDD (all connections of the nodes and branches). Therefore the resulting BDD is shown in figure 2.5. The ite method has the advantage that it does not need to know the minimal cut sets prior to calculating the top event probability. However, a disadvantage is that it does not encode the minimal cut sets directly. The next section discusses a technique that further transforms the BDD into a BDD that does just encodes the minimal cut sets.

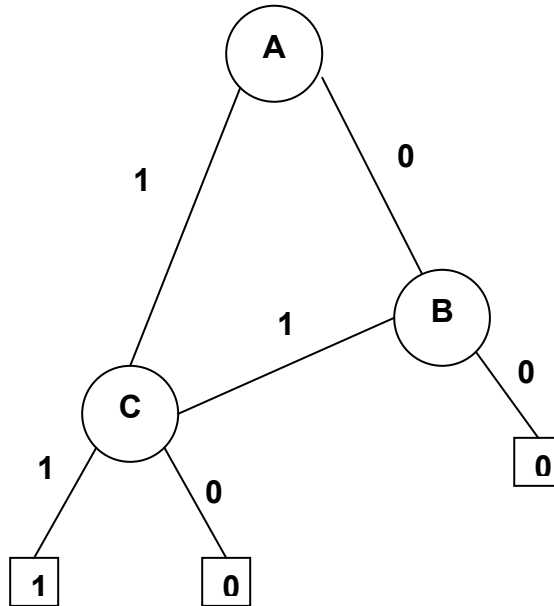


Figure 2.5: BDD constructed from fault tree in figure 2.3



#### 2.7.4 BDD Minimisation

If the BDD just produces minimal cut sets then the BDD is in a minimal form. For the majority of fault trees the BDD is not of this form. Rauzy [3] developed a method to create a minimized BDD from the initial one.

A general node with the ite structure in a BDD is:

$$F = ite(x, f1, f2) \quad (2.23)$$

If  $\delta$  is a minimal solution of  $f1$  and not a minimal solution of  $f2$  then  $x \cap \delta$  is a minimal solution of  $F$ . The minimal solutions of  $f2$  will also be a minimal solution of  $F$ . Therefore the set of all minimal solutions of  $F$  are denoted by  $sol_{\min}(F)$  and are expressed as follows.

$$sol_{\min}(F) = [\{\delta \cap x\}] \cup [sol_{\min}(H)]$$

Recursive application of this rule successfully computes a BDD which is minimized.

#### 2.7.5 Top event probability

To calculate the probability of the top event from the BDD first the paths through the BDD to a terminal 1 node are obtained. For example the BDD in figure 2.5 has two paths that terminate in a 1 which are:

- 1)  $AC$
- 2)  $\overline{A} BC$

If at least one of these paths occurs then the top event will occur. The probability of the top event occurring is the sum of the probabilities of these

paths occurring. Since all the paths through the BDD are disjoint. The calculation to obtain the top event probability for the example is as follows:

$$\begin{aligned}
 Q_{sys} &= \Pr( AC + \overline{A}BC ) \\
 &= \Pr( AC ) + \Pr( \overline{A}BC ) \\
 &= q_A q_C + (1 - q_A) q_B q_C
 \end{aligned}$$

In summary, all the previous parameters, Qualitative and Quantitative Analysis can be applied for single phase system. However, the nature of this project is Phased Mission System. The next section discusses how all the previous methods can alter to account for a Phased Mission System.

### **3 Phased Mission Systems**

#### **3.1 Introduction**

If a system must operate successfully over multiple, consecutive and non-overlapping periods (phases) then it is a Phased Mission System (PMSs). Many practical systems are PMSs, for example an aircraft flight mission that is divided into: taxiing to the runway, take-off, climbing to the correct altitude, cruising, descending, landing and taxiing back to the terminal. PMS analysis has received substantial attention over the last three decades. It is complicated as unlike usual system assessment the system failure does not always correspond with a component failure. A component may fail in any phase in the mission but may or not contribute to the failure of that phase. Each phase is identified by a phase index, phase length, success criteria, and failure parameters. A PMS may vary in configuration, success criteria, and component behaviour in different phases.

The PMS reliability is defined as the probability that the system operates successfully in all of its phases. The PMS unreliability is defined as the probability that at least one phase will not operate successfully. The reliability of the mission may not in general be obtained by the simple multiplication of the individual phase reliabilities since the phases are statistically dependent; if this situation is true, then every component in every phase must be working at the beginning of the phase and the phases must have no component in common. Hence PMS analysis is much more complex than a single phase mission. Special techniques must be obtained to take care of the dependencies between the phases. To find out how the system fails, which is referred to as the failure modes, a method is required to express the system failure in terms of the components' failure (cut sets): this is called qualitative analysis. Once the components' failure probabilities are known and the qualitative analysis has been done, the system failure probability may be calculated and hence the reliability of the system: this is called quantitative analysis of the system.

PMS can be divided into two categories - appropriate for either non-repairable or repairable systems. Once a component has failed in the non-repairable system it will remain in the failed state for the rest of the mission. Once a component has failed in the repairable system it is possible that the component may be restored back into the working state during the mission. The following section discusses techniques that have been found appropriate for non-repairable systems.

## 3.2 Non-Repairable System

### 3.2.1 Introduction

Esary and Ziehms [6] were the first to consider the analysis of PMSs. Each phase configuration is represented by a reliability block diagram or by a fault tree. The components perform independently of each other, and each of them may be in one of two states, functioning or failed. Since the system is non-repairable, once component failure has occurred it will stay in the failed state for the rest of the mission. Esary and Ziehms presented a method which transforms and simplifies the PMS into an equivalent single phase system. This method is discussed in the next section.

If the Multi-Phased Mission can be transformed into a single phase then existing methods for single phase mission analysis can be used.

Let  $X_i$  denote the event that the component  $X$  is in the failed state at the end of the phase  $i$  which implies that the component failure occurred in phase  $i$  or in one of the previous phases e.g.  $X_i = 1$  means that this event is true. Conversely,  $\bar{X}_i$  denotes the event that the component  $X$  is in the working state at the end of the phase  $i$  which implies that the component has worked through out phase  $i$  and all of it previous phases. One of the disadvantages of this Boolean variable is that it does not indicate where the failure occurs. Therefore let  $x_i$  denote the event that component  $X$  failure occur in phase  $i$ .

The relationship of these two variables is that the failed state of the component can be replaced by the sum (where '+' means logical OR) of components failures occurring in phase i and all of the previous phases. The equivalent in fault tree PMS analysis is replacing the basic event  $X_i$  by the gate shown in figure 3.1.

$$X_i = x_1 + x_2 + ..... + x_i \quad (3.1)$$

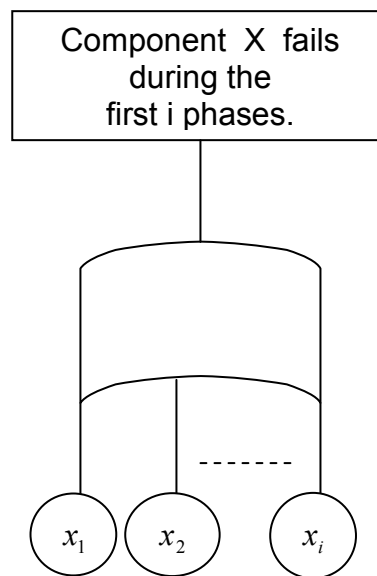


Figure 3.1: Component phase failure representation.

From now on in the review we will just consider the mission failure causes expressed using fault trees. Consider an example phased mission system.

The failure criteria of each phase, represented by the fault trees given by La Band and Andrews [13] are shown in figure 3.2. The procedure of transformation consists of a two step process:

- 1) Replace every component in the fault trees by the gate structure shown in figure 3.1.

2) Join every phase failure fault tree by an OR gate to represent the cause of mission failure.

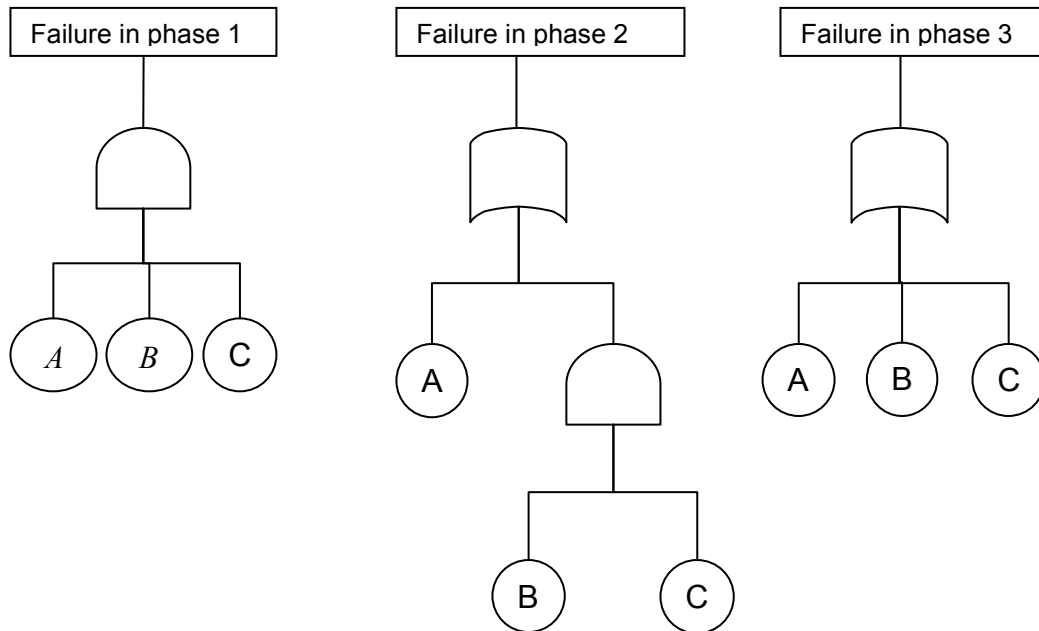


Figure 3.2: Example Phase Failure conditions.

Applying the transformation procedure the multi-phase mission failure cause becomes the single equivalent fault tree shown in figure 3.3.

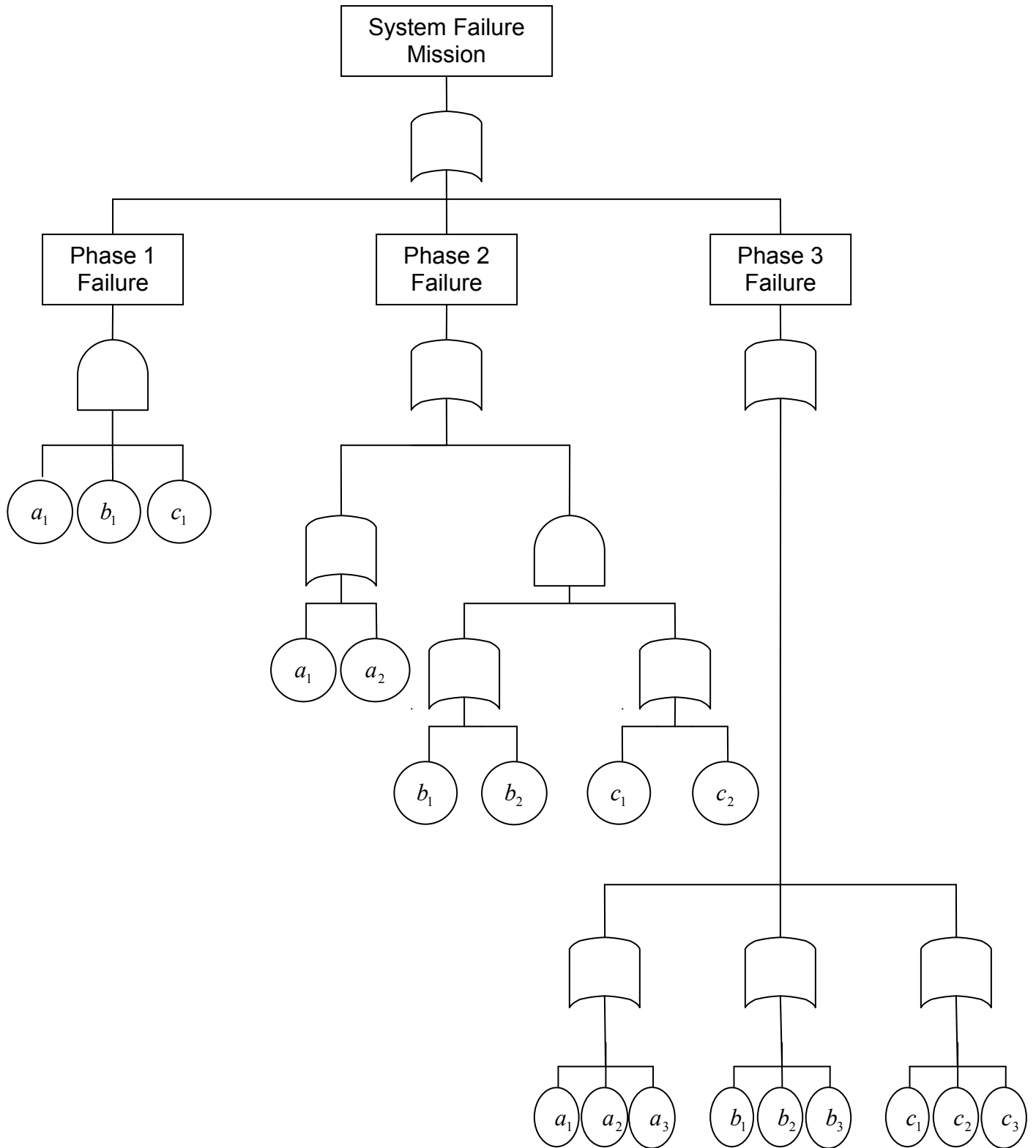


Figure 3.3: Equivalent Fault tree for the mission failure causes.

The disadvantage of the procedure shown above is that each component in the system is expanded to many variables to represent its failure in each phase. The problem then gets computationally intensive as the number of variables becomes very large. Methods can be used to reduce the number of components of the PMS. One of the popular methods is discussed in the following section.

### 3.2.2 The Cut Set Cancellation Method

Several methods can be used to simplify the phase configuration before the transformation procedure. One method was used by Esary and Ziehms [6], is called cut set cancellation.

Cut set cancellation works such that if a minimal cut set of an earlier phase contains any minimal cut set from a later phase, the minimal cut set may be removed from the earlier phase. For example consider the phases which are illustrated by the fault trees in figure 3.2.

The minimal cut sets in each phase are:

$\{A, B, C\}$	phase 1
$\{A\}, \{BC\}$	phase 2
$\{A\}, \{B\}, \{C\}$	phase 3

$\{A, B, C\}$  can be removed from phase 1 since each individual component occurs as a minimal cut set in phase 3. For the same reason  $\{A\}$  in phase 2 can be removed. The minimal cut set  $\{B, C\}$  can also be removed since  $\{B\}$  and  $\{C\}$  occur in phase 3. The cancellation leaves the PMS in a concise form of only the phase 3 cut sets. Hence the fault tree evaluation will be much simpler. In this example the step 2 of the transformation is not even needed since one single phase is left. The equivalent single fault tree is shown in figure 3.4. Therefore the entire system will fail if any of the events  $a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3$  occur. The probability of this event is the unreliability



of the entire PMS and can be calculated by the inclusion-exclusion expansion from equation (2.6).

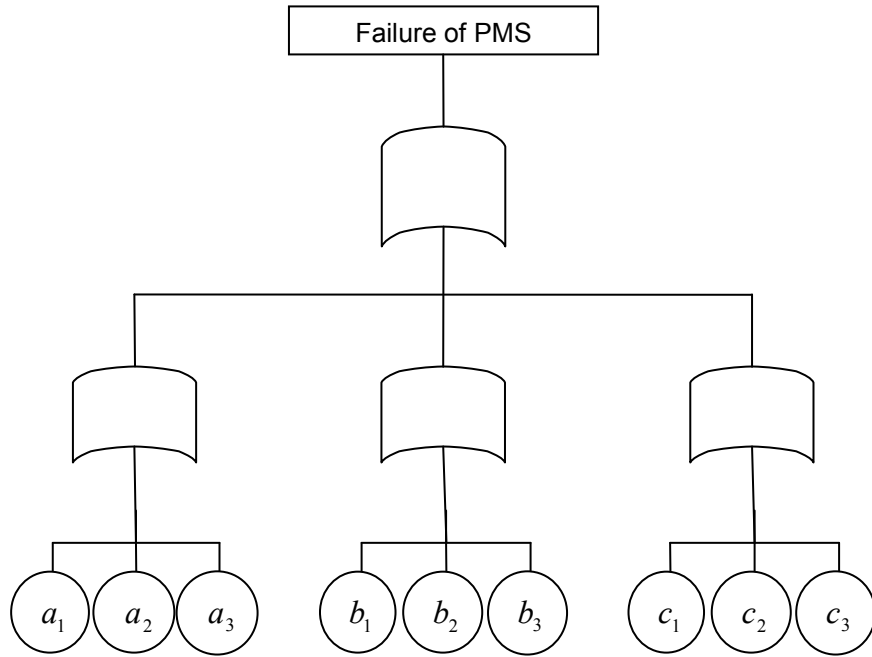


Figure 3.4: Equivalent single phase fault tree.

Esary and Ziehms [6] state and prove that cut cancellation does not affect mission reliability.

In summary, the method presented by Esary and Ziehms is very useful in transforming the multi-phased mission into an equivalent single phase, which enables existing techniques to be used to calculate the entire mission reliability. Cut set cancellation simplifies the quantitative analysis of the entire mission by removing irrelevant minimal cut sets. The disadvantage of cut set cancellation is that only the unreliability of the entire mission can be calculated, and not the unreliability of each of the phases. If after simplification the phased mission analysis is still very large e.g. a large number of variables, then it might be necessary to use an approximation. The following discusses approximation methods for PMS.

### 3.2.3 PMS Unreliability Approximation

Esary and Ziehms [6] presented a method for approximating mission reliability by calculating the reliability of every individual phase and then multiplying them together. There are two choices of component reliabilities to use, conditional or unconditional. It was proved that, if the conditional component reliabilities are used, the method gives an upper bound for mission reliability and unconditional component reliabilities gives a lower bound for mission reliability.

The conditional phase reliability for a given component is the probability that the component failure occurs in the phase given that the component was working at the beginning of the phase. The unconditional phase reliability for a given component is just the probability that the component failure occurs in the phase.

Burdick et al [7] developed and reviewed Esary and Ziehms' methods; four methods for unreliability mission approximation were found:

#### A) The INEX method

This approximation is denoted by  $Q_{MISSINEX}$  and the procedure of calculating it is described in the following steps:

- 1) From the fault tree or an appropriate logic model of each phase obtain the minimal cut sets for each phase.
- 2) Calculate the unreliability of each phase  $Q_i$  by using the inclusion – exclusion expansion.
- 3) The inclusion-exclusion expansion is used again on the set of phase unreliabilities  $Q_j$  where  $j = 1, \dots, n$  and  $n$  is the total number of phases. The expansion will give successive upper and lower bounds.

Usually this will be approximated by the rare event upper bound, or any other upper bound order of the expansion.

$$Q_{MISSINEX} = \sum_{j=1}^n Q_j \quad (3.2)$$

#### B) The INEX-CC method

This approximation is denoted by  $Q_{MISSINEX-CC}$ .

The procedure is very similar to method A the only difference is an additional step between stages 1 and 2. The extra step is the cut-set cancellation which is described in section 2.2.2. The approximation will be generally less than the method in A since there are fewer cut-sets.

#### C) The MCB method

This approximation is denoted by  $Q_{MISSMCB}$  and the procedure of calculation is described in the following steps:

- 1) From the fault tree or an appropriate logic model of each phase obtain the minimal cut sets for each phase.
- 2) Let  $MCS_{ij}$  denote the  $i$ th minimal cut-set in phase  $j$ . Calculate the probability of the minimal cut set by:

$$P(MCS_{ij}) = \prod_{l=1}^{l=n_{ij}} P(x_l) \quad (3.3)$$

Where  $n_{ij}$  is the total number of basic event in the  $i$ th minimal cut-set of the  $j$ th phase and  $x_l$  for  $l = 1, \dots, n_{ij}$  are the basic events of the minimal cut-set.

3) Calculate the individual phase unreliabilities  $Q_j$  estimate by

$$Q_j = 1 - \prod_{i=1}^{k_j} (1 - P(MCS_{ij})) \quad (3.4)$$

Where  $k_j$  is the total number of minimal cut sets in phase  $j$ .

4)  $Q_{MISSMCB}$  is calculated in the same way as in step 3 of the INEX method.

#### D) The MCB-CC method

This approximation is denoted by  $Q_{MISSMCB-CC}$ .

This procedure is very similar to method C the only difference is an additional step between stages 1 and 2. The extra step is the cut-set cancellation which is described in section 2.2.

Burdick et al [7] reviewed that the ordering of the bounds are as follows:

$$Q_{MISS} \leq Q_{MISSINEXCC} \leq \frac{Q_{MISSMCBCC}}{Q_{MISSINEX}} \leq Q_{MISSMCB} \quad (3.5)$$

Where  $Q_{MISS}$  is the exact mission unreliability.

All of these methods are very useful for PMS with a large number of components and phases since otherwise the calculations would become too computationally intensive.

### 3.2.4 Individual Phase Unreliability

#### 3.2.4.1 Introduction

One of the disadvantages of calculating the unreliability by the Esary and Ziehms method shown in section 3.2.1 is that it does not calculate the unreliability of the individual phases. La Band and Andrews [13] present a method that calculates the unreliability of the individual phases as well as the unreliability of the entire mission.

The method used to obtain the unreliability of any phase  $i$  uses a fault tree structure which combines the causes of success in previous phases  $i-1, \dots, 1$  with the causes of failure of phase  $i$ . Therefore the causes of system unreliability in phase  $i$  denoted by  $Q_i$  are represented by the AND of the success of phases  $1..i-1$  and the failure in phase  $i$  which is illustrated by the fault tree in figure 3.5.

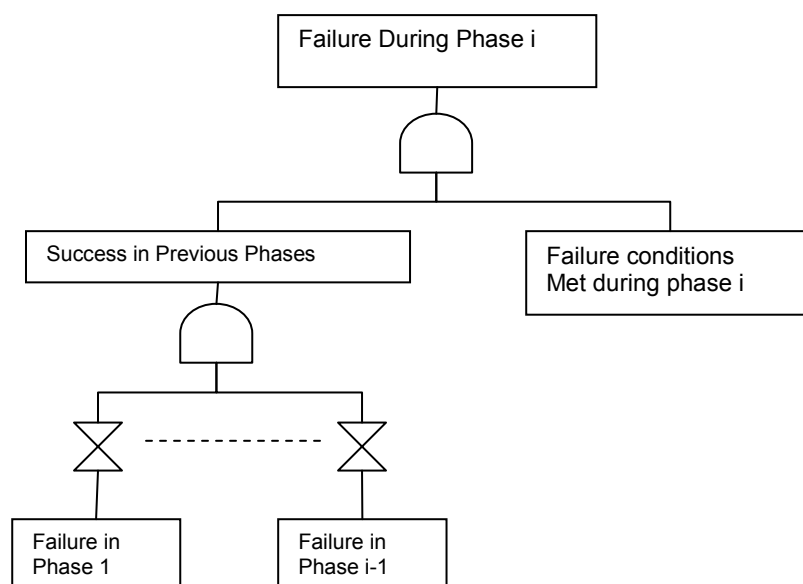


Figure 3.5: Fault Tree for failure during phase  $i$ .

Once all the Phase unreliabilities have been calculated then the entire mission unreliability denoted by  $Q_{MISS}$  can be obtained using:

$$Q_{MISS} = \sum_{i=1}^n Q_i \quad (3.6)$$

Where n is the total number of phases.

Previously the minimal cut sets of the fault tree have been obtained by either a top-down or a bottom-up method. In this case there were no NOT gates in the tree which makes the fault tree coherent. Now the combinations of basic event working and failed states that lead to the occurrence of the top event are not referred to as minimal cut sets, instead they are referred to as the Prime Implicants.

This fault tree requires significantly more effort to solve, since Cut set Cancellation cannot be used. The next section discusses techniques which can be used to reduce the problem complexity.

### **3.2.4.2 Fault Tree Restructuring Techniques**

Before the Prime Implicants are obtained fault tree restructuring techniques can reduce the size of the fault tree and therefore obtain the Prime Implicants more efficiently using less memory and time requirements.

First the not logic is pushed down the fault tree using De Morgans's laws so the not logic applies to the basic events.

There are three stages of this technique: contraction, factorisation and extraction. Each is described in the following sections.

## Contraction

Subsequent gates of the same type in the fault tree are contracted to form a single gate. This will result in alternating sequence of AND and OR gates.

## Factorisation

Replacing the basic events that always occur together in the same gate type by a complex event, which is usually represented by a unique number greater or equal to 2000. However, since NOT gates are included in the fault tree to be taken out as a factor then the event has to occur such that either of the following occur in the fault tree.

$$2000 = A + B \quad \overline{2000} = \overline{A} \cdot \overline{B} \quad (3.7)$$

$$2001 = A \cdot B \quad \overline{2001} = \overline{A} + \overline{B} \quad (3.8)$$

## Extraction

If the fault tree contains a certain structure as shown in figure 3.6 a and b it may be replaced by simpler structure.

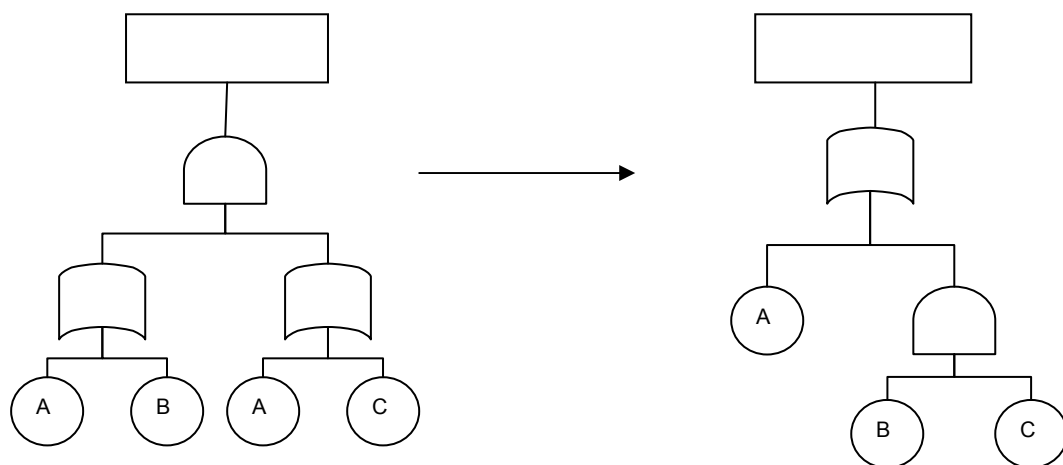


Figure 3.6.a: Extraction Operation

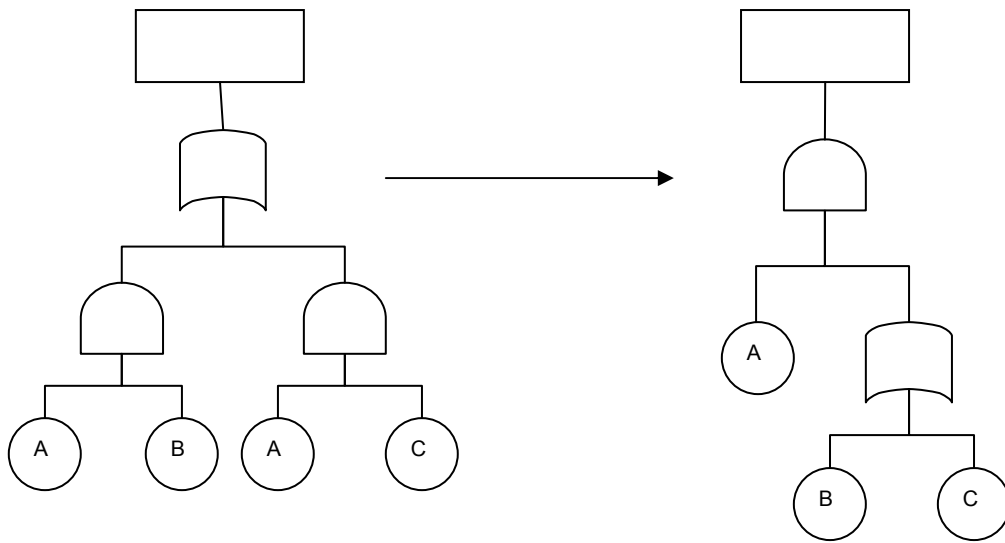


Figure 3.6.b: Extraction Operation

### 3.2.4.3 Obtain the Prime Implicants PMS

Due to the non-repairable nature of the component failure event a, new algebra is used to obtain the minimal cut sets or prime implicants for phased mission [13].

A summary of the new algebraic laws are:

$$x_i x_i = x_i$$

$$x_i x_j = 0$$

$$x_i x_{i,j} = x_i$$

$$\bar{x}_i x_i = 0$$

$$\bar{x}_i \bar{x}_{i+1} \dots \bar{x}_j = \bar{x}_{i,j}$$

$$x_i + x_{i+1} + \dots + x_j = x_{i,j} \quad (3.9)$$



Where  $x_{i,j}$  is used to denote the event that component failure occurs in phases i to j and  $\overline{x_{i,j}}$  is that the event X does not occur throughout phases i to j, where  $j > i$ .

Once the Prime Implicants have been obtained from the fault tree by using the algebraic Boolean laws, then the unreliability can be calculated by substituting the Prime Implicants in the inclusion-expansion.

In summary this method of calculating the unreliability of a individual phase i consists of the follow steps:

- 1) construct the fault tree which is a combination success phases 1..i-1 and the failure of phase i.
- 2) reduce the size of the fault tree by applying contraction, factorisation and Extraction techniques.
- 3) Obtain the Prime Implicants by using the old and new Boolean algebraic laws.
- 4) Use the inclusion-exclusion and the component failure probabilities to calculate the phase unreliability.

Once the unreliability has been calculated for every phase then the unreliability of the entire mission is obtain by the sum of these unreliabilities.

### **3.2.5 Laws of Boolean Phase Algebra**

This section discusses methods for obtaining the unreliability of a PMS by using Boolean algebra. Dazhi and Xiaozhong [9] present method which expresses the combinations of components that exist in the failed state in different phases to fail the mission. The method is based on the concept of

component failure existence, instead of occurrence that was used in previous methods. Therefore new algebra laws are developed to account for this.

The intersection and union of the Boolean events  $X_i$  and  $X_j$  were considered, where  $i$  and  $j$  are general phases with the order  $j \geq i \geq 1$ . The concept of equation (3.10) is used and simplified as shown below:

$$\begin{aligned}
 X_j &= x_1 + x_2 + \dots + x_j \\
 &= \bigcup_{k=1}^i x_k + \bigcup_{k=i+1}^j x_k \\
 &= X_i + \bigcup_{k=i+1}^j x_k \quad (3.10)
 \end{aligned}$$

Expression (3.10) is also used to simplify the intersection of two events as shown below:

$$\begin{aligned}
 X_i X_j &= X_i \left( X_i + \bigcup_{k=i+1}^j x_k \right) \\
 &= X_i + \bigcup_{k=i+1}^j (X_i x_k) \\
 &= X_i \quad (3.11)
 \end{aligned}$$

For the union of two events:

$$\begin{aligned}
 X_i + X_j &= X_i + \left( X_i + \bigcup_{k=1+i}^j x_k \right) \\
 &= X_i + \bigcup_{k=1+i}^j x_k \\
 &= X_j
 \end{aligned} \tag{3.12}$$

Similarly to the component failure, if the system is in the failed state at phase j then the phase failure could have occurred in any phases 1.....j. This is expressed in equation (3.13).

$$P_j = p_1 + p_2 + ..... + p_j \tag{3.13}$$

Where  $P_j$  this the event that the system is in the failed state in phase j and  $p_i$  is the event that the system failure occurs in phase i, which can be obtain by

$$p_j = \bigcup_{k=1}^{N_{mscj}} C_{j,k} \tag{3.14}$$

Where  $C_{j,k}$  is the event that the kth cut set exist in phase j and  $N_{mscj}$  is the total number of cut sets in phase j.

The mission unreliability can be obtained by putting the equations (3.13) and (3.14) together and then calculating the probability of this expression.

$$\begin{aligned}
Q_{miss} &= \Pr(P_n) \\
&= \Pr\left(\bigcup_{j=1}^n p_j\right) = \Pr\left(\bigcup_{j=1}^n \left(\bigcup_{k=1}^{N_{mscj}} C_{j,k}\right)\right) \quad (3.15)
\end{aligned}$$

The cut sets cancellation technique (in section 3.2.2) is automatically implemented in equation (3.15). Dazhi and Xiaozhong applied the Boolean algebra and expressions described above to compute the probability of accident sequences.

Obtaining the unreliability of a PMS by equation (3.15) and applying the Boolean laws to simplify the algebra avoids the need in the Esary and Ziehms method, for the failure of a component in different phases to be represented separately as different basic events and also converting the mission into a single phase system. Therefore Dazhi and Xiaozhong method is less computationally intensive compared to Esary and Ziehms.

This work is extended by Kohda et al [10] which obtain further Boolean laws by using the minimal cut set and path sets of each phase.

Somani and Trivedi [11] present a method for PMS unreliability analysis which is also based on Boolean algebraic methods. The unreliability calculation is exact and computationally efficient.

Instead of transforming the PMS into an equivalent single phase mission where the fault tree suffers from a large number of variables, as occurs in the Esary and Ziehms method. The Somani and Trivedi method solved each phase fault tree individually. However since the phases are dependent there must be information passed from phase to phase to account for this.

One of the main concepts used in this method is the cumulative distribution functions with a mass at the origin, which is used for the components failure probability. A component  $X$  has a cumulative distribution function at time  $t$  given by

$$q_X(t) = (1 - e^{-\lambda T_1}) + e^{-\lambda T_1} (1 - e^{-\lambda t}) \quad (3.16)$$

Where  $T_1$  is the time at the start of the phase and  $\lambda$  is the failure rate.

This function has a mass at the origin given by  $(1 - e^{-\lambda T_1})$  which is the probability that the component is in failed state at the start of the phase, the second term of the function represents the continuous part of the distribution function which is the probability that the component is in the failed state time  $t$  AND that the component has been working up to time  $T_1$ .

Somani and Trivedi [11] considered phase- independent failure criteria. This means that the phases have the same system configuration and failure criteria, but the components failure rate may change from phase to phase. Hence only the components failure combinations are obtained from the fault tree for the last phase. Three situations were considered for this problem corresponding to three different versions of component failure distributions with mass at the origin – phase dependent failure rate, age dependent failure rate, and random phase durations. Further a method was presented for phase-dependent failure criteria which is more complex than the previous situation since the system configuration and failure criteria varies from phase to phase. A combination of components failure may exist before the relevant phase that it is contained in the failure criteria, which means all of the individual components failure have occurred in an earlier phase, this would result in an instant system failure at the start of the relevant phase this is referred to as latent failure. There are four possible cases for a combination of components failure across a phase boundary that could affect the state of the system which are as follows.

1. A combination of component failures does not lead to system failure in both phases  $i$  and  $i+1$ .
2. A combination of component failures leads to system failure in both phases  $i$  and  $i+1$ .
3. A combination of component failures does not lead to system failure in phase  $i$  but leads to system failure in phase  $i+1$ .
4. A combination of component failures leads to system failure in phase  $i$  but not in phase  $i+1$ .

In the first two cases the failure criteria do not change with respect to the failure combination from phase  $i$  to  $i+1$ . The third case can be treated as failure of the system if the combination occurs in either phase, since if the combination occurs in phase  $i$  then it will exist in phase  $i+1$  which will result in a failure at the transition point. The unreliability of these three cases can be evaluated by just quantifying the fault tree for phase  $i+1$ , this is the same approach as the phase-Independent failure criteria situation. The fourth case is more complex. A method is presented to account for the fourth case. The failure combinations are divided into two categories – common failure combinations and phase failure combination.

### **Common Failure Combinations**

The common failure combinations are the component failure combinations which are common to all of the phase after the stringent criterion as been applied. Therefore the combinations are obtained from the last phase in the mission.

The unreliability for common failure combinations are computed by obtaining the failure distribution for each component in the combination and solved the fault tree for the last phase. This is the same method as the case phase independent failure criteria.

## Phase Failure Combination

The phase failure combination for phase i which is the component failure combinations which contribute to system failure in phase i but not contribute to system failure in any phases after i. Phase failure combination for phase j ( $PFC_j$ ) is expression:

$$PFC_j = (..((E_j \cap \bar{E}_{j+1}) \cap \bar{E}_{j+2}) \cap \bar{E}_{j+3}).... \cap \bar{E}_n) \quad (3.17)$$

Where  $E_j$  is a Boolean expression that represents the failure combinations of phase j. Equation (3.17) may be simplified by using De Morgans's law that results in:

$$PFC_j = \bar{E}_j (\bar{E}_{j+1} \cup ..... \cup \bar{E}_n) \quad (3.18)$$

The Boolean variable used for the components existing in the failure state in the jth phase, which will be in the Boolean expression for  $E_j$ , is the same has the equation (3.1). In addition to this the Boolean variable representing a component existing in the working state in the jth phase shown in equation (3.19) will be needed, since  $\bar{E}_j$  will need to be obtained:

$$\bar{X}_j = \bar{x}_1 \bar{x}_2 ..... \bar{x}_j \quad (3.19)$$

Since the  $PFC_j$  are in terms of  $X_j, \bar{X}_j$  for the relevant components. Algebra rules are required to simplify  $PFC_j$  for each phase. The Boolean laws which are used in the simplification:

$$1. \bar{X}_i \bar{X}_j \rightarrow \bar{X}_j$$

$$2. X_i X_j \rightarrow X_i$$

$$3. X_i \bar{X}_j \rightarrow 0$$

$$4. \bar{X}_i + \bar{X}_j \rightarrow \bar{X}_i$$

$$5. X_i + X_j \rightarrow X_j$$

$$6. \bar{X}_i + X_j \rightarrow 1 \quad (3.20)$$

The sixth law shows the deficiency, since from the Boolean law of complementation, an event OR its complement become true. Hence the correct expression would have been  $\bar{X}_i + X_i \rightarrow 1$ , which is not the same as rule six. Hence approximate results will be obtained by the application of this simplification technique.

The expressions  $\bar{X}_i X_j$  and  $X_i + \bar{X}_j$  cannot be simplify any further,  $\bar{X}_i X_j$  is the event that the component X is in the working state at the end of the phase i, and then fails in phase j. The expression  $X_i + \bar{X}_j$  has no physical meaning. The probability of these events are more complex than the other but can be obtained by:

$$\begin{aligned} P(\bar{X}_i X_j = 1) &= E[\bar{X}_i X_j] = E[\bar{X}_i (1 - \bar{X}_j)] \\ &= E[\bar{X}_i] - E[\bar{X}_i \bar{X}_j] = P(\bar{X}_i = 1) - P(\bar{X}_j = 1) \end{aligned} \quad (3.21)$$



$$P(X_i + \bar{X}_j) = P(X_i = 1) + P(\bar{X}_j = 1) \quad (3.22)$$

(since  $X_i$  and  $\bar{X}_j$  are disjoint)

The system unreliability is obtained by first computing the PFC for every phase then is given by:

$$Q_{MISS} = \Pr(E_n) + \sum_{j=1}^{n-1} \Pr(PFC_j) \quad (3.23)$$

Where  $n$  is the total number of phases and  $\Pr(E_n)$  is the probability of the top failure event of the fault tree for phase  $n$ .

Somani and Trivedi [11] also obtain an expression for the unreliability at the end of the  $k$ th phase which is given by:

$$Q_k = \sum_{j=1}^k \Pr(PFC_{j,k}) \quad (3.24)$$

Where  $PFC_{j,k}$  is very similar to  $PFC_j$  the only different is that the phase  $k$  is treated as the last phase in the expression and any phase after  $k$  will be ignored. It can be obtained by:

$$PFC_{j,k} = PFC_{j,k-1} \cap \bar{E}_k \quad (3.25)$$

Somani and Trivedi identifies the jump in unreliability between the phases which is due to the different failure criteria in the next phase. Somani and Trevedi demonstrated this method by an example of a PMS of 3 components

and 3 phases. It was assumed that the phases could occur in any order. Therefore all permutations were calculated.

Ma and Trivedi [12] extended this work by computing the mission unreliability in the form of disjoint products. The algorithm was implemented in the SHARPE software package.

### 3.2.6 Binary Decision Diagrams for Phased Mission Systems

Converting a single fault tree into a BDD was demonstrated in section (2.7). Computing the unreliability of a BDD is very computationally efficient, which is useful when dealing with large systems. BDD transformation can also be applied to PMS, which is useful since PMS do become large. This section reviews methods that convert PMS, phase failure conditions are generally represented by fault trees, into a single BDD and then calculates the unreliability from the BDD.

Trivedi et al [14] present a method that converts multiple phase missions into a single BDD. A component is in the failed state in a phase if the failure occurred in the phase or any of the previous phases as described in section 3.2.1 and illustrate in figure 3.1.

The function  $q_{xj}(t)$  is defined as the probability that the component X is in the failed state at t given that it was in the working state at the beginning of the phase. This function is used to define the failure cumulative distribution function  $F_{xj}(t)$  which is:

$$F_{xj}(t) = \left[ 1 - \prod_{i=1}^{j-1} [1 - q_{xi}(T_i)] \right] + \left[ \prod_{i=1}^{j-1} [1 - q_{xi}(T_i)] \right] \cdot q_{xj}(t) \quad (3.26)$$

Where  $T_i$  is the duration of phase i and t is the time measured from the start of the phase j,  $0 \leq t \leq T_j$ .

The first term in the function is the probability that the component X failed in any of the phases 1.....j-1. The second term is the probability that the component is functioning until phase j-1 and then fails in phase j.

As in the original method for constructing a BBD (shown in section (2.7)) an order for the variables is required. Trivedi et al chooses two methods for ordering the basic events. These correspond to two different operators. The orders are as follows.

- 1) Forwards Phase-Dependent Operation (PDO) is defined as any order in which the basic events that belong to the same component stay together and then are ordered as the same order as the phases, for example:

$$x_1 < x_2 < \dots < x_n \quad (3.27)$$

- 2) Backwards Phase-Dependent Operation (PDO) is similar to the previous order, the difference is that the order of the basic events that belong to the same component are in the reverse order of the phases, for example:

$$x_1 > x_2 > \dots > x_n \quad (3.28)$$

A new operator for its structure is used which deals with the case of basic events belonging to the same component. This operator is demonstrated below.  $E_i$  and  $E_j$  represent two nodes of its structure, which are:

$$E_i = ite(x_i, G_1, G_2) \quad (3.29)$$

$$E_j = ite(x_j, H_1, H_2) \quad (3.30)$$

Where  $i$  and  $j$  are general phases with order  $i < j$ ,  $G_1$ ,  $G_2$ ,  $H_1$  and  $H_2$  are also  $ite$  structure coming off the branches.

The binary operator  $*$  is applied to  $E_i$  and  $E_j$  and the result is as follows:

Forwards PDO :

$$E_i * E_j = ite(x_i, G_1, G_2) * ite(x_j, H_1, H_2) = ite(x_i, G_1 * H_1, G_2 * E_j) \quad (3.31)$$

Backwards PDO:

$$E_i * E_j = ite(x_i, G_1, G_2) * ite(x_j, H_1, H_2) = ite(x_j, E_i * H_1, G_2 * H_2) \quad (3.32)$$

The ordering of the basic events is important, since the size of the BDD will be affected by the order. The backwards PDO will generally generate a smaller BDD than the forwards PDO.

An algorithm follows, which converts a logic model of a PMS, which is represented by a set of fault trees, into a single BDD.

1. Use equation (3.26) to obtain the value for each variable.
2. Order components and their corresponding variables using the heuristic method.
3. Generate the BDD for each phase using ordinary logical operations.
4. Use phase-algebra and the corresponding backward PDO to combine these BDD to obtain the final BDD from the BDD of each phase.
5. Calculate the unreliability of PMS (using an evaluation algorithm) from the final BDD.

Once the BDD is constructed (step 4) by the backwards ordering methods the branches will either connect variables (nodes) that belong to different or the same component. The `0` branches will always connect variables that belong to different component since the PDO has cancelled this type of common component connection. However, the `1` branches may connect variables that belong to different or the same component. This is all taken into account when evaluating the BDD.

Consider a general node in the BDD.

$$G = ite(X_j, G_1, G_2) = X_j \cdot G_1 + \bar{X}_j \cdot G_2 \quad (3.33)$$

$G_2$  will not contain any variables that belong to  $X$  since the `0` branch will always connect variables that belong to different components. Therefore the events  $X_j$  and  $G_2$  are always statistically independent. This implies that

$$\Pr(X_j \cdot G_2 = 1) = \Pr(X_j = 1) \cdot \Pr(G_2 = 1) \quad (3.34)$$

There are two cases which are considered for computing the probability of  $G$  they are as follows:

1. When the `1` branch connects variables of different components, which means that  $G_1$  does not contain any variables that belong to  $X$ .
2. When the `1` branch connects variables of the same component, which means that  $G_1$  does contain a variable that belongs to  $X$ .

The method for evaluating case 1 is the same as for evaluating BDD for a single phase. The calculation is as follows:

$$\Pr(G=1) = E[G] = E[X_j \cdot G_1 + \bar{X}_j \cdot G_2]$$

$$\begin{aligned}
&= E[X_j] \cdot E[G_1] + E[\bar{X}_j] \cdot E[G_2] \\
&= E[G_1] + (1 - E[X_j]) \cdot (E[G_2] - E[G_1]) \\
&= \Pr(G_1 = 1) + (1 - \Pr(X_j = 1)) \cdot (\Pr(G_2 = 1) - \Pr(G_1 = 1))
\end{aligned} \tag{3.35}$$

For the second case which is more complex since the events  $X_j$  and  $G_1$  are not independent, to overcome this problem phase algebra is applied from equation (3.20) to account for this. The calculation is as follows.

First consider a general ite structure  $G_1$  which contains a variable that belongs to component X.

$$G_1 = ite(X_i, H_1, H_2) = X_i \cdot H_1 + \bar{X}_i \cdot H_2 \tag{3.36}$$

The calculation of G is as follows:

$$\begin{aligned}
\Pr(G = 1) &= E[G] = E[X_j \cdot G_1 + \bar{X}_j \cdot G_2] \\
&= E[X_j \cdot (X_i \cdot H_1 + \bar{X}_i \cdot H_2)] + E[\bar{X}_j] \cdot E[G_2] \\
&= E[X_j \cdot X_i \cdot H_1 + X_j \cdot \bar{X}_i \cdot H_2] + E[\bar{X}_j] \cdot E[G_2] \\
&= E[X_i \cdot H_1 + \bar{X}_i \cdot H_2] - E[\bar{X}_j] \cdot E[H_2] + E[\bar{X}_j] \cdot E[G_2]
\end{aligned}$$

$$\begin{aligned}
&= E[G_1] + E[\bar{X}_j] \cdot (E[G_2] - E[H_2]) \\
&= \Pr(G_1 = 1) + (1 - \Pr(X_j = 1)) \cdot (\Pr(G_2 = 1) - \Pr(H_2 = 1))
\end{aligned}
\tag{3.37}$$

The Unreliability of the PMS is calculated in step 5 by evaluating the BDD by using either equations (3.36) or (3.37) depending on if the variables from the nodes being connected belong to the same component or different components.

Trivedi et al also consider the unreliability jump at the phase boundaries. This is due to components failure which occurred in the previous phases, which did not affect the system at the time but do in the new phase, this is referred to as a latent failure. The unreliability jump was calculated by computing the difference of unreliability at an instant before and after the boundaries of the phases.

Xing and Dugan [16] consider the limitations of Trivedi et al method. The method obtains the correct BDD for the PMS if the following ordering rules are satisfied:

- 1) For every phase the ordering schemes must be consistent or the same.
- 2) All variables which belong to the same component must stay together in the ordering. Then ordered in the same order as the phases or the reverse of the phases depending on if the forwards or backwards ordering is chosen.

If any ordering scheme was implemented then the method would not obtain the correct BDD, since the operator (PDO) do not account for a general

ordering scheme. The incorrect BDD would contain impossible paths. For example using the backwards PDO to construct the BDD for the mission, which may contain paths where you have the success of a component in a phase then later on in the path the failure of the same component but in an early phase. For example figure 3.7 shows an impossible path (  $A_2 = 0$   $B_2 = 0$   $A_1 = 1$  ).

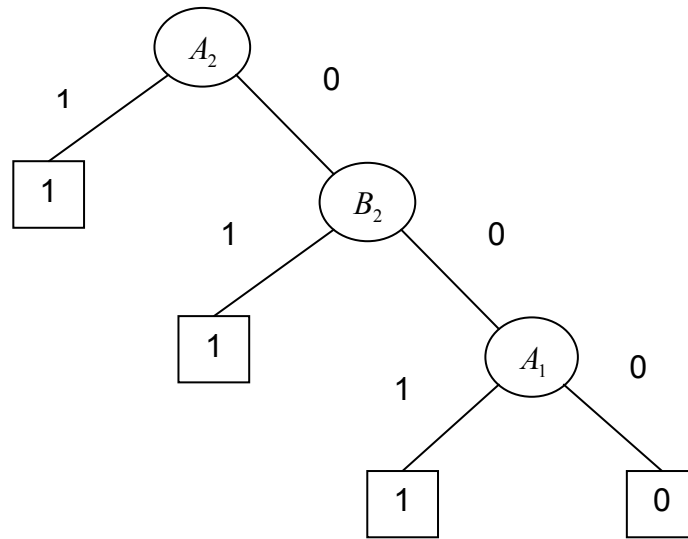


Figure 3.7: BDD with an arbitrary ordering scheme.

However, impossible paths can be removed which will result in the correct BDD. The removed process is as follows: The input branch into the node which makes the path impossible is reconnected to the right son of that node, therefore the left son and any nodes which follow after are removed from the BDD. For example using this technique on the BDD in figure 3.7 results in the BDD shown figure 3.8.



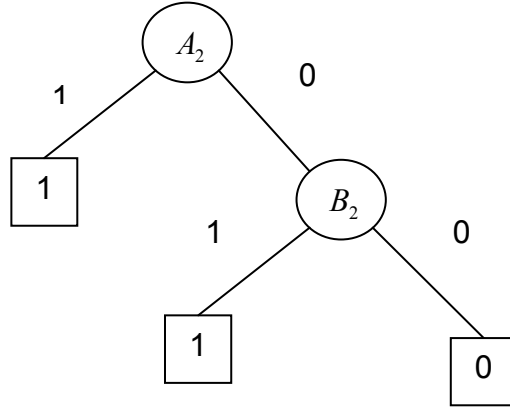


Figure 3.8: BDD where impossible paths have been removed.

A similar technique is used when a forward PDO is used. Removing the impossible paths eliminates the limitations on the ordering of BDD, therefore any ordering scheme may be implemented to construct the final BDD for the PMS.

Summary, the method presented by Trivedi et al [14] successfully constructs an efficient BDD which represents the combination of the components which would fail the PMS. Xing and Dugan [16] consider the ordering limitations of the Trivedi et al method and present a technique to eliminate them.

### 3.2.7 Imperfect Fault Coverage

The papers that have been reviewed so far have only considered two outcomes of a system and component, which are failure and success. In practice systems and components can have more than one failure mode. For example a component might have a failure mode which cause immediate failure of the entire mission even if fault-tolerance mechanisms exist; these are called single point failure. Xing and Dugan [15],[17] considered imperfect fault coverage in a PMS. Imperfect coverage means that the components in the system have two failure outcomes: uncovered failure (single point failure) and covered failure (local effect to the component). A generalized PMS method is proposed to account for imperfect fault coverage. Xing extended this work [19] by developed an efficient method for calculating the reliability of

PMS which accounts for common cause failure as well as imperfect fault coverage.

Tang and Dugan [20] present a method, which is based on BDD, for computing the reliability of a PMS which also accounted for multimode failures which is a generalization of imperfect fault coverage. The method presents more dependence algebra to deal with dependences between the failure modes.

### **3.2.8 Cause-Consequence-Analysis**

Cause-Consequence-Analysis is an alternative approach for modelling system failure; the analysis is based on a diagram. A cause-consequence-diagram represents logically all the system outcomes and its subsequent quantification. These features are useful when modelling a PMS. The disjoint nature of the diagram makes the quantification process more efficient. Vyzaite el at [21] outline the methods for using a cause-consequence-diagram for modelling the reliability of a PMS.

## **Chapter 4: Development of new method for PMS Analysis using BDD**

### **4.1 Introduction**

Previous research published on non-repairable PMS shows that the more efficient approaches for PMS analysis result when the method uses BDDs. Therefore in this chapter the BDD method is investigated and approaches developed to make further advances in its efficiency.

### **4.2 Efficient BDD method for PMS Analysis**

Prescott et al [24] presents a fast analysis method based on the BDD for calculating the individual phase failure probability and the entire mission unreliability of a PMS. The method treats component failures in different phases as independent variables when building the BDDs. This means that a global variable ordering is not required and therefore the ordering of each phase's variables can be separated and is just dependent on the fault tree structure of the failure causes of a particular phase. If the mission is to be formed from a list of potential phase operations, the BDDs of the individual phases can be built before the configuration of a mission is known. This allows the phase failure BDDs to be obtained quickly and at short notice before the mission is due to begin.

After the phase failure occurrence BDDs have been built the next stage is to evaluate them to obtain the phase unreliability's. This requires the failure data for the components. The notation used for a non-repairable component A in the failed state at some point in phase i is as follows:

$$A_{(0, t_i]}$$

Where  $t_i$  is the time when phase i ends.

The interval 0 to  $t_i$  means that the component failure could have occurred anywhere in this interval in order for it to be in the failed state at some point in phase i.

The notation used for a Component A in the working state throughout phase i is:

$$A_{(t_i, \infty)}$$

If the component failure occurs in the interval  $t_i$  to  $\infty$  then the component is in the working state in phase i and all preceding phases.

This interval notation is very useful for a fast evaluation of the BDDs since expressing the period in which the failure can occur deals with the dependency of variables that represent the same component failure but in different phases. The dependencies are dealt with as follows: when tracing through the BDD and two variables representing the same component failure are encountered the intersection of the variable time intervals is taken as the time period in which both component failure events can be satisfied simultaneously.

Paths representing causes of a phase failure due to combinations of component conditions which cannot occur at the same time can be produced in the BDD. Such impossible paths in the BDD may occur since all the component variables were falsely assumed to be independent in the building stage. The impossible paths are recognised when two variables representing the same component have associated failure time intervals that do not intersect. This is accounted for by ignoring this path and moving on to the next one.

## The procedure of the method

The procedure for the method is shown by running through a simple example. First given that a fault tree for each of the phases is known, each individual phase BDD can be obtained before the mission is known. For example the fault trees that represent the phase failure logic for three possible phases of a mission are shown in figure 4.1.

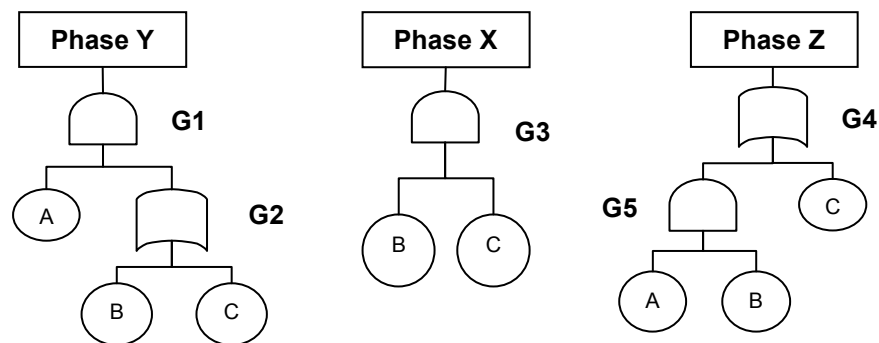


Figure 4.1 Simple phased mission failure causes

The three fault trees in figure 4.1 are converted to BDDs by providing individual orderings of the basic events which is shown in figure 4.2. So far the method has been independent of the mission configuration. However the next step depends on the mission configuration and is performed when the mission has been defined.

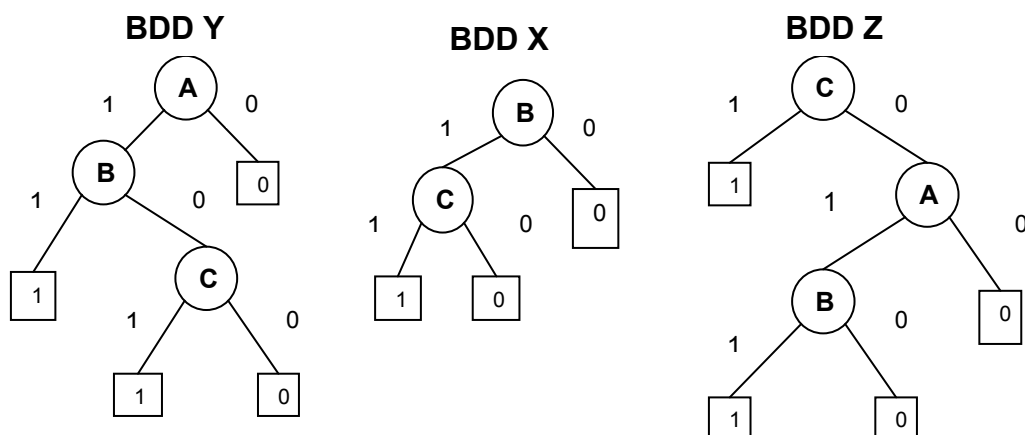


Figure 4.2: BDDs that are converted from the fault trees from figure 4.1

The procedure to obtain a BDD that represents the logic for system failure in a particular phase  $i$  is as follows. Convert all the previous phase BDDs  $1 \dots i-1$  to success BDDs. The success BDD represents the success logic of the phase and is the dual of the failure BDD. To convert from one form to the other requires the one and zero terminal nodes to be inter-changed. Now join these success BDDs for the phases to be performed prior to each phase  $i$  in the mission with the failure BDD of phase  $i$  to obtain the causes of failure in phase  $i$  having successfully reached this phase. This is done by reconnecting the branches of the success BDD phase 1, that have a terminal one node, to the root node of the success BDD phase 2. This BDD joining procedure is continued with the next success phase BDD and so on until the last success BDD phase  $i-1$  has been joined and then the failure BDD  $i$  is joined. Continuing with the example where the mission is known and is phases ZXY. The BDD for the failure in phase 1 is just BDD Z. The BDD for failure in phase 2 and phase 3 are shown in figures 4.3 and 4.4

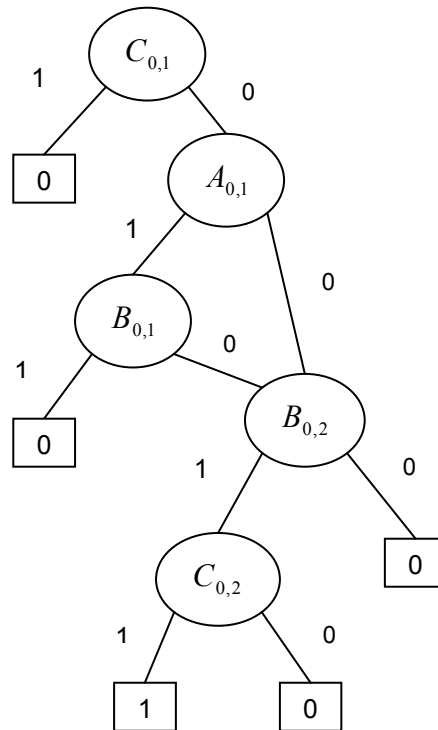


Figure 4.3: BDD for failure in phase 2

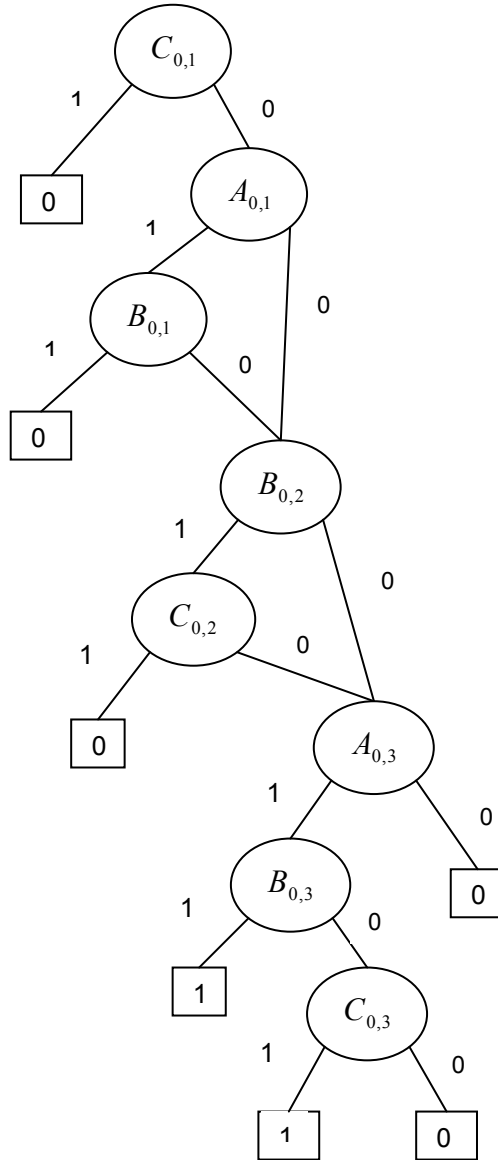


Figure 4.4: BDD failures in phase 3

Once the BDDs have been built the second stage is to use them to evaluate the unreliability. Component failure data is required for this stage. The method is then to trace from the root node of the BDD and record the variables encountered on the paths that terminate in a one node. Phase variables on a path which represent the same component are combined by taking the intersection of their related time intervals. If the time intervals do not intersect then the path is an impossible path and is ignored. Continuing with the example the paths through the 3 phase failure BDDs generated are traced to

obtain the paths and then simplified to eliminate the dependencies between the variables, this process is shown below.

The paths from BDD Z failure in phase 1. (see Figure 4.2) are:

$$C_{(0,1]}$$

$$C_{(1,\infty)} A_{(0,1]} B_{(0,1]}$$

The paths from the BDD generated failure in phase 2. ( ie phase Z works and phase X fails), Figure 4.3 are:

$$C_{(1,\infty)} A_{(0,1]} B_{(1,\infty)} B_{(0,2]} C_{(0,2]} \rightarrow C_{(1,2]} A_{(0,1]} B_{[1,2]}$$

$$C_{(1,\infty)} A_{(1,\infty)} B_{(0,2]} C_{(0,2]} \rightarrow C_{[1,2]} A_{(1,\infty)} B_{(0,2]}$$

The paths from the BDD generated for failure in phase 3 ( ie phase Z works, phase X works, phase Y fails), Figure 4.4, are:

$$C_{(1,\infty)} A_{(0,1]} B_{(1,\infty)} B_{(0,2]} C_{(2,\infty)} A_{(0,3]} B_{(0,3]} \rightarrow C_{(2,\infty)} A_{(0,1]} B_{[1,2]}$$

$$C_{(1,\infty)} A_{(0,1]} B_{(1,\infty)} B_{(0,2]} C_{(2,\infty)} A_{(0,3]} B_{(3,\infty)} C_{(0,3]} \rightarrow \text{Impossible path}$$

$$C_{(1,\infty)} A_{(0,1]} B_{(1,\infty)} B_{(2,\infty)} A_{(0,3]} B_{(0,3]} \rightarrow C_{(1,\infty)} A_{(0,1]} B_{[2,3]}$$

$$C_{(1,\infty)} A_{(0,1]} B_{(1,\infty)} B_{(2,\infty)} A_{(0,3]} B_{(3,\infty)} C_{(0,3]} \rightarrow C_{[1,3]} A_{(0,1]} B_{(3,\infty)}$$

$$C_{(1,\infty)} A_{(1,\infty)} B_{(0,2]} C_{(2,\infty)} A_{(0,3]} B_{(0,3]} \rightarrow C_{(2,\infty)} A_{[1,3]} B_{(0,2]}$$



$$C_{(1,\infty)}A_{(1,\infty)}B_{(0,2]}C_{(2,\infty)}A_{(0,3]}B_{(3,\infty)}C_{(0,3]} \rightarrow \text{Impossible path}$$

$$C_{(1,\infty)}A_{(1,\infty)}B_{(2,\infty)}A_{(0,3]}B_{(0,3]} \rightarrow C_{(1,\infty)}A_{[1,3]}B_{[2,3]}$$

$$C_{(1,\infty)}A_{(1,\infty)}B_{(2,\infty)}A_{(0,3]}B_{(3,\infty)}C_{(0,3]} \rightarrow C_{[1,3]}A_{[1,3]}B_{(3,\infty)}$$

The probability of a path is calculated by the product of the probabilities of the events (variables) in the path since a path does not contain any dependencies between the variables after it is simplified.

$$\text{Path Probability} = \prod_{i=1}^N \Pr( x^i_{(j,k)} ) \quad (4.1)$$

Where N is the total number of variables in the simplified path and  $x_{(j,k)}$  is the variable that component  $x$  fails in the interval (j,k). Then once the probabilities of all the paths have been calculated the probability of the failure in that phase is the sum of the probabilities of the paths, since all the paths are disjoint

$$\text{phase i unreliability} = \sum_{i=1}^N \Pr( path_i ) \quad (4.2)$$

Where N is the total number of paths in the BDD for failure in phase i.

### 4.3 Concise representation of mission phase failure ( Method 1)

Considering the method described in the previous section [24] it is unnecessary to build all the phase failure BDDs separately since potentially all of the information can be obtained from the final phase failure occurrence BDD. This process is performed as follows. The terminal one of the individual phase BDD is replaced by the phase number. Considering again the last example where the mission is phase Z then X then Y and the individual phase BDDs are shown in figure 4.2. The results of converting there to phase BDDs are shown in figure 4.4.

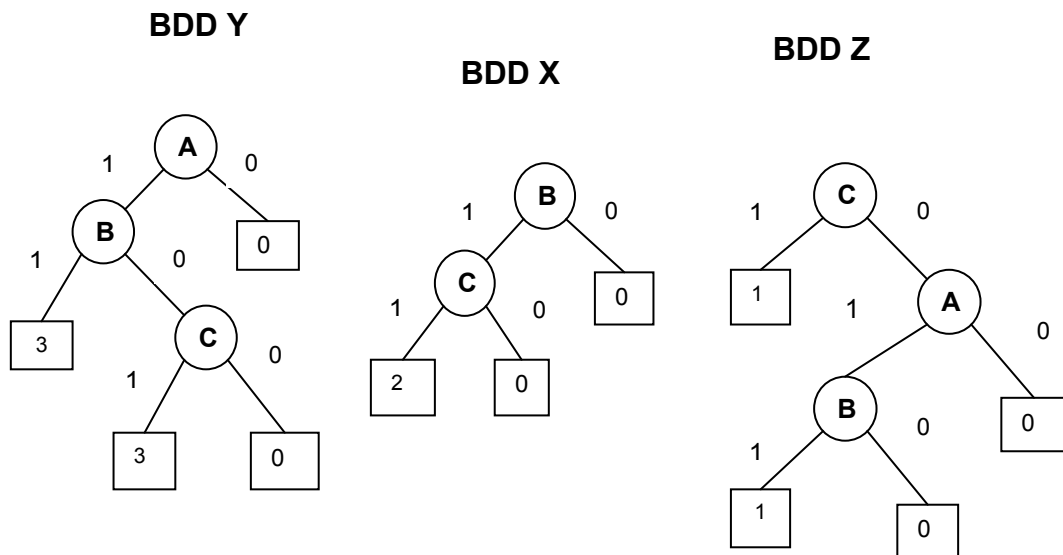


Figure 4.4: BDDs for phases Y,X AND Z

The individual phase BDDs are now taken in the order in which they occur in the mission. Developing BDDs for failure in a phase which accounts for successful operation in previous phases is carried out and results in the BDD for the final shown in figure 4.5.

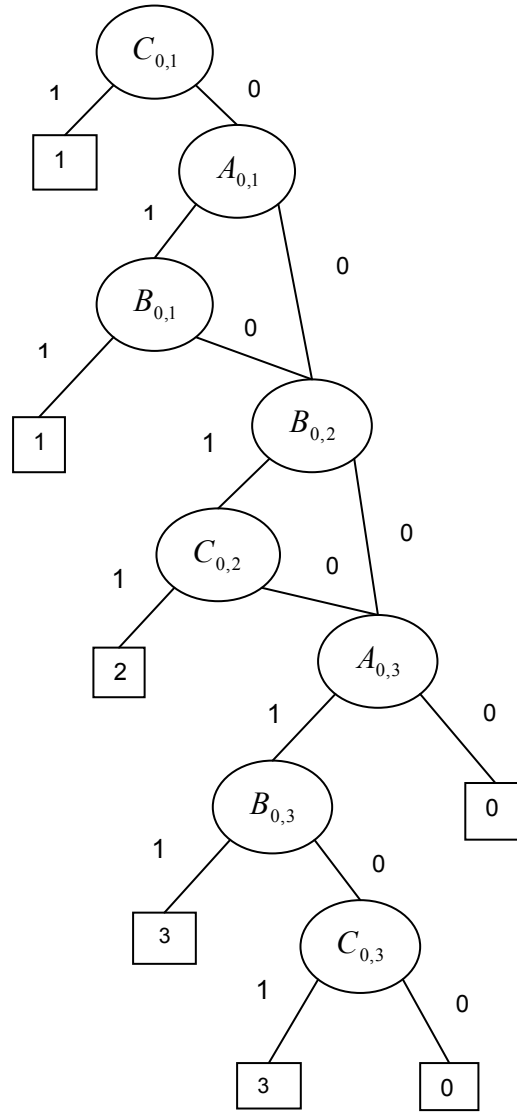


Figure 4.5: The main BDD that contain the failure logic for every phase

Evaluating this BDD is similar to the original method (Prescott method) except that the probability of failure in phase  $i$  is the sum of probabilities of the paths which terminate at node  $i$ . This means that the unreliability of the phases is calculated simultaneously unlike the other method which calculates them one phase at a time. This method will be called **method 1** from now on.

## Results

To test the new method fault trees that represent the failure logic of six mission phases were considered. The number of gates, events and other characteristics of the fault trees are shown in table 4.1 and the fault trees are shown in diagrams in appendix B under mission set 5. Since all of the phase fault trees are converted to a BDD before the mission is analysed the number of nodes in each phase BDD are also shown in table 4.1.

Phase	Number of Gates (OR,AND)	Number of Events (Number of different events)	Percentage Of Different Events	Number of Nodes In BDD
1	17 (9,8)	25 (19)	76%	27
2	14 (8,6)	20 (14)	70%	20
3	15 (7,8)	22 (16)	72%	25
4	15 (7,8)	22 (16)	72%	25
5	14 (8,6)	20 (14)	70%	20
6	17 (9,8)	26 (20)	76%	31

Table 4.1 Characteristics of fault trees of the possible phases from the first mission

Different missions were created by taking different combinations of phases 1-6 and different mission lengths. Phases could be repeated. The mission configurations and the times taken to calculate the mission unreliability using the two methods: Prescott method (original method) and **method 1** (new modified) are shown in table 4.2.

Number of phases	Mission Configuration	Time of old Method (Prescott method)	Time of new Method (method 1)	Percentage improvement
6	1,2,3,4,5,6	0.06s	0.06s	0%
8	1,4,3,2,3,5,6,1	0.34s	0.28s	17.6%
18	1,2,3,4,5,6,1,2,3,4, 5,6,1,2,3,4,5,6	37.67s	31.66	16%
22	1,5,4,6,2,1,3,2,3,4, 5,3,1,3,5,6,3,3,2,1, 2,3	3min 2.64s	2min 20.9s	22.8%
26	1,5,4,6,2,1,3,2,3,4, 5,3,1,3,5,6,3,3,2,1, 2,3,1,2,3,4	6min 33.99s	4min 56.29s	24.4%
34	1,5,4,6,2,1,3,2,3,4, 5,3,1,3,5,6,3,3,2,1, 2,3,1,2,3,4,,2,3,4,5, 3,5,6,3	38min 21.54s	30min 31.1s	20.3%

Table 4.2: Result time of mission calculation

The results show that generally the more phases in the mission the greater the improvement in the analysis times obtained from **method 1** when compared to the Prescott method. This would be expected as more saving is made on the time taken in building the BDDs and evaluating them. Further experiments were carried out where more phases were created with fault trees larger than those used in the pervious mission model experiments. The fault tree characteristics used for the new phase failure causes are shown table 4.3 and the fault trees are shown, in file format, in appendix A under mission set 2.

Phase	Number of Gates (OR,AND)	Number of events	Number of Nodes In BDD
1	13(10,3)	30	30
2	13(10,3)	32	31
3	13(10,3)	30	51
4	13(10,3)	29	29
5	13(10,3)	32	47
6	13(10,3)	33	22

Table 4.3 Characteristics of fault trees of the possible phases from the second mission

Considering the information given for the fault trees in table 4.3 the trees and the BDDs that are created do not seem significantly larger than those used in the previous experiment. However the information does not indicate the occurrence of common sub BDDs where there is more than one input branch going into the same node. In a simple BDD where the basis events represented by the nodes are all independent it would not be much more effort to evaluate a BDD with common sub BDDs since modularisation may be applied. However in this method the variables in the BDD are dependent and therefore modularisation may not be applied and every path must be traced through the structure. The information in table 4.4 shows the number of nodes including repeated ones in the BDDs and is obtain by tracing through every possible path of the BDD starting from the root node, assuming there are no impossible paths.

Phases for first Missions considered	Number of nodes	Phases for second Missions considered	Number of nodes
1	10	1	88
2	5	2	40
3	28	3	175
4	28	4	151
5	5	5	514
6	34	6	63

Table 4.4 Number of nodes for the BDDs from the two missions

Table 4.4 shows that the new fault trees are larger than the previous trees in terms of the work required for their analysis. Different missions were again created by taking different combinations of phases 1-6 and different mission lengths allowing some phases to be repeated. The mission configurations and analysis times for the two methods are shown in table 4.5.

Number of phases	Mission Configuration	Time of old Method (Prescott method)	Time of new Method ( <b>method 1</b> )	Percentage improvement
3	1,4,5	8.9s	8.8s	1%
3	6,2,3	0.69s	0.66s	4%
3	4,3,1	3.4s	3.4s	0%
3	2,1,5	2.4s	2.3s	4%
4	2,3,4,1	54.39s	54.33s	0.1%
4	5,6,1,2	2min 31.9s	2min 29.1s	2%
4	2,3,6,1	43.2s	42.8s	1%
5	3,1,6,4,2	2hr 55min 50.3s	2hr 54min 8.6s	1%
5	5,1,4,2,6	6hr 26min 6.3s	6hr25min7.5s	0.0025%

Table 4.5 Running time of the two PMS analysis method

The results of the improvement of the **method 1** (new method) compared to the Prescott method (original method) on the missions shown in table 4.5 are significantly less than the improvement obtained considering the smaller trees. Observing the results, every time the number of mission phases is increased by one the calculation time significantly increases. This is because when a phase BDD is joined to the mission so far BDD, its evaluation is performed by tracing through the structure. The number of times a node is traced is potentially increased by the product of the number of terminal one nodes on the BDD to which the new phase BDD will be joined times the trace of the additional BDD.

## **Conclusion**

The main advantage of the newly developed method compared to the previous method is that only one BDD has to be built and evaluated to calculate the unreliability of all the phases. This BDD will be the same size and structure as the BDD created for the analysis of the final phase in the previous method. Therefore the new method will take less computational effort to calculate than the previous one. However, the last BDD will be significantly larger than those for previous phases. A disadvantage of the new approach is that the unreliability of the phases can only be calculated once the whole procedure is over, however the original approach obtained the phase unreliability's as it progressed, which in some cases would be an advantage. The problem with both of these methods is that every time an additional phase is included in the mission the number of nodes including repeated ones increases dramatically, therefore different approaches must be considered that simplify the BDD to reduce the number of nodes including repeated ones.

### **4.4 Alternative approach for Phased Mission Analysis using BDDs based on the Trivedi method (Method 2)**

Methods to improve the BDD analysis of PMS as presented in the previous section treat the phases independently until the point when the mission BDDs are evaluated. Therefore making the analysis fast until the evaluation stage is



performed when the advantages gained are lost for larger problems. The method presented in this section which is referred to as **Method 2** explores an extension of the Trivedi method [14] discussed in chapter 3 and which deals with the dependency between the phases as the BDDs are being built and evaluated. The method is extended by calculating the unreliability of the individual phases as well as the overall mission. The method is discussed and explained below by demonstrating its application to an example.

The fault trees of the possible phases in the mission are those used earlier shown in figure 4.1.

## Ordering

Since the event dependencies are going to be dealt with at the stage of building the mission BDDs a global order of the events must be decided. The method goes through every fault tree labelling events using a depth-first traversal starting at the top gate and then to the first son. Once the sub-tree of the son gate has been traced through completely then the next son is considered and so on. When each gate is being visited the input events are listed in the order they appear unless they have already been listed. Going back to the example and starting with phase Y. First gate G1 is considered and the input event is listed A. Now the son of gate G1 is considered which is gate G2. The input events of gate G2 are recorded B and C. Since there are no gate inputs to G2 we therefore trace back to gate G1 and as all the inputs to this gate have been traced, then the phase Y ordering has been completed. The current order is  $A < B < C$ . The next two phase fault trees are now traced through, however all of the events have already been listed.

## The mission

Now the mission is created from all the possible phases. A phase can be repeated several times throughout the mission. All the calculations done until this stage have been done off-line. Continuing with the example the mission which is selected is phase Z first, phase Y second and then phase X last.

## Component failure models

In the method, developed by Prescott et al [24], the component failure data for the individual phases are obtained from files at the beginning of the code. However this method uses a more complex way of calculating the component failure data for the individual phases, which is more realistic to a UAV mission. The method uses the exponential distribution function for calculating the probability that component failure occurs in phase  $i$  which is shown in the expression below.

$$e^{-\lambda_1 T_1} e^{-\lambda_2 T_2} \dots e^{-\lambda_{i-1} T_{i-1}} (1 - e^{-\lambda_i T_i}) \quad (4.3)$$

Where  $\lambda_1 \lambda_2 \dots \lambda_i$  are the component failure rates in the phases of the mission and  $T_1 T_2 \dots T_i$  are the durations of the phases. The expression depends on the order of the phases in the mission, therefore this calculation has to be done after the mission is known, which was not the case in previous method (**method 1**) in this chapter.

## Phase ordering components

In the Trivedi method there are two options for ordering the component failure events, forward and backward phase ordering, which was discussed in Chapter 3 section 3.2.6. The new method presented here uses backward ordering. Each component in the global order is replaced by the set of phase variables for that component in the order of the last phase to the first phase. For the example the ordering is as below.

$$A_{0,3} < A_{0,2} < A_{0,1} < B_{0,3} < B_{0,2} < B_{0,1} < C_{0,3} < C_{0,2} < C_{0,1} \quad (4.4)$$

## Building the individual phase BDDs

The phase fault trees are converted to the equivalent BDD form. After each individual phase BDD has been obtained it is copied into the main BDD two dimensional array and also the variables are converted to account for the phase ordering. The phase fault trees from the example are converted to BDDs and the variables are changed to phase ordering as shown in figure 4.6.

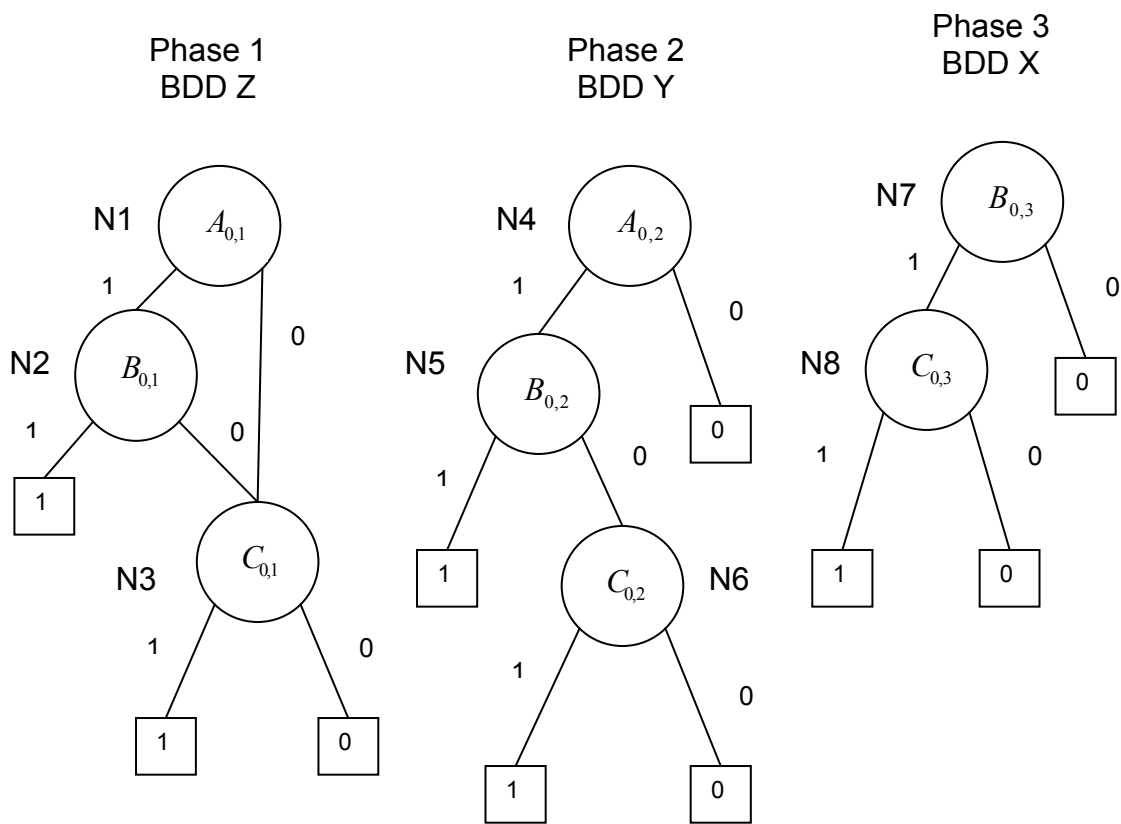


Figure 4.6: BDD that have be converted from phase fault trees

## Building and evaluating the phase failure BDD

When combining the phase BDDs with each other, node variables belonging to the same component but different phases are encountered and the

dependencies are dealt with. When evaluating the phase BDDs an alternative formula will be used for nodes with dependencies. Each node will only have to be evaluated once unlike the previous method which has to trace through the nodes several times.

The previous methods for combining the BDDs to obtain phase failure BDDs was accomplished by changing all the previous phase failure BDDs to success BDDs, by changing the terminal nodes around. All of these success BDDs were combined with the failure BDD of the particular phase with a AND operator. However considering the current method an approach based on the Trivedi method is used. The BDD computation formula shown in equation 3.31 and 3.32 in chapter 3 and the global ordering means that when building phase BDDs it is computationally intensive when compared to **Method 1**. However there is an alternative way to do this part of the calculation that will be less computationally intensive. Instead a sequence of attaching the next phase failure BDD to the last phase BDD by an OR operator is used after each phase has been attached the BDD evaluated. These BDDs represent the events that the system has failed in that particular phase or any previous phase. So taking any probability of these BDDs in this sequence and subtracting the previous probability in this sequence gives the probability that the system failure occurred in that phase. The proof that it is equal is shown below.

For this proof the Boolean variable  $p_i$  is used which represents the failure logic from the fault tree of phase i and  $\bar{p}_i$  represents the success logic from the dual of the fault tree of phase i. The left hand side of the equation 4.5 below represents the system has been successful in phases 1 to i. The right hand side is just the original expression copied i-1 extra times. The extra expressions are modified by replacing  $\bar{p}_i$  by  $p_1$  to  $p_{i-1}$  in turn one for each extra expression. The sum of all these extra expressions is equal to 0 therefore they can be added on (Note + is used for OR in this logic

expression). Now the intersection (AND) of these events  $\bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1}$  can be taken out as a factor of this expression.

$$\begin{aligned}
\bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1} \bar{p}_i &= \bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1} \bar{p}_i \\
&+ \bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1} p_1 \\
&+ \bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1} p_2 \\
&\vdots \\
&\vdots \\
&+ \bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1} p_{i-1} \\
&= \bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1} (\bar{p}_i + p_1 + p_2 + \dots + p_{i-1}) \quad (4.5)
\end{aligned}$$

Now the NOT operator is applied to both sides of the equation 4.5 and De Morgan's law [1] is used to expand the terms out. The outcome of this is equation 4.8.

$$\overline{\bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1} \bar{p}_i} = \overline{\bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1} (\bar{p}_i + p_1 + p_2 + \dots + p_{i-1})} \quad (4.6)$$

$$\begin{aligned}
&p_1 + p_2 + \dots + p_{i-1} + p_i \\
&= p_1 + p_2 + \dots + p_{i-1} + \overline{(\bar{p}_i + p_1 + p_2 + \dots + p_{i-1})}
\end{aligned}$$

$$= p_1 + p_2 + \dots + p_{i-1} + (p_i \bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1}) \quad (4.7)$$

Now take the probability of both sides.

$$\begin{aligned} & \Pr(p_1 + p_2 + \dots + p_{i-1} + p_i) \\ &= \Pr(p_1 + p_2 + \dots + p_{i-1} + p_i \bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1}) \end{aligned} \quad (4.8)$$

These two expressions can be separated because the events are mutually exclusive.

$$\begin{aligned} & \Pr(p_1 + p_2 + \dots + p_{i-1} + p_i) \\ &= \Pr(p_1 + p_2 + \dots + p_{i-1}) + \Pr(p_i \bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1}) \end{aligned} \quad (4.9)$$

Now take the first term of the right-hand over to the left hand side.

$$\begin{aligned} & \Pr(p_1 + p_2 + \dots + p_{i-1} + p_i) - \Pr(p_1 + p_2 + \dots + p_{i-1}) \\ &= \Pr(\bar{p}_1 \bar{p}_2 \dots \bar{p}_{i-1} p_i) \end{aligned} \quad (4.10)$$

Equation 4.10 means that the probability of the mission being successful from phase 1 up to phase  $i-1$  and then failing in phase  $i$  is equal to the probability that the mission fails in any phase 1 to  $i$ , minus the probability that the mission fails in any phase 1 to  $i-1$ . This expression is very helpful when building and

evaluating the phase BDDs since the BDDs that represent the terms on the left-hand side of equation 4.10 can all be built by adding on a phase on the current expression. Therefore this is more efficient than starting to build each failure phase BDD from scratch to obtain the right-hand side of equation 4.10. Continuing with the example, building and evaluating the phase failure BDDs is as follows.

The formulas 3.31, 3.32, 3.35 and 3.37 from Chapter 3 section 3.2.6 are used for combining the BDDs together and for evaluating them for the probabilities. The probability of failure in the first phase is just the evaluation of the phase 1 BDD in figure 4.6 which is the evaluation of node 1 which is shown below.

Probability of node 2:

$$\Pr( N_2 = 1 ) = 1 + (1 - \Pr( B_{0,1} = 1 )) \cdot (\Pr( C_{0,1} = 1 ) - 1)$$

Probability of node 1:

$$\Pr( N_1 = 1 ) = \Pr( N_2 = 1 ) + (1 - \Pr( A_{0,1} = 1 )) \cdot (\Pr( C_{0,1} = 1 ) - \Pr( N_2 = 1 ))$$

Probability of failure in phase 1

$$\Pr( p_1 = 1 ) = \Pr( N_1 = 1 )$$

The first and second phase are combined by an OR as shown in figures 4.7 and 4.8 and the final BDD with top node 11 shown in figure 4.10.

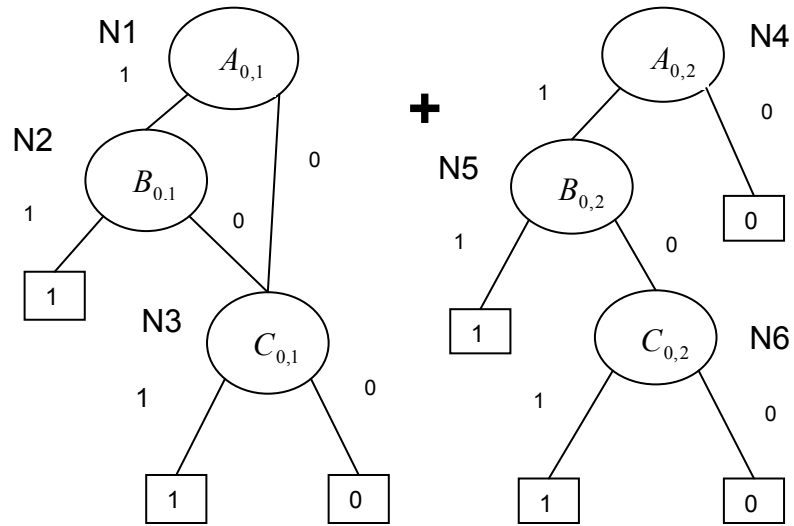


Figure 4.7: Phase 1 OR Phase 2 BDDs

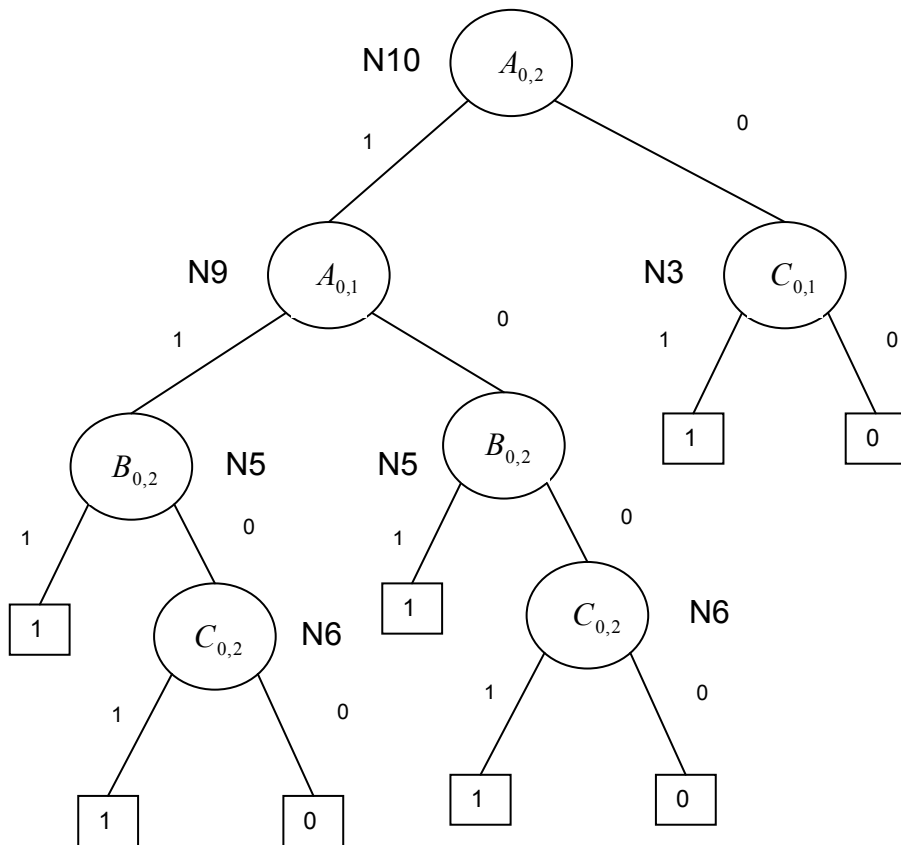


Figure 4.8: Phase 1 OR Phase 2 BDDs combined



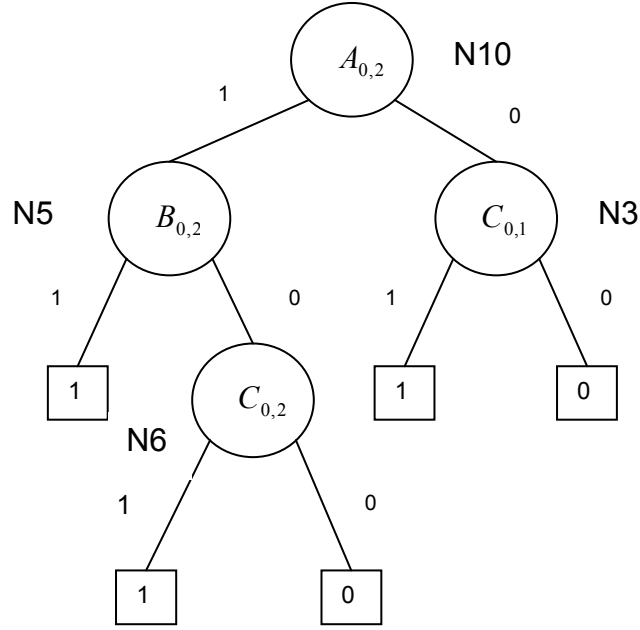


Figure 4.9: Phase 1 OR Phase 2 BDDs simplify

Now node 11 is evaluated as shown below. This is the probability that the mission fails in phase 1 or 2. However the probability that the mission is successful in phase 1 and failure occurs in 2 is the evaluation of node 10 minus node 1 shown in equation 4.12.

Probability of node 5:

$$\Pr( N_5 = 1 ) = 1 + (1 - \Pr( B_{0,2} = 1 )) \cdot (\Pr( C_{0,2} = 1 ) - 1)$$

Probability of node 10:

$$\Pr( N_{10} = 1 ) = \Pr( N_5 = 1 ) + (1 - \Pr( A_{0,2} = 1 )) \cdot (\Pr( C_{0,1} = 1 ) - \Pr( N_5 = 1 ))$$

Probability of failure in phase 1 or Phase 2:

$$\Pr(( p_1 + p_2 ) = 1) = \Pr( N_{10} = 1)$$

Probability of failure occurs in 2:

$$\Pr( \bar{p}_1 p_2 = 1 ) = \Pr( N_{10} = 1 ) - \Pr( N_1 = 1 )$$

(4.11)

The same procedure is applied to phase 3 has the previous phase and is shown below in Figure 4.10 and 4.11.

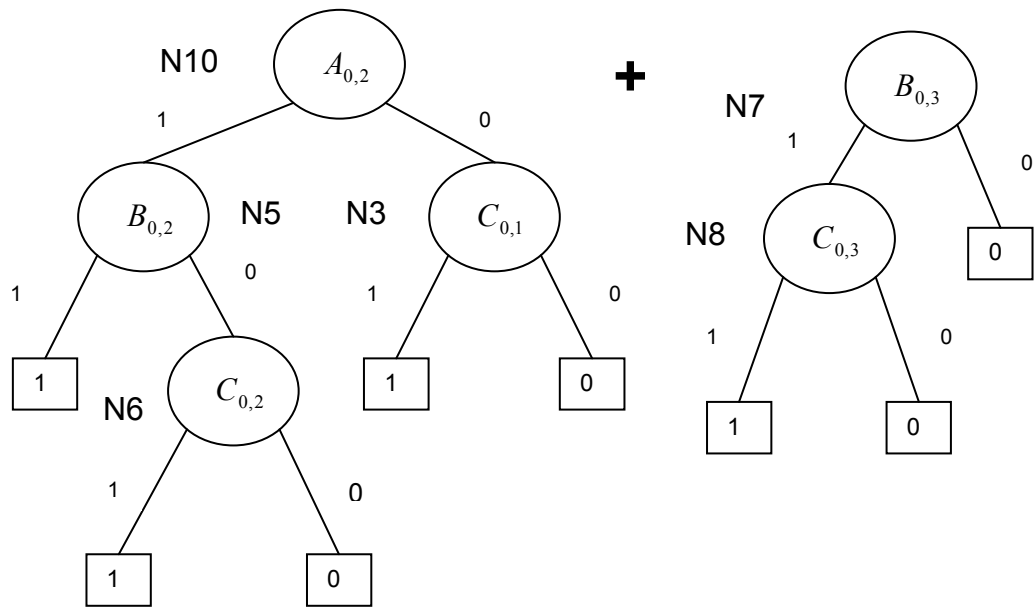


Figure 4.10: (Phase 1 OR Phase 2) OR Phase 3 BDDs combined

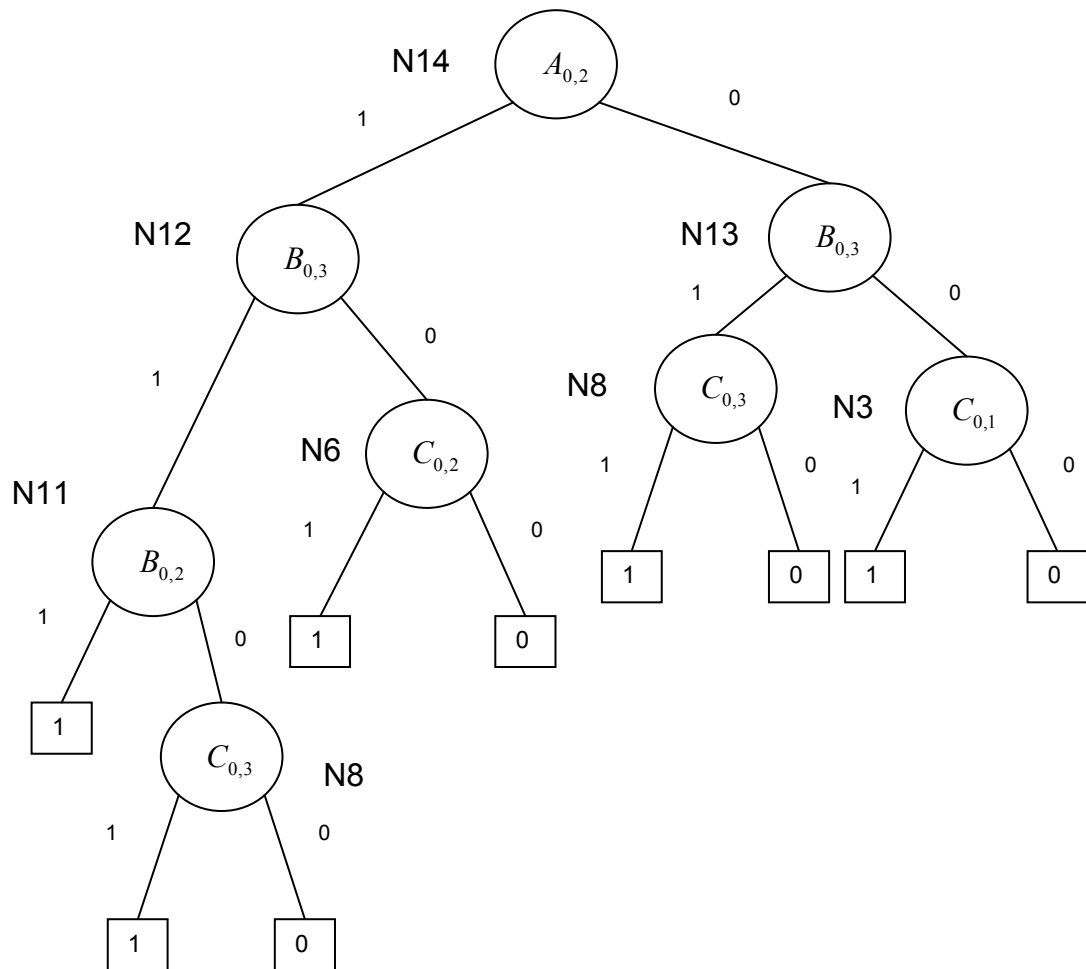


Figure 4.11: (Phase 1 OR Phase 2) OR Phase 3 BDD

Probability of node 11:

$$\Pr(N_{11} = 1) = 1 + (1 - \Pr(B_{0,2} = 1)) \cdot (\Pr(C_{0,3} = 1) - 1)$$

Probability of node 12:

$$\Pr(N_{12} = 1) = \Pr(N_{11} = 1) + (1 - \Pr(B_{0,3} = 1)) \cdot (\Pr(C_{0,2} = 1) - \Pr(N_{11} = 1))$$

Probability of node 13:

$$\Pr(N_{13} = 1) = \Pr(C_{0,3} = 1) + (1 - \Pr(B_{0,3} = 1)) \cdot (\Pr(C_{0,1} = 1) - \Pr(C_{0,3} = 1))$$

Probability of node 14:

$$\Pr(N_{14} = 1) = \Pr(N_{12} = 1) + (1 - \Pr(A_{0,2} = 1)) \cdot (\Pr(N_{13} = 1) - \Pr(N_{12} = 1))$$

Probability of failure in phase 1 or Phase 2 or Phase 3

$$\Pr((p_1 + p_2 + p_3) = 1) = \Pr(N_{14} = 1)$$

Probability of failure occurs in 3:

$$\Pr(\bar{p}_1 \bar{p}_2 p_3 = 1) = \Pr(N_{14} = 1) - \Pr(N_{10} = 1)$$

(4.12)

#### 4.5 Fault Tree Modularisation

Modularisation methods identify independent sub trees of a fault tree that can be calculated separately. This simplifies the fault tree, making its easier to analyse. These independent sub trees are known as modules which are defined as a section of the fault tree which is completely independent from the rest of the tree so that all the gates and events it contains do not occur anywhere else in the rest of the tree. These modules can be treated as individual fault trees and analysed separately. There are several methods that perform modularisation, however the most efficient which is going to be discussed in this section is linear-time algorithm [25]. This method is the most efficient since it only scans through the tree twice to obtain all the modules.

## Linear - Time Algorithm

This algorithm is performed in two depth-first traversals of the fault tree. The first passes through the tree structure and records, step by step, the order in which gates and events are visited. Therefore the information of the step number at the first, second and final visit to every gate and event is obtained. This is explained by going through the technique performed on the fault tree shown in figure 4.12. Going through the tree in a depth-first manner and recording the step number at which the gates and events visited, the visited order is shown in table 4.6. The algorithm always considers the event inputs ahead of the gate inputs when scanning through the tree. Every gate will be visited at least twice once scanning down the structure and the other time tracing back up. If the gate is repeated more than once then when visiting the gate after the first encounter the gate is just listed but the events below it are not considered again, therefore treated like an event. This is shown in table 4.6 when gate G4 is visited again on its second appearance on the 30th visit

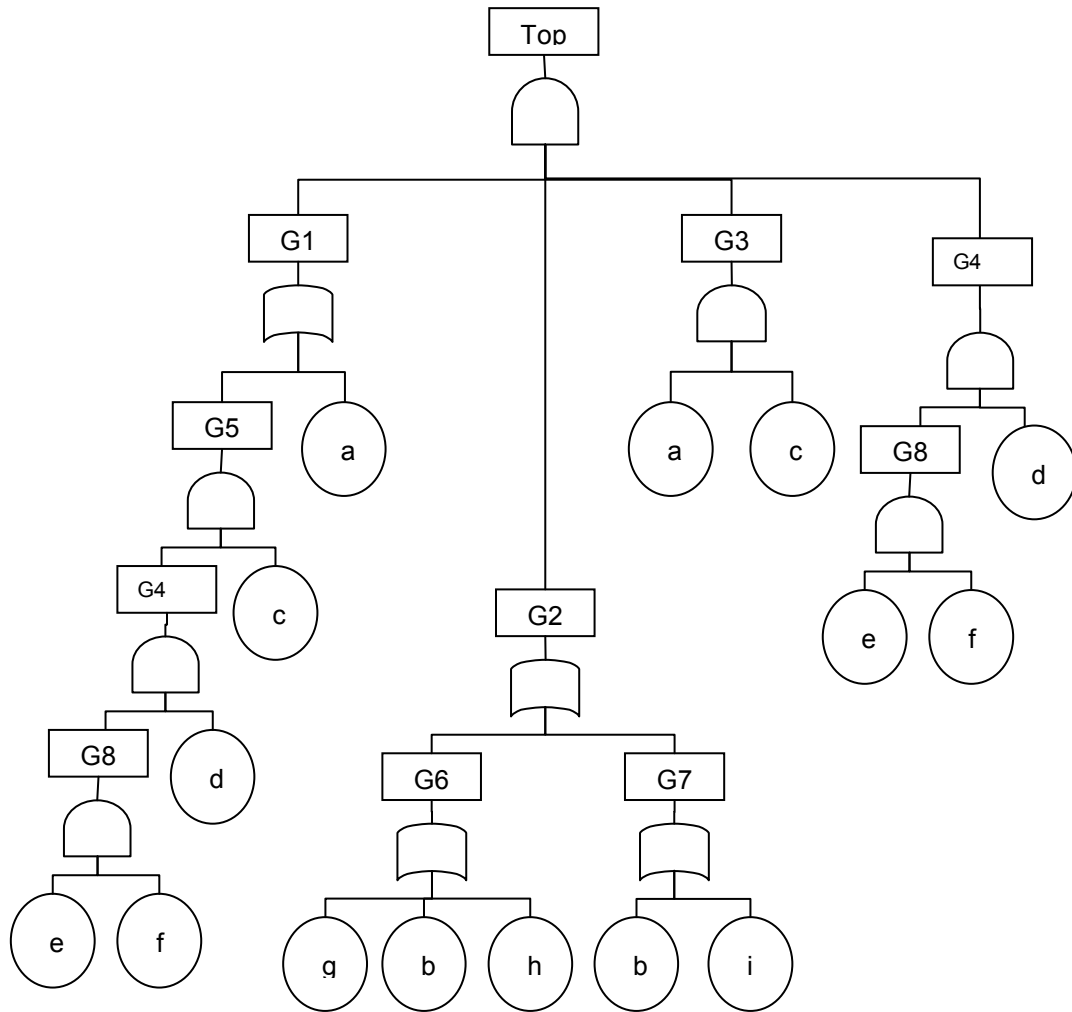


Figure 4.12: Example fault tree to explained the linear-time algorithm

Step number	1	2	3	4	5	6	7	8	9	10	11
Node	Top	G1	a	G5	c	G4	d	G8	e	f	G8

Step number	12	13	14	15	16	17	18	19	20	21	22
Node	G4	G5	G1	G2	G6	g	b	h	G6	G7	b

Step number	23	24	25	26	27	28	29	30	31
Node	i	G7	G2	G3	a	c	G3	G4	Top

Table 4.6: The list of order of visited from the depth-first traversal of the fault tree in figure 4.12

From the visit list in table 4.6 information is obtained for every gates first, second and last visit step number which are shown in table 4.7. In the table the second and the final visit for the gates are the same except for gate G4 which is different since it appeared more than once in the tree. Also the equivalent information is obtained for the events and shown in table 4.8.

Gate	Top	G1	G2	G3	G4	G5	G6	G7	G8
1 <sup>st</sup> visit	1	2	15	26	6	4	16	21	8
2 <sup>nd</sup> visit	31	14	25	29	12	13	20	24	11
Final visit	31	14	25	29	30	13	20	24	11
Min	2	3	16	3	7	5	17	18	9
Max	30	27	24	28	11	28	22	23	10

Table 4.7: Visits number and MAX and MIN value for the gates in the fault tree

Event	a	b	c	d	e	f	g	h	i
1 <sup>st</sup> visit	3	18	5	7	9	10	17	19	23
2 <sup>nd</sup> visit	27	22	28	7	9	10	17	19	23
Final visit	27	22	28	7	9	10	17	19	23

Table 4.8: Visits number for events in the fault tree

Now the algorithm moves on to the second pass through of the tree which obtains the maximum (Max) of the last visits and the minimum (min) of the first visits of descendants of each gate this is also shown in table 4.7. Now there is enough information for the algorithm to identify the modules. If a gate has any descendant which has a smaller first visit step number than the first visit step number of the gate, then that descendant must occur beneath another gate. Similarly, if a gate has any descendant which has a last visit step number greater than the second visit step number of that gate, then also that

descendant must occur beneath another gate. Therefore a gate can be identified as a module if and 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.

If these conditions are satisfied then it means that none of the descendants can appear anywhere else in the tree (except for beneath another appearance of the same gate) which makes the gate a module. Therefore the algorithm goes through each gate and compares the min and max values with the first and second visit step number of its descendants.

Going back to the example shown in table 4.7 shows that gates G1, G5 and G6 max values are greater than their second visit step numbers therefore the gates cannot be modules. Also gates G3 and G7 cannot be modules because their min values are smaller than their first visit step number. Therefore the gates that satisfy the module conditions are gates G2, G4, G8 and also the top gate which is always a module. Each of these module gates are now replaced in the fault tree by a module event. This is shown in figure 4.13 which assigns modular event M1 to gate G2, modular event M2 to gate G4, and modular event M3 to gate G8. Now the module gates can be analysed separately and the results used for the module events which they replace in the main fault tree. Modularization significantly reduces the computations needed for the analysis.

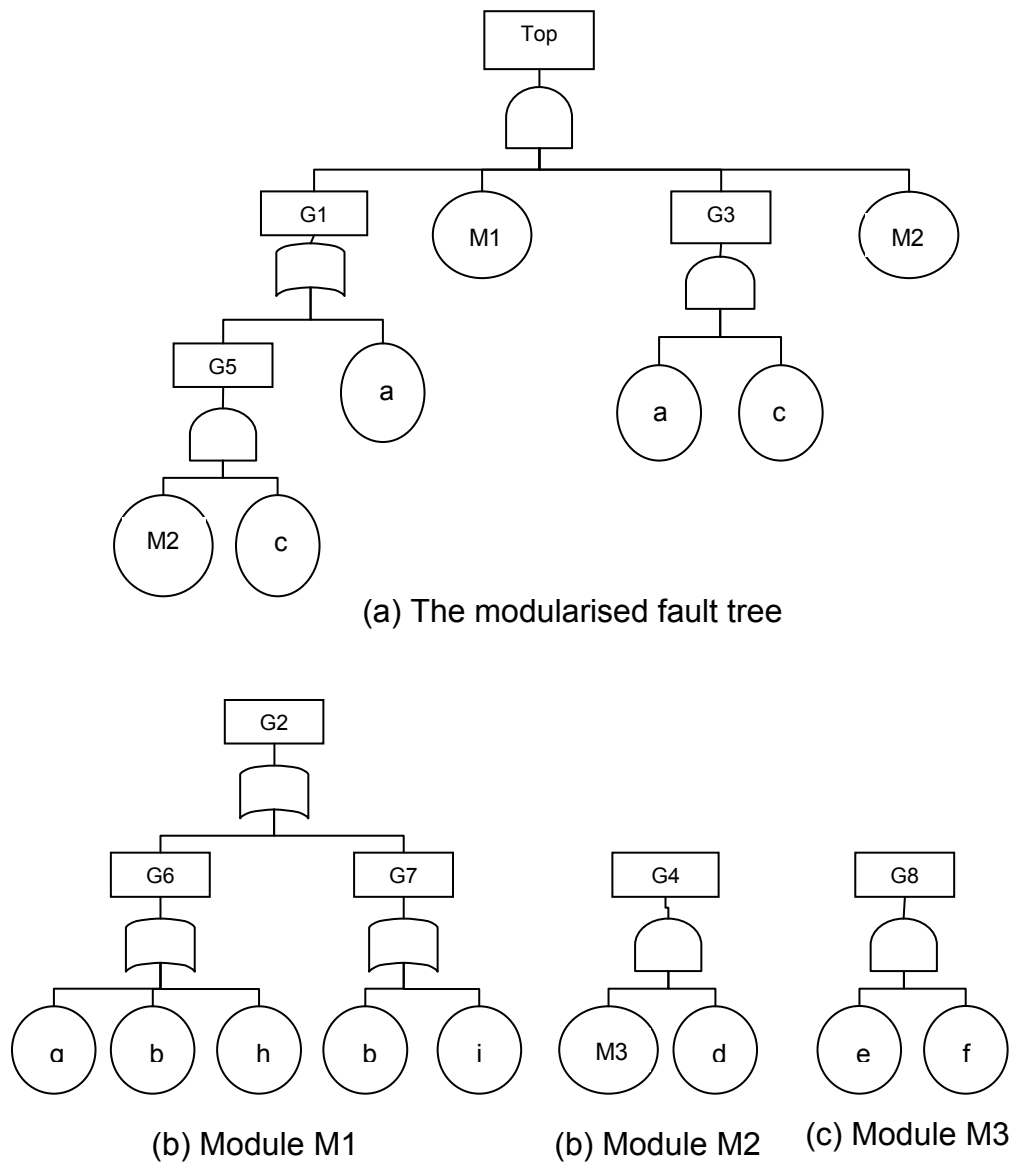


Figure 4.13: The fault tree after the modules have been taken out

#### 4.6 Applying Modularisation to phase mission analysis (Method 3)

This section discusses a new method, which is referred to as **Method 3**, for performing a phase mission analysis that applies the modularization technique from section 4.5 to the Trivedi approach discussed in section 4.4. Modularization has been explored in the context of performing a single phase analysis and been found to be very successful in its efficiency. Its benefits have not been overly explored for a phase mission analysis. This new method is going to be discussed and explained by considering an example for which



the fault trees for the possible phases are shown in figure 4.14. This example has been specially chosen because it covers all of the special cases of the method. The method takes out module gates of the mission. However depending on which phases are chosen for the mission a gate may be a module or not.

### Identify modules

First the gates which are modules are identified for each of the phase fault trees. The linear time algorithm is used which was discussed in the previous section. It is applied to the fault trees shown in figure 4.14 to identify its modules. The modules are listed in the first row and the top event gates that belong to it are listed in the second row of table 4.9.

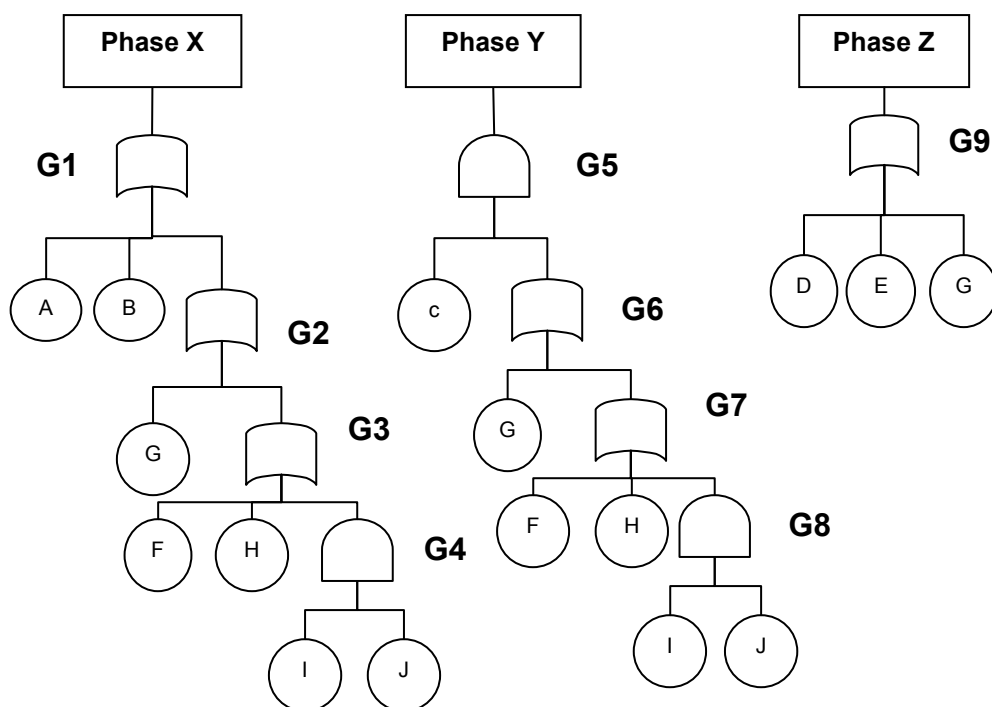


Figure 4.14: Example of three phase fault tree

### **Identify which modules are beneath other modules.**

The information of which modules are beneath other modules is essential when it comes to building and evaluating the BDD of the module gates since a BDD for a module cannot be built unless all the modules beneath it are built. This information is obtained as follows. Every possible pair of modules is considered. First it is checked if they belong to the same phase since if they didn't it would be impossible for one of them to be beneath another. The first gate in the pair is referred to as the 'above gate' and the second the 'below gate'. Now the visit information from the linear time algorithm is used. If the first visit step number of the 'above gate' is less than the first visit step number of the 'below gate' and if second visit step number of the 'above gate' is greater than the second visit step number of the 'below gate' then the 'below gate' is beneath the 'above gate'. Also if any of the visit step numbers, of the 'below gate', after the second visit are greater than first visit step number of the 'above gate' and less second visit step number of the 'above gate' then also the below gate is beneath the 'above gate'. These rules are shown below. Continuing with the example, all the modules beneath other modules are shown on row four of table 4.9.

**IF(**

first visit step number of the above gate < first visit step number of the below gate

**AND**

second visit step number of the above gate > second visit step number of the below gate

**)**

**Then** below gate is beneath the above gate.

**OR**

**IF(**

first visit step number of the above gate

< visit step numbers of the below after the second visited <

second visit step number of the above gate

)

**Then** below gate is beneath the above gate.

Module	M1	M2	M3	M4	M5	M6	M7	M8	M9
Module Gate	G1	G2	G3	G4	G5	G6	G7	G8	G9
Phase that Belongs to	X	X	X	X	Y	Y	Y	Y	Z
Modules Beneath	G2 G3 G4	G3 G4	G4	-	G6 G7 G8	G7 G8	G8	-	-
Node for the module logic	N25	N23	N22	N20	N26	N23	N22	N20	N28
Module Gate equal	-	G6	G7	G8	-	G2	G3	G4	-
Contradiction phase	Y Z	Z	-	-	X Z	Z	-	-	X Y

Table 4.9: All the information of the potential modules

## Ordering

The variables are ordered by a depth first ordering followed by the module events. The ordering of the variables in the example is shown below in equation 4.13.

$$A < B < G < F < H < I < J < C < D < E < M1 < M2 < M3 < M4 < M5 < M6 < M7 < M8 < M9 \quad (4.13)$$

### Building the BDDs for all the possible modules

It is unknown which of the modules that are listed in table 4.9 will be modules in the mission, since the mission is unknown at this stage. Therefore all the BDDs are built for the modules since potentially any of them could be modules in the mission. When building the BDDs all of the components are treated as single variables instead of multi-phase. There are two reasons for the variables been treated as a single phase. One is that the calculation for the common logic throughout the possible phases will only be performed once. Secondly if there are two or more modules which are logically equivalent but are labelled as different gates or are structured differently then since only one order is used the algorithm will recognize this. Therefore it will reduce the calculations. When the BDD of a module is built the gate to that module is replaced by its module event in the faults tree. A BDD is only built for a module if all of the modules beneath it have had a BDD built for it. Considering the example, first the single node BDDs are created for the events and module events these are nodes N1 to N19 and are shown in figure 4.15.

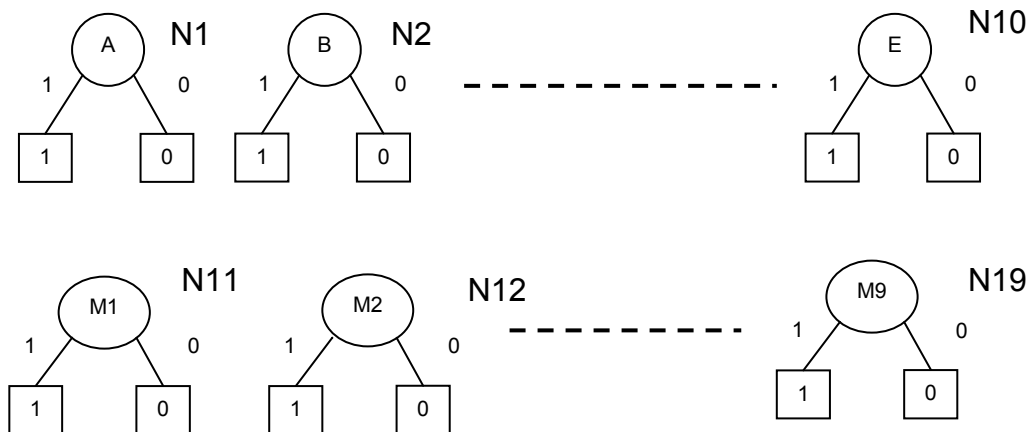


Figure 4.15: Single node BDD for events and possible modules

Now the first module BDD that is built is for module 4 because it does not have any modules beneath it. The node that is created for the logic of the module is N 20. This is recorded in the fifth row of table 4.9 .Therefore the module event 4 replaces gate G4 in the fault trees. Also gate G8 is replaced by module event 4 because that it is logically equivalent to G4. These changes to the fault trees and the module BDDs are shown in figure 4.16.

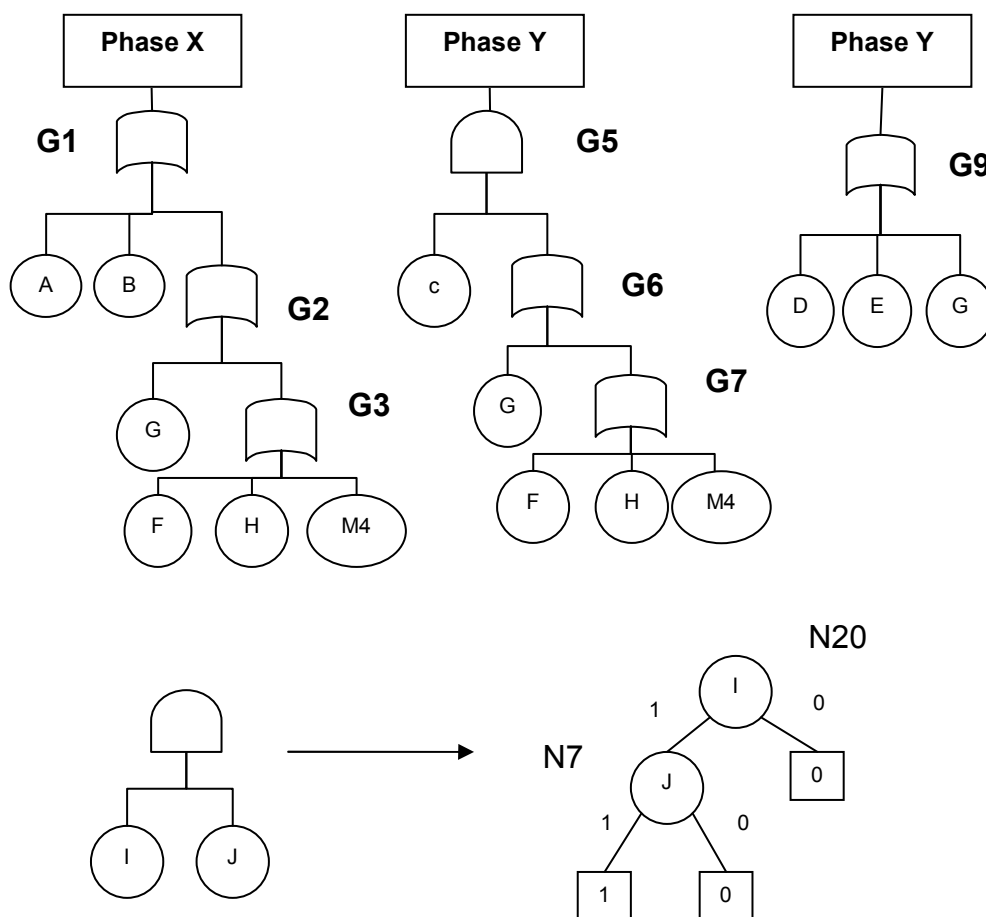


Figure 4.16: The fault trees after module 4 replaces gate G4 and G8

The next module BDD that is built is for module 3 which is represented by node 22 and is also logically equivalent to gate G7. The module gates G3 and G7 are replaced by module 3 and this is shown in figure 4.17.

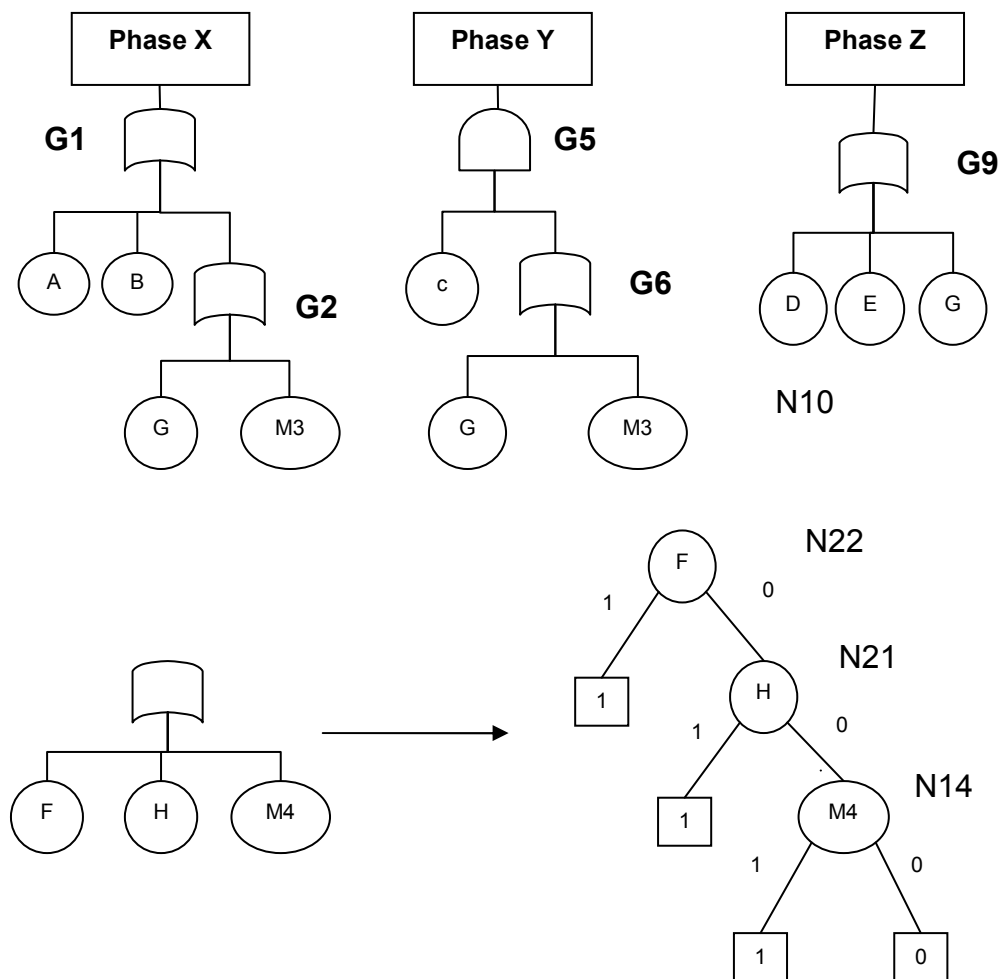


Figure 4.17: The fault trees after module 3 replaces gate G3 and G7

Now the next module BDD that is built is for gates G2 and G6 which are logically equivalent and all the modules beneath them have been built. The BDD is represented by node 23 and is shown in figure 4.18 with the changes to the fault trees.

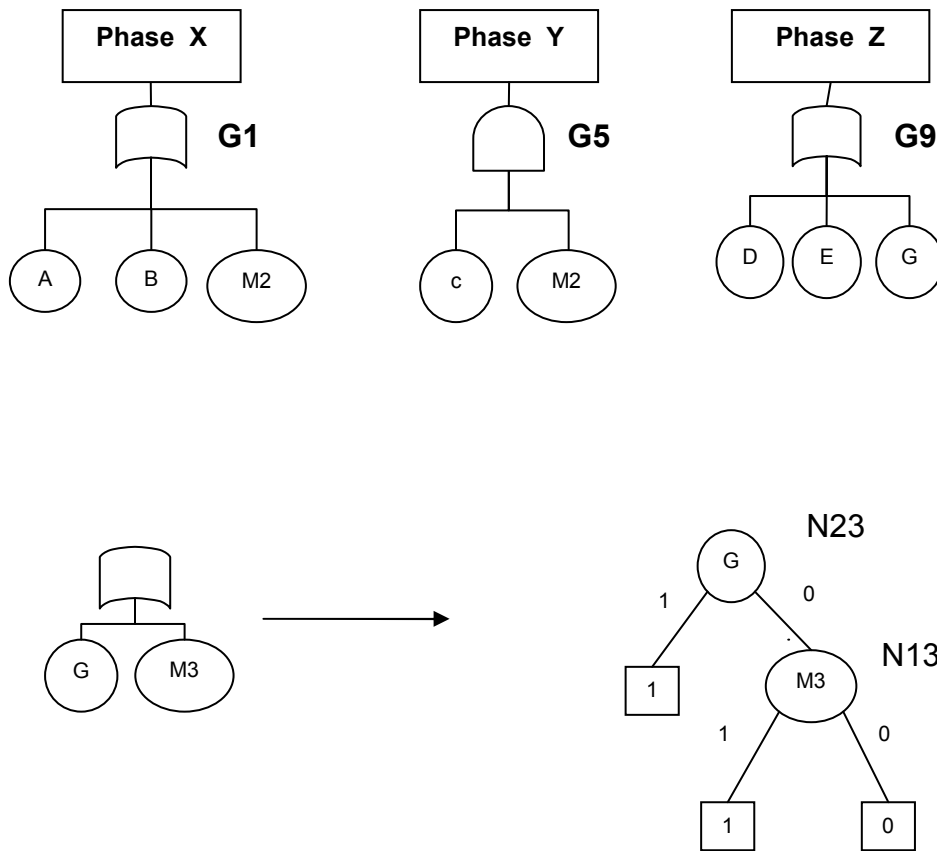


Figure 4.18: The fault trees after module 2 replaces gate G2 and G6

The top gates to the fault trees are modules and therefore the BDDs are built for them and represented by nodes N25, N26 and N28 this is reported in table 4.9 and shown in figure 4.19.

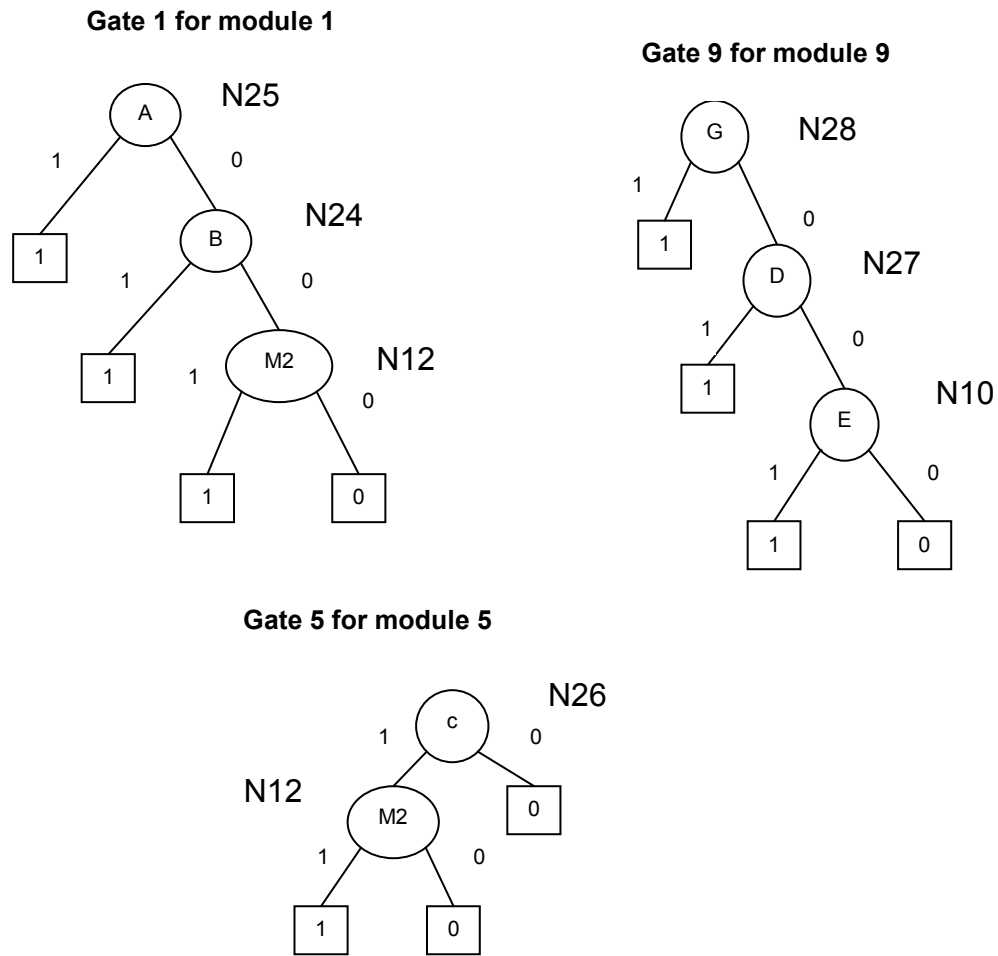


Figure 4.19: BDDs of the fault trees phase if all modules were take out

### Contradiction phases to the modules to a single phase

When phases are combined to form a mission some of the modules of the individual phases will not be modules of the combined logic function. It is established which potential modules are mission modules and which are not for every module of the individual phases. However at is stage the mission is still unknown so a list of phases for every potential module is obtained which are referred to as contradiction phases. A contradiction phase for a particular potential module is defined as a phase such that if it was chosen for the mission then that potential module is not a module in the mission. So when the mission is known the algorithm can identify which potential modules are modules of the mission by comparing their list of contradiction phases to the



list of phases in the mission. The algorithm obtains the contradiction phases for a potential module by listing all of the gates and events that appear under it. Then it scans through a phase fault tree and lists all of the components in a second list. The two lists are then compared to see if they have a common component. If they do then that phase is a contradiction phase for that module. This is done for every phase. However there is a special case, if a phase contains a module which is logically equivalent to the potential module then that phase is not a phase contradiction. This procedure needs to be performed on the fault trees before the modules were extracted therefore the program copies the fault trees before the modules are taken out and the procedure is performed on the copied versions. Returning to the example the phase contradictions to the modules are shown on row 7 of table 4.9. Considering gate G1 which has two phase contradictions Y and Z. Phase Y because it has components G, F, H, I and J in common and phase Z because it has component G in common. Considering gate G2 as well which has contradiction phase Z because it has component G in common. In this case phase Y is not a contradiction phase even though it has components in common since these components appear under gate G6 which is logically equivalent to gate G2. The algorithm does this for each gate module of a single phase.

## **Mission**

Now the mission is entered, defined in terms of all the possible phases. The phases can appear more than once in the mission. Two example missions are going to be considered and these are as follows:

### **Mission one**

Mission one is defined as phase X then Y then Z. Now that the mission is known the component failure probabilities can be calculated as the same in the current method described in section 4.4.

## Identify the mission modules for mission one

A mission module is a module for a single phase which is also a module for the whole mission. To establish the modules which are not mission modules the contradiction phases shown on row seven of table 4.9 are scanned. If any of its contradiction phases is contained in the mission then it cannot be a mission module. Therefore if a module is not a module in the mission it is put back by replacing it with its top gate in the fault trees. Considering the example, gates G1, G2, G5, G6 and G9 all have phase contradictions in the mission therefore they are all replaced in the fault tree structure by their top event gates. This is shown in figure 4.20.

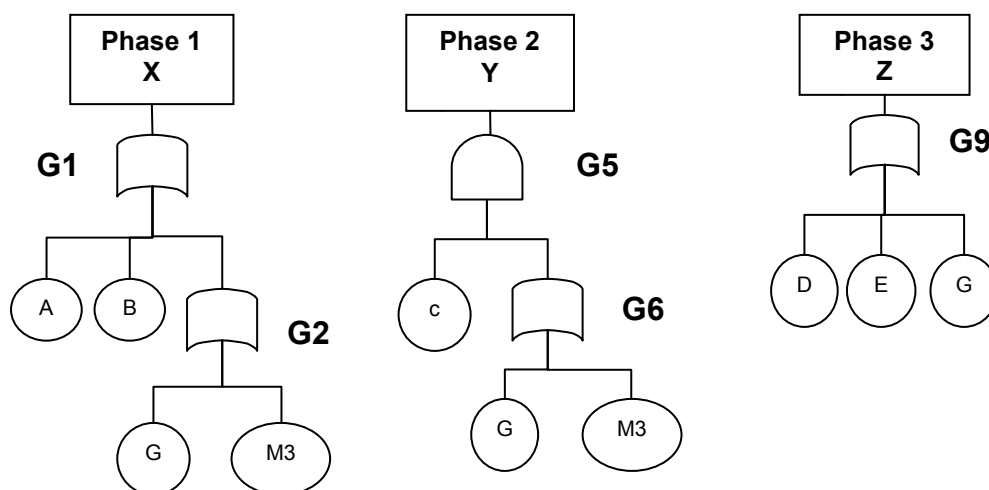


Figure 4.20: The phase fault trees after all the mission modules have been taken out for mission 1

## Phase failure probabilities for the modules of mission 1

So far each module has been taken out of the tree structure and treated as a single event. Now the module phase failure data must be calculated. It is calculated by treating the module as a mini phased mission with the logic being the same in every phase. A module's BDD can only be evaluated once all of the modules in that BDD have been evaluated. The example mission 1

contains two modules M3 and M4 which have BDDs that are represented by the nodes N22 and N20 respectively. First module M4 that is represented by node N20 is evaluated since it doesn't have any modules contained within it. Each phase BDD will be the same logic therefore the only difference will be variable labels which represent the failure in different phases. These are shown in figure 4.21.

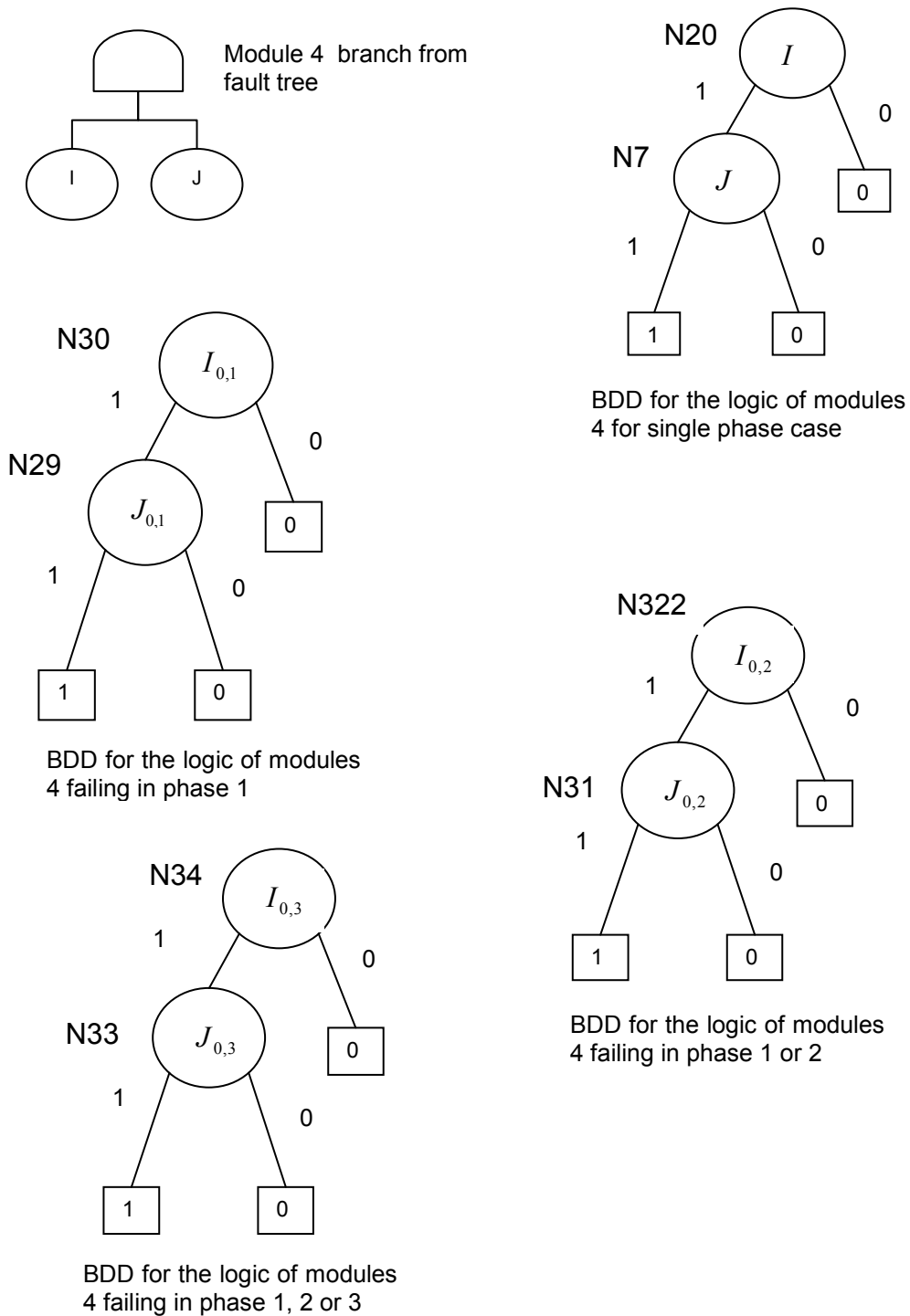


Figure 4.21: Phases BDDs for module 4

The phase module BDD failure probabilities are calculated by using equation 4.10 and the node probabilities are evaluated by the formulas 3.35 and 3.37 from Chapter 3 section 3.2.6. However the evaluation is simplified because the phase logic is identical for each phase so when they are ORed together then the results is just a logic expression of the last phase. The calculations for this procedure are shown below.

Probability of node 30:

$$\Pr( N_{30} = 1 ) = \Pr( I_{0,1} = 1 ) \cdot \Pr( J_{0,1} = 1 )$$

Probability of node 32:

$$\Pr( N_{32} = 1 ) = \Pr( I_{0,2} = 1 ) \cdot \Pr( J_{0,2} = 1 )$$

Probability of node 34:

$$\Pr( N_{34} = 1 ) = \Pr( I_{0,3} = 1 ) \cdot \Pr( J_{0,3} = 1 )$$

The probability of module 4 N20 fails in phase 1:

$$\Pr( N_{30} = 1 )$$

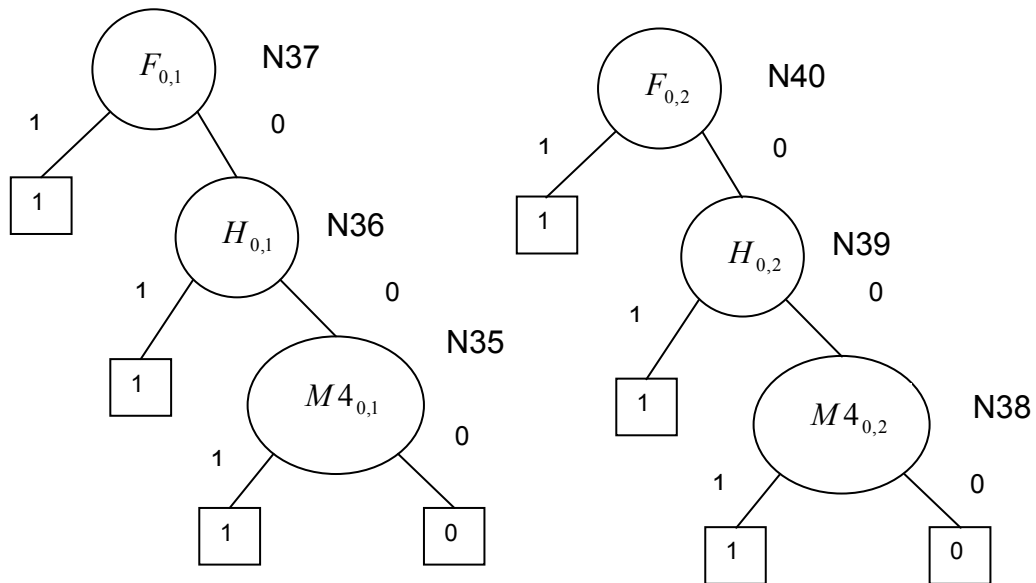
The probability of module 4 fails in phase 2:

$$\Pr( N_{32} = 1 ) - \Pr( N_{30} = 1 )$$

The probability of module 4 fails in phase 3:

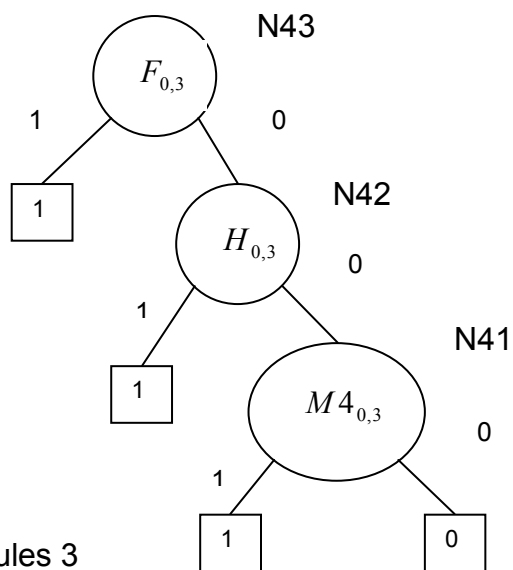
$$\Pr( N_{34} = 1 ) - \Pr( N_{32} = 1 )$$

Now module 3 can be evaluated since the module it contains has been evaluated. The same procedure is applied as when evaluating module 4. The phase BDDs for module three are shown in figure 4.22.



BDD for modules 3  
Failure in phase 1

BDD for modules 3  
Failure in phase 1  
or 2



BDD for modules 3  
Failure in phase 1  
or 2 or 3

Figure 4.22: Phases BDDs for module 3

The calculations for the probability of module 3 in the phases are shown below.

Probability of node 37:

$$\begin{aligned} \Pr( N_{37} = 1 ) = & \Pr( F_{0,1} = 1 ) + (1 - \Pr( F_{0,1} = 1 )) \cdot \Pr( H_{0,1} = 1 ) + \\ & (1 - \Pr( F_{0,1} = 1 )) \cdot (1 - \Pr( H_{0,1} = 1 )) \cdot \Pr( M_{4_{0,1}} = 1 ) \end{aligned}$$

Probability of node 40:

$$\begin{aligned} \Pr( N_{40} = 1 ) = & \Pr( F_{0,2} = 1 ) + (1 - \Pr( F_{0,2} = 1 )) \cdot \Pr( H_{0,2} = 1 ) + \\ & (1 - \Pr( F_{0,2} = 1 )) \cdot (1 - \Pr( H_{0,2} = 1 )) \cdot \Pr( M_{4_{0,2}} = 1 ) \end{aligned}$$

Probability of node 43:

$$\begin{aligned} \Pr( N_{43} = 1 ) = & \Pr( F_{0,3} = 1 ) + (1 - \Pr( F_{0,3} = 1 )) \cdot \Pr( H_{0,3} = 1 ) + \\ & (1 - \Pr( F_{0,3} = 1 )) \cdot (1 - \Pr( H_{0,3} = 1 )) \cdot \Pr( M_{4_{0,3}} = 1 ) \end{aligned}$$

The probability of module 3 fails in phase 1:

$$\Pr( N_{37} = 1 )$$

The probability of module 3 fails in phase 2:

$$\Pr( N_{40} = 1 ) - \Pr( N_{37} = 1 )$$

The probability of module 3 fails in phase 3:

$$\Pr( N_{43} = 1 ) - \Pr( N_{40} = 1 )$$

Now that all the modules have been taken out and quantified for the Phase failure probabilities. The rest of the method stages performed to obtain all the phase unreliabilities are: phase ordering components, building the individual phase BDDs and building and evaluating the phase failure BDD. This is

exactly the same has **method 2** presented in section 4.4 under the sub headings:

- 1) Phase ordering components
- 2) Building the individual phase BDDs
- 3) Building and evaluating the phase failure BDD

A second mission is demonstrated which is mission 2 and contains phases X and Y.

### Identify the mission modules for mission 2

Mission two contains phases X then Y. The gates G1 and G5 are not modules in this mission since they have phase contradictions. Therefore these two gates are put back in the fault trees which are then shown in figure 4.23.

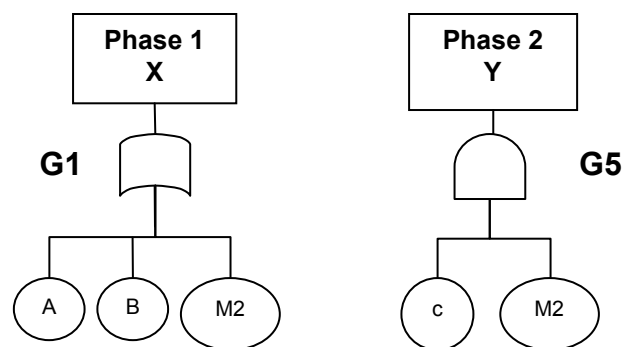


Figure 4.23: The phase fault trees after all the mission modules have been taken out for mission 2

### Phase failure probabilities for the modules of Mission2

The phase BDDs for module 2 are shown in figure 4.24. Module 3 is contained in module 2 and since the evaluation for this was discussed in the previous mission it will not therefore be repeated.

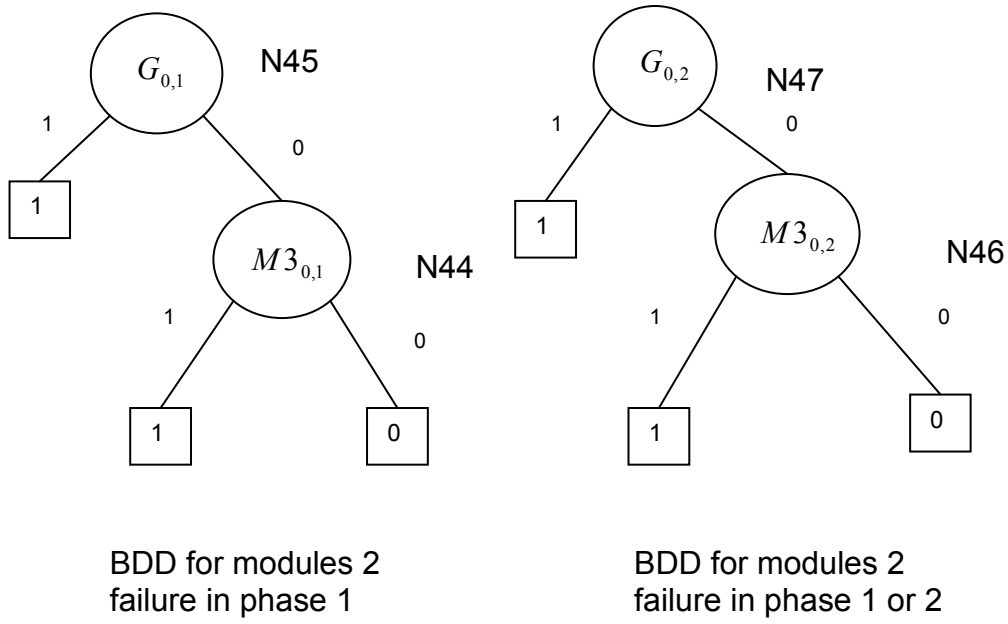


Figure 4.24: Phases BDDs for module 2

Following the same procedure as before, the calculations for the module probabilities in different phases are evaluated using the phase BDD's in figure 4.24:

Probability of node 45:

$$\Pr( N_{45} = 1 ) = \Pr( G_{0,1} = 1 ) + (1 - \Pr( G_{0,1} = 1 )) \cdot \Pr( M3_{0,1} = 1 )$$

Probability of node 47:

$$\Pr( N_{47} = 1 ) = \Pr( G_{0,2} = 1 ) + (1 - \Pr( G_{0,2} = 1 )) \cdot \Pr( M3_{0,2} = 1 )$$

The probability of module 2 fails in phase 1:

$$\Pr( N_{45} = 1 )$$

The probability of module 2 fails in phase 2:

$$\Pr( N_{47} = 1 ) - \Pr( N_{45} = 1 )$$



Now that the modules have been extracted and quantified for the Phase failure probabilities. The rest of method stages to obtain all the phase unreliabilities are: phase ordering components, building the individual phase BDDs and building and evaluating the phase failure BDD.

#### **4. 7 Results**

All of the methods presented in this chapter have been tested for the running time by analysing eight mission sets which are made up of from 5 to 9 possible phases. Information of the fault trees such as: structure characteristics, number of gates and events are shown with the fault trees, in file format, in appendix A. The missions are split into two groups. In the first group which consists of mission sets 1 to 4, each mission set contains an individual standard fault tree structure for all its possible phases and the events in these trees are randomly positioned. Therefore the fault trees do not represent any particular real system. The second group which also consists of 4 missions represent UAV missions made up of 5 to 8 possible phases.

For the first 4 mission sets, the information shown in the Appendix A includes a diagram of a fault tree structure which represents the phase failure for all possible phases in that mission. Also there is a table which contains information on the characteristics of the fault trees. The first column contains the phase number, the second the number of gates, the third the number of events including repeated events, the fourth the number of events not included the repeated events, and the fifth the number of common events such as if the event is contained within another phase. Also a second table contains the information of the number of modules, gates and events not including repeated ones in the entire mission. The last piece of information shown for the mission is the fault tree of the phases in data format. For missions 5 to 8, which represent UAV missions, information is shown in Appendix B. The information is the same as for missions 1 to 4 described above except that the fault trees in these missions do not have the same structure in each phase and therefore a standard structure diagram of the fault tree is not shown. The fault trees representing the phase failure in mission 5

are relatively small therefore they are shown in a diagram. The fault trees representing the phase failure in the missions 6-8 are large and therefore are just shown in data format. The subsystems which the fault trees are constructed from are shown in diagrams in Appendix F with a descriptions. The missions 6 to 8 are just one mission progressively getting more complex by expanding the tree branches that represent failure of the power supplies such as DC power, AC power and hydraulic power. In mission 6 the power supplies are represented by basic events and in missions 7 and 8 they are represented by sub fault trees that are complex.

First missions are analysed using **Method 1** (section 4.3), **Method 2** (section 4.4) and **Method 3** (section 4.6) for a comparison. The run times of the performance are shown in table 4.10. The first column contains which mission set being analysed, the second the mission configuration, the third the online run time of **Method 1**, the fourth the number of modules taken out of the mission, the fifth which is split into two, the online and off-line run times of **Method 2** and the sixth which is also split into two, the online and off-line run times of **Method 3**.

Comparing the online times of **Method 2** and **Method 1** shows that for the mission configurations from set 5 improvement increase dramatically as the number of phases in the configuration increases. For example, as the number of phases in the mission increase to 18 the online running time was hundreds of times faster. As the number of phases in the mission increases to 25 analysis times are thousands of times faster for **Method 2** compared to **Method 1**. This significant improvement is because the mission set is from a UAV mission and therefore has a structure featuring a lot of common logic between the phases. The common logic is an advantage for **Method 2** since it deals with its dependencies at the building stage of the BDD's. Also contributing to this improvement are aspects of looking up any BDD's computation that has already been done. This will be more advantageous where there is more common logic between the phases. **Method 1** takes significantly longer to calculate these missions as the number of phases increases since every time a phase is added the number of tracing operations

in the BDD analysis dramatically increases. For the missions from mission set 1 **Method 2** did not out perform **Method 1** as it did in the missions from set 5. These fault trees were from a random mission set and, as such, had a different type of structure which did not feature a lot of common logic between them. Therefore the functions such as looking up the previously performed computations of BDDs operations and nodes did not substantially reduce the time of calculation. The two methods generally performed the same however when the number of phases within the mission increased to five **Method 2** ran out of memory because of the amount of BDD lines created and the computation operations to be stored for the lookup function. However **Method 1** does not suffer from this since it evaluates a path at a time and then deletes it.

Comparing **Method 2** with **Method 3** using the missions from set five there was not that much difference in the running time calculation since they were small fault trees and therefore they were not large enough for a proper comparison. From the 7 missions from mission set 1, four of them were faster on **Method 3**. However there was not a significant difference in the two methods performance since the fault trees are random and depending on which phases are picked for the missions the number of modules and their size will be different. The off-line analysis times for these missions was mainly the management of the data storage for the code and therefore as the mission size increases the memory increases and the off-line time increases.

Mission set	Mission Configuration	Method 1	No modules	Method 2		Method 3	
				online	offline	online	offline
5	1,2,3,4,5,6	0.06s		0.03s	(0.06s)	0.03s	(1.66s)
5	1,4,3,2,3,5,6,1	0.28s	4	0.05s	(0.08s)	0.05s	(1.66s)
5	1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6	31.66	4	0.13s	(0.09s)	0.13s	(1.72s)
5	1,5,4,6,2,1,3,2,3,4,5,3,1,3,5,6,3,3,2,1,2,3	2min 20.9s	4	0.16s	(0.11s)	0.17s	(1.75s)
5	1,5,4,6,2,1,3,2,3,4,5,3,1,3,5,6,3,3,2,1,2,3,1,2,3,4	4min 56.29s	4	0.19s	(0.14s)	0.22s	(1.75s)
5	1,5,4,6,2,1,3,2,3,4,5,3,1,3,5,6,3,3,2,1,2,3,1,2,3,4,2,3,4,5,3,5,6,3	30min 31.1s	4	0.30s	(0.16s)	0.29s	(1.76s)
1	1,4,5	8.8s	9	8.45s	0.85s	1.02s	1.83s
1	6,2,3	0.66s	2	4min 57.53s	4.39s	1min50.29s	4.44s
1	4,3,1	3.4s	5	9.63s	0.88s	2.23s	1.41s
1	2,1,5	2.3s	6	0.08s	0.17s	0.84s	1.19s
1	2,3,4,1	54.33s	5	2min 56.2s	3.45s	4min 16.04s	7.86s
1	5,6,1,2	2min 29.1s	3	1min 52.24s	2.88s	1min 35.85s	7.88s
1	2,3,6,1	42.8s	1	24min 56.84s	9.61s	24min 17.06s	15.62s
1	3,1,6,4,2	2hr 54min 8.6s	3	3 hour 9min 027s	9.61s	3 hour 1min 45.49s	37.08s
1	5,1,4,2,6	6hr25min7.5s	4	1 hour 32 min 47.43s	19.36s	1 hour 56min 26.2s	31.64s

Table 4.10: Online and offline running times for **Methods 1,2 and 3** on mission set 1 and 5

Mission set	Mission Configuration	No modules	Method 2		Method 3	
			online	offline	online	offline
1	1,2,3	2	1.75s	0.41s	1.97s	1.31s
1	1,2,3,4	5	4min 2.15s	3.89s	5min 13.8s	9.38s
1	1,2,3,4,5		35min 52.69s	9.36s	1 hour 19min 56.24s	63.71s
1	1,2,4	9	0.29s	0.27s	0.53s	1.19s
1	1,2,4,6	7	33.82s	1.55s	35.51s	3.38s
1	1,2,4,6,5	4	1 hour 25min 6.99s	20.94s	1 hour 14min 50.00s	25.84s
1	2,4,6	10	12.64s	0.98s	3.52s	2.03s
1	2,4,6,5	5	1hour 13min 4.15s	17.12s	42min 56.05s	20.27s
1	2,4,6,5,1	4	1hour 15min 9.51s	17.13s	1hour 14min 10.45s	24.09s
1	2,3,5	4	12.42s	0.78s	29.61s	2.53s
1	2,3,5,1	4	14.5s	0.88s	33.13s	2.52s
1	2,3,5,1,4	6	3min 34.94s	3.70s	13min 55.62s	12.91s
1	1,2,7	4	0.86s	0.29s	1.05s	1.30s
1	1,2,7,3	3	20min 33.37s	8.63s	29min 31.61s	17.17s
1	3,4,8	7	M	M	19min 45.49s	14.49s

Table 4.10 continued: Online and offline running times for **Methods 2 and 3** on mission set 1

22 Missions were analysed by **Methods 2** and **3** from mission sets 2, 3 and 4, the results of which are shown in tables 4.11, 4.12 and 4.13. Five online running times of these missions were the same, 13 were faster for **Method 2** and 4 was a faster **Method 3**. When one method was faster than the other it was however not significantly faster than the other method (less than a factor of 2). The number of modules extracted ranged from 1 to 11. The lack of improvement by taking out modules was because the fault trees from the mission did not feature characteristics which enabled the extraction of modules that were large enough for it to result in significant improvements. This is down to the randomness of the trees.

The extra time that **Method 3** took to perform the analysis was down to the extra calculations for applying a modularized method which because of the tree characteristics did not prove to be worth the investment.

Mission set	Configuration	No modules	Method 2		Method 3	
			online	offline	online	offline
2	1,2,3	7	0.06s	0.11s	0.86s	2.58s
2	1,2,3,4	6	14.14s	1.03s	2min 31.73s	6.48s
2	1,2,5	11	0.03s	0.09s	0.44s	0.56s
2	1,2,5,6	7	1.23s	0.38s	8.55s	3.69s
2	1,2,5,6,3	6	1min 35.6s	2.42s	23min 21.74s	15.45s
2	3,4,5,6	7	M	M	2min 50.59s	6.52s

Table 4.11: Online and offline running times for **Methods 2** and **3** on mission set 2

Mission set	Configuration	No modules	Method 2		Method 3	
			online	offline	online	offline
3	1,2,3	2	0.01s	(0.08s)	0.01s	1.51s
3	1,2,3,4	2	0.03s	(0.09s)	0.03s	1.52s
3	1,2,3,4,5	3	0.09s	(0.13s)	0.11s	1.86s
3	1,2,3,4,5,6	1	1.64s	(0.36s)	1.92s	1.89s
3	1,2,3,4,5,6,1,2	1	2.30s	(0.44s)	2.40s	1.94s
3	1,2,3,4,5,6 1,2,3	1	3.36s	(0.55s)	6.63s	2.16s
3	1,2,3,4,5,6 1,2,3,4	1	8.92s	(0.78s)	18.44s	2.16s
3	1,2,3,4,5,6 1,2,3,4,5	1	19.63s	(1.06s)	33.05s	2.95s
3	1,2,3,4,5,6 1,2,3,4,5,6	1	32.36s	(1.50s)	48.31s	3.22s
3	1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6	1	1min 31.41s	(2.28s)	2min 29.75s	4.48s

Table 4.12: Online and offline running times for **Methods 2** and **3** on mission set 3

Mission Set	Configuration	No modules	Method 2		Method 3	
			online	offline	online	offline
4	1,2,3	0	0.14s	(0.13s)	0.17s	(1.73s)
4	1,2,3,4	0	1.79s	(0.38s)	2.03s	(2.00s)
4	1,2,3,4,5	1	29.14s	(1.39s)	32.31s	(3.17s)
4	1,2,3,4,5,1,2,3	1	4min 38.23s	(4.09s)	3min 23.43s	(5.38s)
4	1,2,3,4,5 1,2,3,4,5	1	10min 46.51s	(6.16s)	8min 0.94s	(7.89s)
4	1,2,3,4,5 1,2,3,4,5 1,2,3,4,5	1	33min 41s	(11.27s)	23min 54s	(11.41s)

Table 4.13: Online and offline running times for **Methods 2** and **3** on mission set 4

Mission set six represents a UAV mission and the results are shown in table 4.14. The results showed that there is no significant difference between the performance of **Method 2** and **3** since the modules are not large enough to make a big impact even though there are a lot modules identified. So basic events representing failure of the energy supplies were replaced with sub trees as shown in appendix B. This develops a new mission set which is referred to as mission set seven. Results of the analysis are shown in table 4.15. The results show that as the number of phases increases **Method 3** performs significantly better due to the modularization. Finally the DC power supply sub tree was replaced with an even larger and more complex sub tree in the mission, referred to as mission set 8. The results of the mission analysis are shown in table 4.16. Again the results show that by taking out larger and more complex modules it has a significant impact on the improvement offered by the modularized **Method 2**. For example the last mission of mission set 8 took 3 min 53.99s on **Method 2** and 1.08s on **Method 3** which is a sufficient improvement. Approximately the time saved by modularization on the online time is just shifted to the off-line time.



Mission	Configuration	No modules	Method 2		Method 3	
			online	offline	online	offline
6	1,2	18	0.02s	(0.09s)	0.001s	(4.33s)
6	1,7	77	0.01s	(0.11s)	0.02s	(4.28s)
6	3,5	77	0.01s	(0.13s)	0.001s	(4.31s)
6	3,4	85	0.02s	(0.14s)	0.016s	(4.27s)
6	3,7	84	0.01s	(0.13s)	0.016s	(4.28s)
6	5,4	90	0.03s	(0.11s)	0.001s	(4.28s)
6	6,7	89	0.01s	(0.14s)	0.01s	(4.25s)
6	4,7	97	0.02s	(0.13s)	0.01s	(4.28s)
6	1,2,8	27	0.01s	(0.34s)	0.02s	(4.31s)
6	1,2,3	51	0.01s	(0.13s)	0.02s	(4.27s)
6	1,3,6	85	0.03s	(0.13s)	0.01s	(5.06s)
6	1,3,5,4	133	0.03s	(0.14s)	0.02s	(4.33s)
6	3,6,4,7	172	0.06s	(0.14s)	0.03s	(4.33s)
6	3,5,6,4,7	214	0.06s	(0.16s)	0.03s	(4.50s)
6	1,2,3,5,6,4,7	230	0.08s	(0.17s)	0.05s	(4.47s)
6	1,2,3,4,5,6,7,8,9	246	0.11s	(0.22s)	0,06s	(4.48s)
6	1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9	246	0.36s	(0.41s)	0.31s	(4.97s)
6	1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9	246	0.73s	(0.58s)	0.73s	(5.50s)

Table 4.14: Online and offline running times for **Methods 2** and **3** on mission set 6

Mission	Configuration	No modules	Method 2		Method 3	
			online	offline	online	offline
7	1,2,3,4,5,6,7,8,9	531	9.30s	(3.27s)	0.09s	(5.56s)
7	1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9	531	11.48s	(3.88s)	0.45s	(6.92s)
7	1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9	531	20.92s	(5.28s)	1.09s	(7.61s)

Table 4.15: Online and offline running times for **Methods 2** and **3** on mission set 7

Mission	Configuration	No modules	Method 2		Method 3	
			online	offline	online	offline
8	1,2,3,4,5,6,7,8,9	661	27.91s	(4.48s)	0.11s	(7.44s)
8	1,2,3,4,5,6,7,8,9 1,2,3,4,5,	661	1min 1.53s	(9.08s)	0.27s	(8.17s)
8	1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9	661	1min 39.02s	(13.00s)	0.48s	(8.77s)
8	1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9 1,2,3,4,5,	661	2min 46.73s	17.66s)	0.77s	(9.29s)
8	1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9 1,2,3,4,5,6,7,8,9	661	3min 53.99s	(21.36s)	1.08s	(9.50s)

Table 4.16: Online and offline running times for **Methods 2** and **3** on mission set 8

## Conclusion

The comparison of 3 methods has been performed by the analysis of 8 mission sets that are separated into two groups containing randomly generated missions and those representing a UAV mission structure. The results showed that **Method 2** Alternative Trivedi and **Method 3** that has modularization significantly out-performed **Method 1**. However **Method 1** is more memory efficient and has a better chance of providing an answer. The Modularized **Method 3** compared to the Non-Modularized **Method 2** is only better where complex modules can be taken out which would take a significant amount of time to calculate online even though it will take a long time. Modularization shifts the analysis time to off-line. This is shown in the results as the mission experiments increased in module complexity then the improvement increases with it. However if fault trees in a mission do not feature this characteristic then the modularized method does not yield sufficient improvement as was shown from the random mission sets 1-4 which performed badly since they did not have these characteristics.

## Chapter 5: Investigating the distribution of analysis time spent on recursive functions

### 5.1 Introduction

Programming recursive functions can be very computationally intensive since they can call their self many times. It is very important to consider how they will be handled when aiming to do a fast analysis. Therefore this section will investigate the amount of time taken up by iterative functions in the codes of the methods discussed so far.

There are two recursive functions in **Methods 2** and **3**. The first recursive function is applied for the qualitative analysis and consists of building the BDDs that represent the failure logic. The second recursive function is applied for the quantitative analysis that consists of evaluating the BDDs. This recursive function is less computationally intensive compared to first recursive function for the qualitative analysis. The number of the times the quantitative recursive function is performed is the number of BDD nodes and will only take several seconds. As such, as the analysis tackles missions with larger fault trees the amount of time taken up by this the quantitative iteration function will not grow huge compared to the rest of the time taken up by other calculations. However the number of qualitative recursive function applications performed is not necessarily the number of nodes in the final BDD but the number in generating the final BDD. This could result in the recursive function for the qualitative analysis taking up large amounts of analysis time. Therefore this chapter will only investigate the recursive function for the qualitative analysis since this has a greater impact on the time of analysis.

### 5.2 Storing the BDD in the code

A BDD is stored in the program as an array of elements that represents a node in the BDD it consists of three integers. For example, figure 5.1 shows a small

BDD consisting of three nodes and how it is stored in file format. The first integer represents the component in the node, the second array entry is the element in the array that represents the node branching off to the left (1-branch), the third array entry is the element in the array that represents the node branching off to the right (0-branch). For representing the terminal failure node 1, -1 is use since 1 could mean the first element of the array.

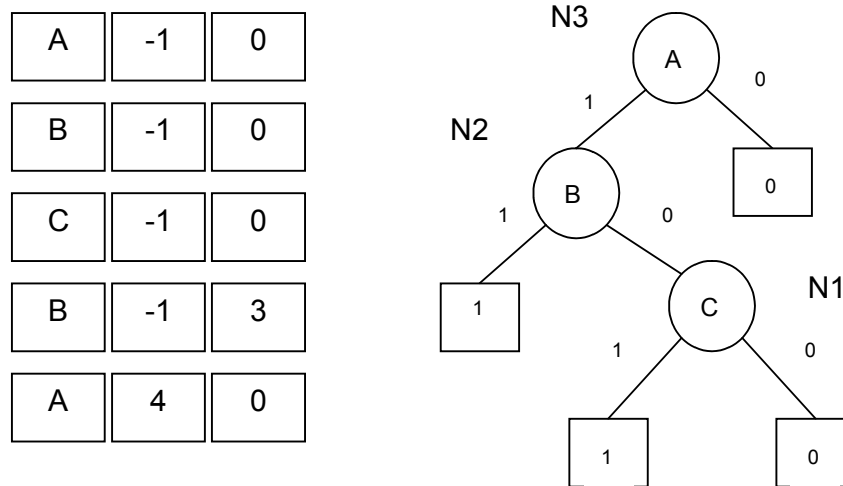


Figure 5.1: BDD in file format

### 5.3 Recursivefunction for the qualitative analysis

A.Rauzy [3] presents a recursive function for the logic operators AND and OR for combining BDDs together which are applied in all the codes of the methods in this thesis. This function is shown in figure 5.2 and will be expanded in this section. As described above the BDDs are stored in an array in the code. So the function parameters are the elements entered for the two nodes of the array which are going to be combined by some logical operator which are referred to as node 1 and node 2. The operator is also a parameter and can be OR or AND. The function returns the elements for the entry for the node which

is a result of the node 1 and 2 be combined together. The function performs the computation of node 1 and node 2 in at most seven stages as follows:

- 1) Checks to see if node 1 and node 2 are equal. If so the function terminates and node1 is return as the result. This is shown on line 4.
- 2) Checks to see if either nodes 1 or 2 are terminal nodes if so the function terminates and returns the Boolean logic of the node 1 and 2 with operator which will be 1, 0, node 1 or node 2 . This is shown between lines 6 to 10.
- 3) Look up to see if the computation has already be done since all the computations are recorded. If so the function terminates and returns the look up result. This is shown between lines 13 to 15.
- 4) This stage is the fundamental stage it obtains the new node by calling itself. This is the ite operation described in chapter 2 section 2.7.3. This is shown between lines 18 to 38.
- 5) Checks to see if the left and right branches of the new node are equal if this is true then the function terminates and returns the left branch node. This is shown between lines 40 and 41.
- 6) Search through all the elements in the array of all nodes created to see if the new node already exists. If it does then the function terminates and returns the already existing node.
- 7) This stage records the computation and returns the new node.

```

1 node_new computation(op, node_1 , node_2)
2 {
3
4     if (node_1 == node_2) return node_1;
5
6     if (node_1 ==0 || node_1 ==-1 || node_2==0 || node_2==-1)
7     {
8         if (op==1) return (node_1 | node_2);
9         else return (node_1 & node_2);
10    }
11    else
12    {
13        int x;
14        if ((x=search_new_compute(op, node_1, node_2))!=0)
15            return x;
16        else
17        {
18            int U, V;
19            if (component_of_node_1==component_of_node_2)
20            {
21                U=computation(op,branch_1_node_1, branch_1_node_2);
22                V=computation(op, branch_2_node_1, branch_2_node_2);
23                x=ite[F][0];
24            }
25            else
26            {
27                if (component_of_node_1< component_of_node_2)
28                {
29                    U=computation(op, branch_1_node_1, node_2);
30                    V=computation(op, branch_2_node_1, node_2);
31                    x= component_of_node_2
32                }
33                else
34                {
35                    U=computation(op, node_1, branch_1_node_2);
36                    V=computation(op, node_1, branch_2_node_2);
37                    x= component_of_node_1;
38                }
39            }
40            if (U==V)
41                return U;
42            else
43            {
44
45                if (U==-1 && V==0) return x;
46
47                int p=R, find=0;
48                while (find==0 && p>0)
49                {
50                    if (branch_1_node_p==U && branch_2_node_p==V &&
51                        component_of_node_p==x)
52                {
53                    record_computation_outcome(); return node_p;
54                }
55                p--;
56            }
57            component_of_node_new=x;
58            branch_1_node_new=U;
59            branch_2_node_new=V;
60            record_computation_outcome(): return node_new;
61        }
62    }
63 }
64 }
65 }

```

Figure 5.2: Function for operating BDD nodes

The final number of elements in the array are not necessary the number of nodes in the BDD. There are a number of nodes that are the result of intermediate calculation used to create the final BDD. All these nodes are added to the array. A problem with the method is that the number of nodes stored can become vast which causes a memory management allocation problem. For every node has an allocation in the array of the BDD but it does not take up the majority of the memory used by the code. There is another array which takes up more. Every computation that is performed is recorded. Every node computation is assigned to an element in an array that stores the results of the computation. Even though this has the disadvantage of taking up a lot of memory, the advantage is that it enables quick check to see if the computation has already been performed.

#### **5.4 Modifying the recursive function for the qualitative analysis for PMS nodes**

The modification to the recursive function for the computation combining two phase nodes together using the Trivedi formula shown in equations 3.31 and 3.32 in chapter 3 are implemented by replacing the lines of code 19 to 39 in figure 5.2 with the piece of code shown in figure 5.3.



```

if (component_of_node_1==component_of_node_2)
{

    if (component_order_of_node_1==component_order_of_node_2)
    {
        U=computation(op,branch_1_node_1, branch_1_node_2);
        V=computation(op,branch_2_node_1, branch_2_node_2);
        x= component_order_of_node_1
    }
    else
    {
        if (component_order_of_node_1< component_order_of_node_2)
        {
            U=computation(op, branch_1_node_1, node_2);
            V=computation(op, branch_2_node_1, branch_2_node_2);
            x= component_of_node_1
        }
        else
        {
            U=computation(op, node_1, branch_1_node_2);
            V=computation(op, branch_2_node_1, branch_2_node_2);
            x= component_of_node_2;
        }
    }
}

}
else{

    if (component_order_of_node_1==componen_order t_of_node_2)
    {
        U=computation(op,branch_1_node_1, branch_1_node_2);
        V=computation(op, branch_2_node_1, branch_2_node_2);
        x= component_order_of_node_1
    }
    else
    {
        if (component_order_of_node_1< componen_order t_of_node_2)
        {
            U=computation(op, branch_1_node_1, node_2);
            V=computation(op, branch_2_node_1, node_2);
            x= component_of_node_1
        }
        else
        {
            U=computation(op, node_1, branch_1_node_2);
            V=computation(op, node_1, branch_2_node_2);
            x= component_of_node_2;
        }
    }
}
}

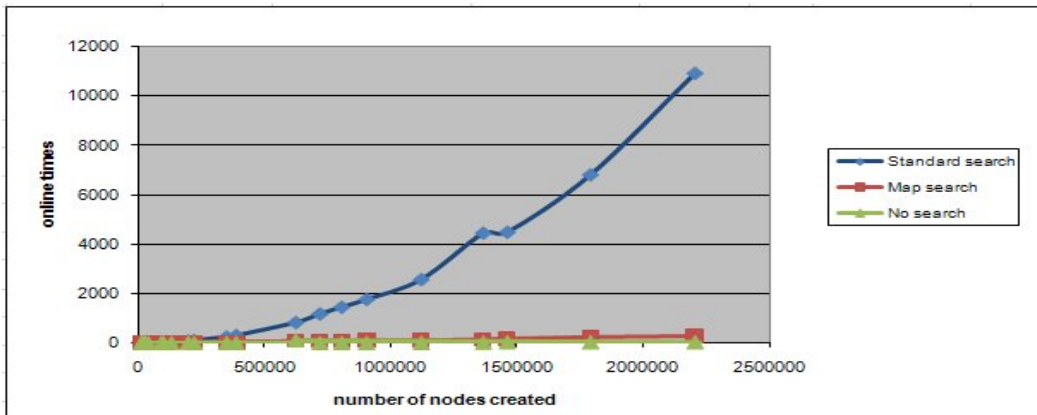
```

Figure 5.3 :function for operating BDD for Phases nodes

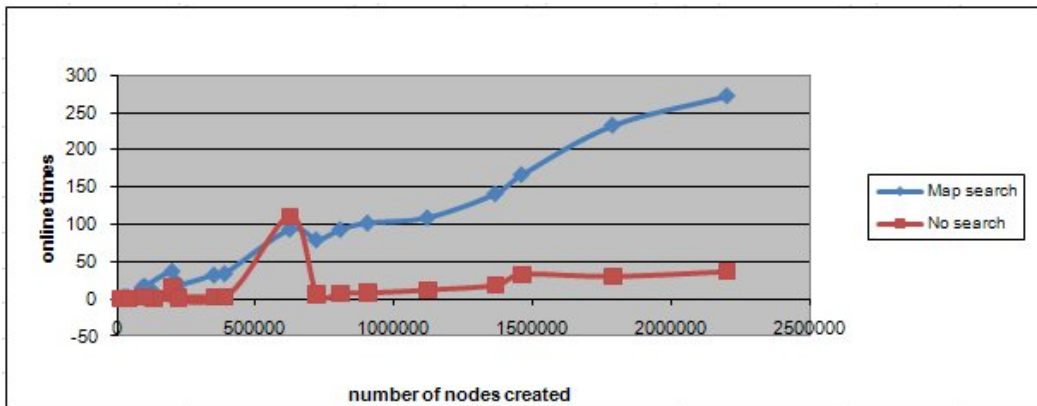
## 5.5 Investigating the analysis time taken for the nodes search function

Out of the seven stages of the computation for combining two nodes discussed above, stage 6 generally takes longer than any other stage. This is because it involves searching through the entire list of nodes that have already been created to check if the new node already exists. The time spent on stage 6 will depend on how many nodes to search through to find the node if it already exists and how many times stage 6 is performed in the analysis. As the analysis proceeds the number of nodes to search increases since new entries are continually added. This can take up a significant amount of time. This is investigated by running **Method 2** and **3**, that are based on the Trivedi method one with modularization and the other without, on 86 missions from 8 different mission sets, shown in appendix A and B, with a standard node search going through each element of the array one by one, without the search of existing nodes, and one with a standard C++ map to assist the search which is a sorted associative array of unique keys and associated data. Relative measurements are recorded and are shown in tables in appendix C for **Method 3** and appendix D for **Method 2**. The first column recorded the mission set the phases belong to. The second column contains the mission configuration. The online and offline times are recorded to compare the performance of the alternative approaches and are shown in columns three to five. In the upper half of the cell contains the online time and lower half contains the offline time. The total number of BDD nodes that are created to perform the analysis with an existing node search and without are recorded in column 6 and 8. The number of successful searches for existing nodes is recorded in column 7. There are two measurements shown for the number of nodes that are created and the number of successful searches for nodes that have already been created. The first number which is contain in the upper half of the cell represents the number of nodes or successful searches for when the failure event occurs in a phase BDD and number in the lower half of the cell is the number of nodes or successful searches for all the individual phase failure BDDs. Generally the measurement for the failure occurring in a phase BDD is much higher than the one for all the individual phase failure BDDs because the combined phase BDDs are more complex than single phase BDDs.

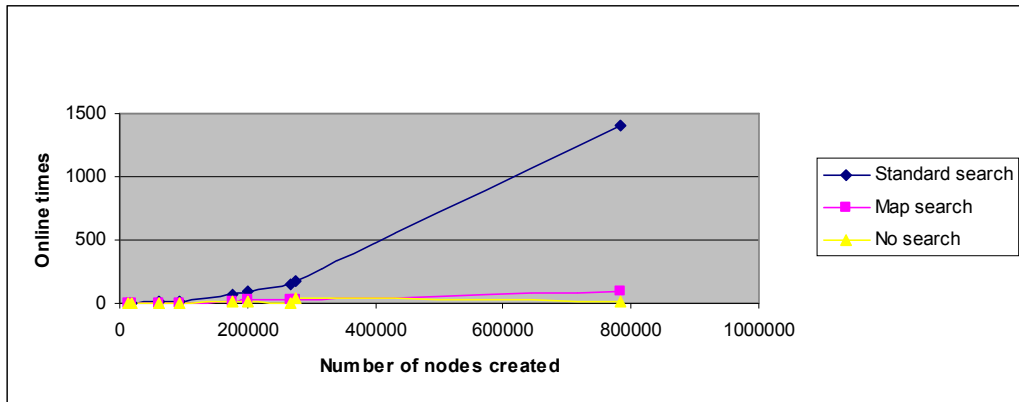
The data for the online times of the analysis of these three approaches are plotted against the size of the analysis, defined by the number of nodes created. For each mission set the data from **Method 3** is plotted in graphs 5.1 to 5.9 and also the data from **Method 2** is plotted in graphs 5.10 to 5.19 .



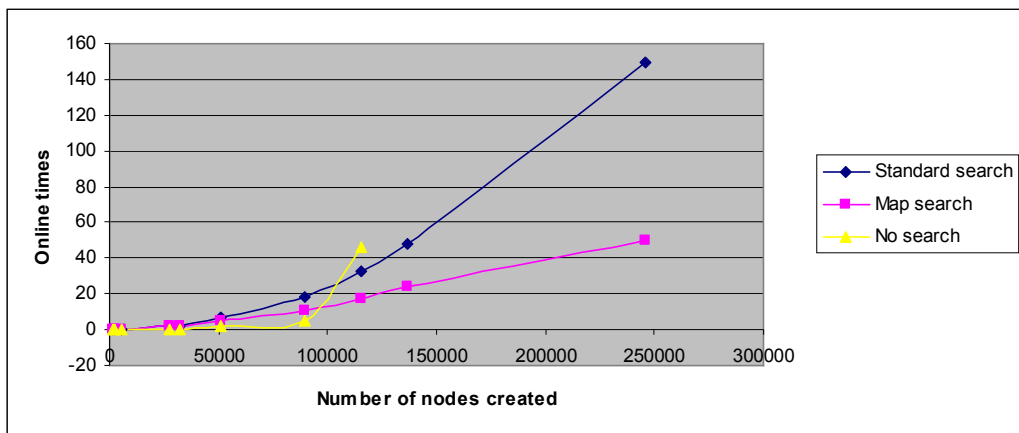
Graph 5.1: Online times of **Method 3** on set 1 with standard search, map search and no search



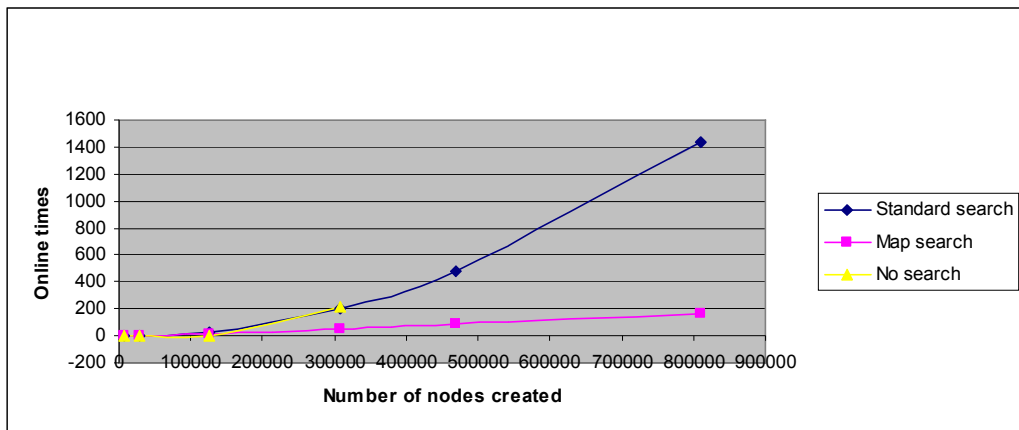
Graph 5.2: Online times of **Method 3** set 1 with map and no search



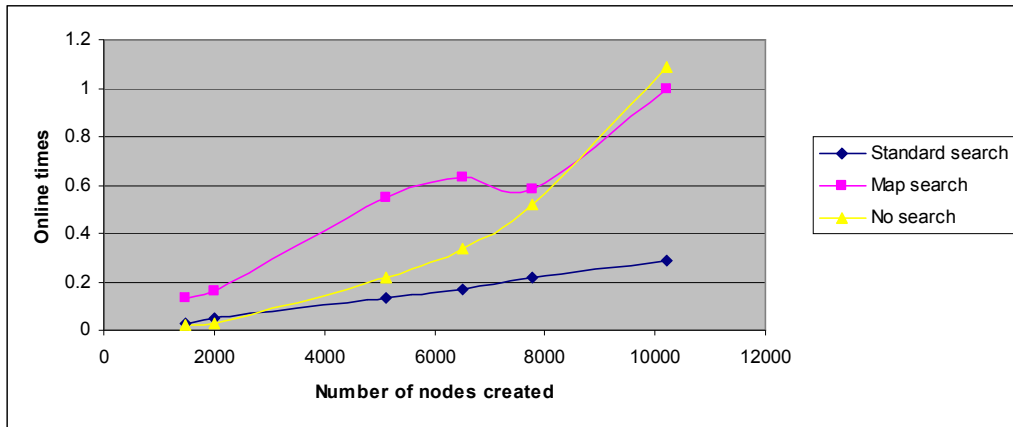
Graph 5.3: Online times of **Method 3** on set 2 with standard search, map search and no search



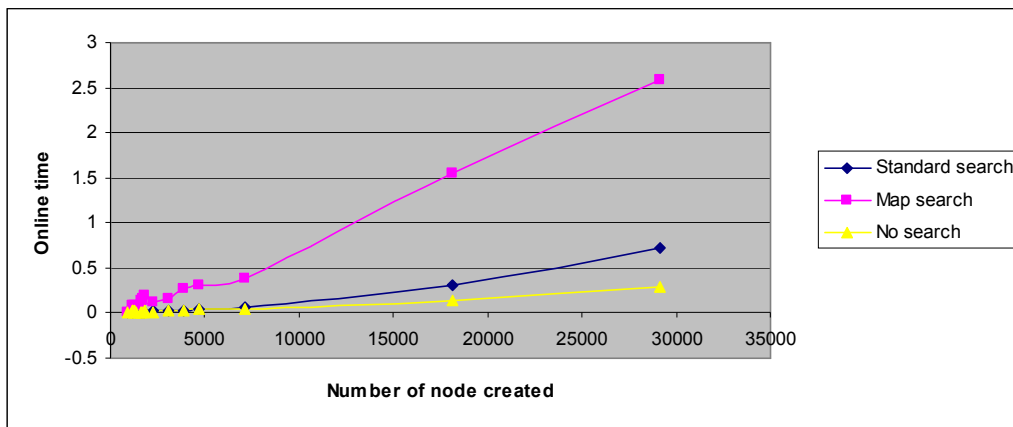
Graph 5.4: Online times of **Method 3** on set 3 with standard search, map search and no search



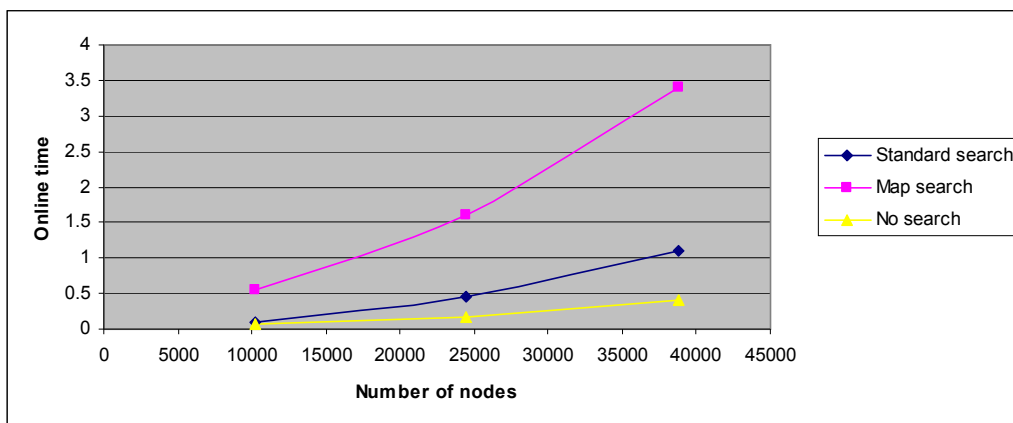
Graph 5.5: Online times of **Method 3** on set 4 with standard search, map search and no search



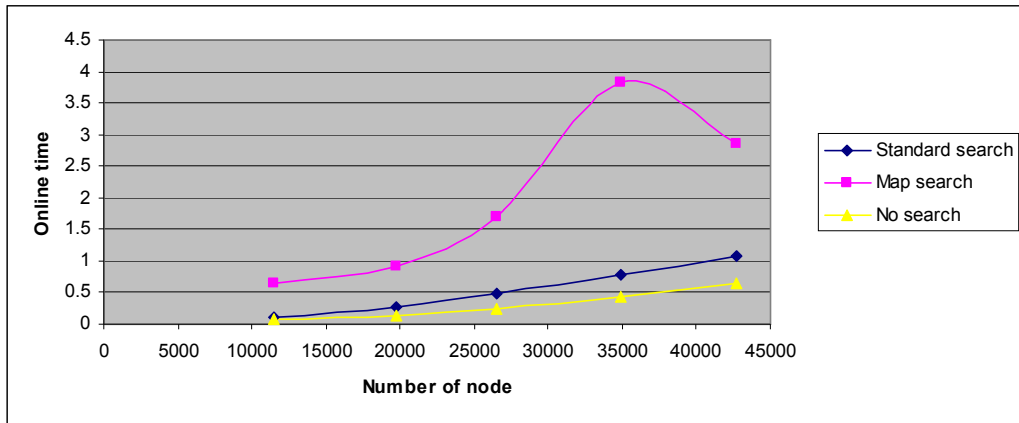
Graph 5.6: Online times of **Method 3** on set 5 with standard search, map search and no search



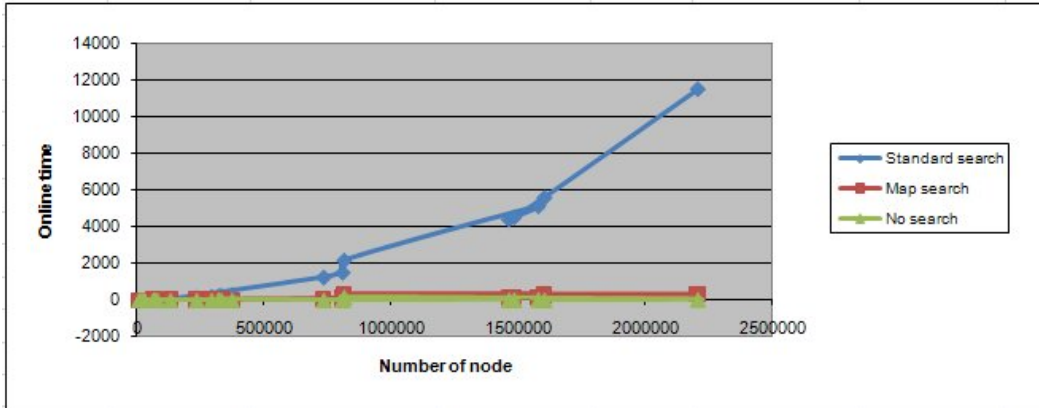
Graph 5.7: Online times of **Method 3** on set 6 with standard search, map search and no search



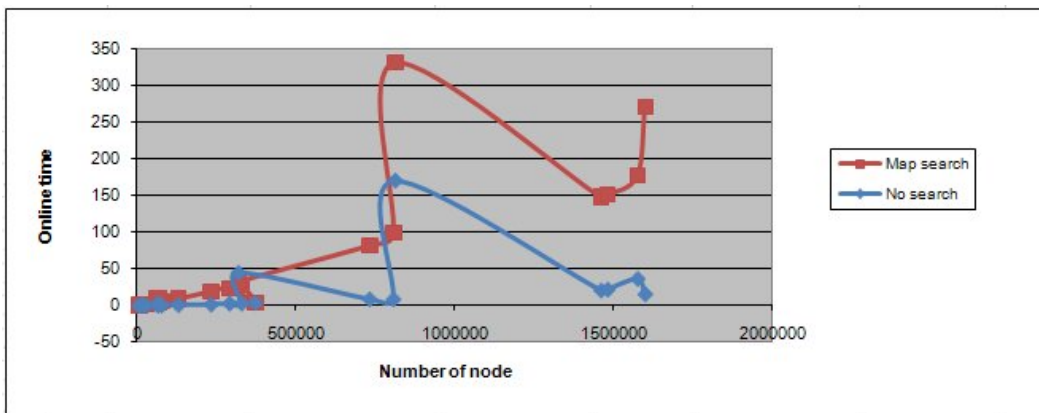
Graph 5.8: Online times of **Method 3** on set 7 with standard search, map search and no search



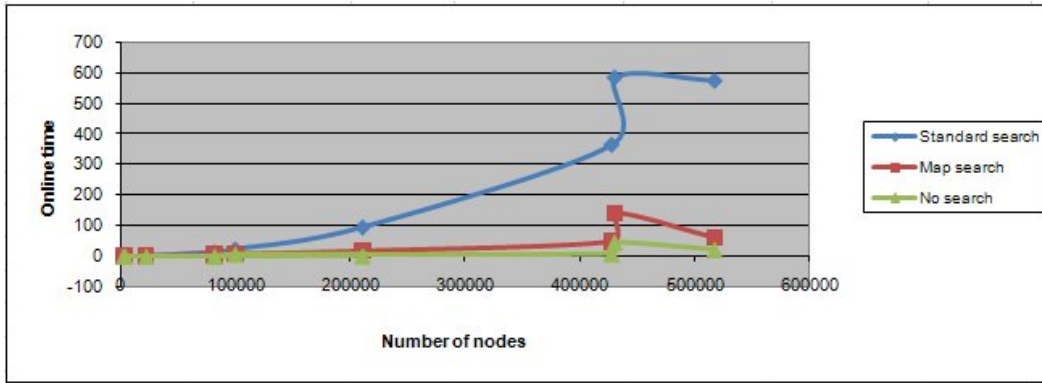
Graph 5.9: Online times of **Method 3** on set 8 with standard search, map search and no search



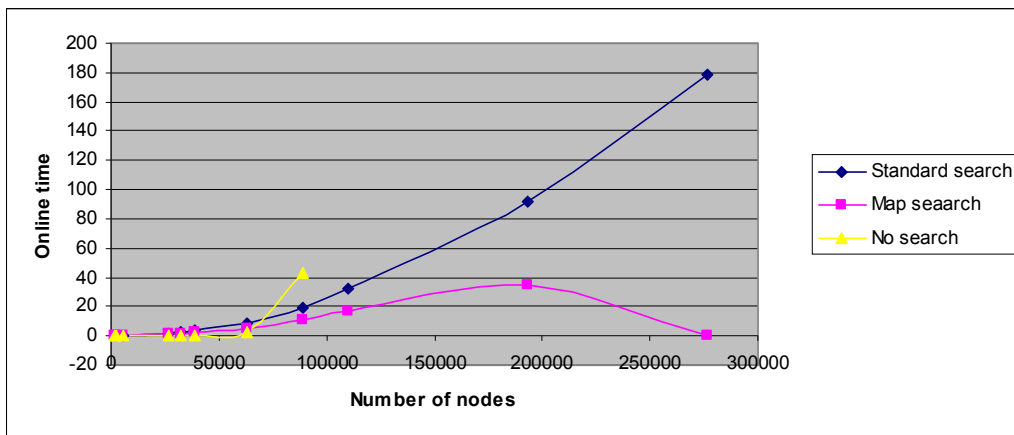
Graph 5.10: Online times of **Method 2** on set 1 with standard search, map search and no search



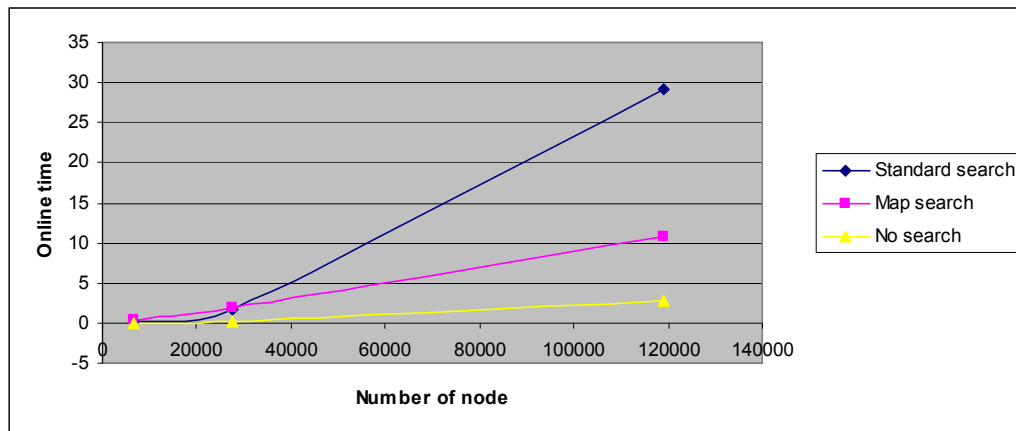
Graph 5.11: Online times of **Method 2** on set 1 with map search and no search



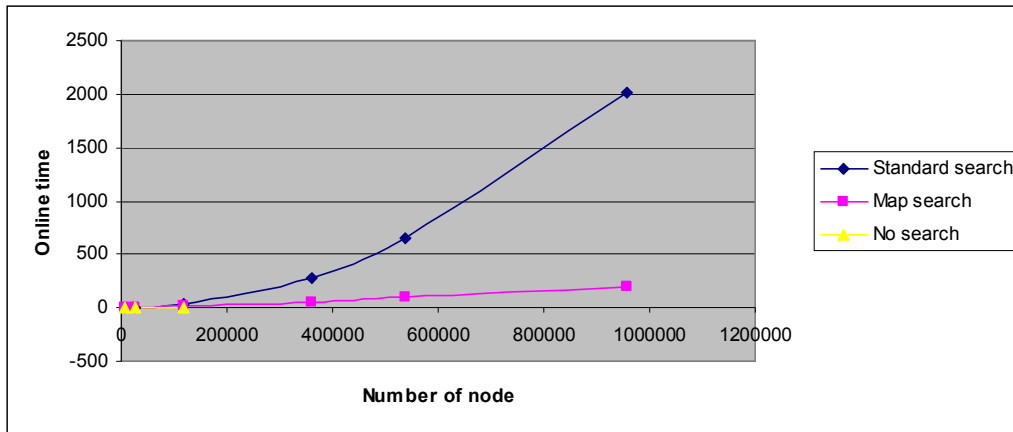
Graph 5.12: Online times of **Method 2** on set2 with standard search, map search and no search



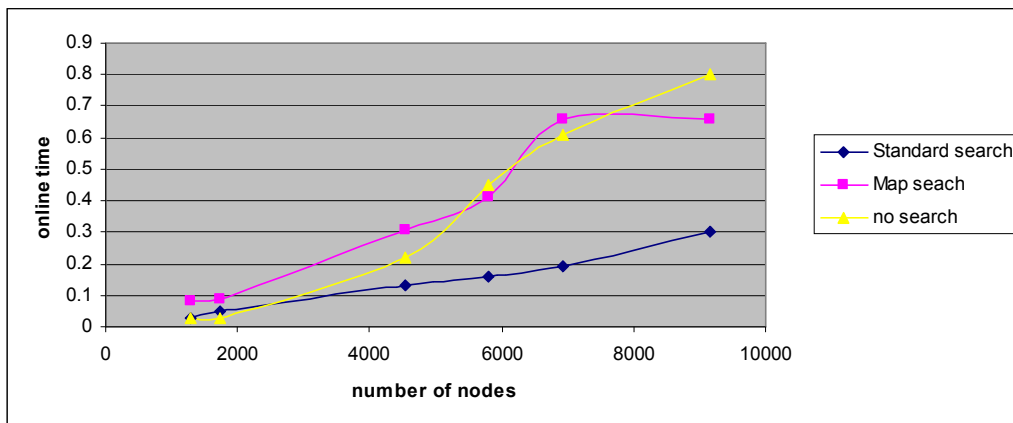
Graph 5.13: Online times of **Method 2** on set 3 with standard search, map search and no search



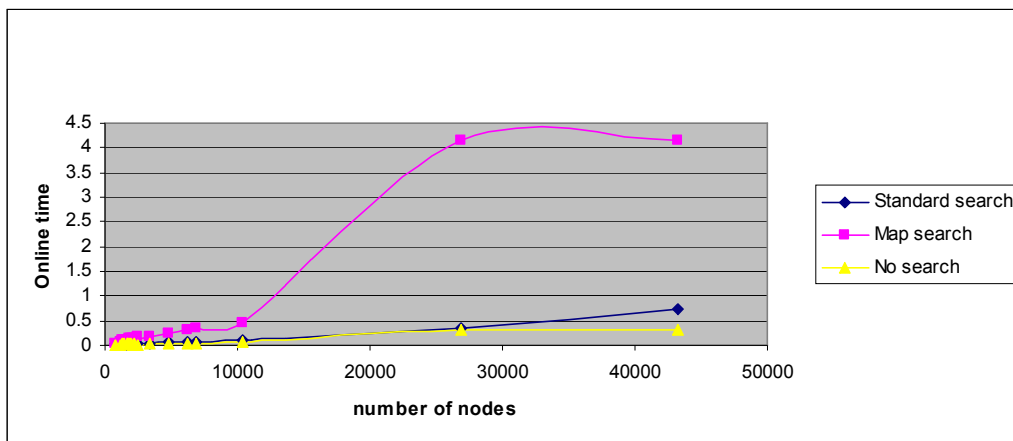
Graph 5.14: Online times of **Method 2** on set 4 with standard search, map search and no search zoom in



Graph 5.15: Online times of **Method 2** on set 4 with standard search, map search and no search zoom out

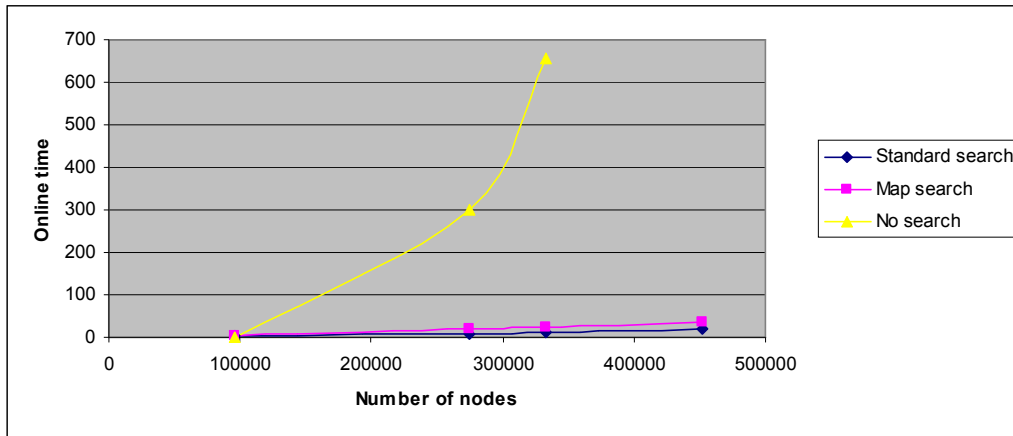


Graph 5.16: Online times of **Method 2** on set 5 with standard search, map search and no search

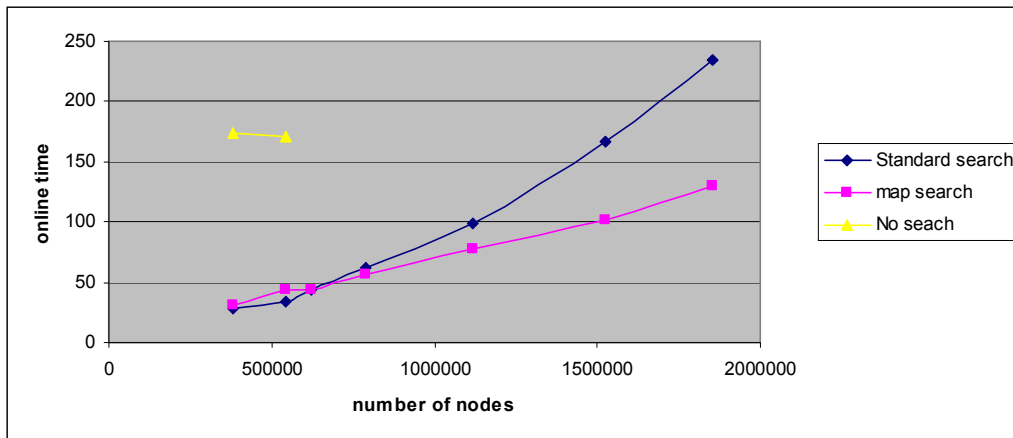


Graph 5.17: Online times of **Method 2** on set 6 with standard search, map search and no search





Graph 5.18: Online times of **Method 2** on set 7 with standard search, map search and no search



Graph 5.19: Online times of **Method 2** on set 8 with standard search, map search and no search

### Discussion of results for Method 3 analysis

First the results shown in appendix C from the analysis performed on **Method 3** are discussed. 74 out of the 84 mission analyses perform faster without a search. For the remaining 10 missions, 7 of them ran out of memory allocation before an answer was obtained. The total number of nodes created when analysed using a normal search, for these 7 missions, was reaching 6 figures with the total number of successful searches approximately doubling this figure. The other 3 missions did not perform better without a node search

because the total number of nodes created is much greater without the search. These missions had a higher amount of successful searches. Graphs 5.1 and 5.2 for mission set 1 show the approaches perform fastest in the order of not having a search, using a map search, and then the standard search. The standard search takes significantly longer than the other approaches and as the number of nodes created grows the gap increases. Comparing not having a search with a map search shown in graph 5.2, it can be seen that as the number of nodes created increases the map search takes significantly longer than not having a search. These large results are due to the number of nodes growing every time the computation function searches for existing nodes and potentially having to scan through the entire list. Also the mission set 1 is created from random fault trees and therefore there will not be that much common logic between them and relatively few repeated nodes. The approach using the map search is more efficient than the standard search since it checks to see if a new node is already in the list faster than the standard search. However the search may just be suited to, how the data is ordered in the array. Similar results are shown in graphs 5.3 to 5.5 for other mission sets 2-4 that are created by random fault trees. However for the mission sets 3 and 4 shown in graphs 5.4 and 5.5 as the number of nodes grows greater than about 100000, not having a search approach causes the online time to grow greater than the other two search approaches. This is because in these two mission sets when the missions starts getting large, the phases start repeating causing more common logic and hence more repeated nodes. The searches are therefore more successful and not having a search is a disadvantage. Also the not having a search approach runs out of memory before the other two approaches. For example mission set 4 when the number of nodes created gets greater than 300000, the memory runs out. These results are shown in the table in the Appendix C. Mission sets 5 to 8 are different to 1 to 4 since the fault trees represent the failure of UAV phases instead of being randomly generated. The results follow a similar trend to the results from mission sets 1-4. Not having a search generally performs the analysis in the fastest times except for mission set 5 where as the number of nodes grows it becomes the slowest time. However all the online times of mission set 5 are small, less than one second. Also the phase fault trees from mission set 5 are small so

missions can be analysed that contained many phases (up to 34) which are constructed from 6 different phases. Therefore the phases are repeated multiple times which increases the common logic which is an advantage for the search approaches. The standard search performs better than the map search for the mission sets 5-8 which was the opposite way around to mission sets 1-4. The standard search is more suited to the order in which the nodes occur in the list for missions 5-8. For mission sets 1-4, as the number of nodes get larger, the no search approach starts to slow down or run out of memory. This is not the case for missions of sets 6-8. However the fault trees that are analysed are not large enough since for this effect to take place a lot of modules would have been taken out. The next results discuss are for **Method 2** where modules are not taken out and the fault trees will be larger and will give a better indicator.

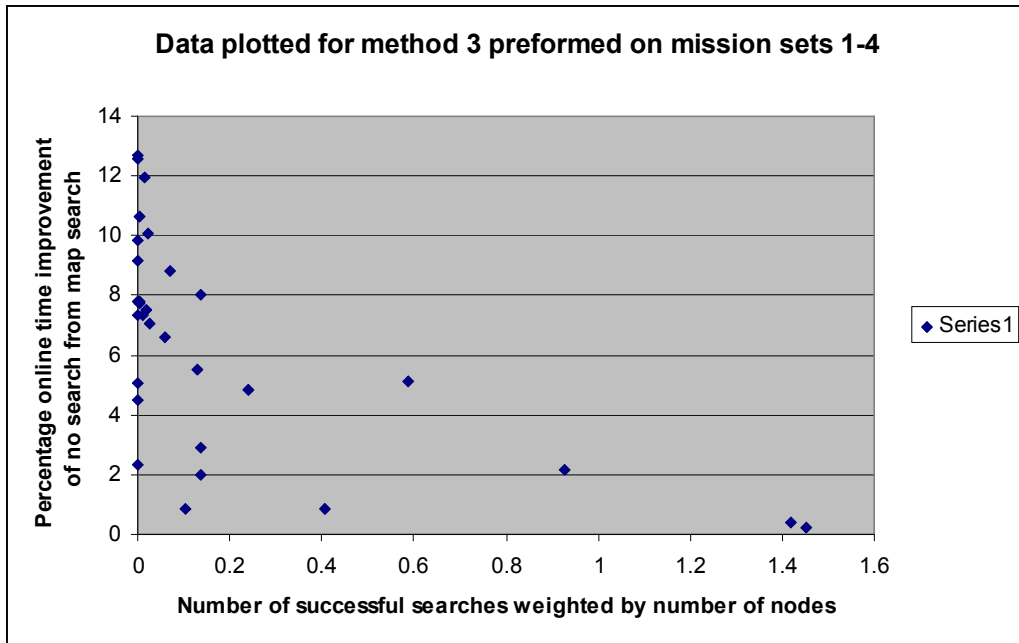
## **Results discuss on the analysis on Method 2**

The three approaches of a search are performed on **Method 2**. **Method 2** is different to **Method 3** in that it does not take out modules. The results are shown in appendix D which consist of 86 missions, 57 from the random mission sets, and 28 the UAV structure mission sets. The missions from the random mission sets will not vary that much compared to the performance on the **Method 3** since they are randomly generated trees and will not have a lot of modules to be taken out. As these missions increase in size the difference in performance increases for the 3 different approaches. The data for mission sets 1-4 are plotted in graphs 5.10 to 5.15. These results are very similar to those on **Method 3** because the mission from sets 1-4 have few modules. The data for the mission sets 5-8 which represent UAV mission analysis, the online time of the three search approaches are plotted in graphs 5.16 to 5.19. The results for **Method 3** on mission sets 5 and 6 were also similar to those perform on **Method 2** since the modules that were taken out are relatively small. However for mission sets 7 and 8 very different results were achieved which are shown in graphs 5.18 and 5.19. The graphs show that not having a search performs badly compared to the analysis using **Method 2**. This is down to the fact that the mission sets 7-8 were significantly reduced by taking out

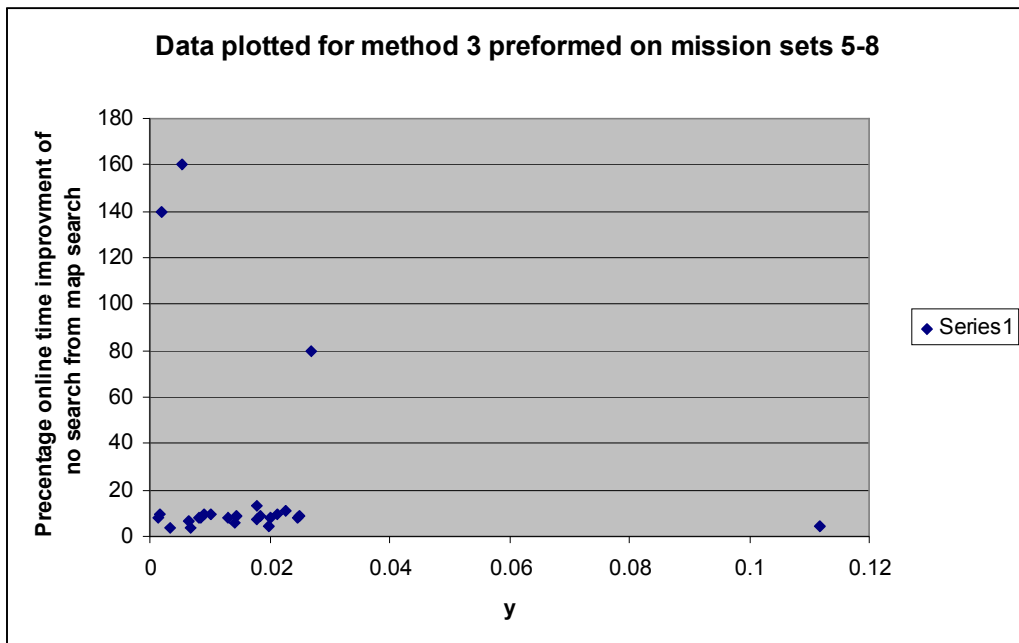
modules and now they are larger. The potential modules left in the faults tree provides a lot of common logic and increases the number of successful searches and favours for the search approaches.

### **Further analysis of the results**

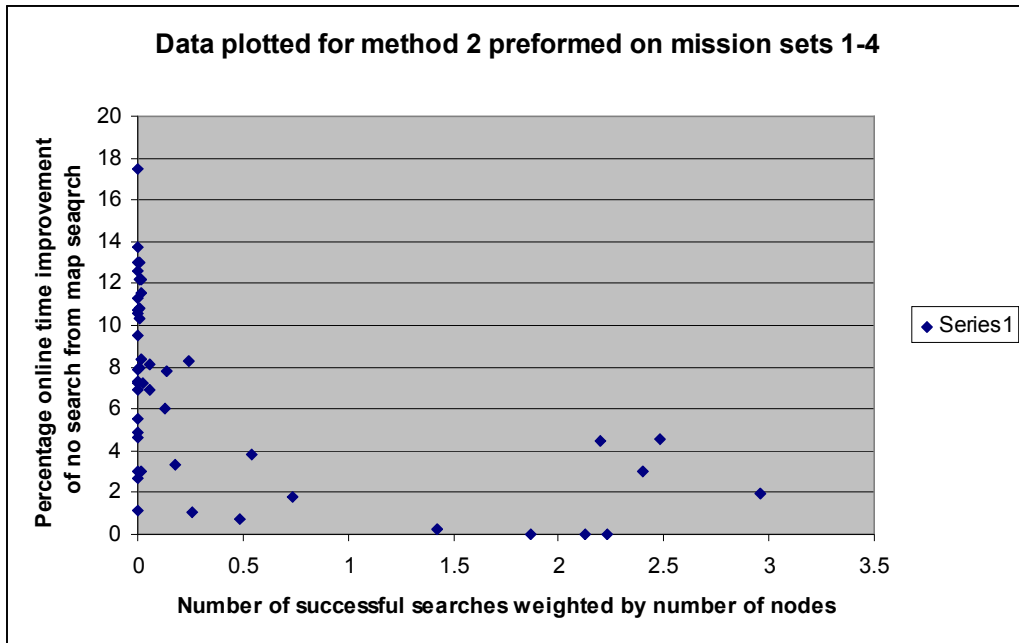
The results so far for the three search approaches show that not having a search is generally the fastest, but as the number of successful searches grows it becomes slower compare to the other two search approaches. The number of successful searches depends on the common logic between the phases and will grow by increasing the number of repeated phases which provides more common logic. If the number of successful searches increases then this will be an advantage for the search approaches over not having search approach. It is shown in four graphs that have been plotted for the percentage of improvement of not having a search approach compared to the map search approach against the number of successful searches weighted (dividend) by the number of nodes created. Graphs 5.20 and 5.21 are for the mission sets 1-4 and 5-8 data from **Method 3** and 5.22 and 5.23 are for **Method 2**. The results show that for the mission sets 1-4 on **Method 3** that as the number of successful searches increases relevant to the number of nodes a decrease in the percent of improvement of the no search approach compared to the map search approach. The results for the mission sets 5-8 on **Method 3** do not give great evidence of this, but the samples of missions are relatively small. The graph 5.22 is similar to 5.20 since the random mission sets 1-4 do not have lot of modules taken out. The graph 5.23 from the mission set 5-8 analysed using **Method 2** has a stronger evidence of the below discussion in the data since the method does not take out modules which leaves a significant amount of common logic for the mission sets 5-8. This is expected since these are UAV mission and will contain a lot of common logic between the phase increase the number of repeated nodes.



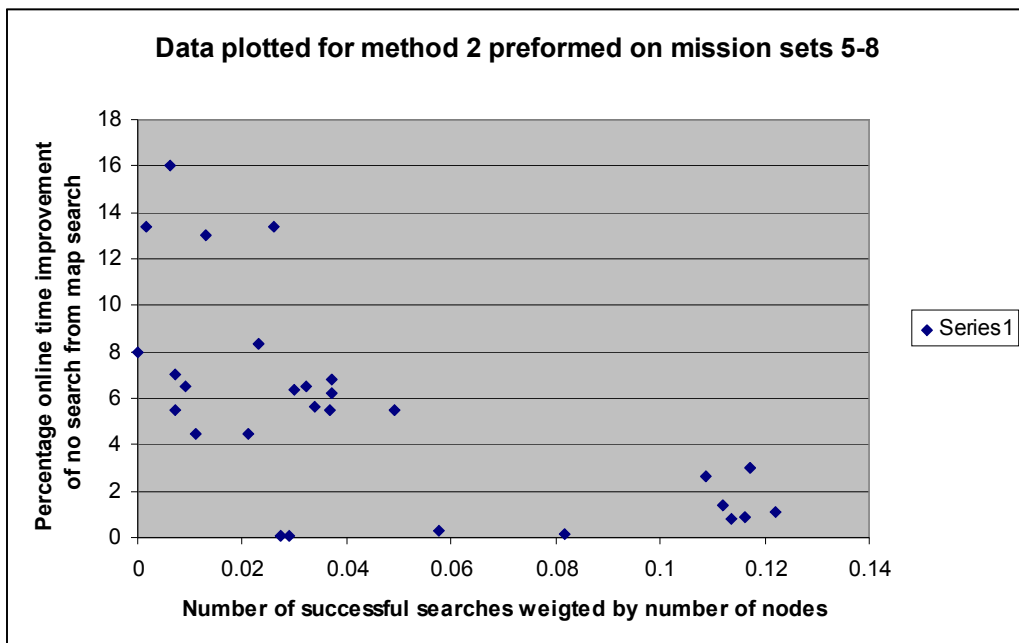
Graph 5.20: Data plotted for **method 3** performed on mission set 1-4



Graph 5.21: Data plotted for **method 3** performed on mission set 5-8



Graph 5.22: Data plotted for **method 2** preformed on mission set 1-4



Graph 5.23: Data plotted for **method 2** preformed on mission set 5-8

## 5.6 Investigating the analysis time taken for the computation look up function

The third stage of the computation function checks in a looks up table to see if the computation that will be performed has already been done before. This is

shown on lines 14 and 15 in figure 5.2. This stage does not take that much online time because of how the results of the computation are stored. Every node is assigned an element in an array. Every element in the array for a node contains all the computations that have been done with it so far and records the operator, the other node in the computation, and the resultant node. So the function directly searches the relevant elements for that node instead of the whole array. Whilst this stage does not take up that much analysis time a drawback is however that storage of all of these computations takes a lot of memory. Approximately 4 times as much as storing the nodes. For a larger problem with a great number of nodes there will be more elements in the array required.

The impact that the look up computations has on the analysis will be investigated by applying **Method 2** with a C++ map search on the missions that are shown in tables 5.1 and 5.2 which have been used throughout this thesis and counting the number of times the look up computation stage is successfully used and recording the online and offline times together with the number of nodes created. Then the analysis is repeated but this time without the lookup computation stage, recording the online and offline times, and the number of nodes created for a comparison. There are two measurements shown for the number of nodes that are created, the number of look up operations for computations that have already been calculated. The top level numbers in the table are for when the failure occurs in a phase BDD and those below are the numbers for all the individual phase failure BDDs. These results are shown tables 5.1 and 5.2.

Mission Set	Configuration	Method 2 with map With out computation Look up		Method 2 with map With computation Look up		
		Times Online (offline)	Number Of nodes Created	Times Online (offline)	Number Of nodes Created	Number Of Look up
1	1,2,4	5min 23.05s (0.08s)	11107 1496	0.61s (0.22s)	11107 1496	4662 212
1	2,1,5	6min 26.5s (0.16s)	5800 2435	0.34s (0.14s)	5800 2435	1027 550
1	1,2,3	1 hour 7min 14.2s (0.17s)	28042 2191	1.72s (0.41s)	28042 2191	14220 296
1	4,3,1	1min 7.03s (0.25s)	67664 2601	4.31s (0.88s)	67664 2601	37586 415
1	2,3,5	15 hr 24min 5.7s (0.25s)	61724 3766	9.19s (0.78s)	61724 3766	158288 795
1	6,2,3	>16hr		39.16s (4.34s)	372684 3079	235186 742
2	1,2,5	1min 49.02s (0.2s)	2630 1105	0.14s (0.11s)	2630 1150	340 154
2	1,2,3	4min 53.01s (0.2s)	3974 1211	0.22s (0.11s)	3974 1211	1037 189
2	1,2,5,6	>24hr		1.34s (0.34s)	22520 1269	10804 167
3	1,2,3	0.59s (0.20s)	1407 477	0.06s (0.06s)	1407 477	150 15
3	1,2,3,4	38.86s (0.75s)	2476 649	0.13s (0.08s)	2476 649	398 38
3	1,2,3,4,5	5hr 3min 16.9s (0.90s)	5736 946	0.30s (0.14s)	5736 946	2290 80
4	1,2,3	5.78s (0.16s)	6661 974	0.39s (0.14s)	6661 974	3038 285
4	1,2,3,4	52.19s (0.17s)	27740 1301	1.83s (0.38s)	27740 1301	20393 348

Table 5.1: The analysis results of the impact of the look up computation



Mission Set	Configuration	Method 2 with map With out computation Look up		Method 2 with map With computation Look up		
		Times Online (offline)	Number Of nodes Created	Times Online (offline)	Number Of nodes Created	Number Of Look up
4	1,2,3,4,5	56min 35.5s (0.18s)	119206 1529	10.83s (1.41s)	119206 1529	136417 446
5	1,2,3,4,5,6	0.64s (0.14s)	1277 206	0.08s (0.03s)	1277 206	191 98
5	1,4,3,2,3,5 6,1	0.98s (0.18s)	1730 206	0.09s (0.06s)	1730 206	298 146
5	1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6	2.14s (0.16s)	4547 206	0.31s (0.08s)	4547 206	699 354
5	1,5,4,6,2,1 3,2,3,4,5,3 1,3,5,6,3,3 2,1,2,3,	2.88s (0.16s)	5803 206	0.41s (0.11s)	5803 206	935 436
5	1,5,4,6,2,1 3,2,3,4,5,3 1,3,5,6,3,3 2,1,2,3,1,2 3,4	3.44s (0.17s)	6930 206	0.66s (0.14s)	6930 206	1150 521
5	1,5,4,6,2,1 3,2,3,4,5,3 1,3,5,6,3,3 2,1,2,3,1,2 3,4,2,3,4,5 3,5,6,3	4.56s (0.17s)	9153 206	0.66s (0.16s)	9153 206	1483 686
6	1,2,8	0.06s (0.15s)	1107 369	0.08s (0.13s)	1107 369	10 20
6	1,2,3	0.09s (0.16s)	1371 633	0.09s (0.11s)	1371 633	10 51
6	1,3,6	23min 11.6s (0.31s)	2092 971	0.13s (0.14s)	2092 971	41 157
6	1,3,5,4	>24hr		0.19s (0.14s)	3363 1167	330 339
7	1,2,3	>24hr		0.42 (0.18s)	5815 4505	10 2704
8	1,2,3	>24hr				

Table 5.2: The analysis results of the impact of the look up computations

The online times of 26 missions from the 8 mission sets analysed by **Method 3** with a computation look up function and without are shown in tables 5.1 and 5.2. All of the 26 missions perform significantly faster with the computation look up function rather than without. The online times of the analysis with the look up function did not exceed a minute, with an average of 2.83 seconds. For the analysis without the look up function 4 of the missions did not even obtain a result after 24 hour for the rest that did had an average online time of 1 hour which is over 1000 times longer than with the look up function. The big different in the results is down to how many successful look ups there are which is shown in the last column of tables 5.1 and 5.2 and each computation look up could potentially save a sufficient amount of calculation. In summary the results show that the computation look up function has a sufficiently large impact on the speed of the analysis. However a disadvantage is that the memory can be used up about 4 times as fast.

## **Summary**

My research shows that the decision to use a search for the repeated computations can have a very sufficient impact on the analysis time. Whether the search will speed up or slow down the analysis will depend on the nature of the mission. If there are many common logic sections in the phase fault trees which cause the same nodes to be built in the BDD, then the search can help the speed of the analysis as shown in the UAV mission example. However when the number of nodes grows then the time spent on an individual search will also grow. If there is not a lot of common logic between the phases then it will not be worth the time spent on the search. This is shown in the mission examples for the randomly generated phase failure condition whose analysis is improved willout the search. If a search approach is implemented then a big factor on the analysis times is which type of search to apply. My research investigated two types of searches: a normal search comparing every individual element in the array in the order the node was created, and a C++ map assisted search. It shows that generally the C++ map performs better on the random mission sets and the normal search performed better on the UAV structure mission sets. Therefore the type of search which performs best will

depend on the structure of the BDD which determines the order that the BDD nodes are listed in the array.

## Chapter 6: Modularization by repeated gates and events method

### 6.1 Introduction

In a fault tree there is a relationship between the positions of repeated gates and events and the number of independent modules which can be extracted. If a gate has at least one repeated gate or event beneath it, which occurs somewhere else in the tree then it cannot be a module. Another way of looking at this is to take a repeated gate or event and to trace up through the tree from every position at which the gate or event appears in the tree until all the paths intersect. All the gates that the paths pass through to the point of convergence cannot be a module. For example consider figure 6.1. The repeated event 1 occurs twice. The paths go through gates 3, 4 and 5 before they intersect at gate 1 therefore gates 3, 4 and 5 are not modules.

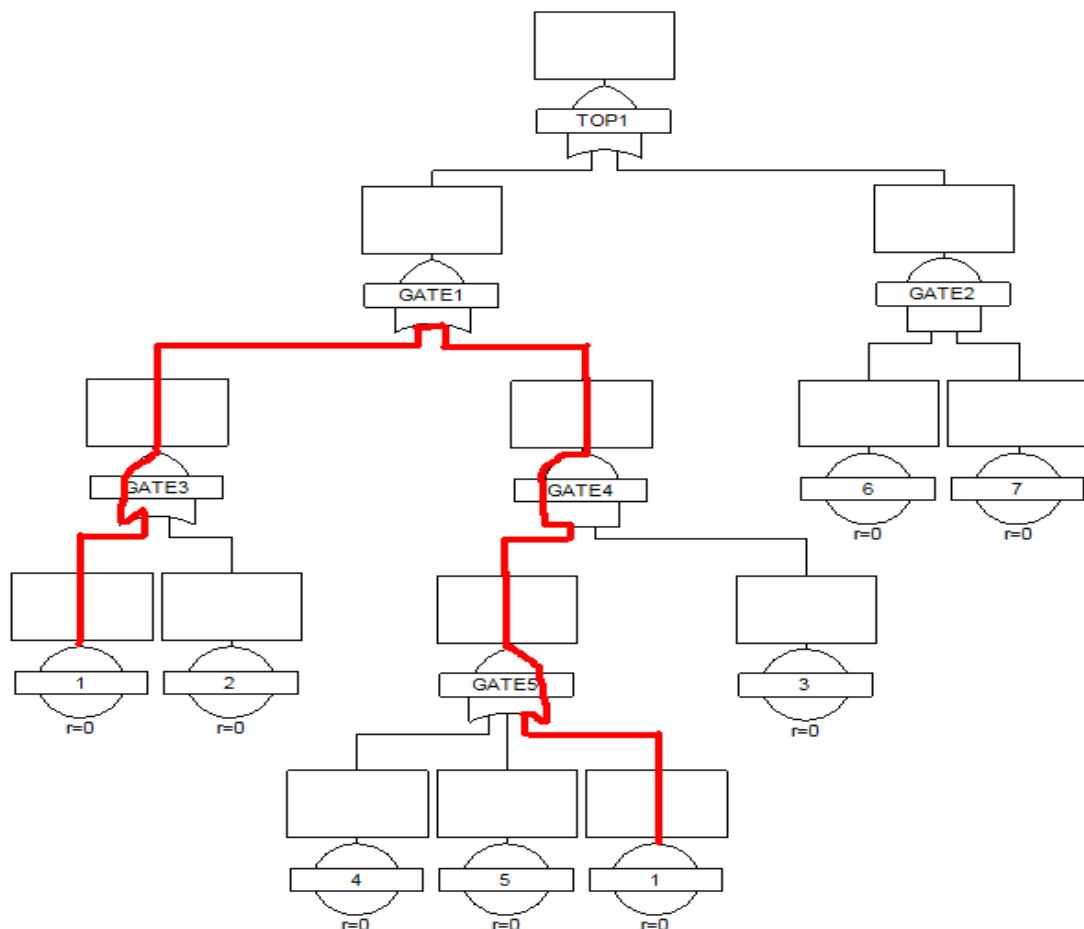


Figure 6.1: Fault tree with the path between the event 1

A method of obtaining the module gates of a fault tree is to go through all the repeated gates and events and perform the technique shown above to find out which gates are not modules. Therefore once all of the repeated gates and events have been considered the gates that have never had a path that go through them and have not therefore been excluded from the module list are modules. This method and Rauzy linear time algorithm both obtain the modules in an efficient way. However this method is applied instead because it also obtains the repeated gates and events that prevent non-module gates from being modules.

The main goal of this project is to develop a faster running calculation for a PMS analysis and this can be achieved by modularizing the problem where possible. In the process described above, the fault tree representation for the same logic function is not unique. If the size of the paths are minimal then the paths will trace through less gates. Therefore restructuring the tree, minimizing the number of non-module gates encountered, will maximize the number module gates. If more modules can be taken out of the tree then the calculation will run faster.

## **6.2 Restructuring technique**

The restructuring technique will be expanded on for the occurrence of repeated gates in the tree structure rather than repeated events. However the method that is applicable for gates will generalize for events also. The fault tree restructuring is carried out using a sequence of processes as described below:

### **6.2.1) Push-up**

Push-up is a similar idea to the contraction process described earlier. Pushing up the repeated gate until the gate type of the next two gates above it have different gate types. For example consider figure 6.2 below where G4 is a repeated gate. As G3 and G2 are the same gate type, G4 can therefore be pushed up to be an input to G2. As shown the gate to which G4 is now an

input (G2) is an OR gate which inputs to an AND gate (G1) and so satisfies the requirement that the two gates immediately above gate G4 are different. This restructuring effectively removes G3 from the path enabling it to be considered as a potential module.

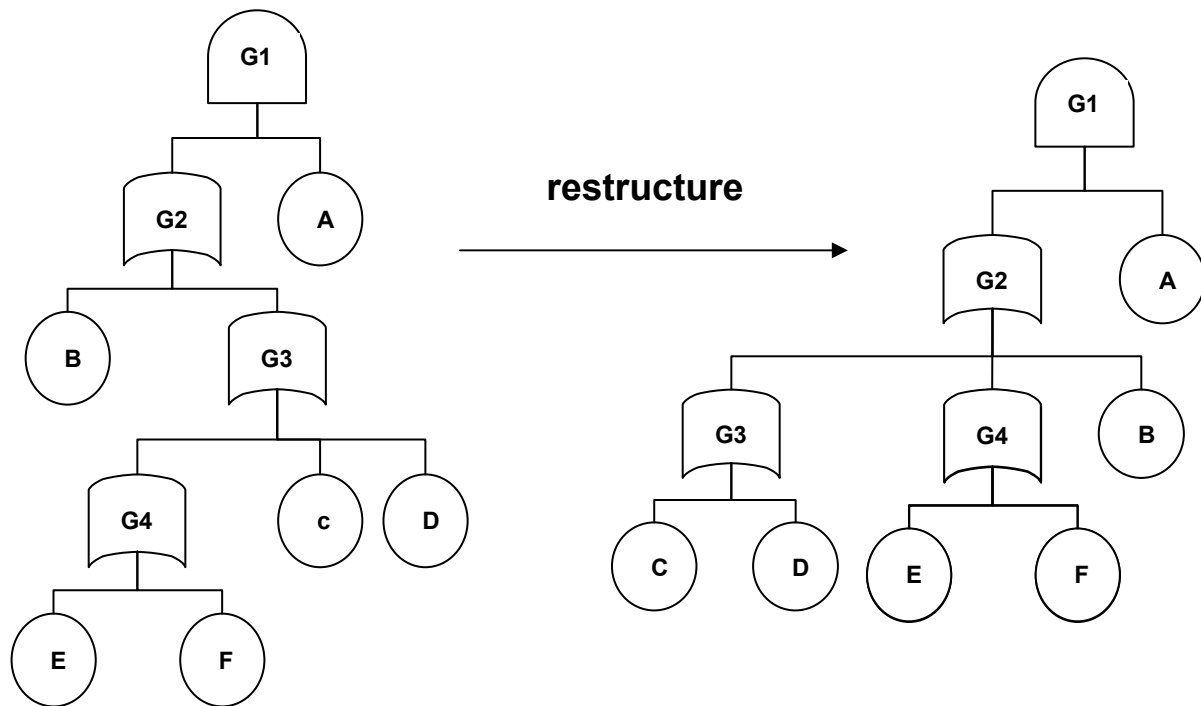


Figure 6.2: Example of the Push-up technique

### 6.2.2) Common-input Push-up

Common-input push-up is a similar idea to the Extraction operation presented earlier. Consider the two fault tree structures shown in figures 6.3 a and b. These have an OR gate and an AND gate at the top respectively. The repeated gate features as an input to each of the branches which input to the top event. Note gates  $G1 - G_n$ ,  $g1 - g_n$  and the repeated gate can be either OR or AND gates. The restructuring identifies that the repeated gate can be taken out as a common factor. This has the effect of removing several gates from the path which can now be considered as potential modules.

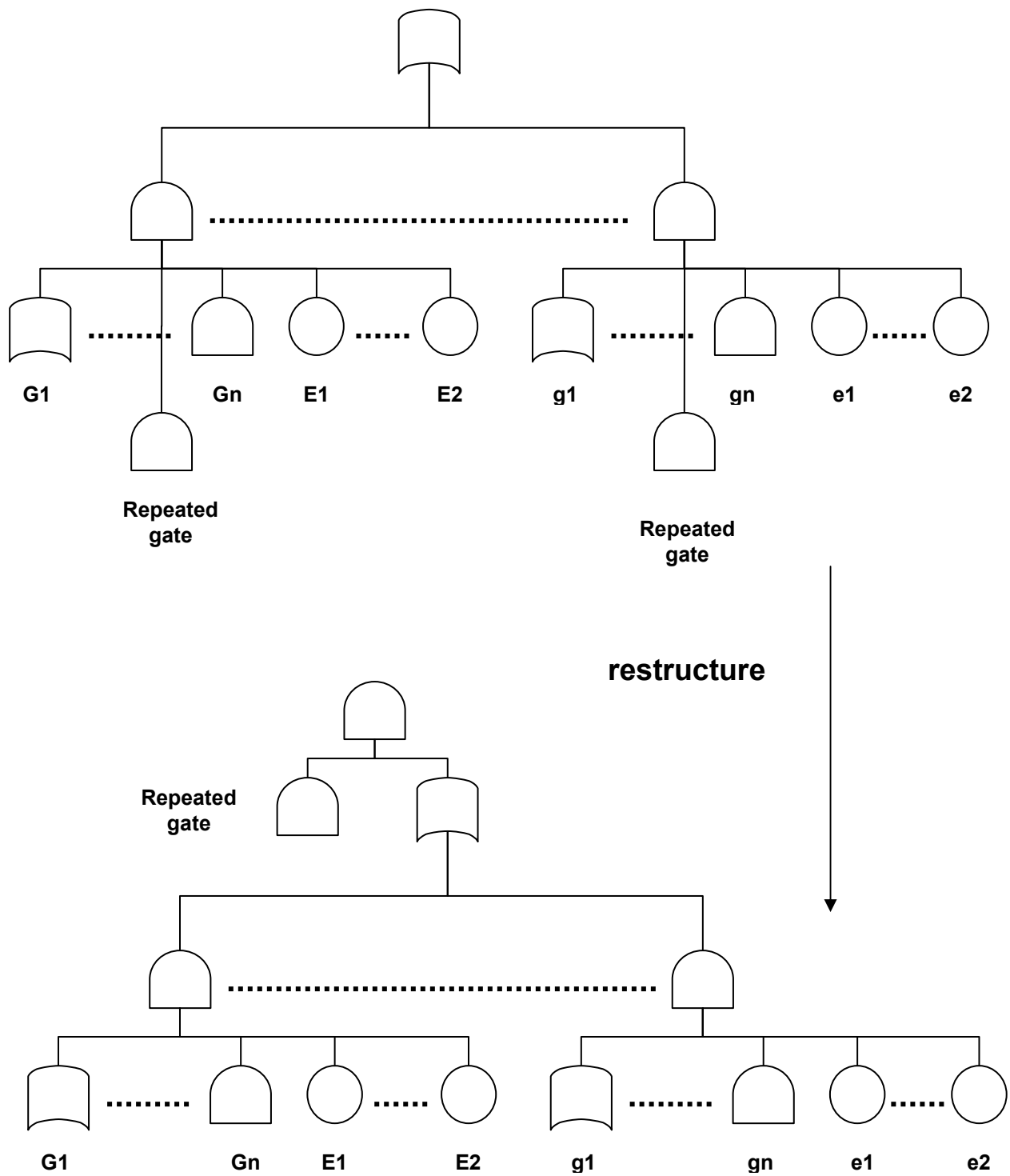


Figure 6.3 a: Common-input Push-up technique

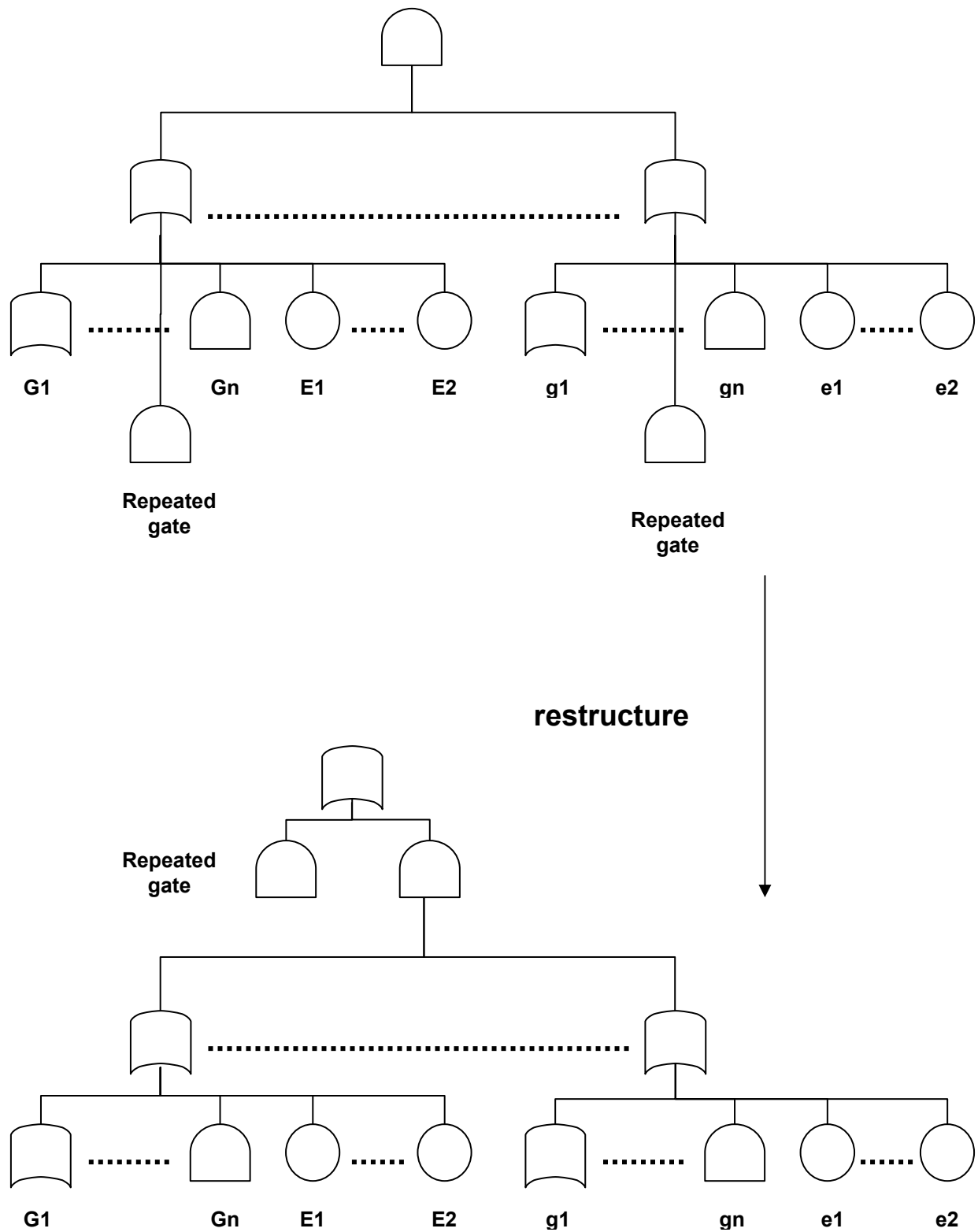


Figure 6.3 b Common-input Push-up technique



### 6.2.3) Elimination

If two repeated gates are positioned in the tree so that one of them is an immediate input to a gate and the second occurrence also features as an input to another gate on a lower level in the fault tree structure under these conditions then elimination is possible. This is shown in figures 6.4 and 6.5 below. The gate to which the repeated gates provide an input is referred to as the primary gate (G1) and the one to which the lower repeated gate is an input is referred to as the secondary gate (G4). If the primary and secondary gates are of the same type then the second occurrence of the repeated gate into the secondary gate can be removed (Figure 6.4). However, if the gates types are different then the secondary gate can be removed (Figure 6.5).

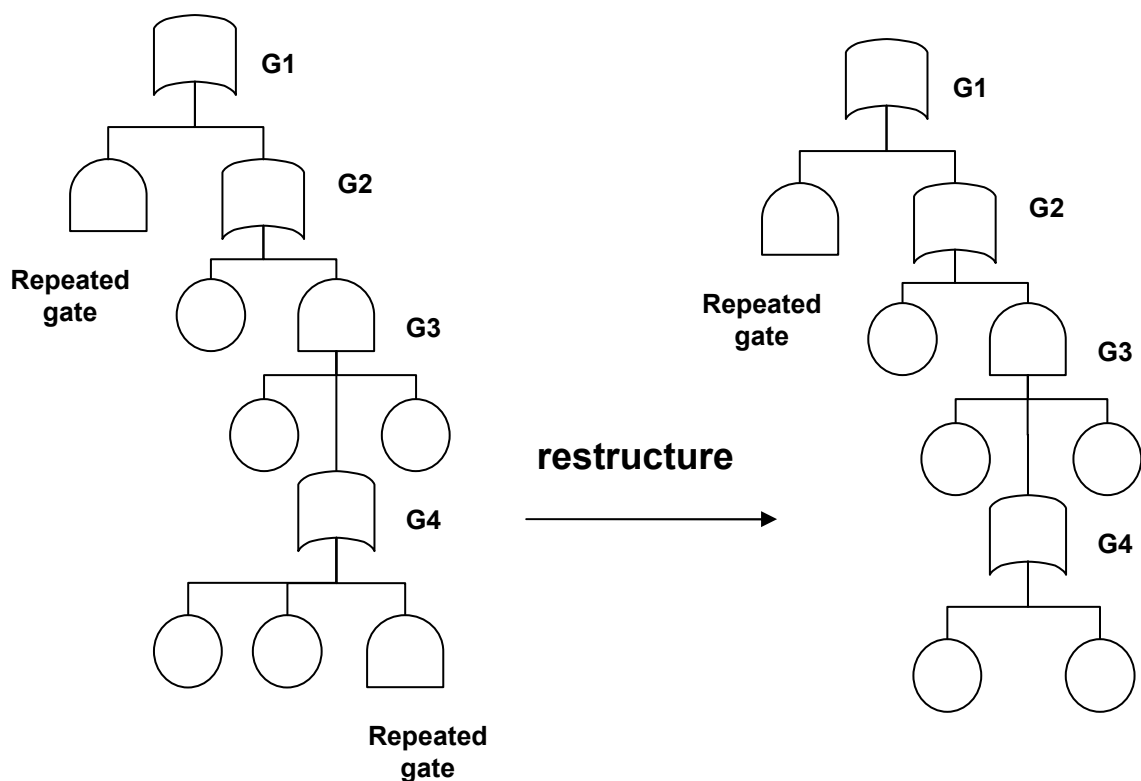


Figure 6.4: Example of an elimination for the same type case

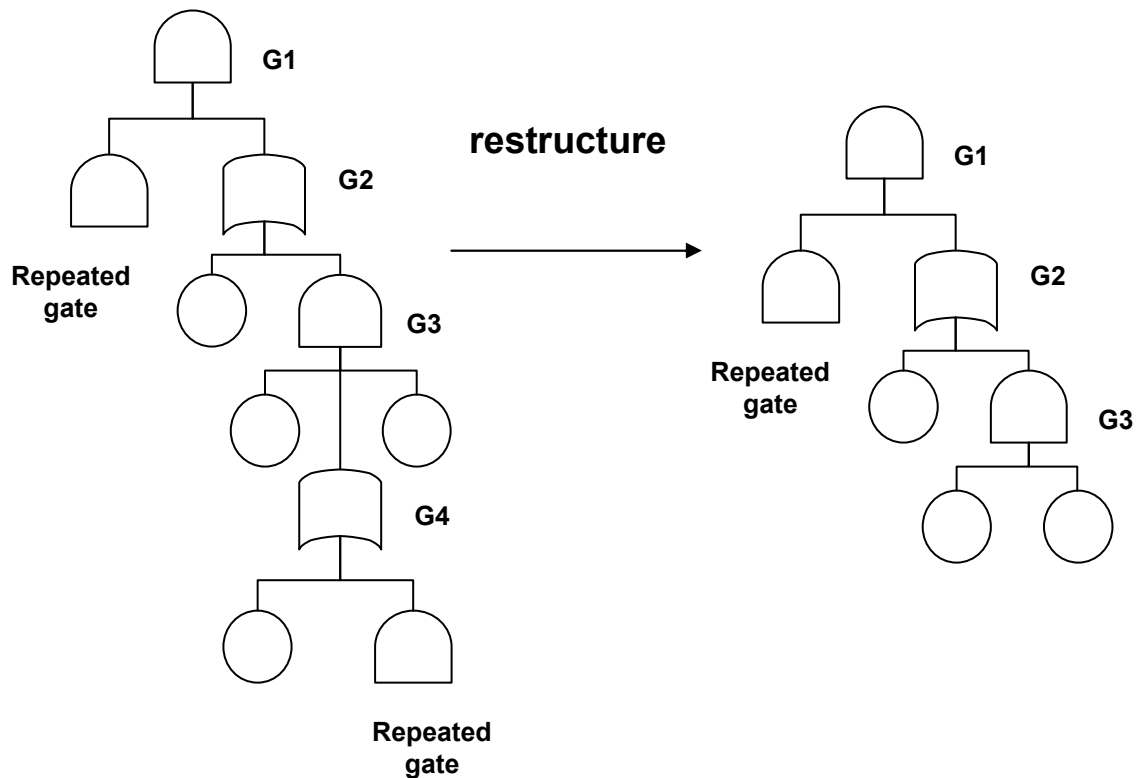


Figure 6.5: Example of an elimination for the different type case

#### 6.2.4) Factorization

This restructuring technique is different from the others presented which reduce the paths between the repeated gates and events in order to maximize the number of potential modules. The factorization technique adds a gate which in turn changes two non-module gates to module gates. Also the added gate will be a module. If two non-module gates always occur together under the same gate type and all the gates and events that occur beneath the two gates only occur there and nowhere else, then these two gates are dependent but independent of the rest of the tree. In this situation their combination can be a module. For example consider the example shown in figure 6.6, Assume that gate G1 and G2 are not modules and that all the gates and events beneath G1 and G2 only occur under G1 and G2 and nowhere

else therefore gates G1 and G2 are dependent of each other but not the rest of the tree. Therefore since gates G1 and G2 always occur together under the same gate type we can combine the gates under a new gate which is of the same gate type as the gate to which they were originally inputs as shown in figure 6.6. This results in G1 and G2 becoming modules.

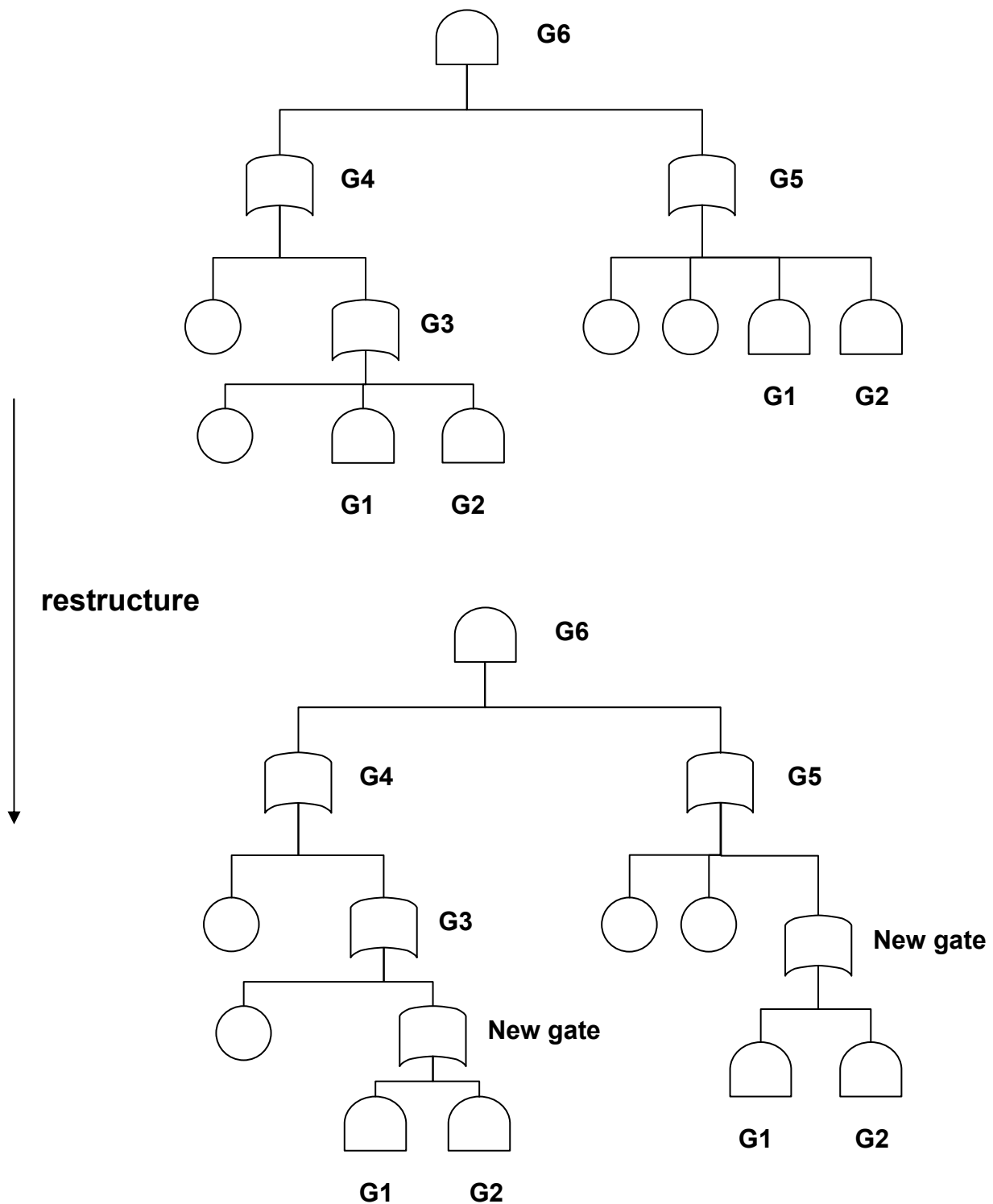


Figure 6.6: Gates G1 and G2 Being combined by Factorization

### 6.3 Worked example

In order to explain the method and the implementation in the program, the method will be applied to the fault tree illustrated below in figure 6.7. This is only a single phase example but is sufficient to demonstrate the features of the method. The method for the single and multi-phase cases are the same in principle since the method would perform the restructuring stage on the possible phases of a mission all at once as if it was a single phase. The objective of the algorithm below is to restructure the fault tree of the mission phases to maximize the number of modules that can be extracted by using the techniques discussed above. The algorithm and example below only expand the restructuring stage of the method since the analysis is performed as in the method discussed in section 4.6.

The program goes through the steps as follow:

- 1) Input the data of all the fault trees of the possible phases. Note all the fault tree phase data files are combined into one since for the UAV example that it attempts to solve the phases have a lot of branches in common. This will avoid repeating these branches in files several times which the code for the pervious methods did. The code identifies the top gates of the phases by searching though the data for gates which appear as outputs but do not appear as inputs.
- 2) Consider every repeated gate  $i$  that could potentially be from a phase in the final mission. Note the restructuring stage does not restructure the phase trees one by one it works on them simultaneously since all the trees are combined together in the same arrays.

2.1) Find the top intersection gate of all the occurrences of gate  $i$

2.2) Push-up (from 6.2.1) all the occurrences of gate  $i$  as much as possible.

- 2.3) Common-input Push-up (from 6.2.2) all the occurrences of gate  $i$  as much as possible.
- 2.4) If (the previous step 2.3) has at least one common push then go back to step 2.1.
- 3) Go through every repeated gate  $i$ , that potentially could be from a phase in the final mission, to identify Elimination (6.2.3) where possible and then eliminate the necessary gate
- 4) Apply steps 2 and 3 to all the repeated events as well.
- 5) Go through every possible non-module gate pair to see if they can be factorised (6.2.4).

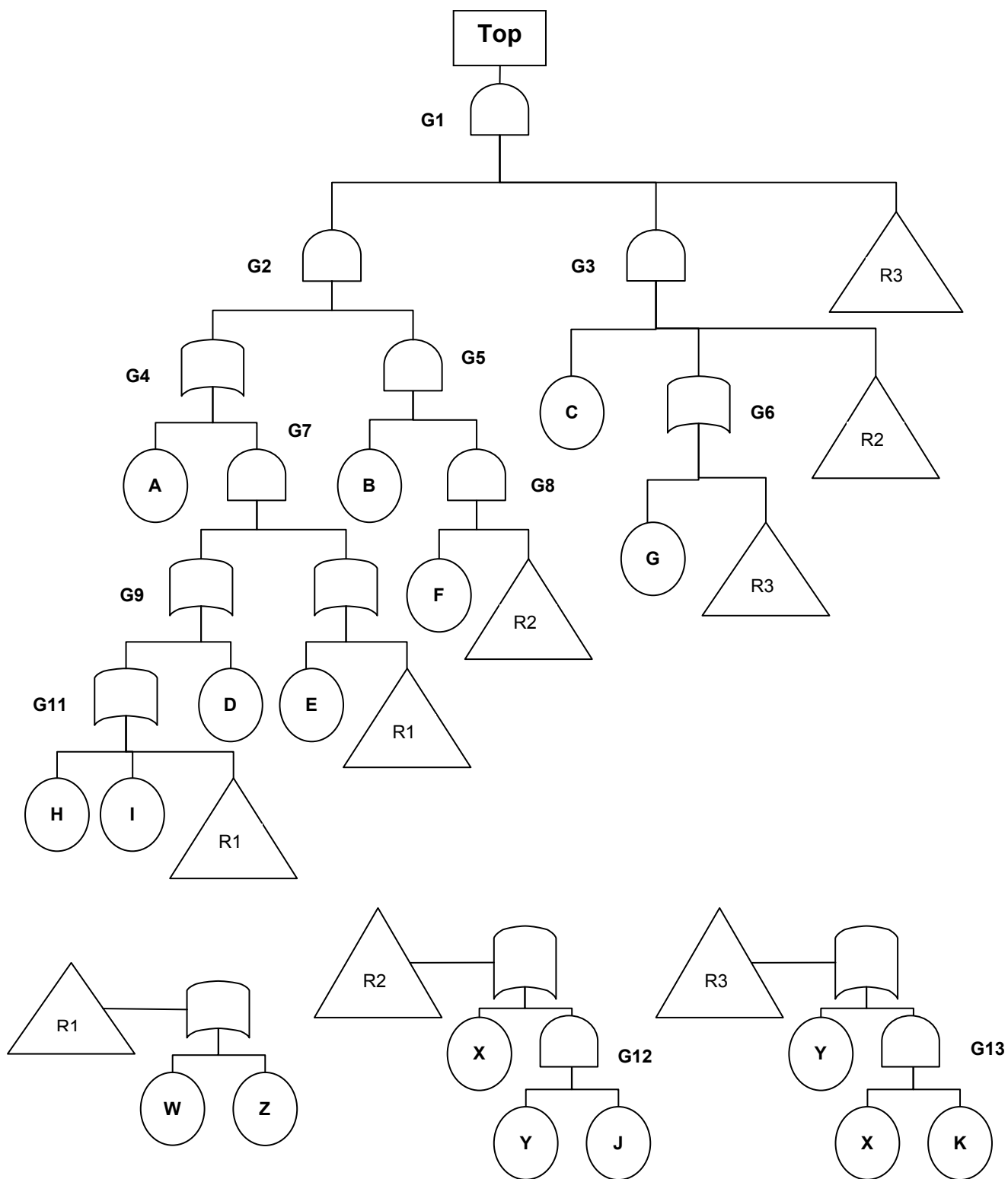


Figure 6.7: An Example fault tree

### 6.3.1 Inputting the fault tree to the program

The first step is to input the fault tree from a file into arrays within the code. Each line in the file represents a gate in the tree. The information given consists of the gate name, type (1 for OR gate and 2 for a AND gate), number of gate inputs, number of event inputs and the input gate and event list follows. Even if the gate is repeated in the tree it will only appear once in the data. The format of the data file for the tree in figure 6.7 is show in table 6.1 below.

Gate name	Gate type	Number of gates	Number of events	Inputs
G1	2	3	0	G2 G3 R3
G2	2	2	0	G4 G5
G4	1	1	1	G7 A
G7	2	2	0	G9 G10
G9	1	1	1	G11 D
G11	1	1	2	R1 H I
R1	1	0	2	Z W
G10	1	1	1	R1 E
G5	2	1	1	G8 B
G8	2	1	1	R2 F
R2	1	1	1	X G12
G3	2	2	1	G6 R2 C
G6	1	1	1	R3 G
R3	1	1	1	Y G13
G12	2	0	2	Y J
G13	2	0	2	X K

Table 6.1 Data file for the fault tree shown figure 6.7

The program reads in the file line by line. Each column forms an array. The data which is a string type in the file is changed to a numerical format because manipulation of numerical information is more efficient than string data. To do this basic events are assigned a number from 1 to 9999 and gates are numbered from 10000 upwards. The numerical version is shown in Table 6.2.

Gate name	Gate type	Number of gates	Number of events	Inputs
10000	2	3	0	10001 10011 10013
10001	2	2	0	10002 10008
10002	1	1	1	10003 1
10003	2	2	0	10004 10007
10004	1	1	1	10005 2
10005	1	1	2	10006 3 4
10006	1	0	2	5 6
10007	1	1	1	10006 7
10008	2	1	1	10009 8
10009	2	1	1	10010 9
10010	1	1	1	10 10014
10011	2	2	1	10012 10010 12
10012	1	1	1	10013 13
10013	1	1	1	11 10015
10014	2	0	2	11 14
10015	2	0	2	10 15

Table 6.2: The numerical fault tree data for figure 6.7



The top gate is identified by searching through the gates to see which one does not occur in any of the input gate list. The inputs array is a 2 dimension array the elements are arranged as follows. The first element 0 to (number of gates -1) are occupied by the gates and the elements (number of gates) to (number of gates + number of the event) are occupied by the events. Other arrays that are created from the input data are the parents arrays for gates and events. The number of parent gates is stored along with the parent gates which they provide input to the child gate, this information is used in applying the restructuring technique.

### 6.3.2 Restructuring the fault tree

Now that the fault tree and parents arrays have been created the restructuring technique can begin. The numerical fault tree is shown in figure 6.8 which corresponds to table 6.2.

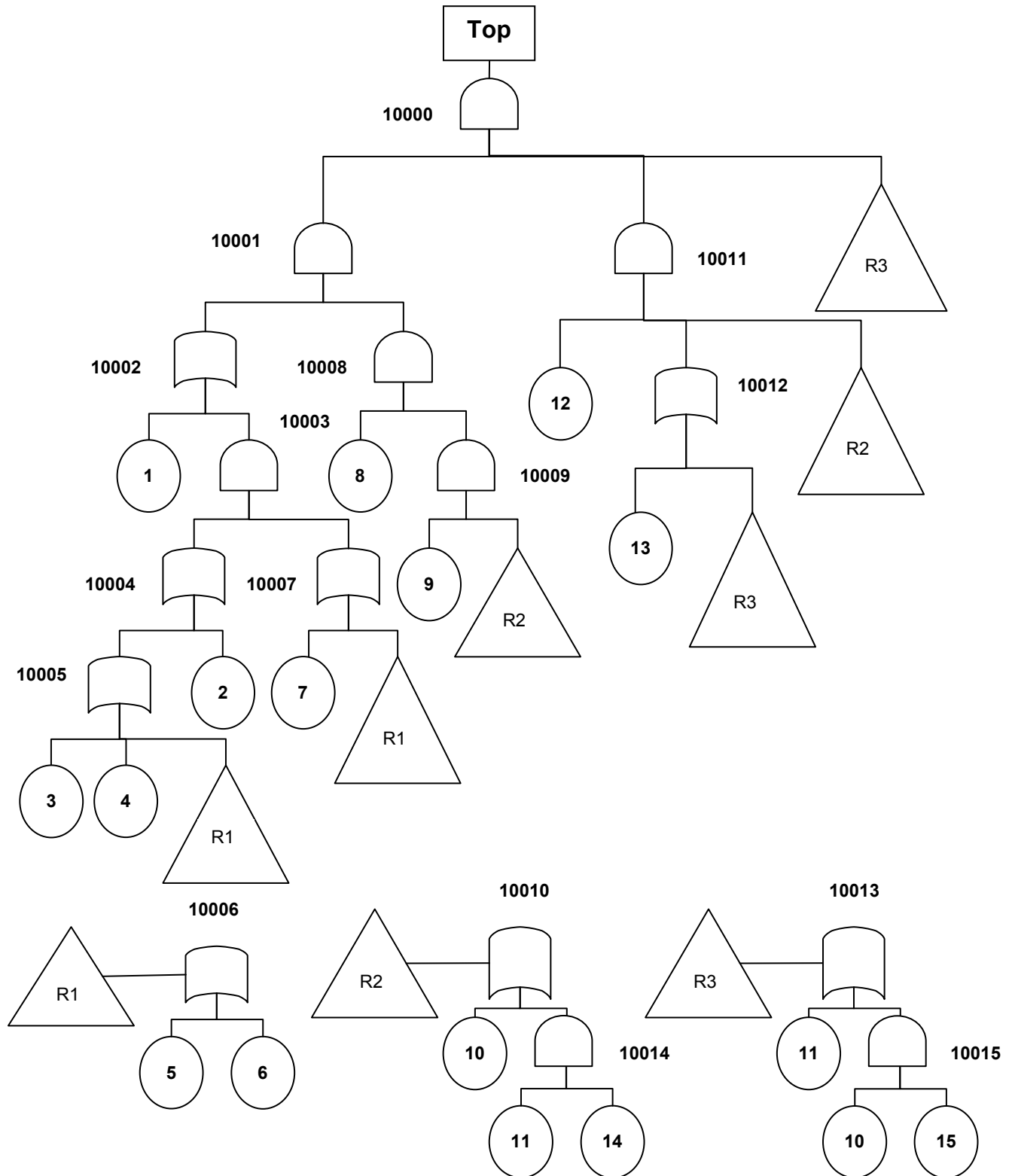


Figure 6.8: The example fault tree in the numerical form

The program identifies and processes each repeated gate one at a time. There are 3 repeated gates in the tree: gates 10006, 10010 and 10013. The code starts with gate 10006 which appears twice under gates 10005 and 10007.

### **Top intersection gate for repeated gate 1**

The top intersection gate of a repeated gate is where all the paths which are obtained by beginning at the appearances of the repeated gates and tracing up the tree to the gate where the paths intersect. This is used when pushing up the repeated gate when the top intersection is reached pushing up stops. Since the goal is to minimize the paths, going past the top intersection may increase the paths. To obtain the top intersection gate of a repeated gate go through each occurrence of the repeated gate and trace a path of gates up the tree until the top gate is reached. For example gate 10006 occurs twice the two paths are: path 1 gates 10005, 10004, 10003, 10002, 10001 and 10000. Path 2 gates 10007, 10003, 10002, 10001 and 10000. The top intersection gate, where all the paths intersect, is gate 10003.

### **Special case**

When tracing up the tree, if the current gate that is getting passed through has more than one parent then that path can make multiple paths as shown in Figure 6.9. However the other parent gates of the current gate can only be explored if the current top intersection is higher than the current gate. This is necessary because if the current top intersection gate is not above the current gate then the current gate should be the top intersection gate. If this is the case and the parent gates of the current gate are explored then these paths will intersect with each other to give a false top intersection gate, which will be higher than the current gate. For example considering figure 6.10 where the top intersection gate is not above the current gate then the parent gates are not considered. However if the current intersection gate is above the current gate then the parent gates are considered as paths as shown in figure 6.11.

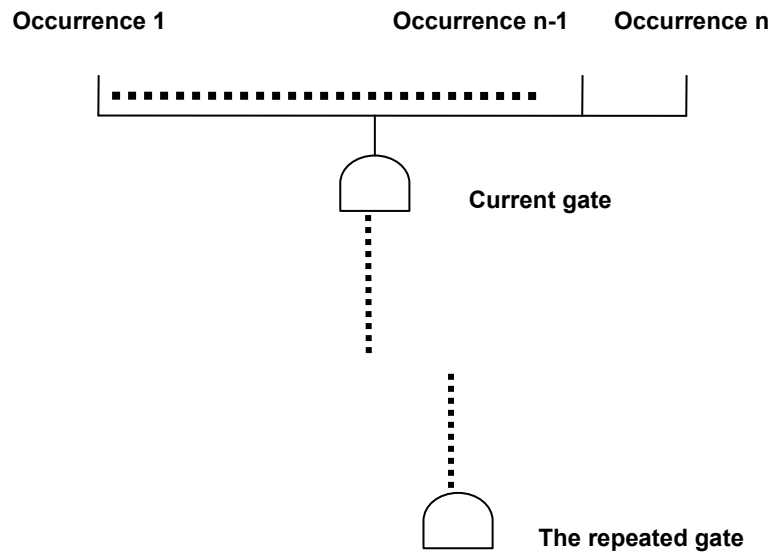


Figure 6.9: The special case

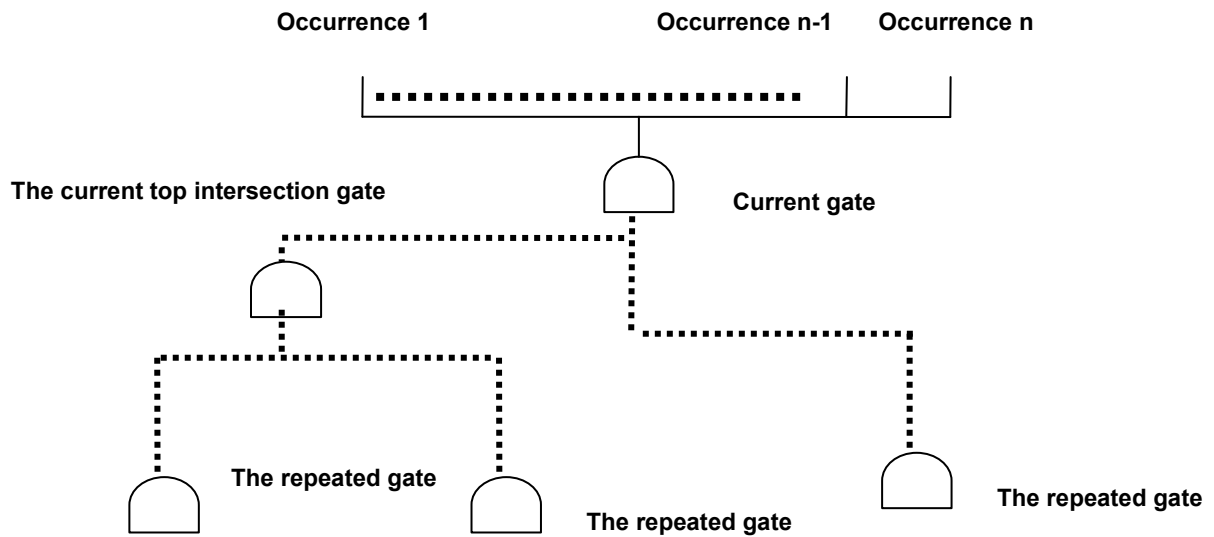


Figure 6.10: Example of the special case for finding the top intersection gate

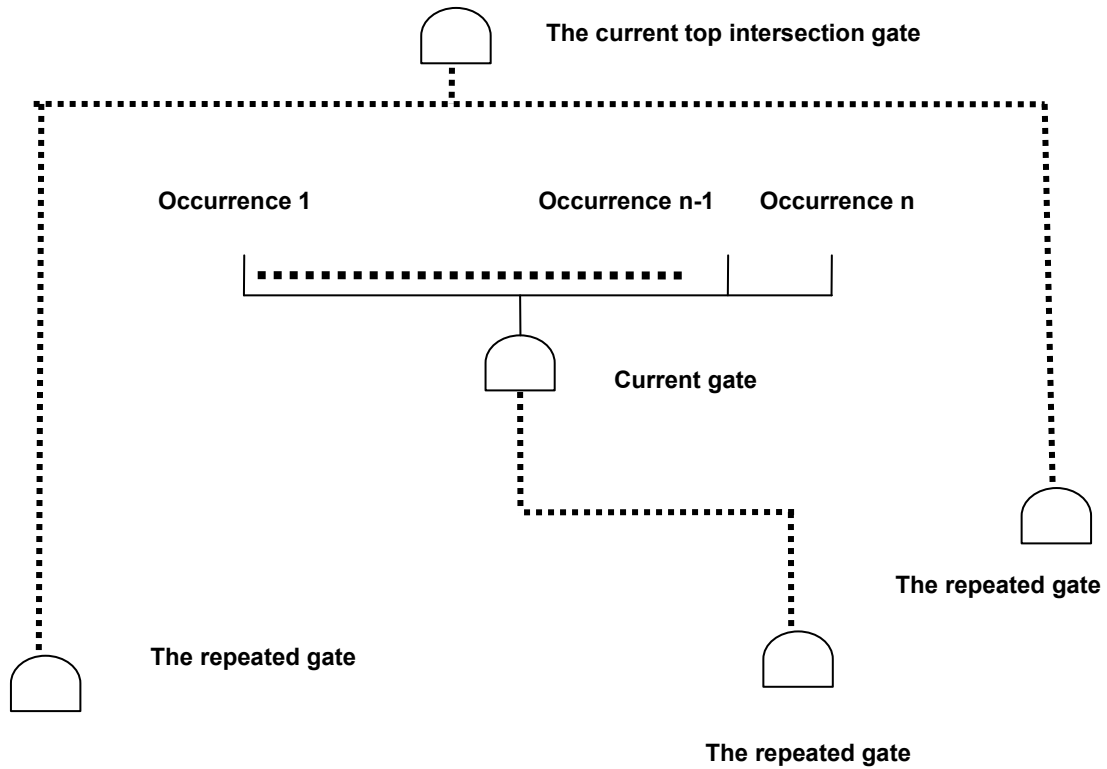


Figure 6.11: Example of the special case for finding the top intersection gate

### Push-up 1

The aim of this stage is to push up all of the occurrences of a repeated gate, so that the distance between them is reduced. A gate which is repeated (referred to as gate  $i$ ) is pushed up by considering the gates at two levels above it, the first gate which has as input gate  $i$  (which is referred to as gate  $j$ ) and second gate which has as input gate  $j$  (which is referred to as gate  $z$ ). If the gate  $j$  and gate  $z$  are of the same type and gate  $j$  is not the top intersection gate then gate  $i$  can be pushed up from gate  $j$  to gate  $z$ . This is shown in figure 6.12 below.

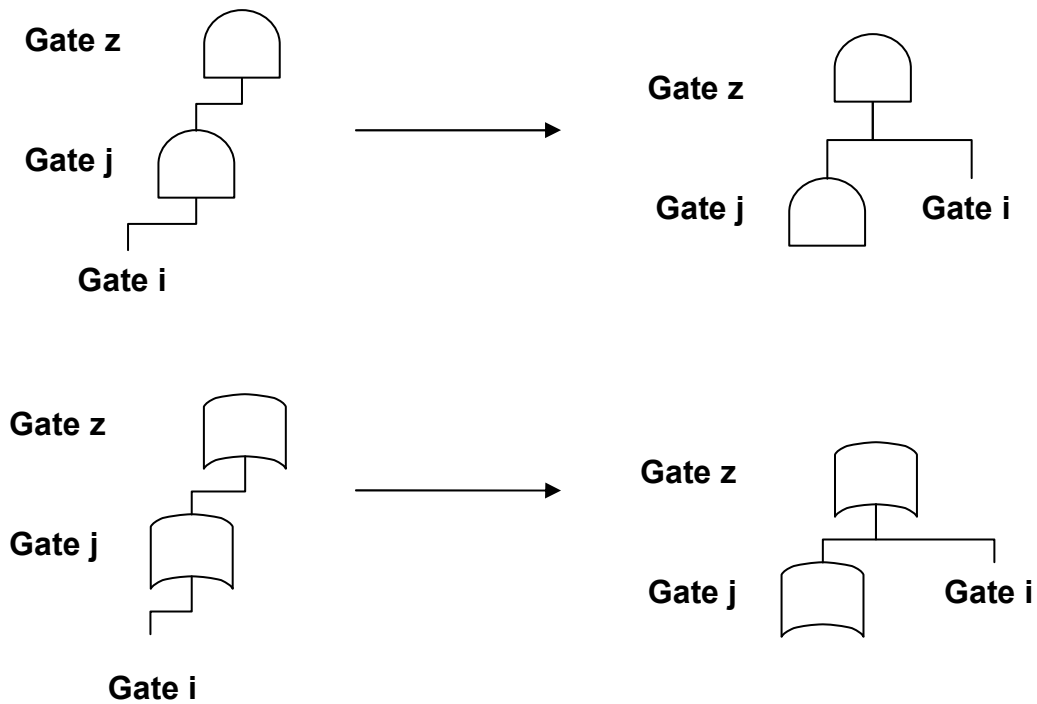


Figure 6.12 demonstration of the push up

However the above push up technique only covers the case if there is only one parent gate (gate z) of gate j in the tree. So if there is more than one parent gate of gate j then all of them must be considered. If all the gates z are of a different type to gate j then no push up is done. But if at least one gate z is the same type as gate j then each gate z will have to be considered one by one. If the gates are of the same type then the same process is applied as in the singular case. However if the gate types are different then a new gate must be created, which is the same type as gate j and which is positioned in between gate j and gate z. Gate i is pushed up to this new gate. This is shown in figure 6.13.

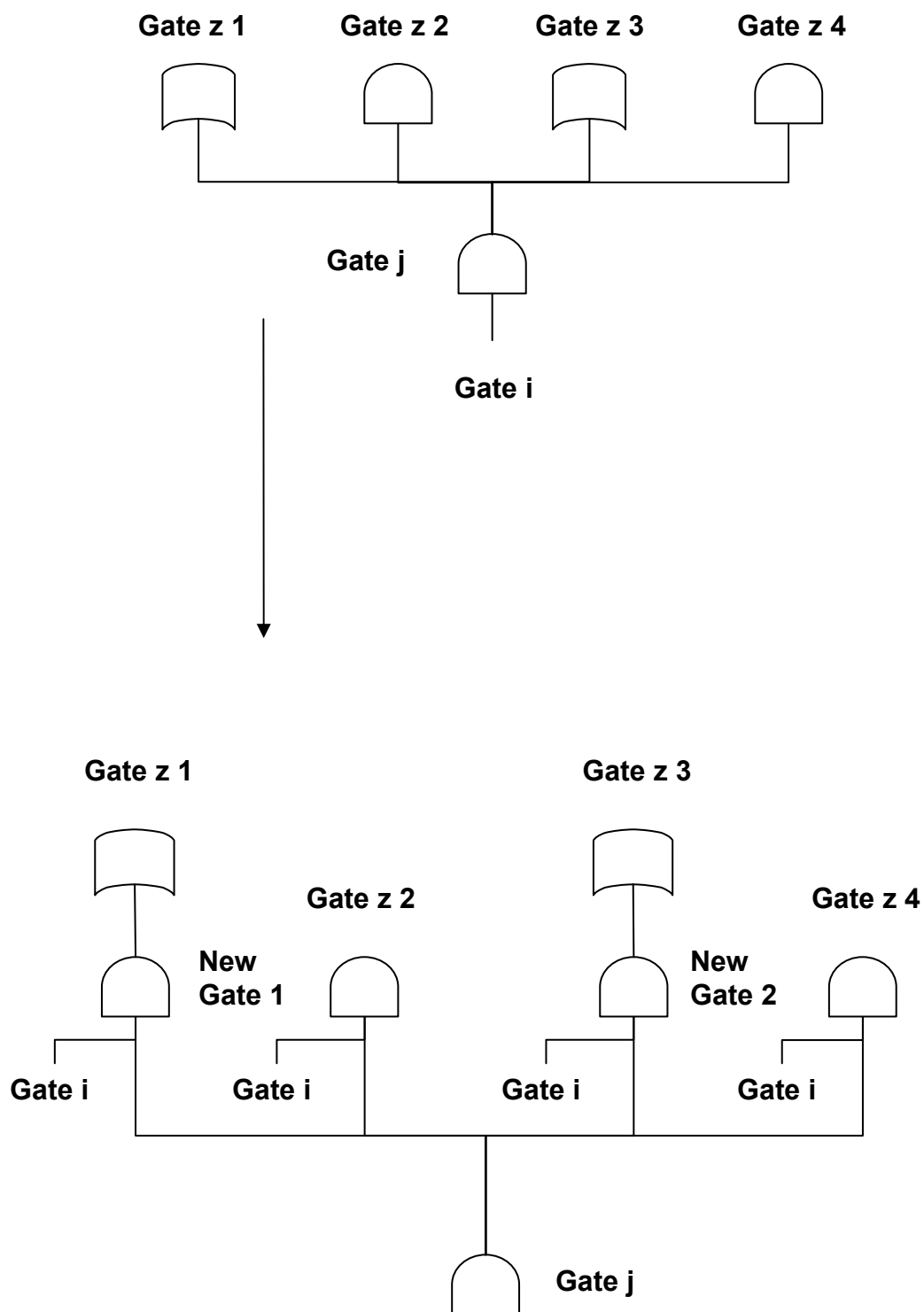


Figure 6.13: Push up technique for multiple appearances of gate *i*

The code applies the push-up procedure as follows. When the code has selected a repeated gate (gate i) and the top intersection gate has been obtained for it then the push up process can begin. The code goes through the parent gates of gate i which are referred to as gate j, one by one. The push up process is applied and the code records all of the gates where gate i is going to be pushed up to, in an array. The code then goes through the array of gates and checks to see if gate i already exists as an input to it. If no then gate i is put has an input to the gate, altering the input and parents arrays. Once this has been done for gate j then gate i is removed from it, altering the input and parents arrays. The code does this for every repeated gate.

Considering again the example, the top intersection gate for gate 10006 as been obtained and is gate 10003, the pushing up process can start. The code looks up the first occurrence of this gate from the list in the inputs array and sets its parent to be gate j which is gate 10005. It considers the possible parents of gate j (10005), which there is only one, gate 10004. Gate 10004 is the same type as gate 10005 (OR) and this is not the top intersection gate. Therefore the code considers gate 10004 and looks up its parents. It only occurs once, as input to gate 10003. Gate 10003 is of different type to gate 10004, therefore the furthest that this first occurrence of gate 10006 can be pushed up to is gate 10004, so the code records this gate. Now the code goes through all of the gates recorded in the array where gate 10006 is going to be pushed up to from its first appearance. There is only one gate in the array gate 10004. First the code scans through gate 10004 inputs to see if gate 10006 is already there. If it is there then there is no need to insert it. However, it does not appear there, therefore gate 10006 is inserted, altering the input and occurrence arrays of the data. Now the input of gate 10006 to gate 10005 is removed. The next occurrence of gate 10006 is considered which has parent gate 10007 that has type OR. It occurs once in the tree as input to gate 10003 of type AND. This is of a different type to gate 10007 therefore the second occurrence of gate 10006 is not pushed up at all. The changes to the tree and data are shown in figure 6.14.



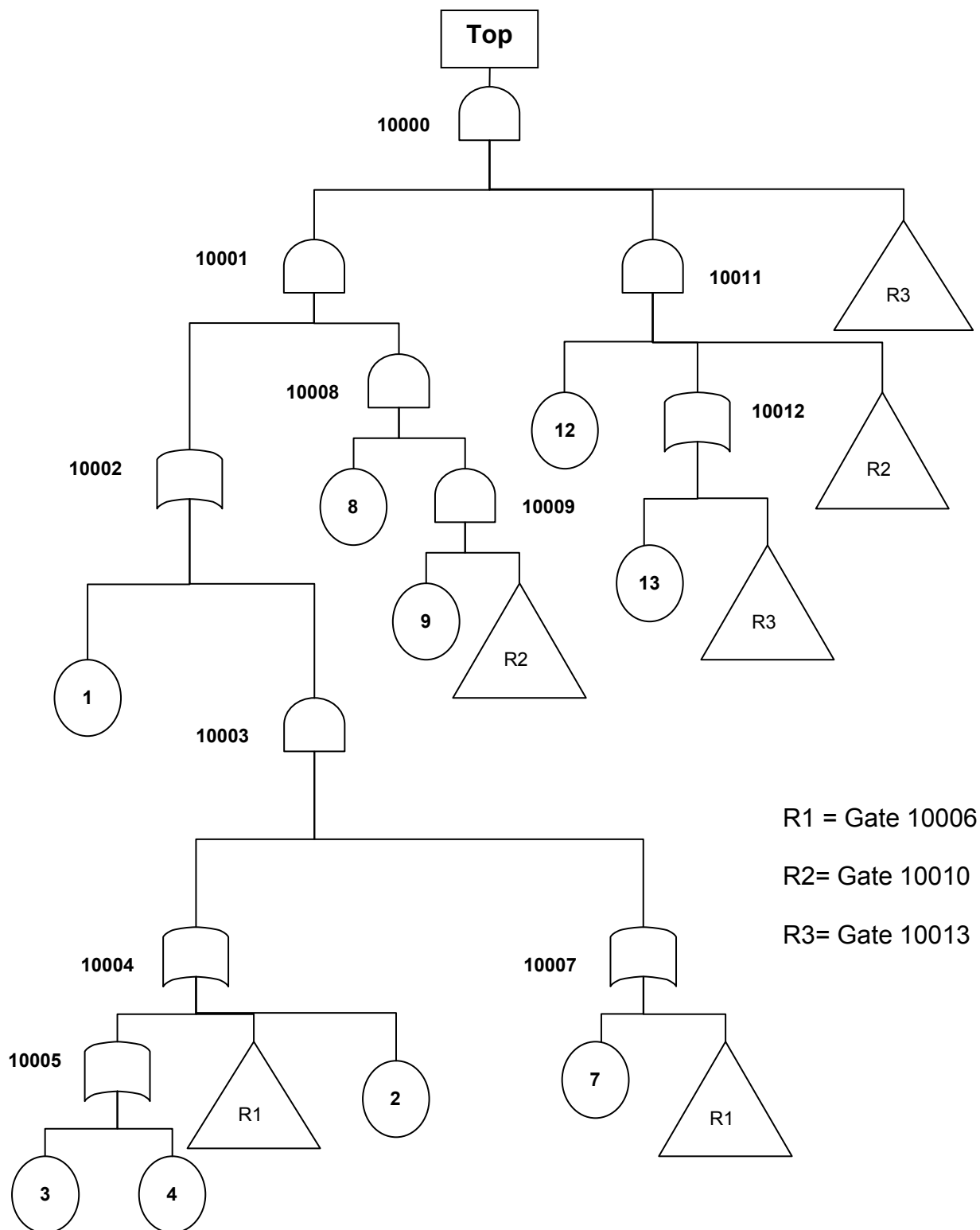


Figure 6.14: The example fault tree after pushing up gate 10006

## **Common-input Push-up 1**

The aim of this stage is searching through the whole tree for the repeated gate  $i$  in the form of structure pattern that appears in Figure 6.3 then applying common push-up to it. The code does this in two steps. The first step identifies and records the gates where the repeated gate is going to be pushed out of, which is referred to as current gate 2. The second step goes through the recorded gates one by one, from this information the code inserts and removes the repeated gates where necessary in the tree data and also alters the parents gates data.

For the first step the code goes through all of the parent gates of the repeated gate. This gate being considered is referred to as current gate 1. Then the code goes through all of the parent gates of current gate 1. This gate is then considered and is referred to as current gate 2. If the current gate 1 has the same type as current gate 2 then the search ends and the code moves on to the next current gate 1. However if the gates are of different type then the search continues. For it to continue it is also required that current gate 2 must not have any input events, just gates. The code goes through the inputs of current gate 2 one by one. This gate is referred to as current gate 3. If all current gates 3 are of a different type to current gate 2 and at least one of their inputs is the repeated gate then a common push can be applied, this is shown in figure 6.15. The code goes through the criteria required and if it passes then current gate 2 is recorded in an array for the second step. Then the whole process is repeated with the next current gate 1.

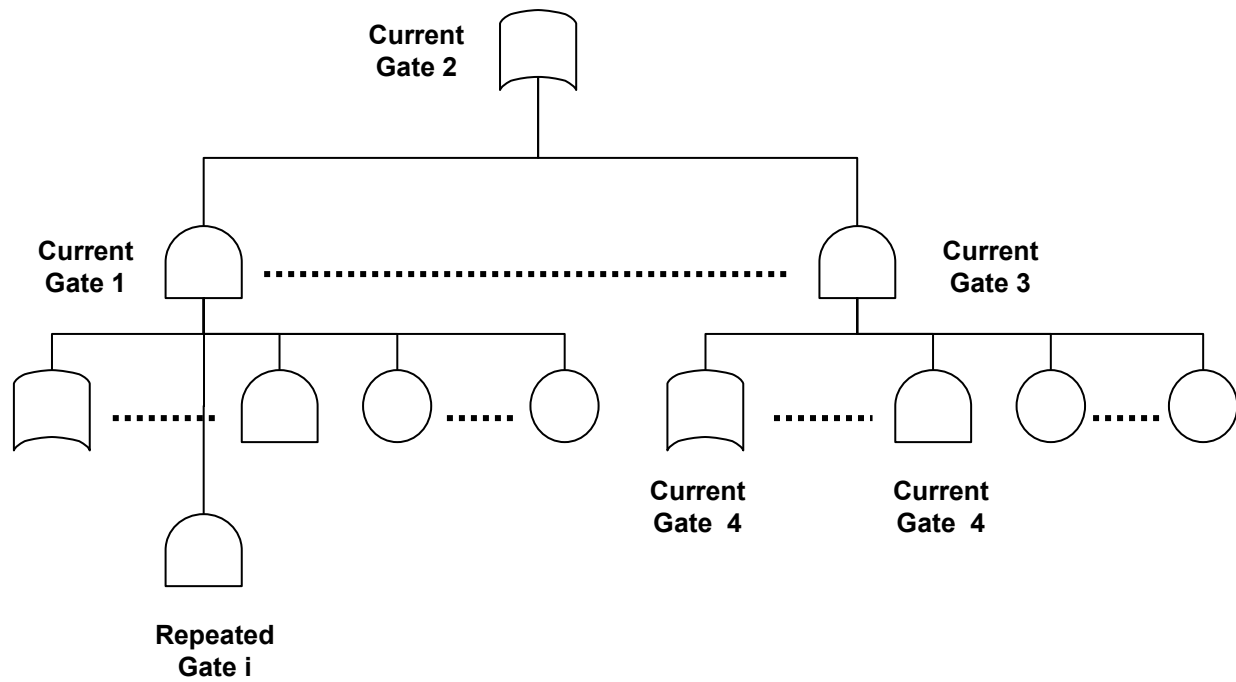


Figure 6.15: Example of common push up technique

The second step has two parts to it, the first part inserts the repeated gate in its new position and the second removes the repeated gate from its old position. The code goes one by one through all the gates (referred to as current gate 2) in the array that was recorded in the first step. The code for the first part goes through all the occurrences of current gate 2. It refers to a parent gate of current gate 2 as 'gate up'. If 'gate up' is a different type than current gate 2 then the repeated gate is inserted as one of 'gate up' inputs only if it does not already appear there. However, if 'gate up' is of the same type as current gate 2 then a new gate is created which goes in between current gate 2 and 'gate up', and is different type to both of them. The repeated gate is inserted as an input to this new gate. The code does this by altering the data of the tree and parents gates array. This is shown in Figure 6.16.

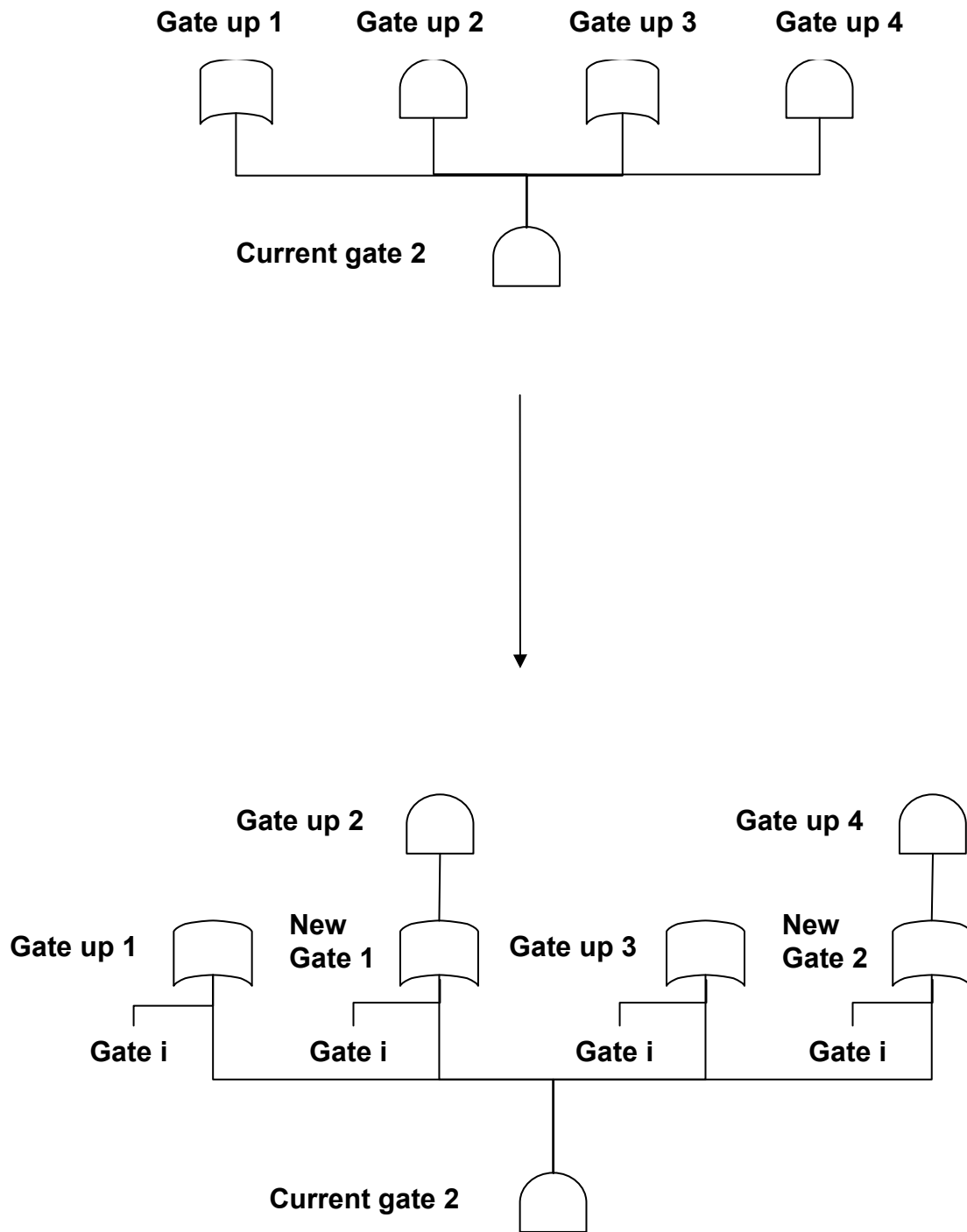


Figure 6.16: Example of inserting the repeated gate for a common push up

The second part deletes the repeated gates as input from current gate 1 as shown in Figure 6.15 to Figure 6.17. However, there is a special case to consider. If current gate 1 has at least one parent gate that is not belonging to the array that records the common push up, then if the repeated gate is deleted then those parent gates will be incorrect. This problem is overcome by creating a new gate which is the same as current gate 1 except that the repeated gate is not included in the inputs. Now the code goes through the parent gates of current gate 1 and if it does belong to the common push up array then the data is changed so that it is an input of the new gate instead of the current gate 1, this is shown in figure 6.18.

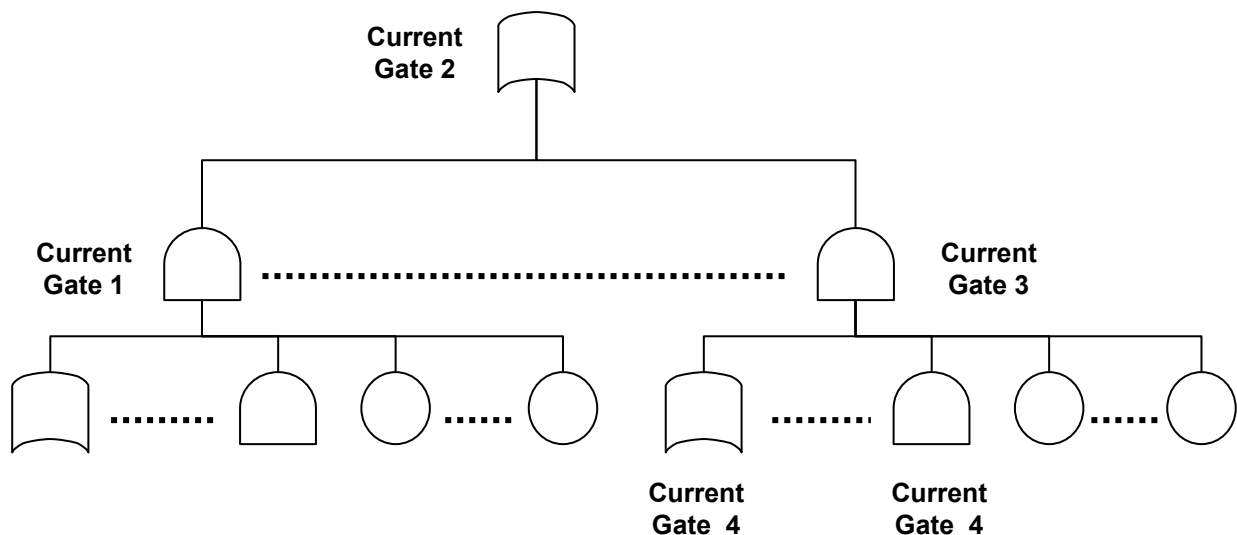


Figure 6.17: Example of removing the repeated gate for a common push up

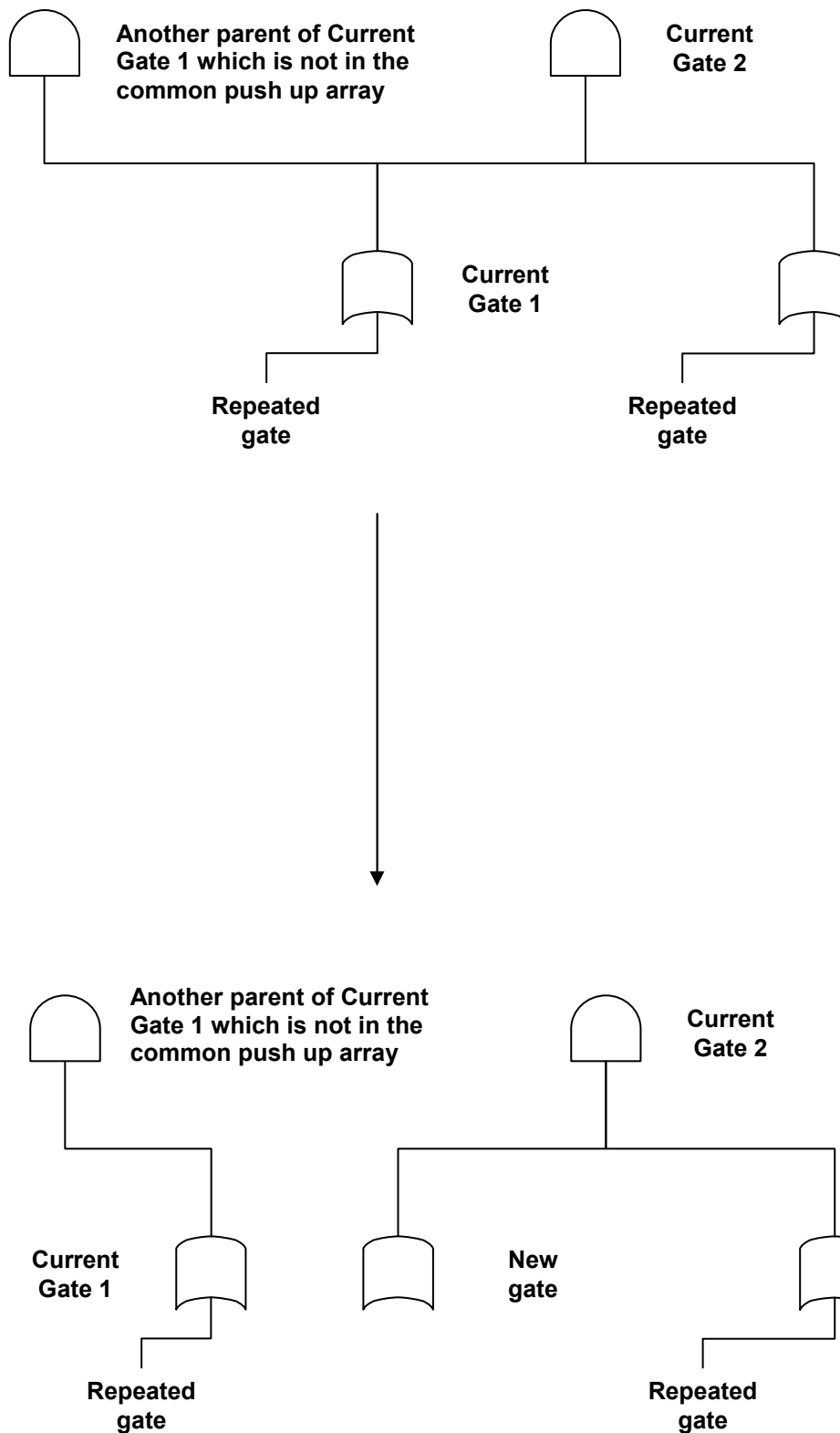


Figure 6.18: Special case for common push up for removing the repeated gate

Now back to the example. In figure 6.14 the current fault tree is shown. The code takes the first occurrence of gate 10006, which has parent gate 10004 and this is referred to as current gate 1. Next the code takes the parent gates of gate 10004 which there is only one, gate 10003, and this is referred to as current gate 2. Current gates 1 & 2 are of different type, therefore continue to the next step. The code scans through the inputs of gate 10003, there are two which are both a different type than gate 10003 and have the repeated gate 10006 as an input. Gate 10003 also has no event inputs therefore the code records gate 10003 in the common push up array. Now the code moves on to the next occurrence of gate 10006, which has parent gate 10007. It also identifies that gate 10003 needs to be recorded in the common push up array, but since it is already recorded it is not necessary to record it again. Now step two, the elements of the common push up array are gone through one by one. There is only one element in this array gate 10003. Therefore the code scans through the parent gates of gate 10003, there is only one gate 10002. Gate 10002 is of a different type to gate 10003 therefore the repeated gate 10006 is inserted to gate 10002. Now the code goes through the remove process part 2 of the step 2. It checks for the special case which was discussed in the algorithm description above. The special case does not occur in this example. Therefore gate 10006 is just removed from gates 10004 and 10007. The changes that have been made are shown in figure 6.19.

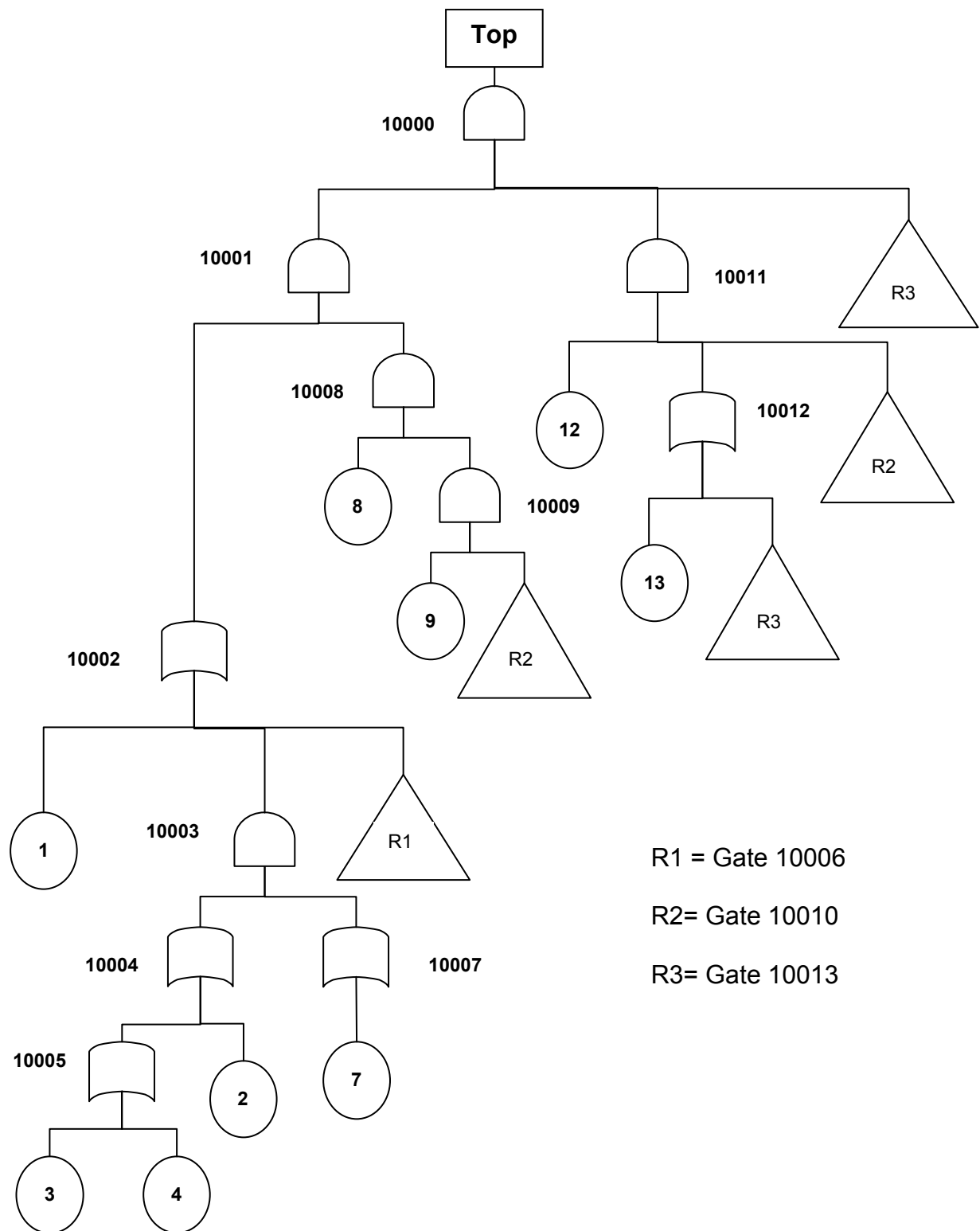


Figure 6.19: Fault tree after a common input push up of gate 10006



## **Push-up 2**

Now because there was a common input push up activated the process is repeated because the tree has now changed, more push-ups may be possible. Going back to the example in figure 6.19 gate 10006 cannot be pushed up the tree structure any more. Since the two gates above it are of different types. Common push up is considered again. Since there is only one occurrence of gate 10006 there is no common push up.

## **Top intersection gate for repeated gate 2 ( gate 10010).**

The two paths which are traced up the tree from the two appearance of gate 10010 (R2) intersect at gate 10000. Therefore gate 10000 is the top gate of intersection for the repeated gate 10010.

## **Push-up 3**

The first parent gate of the first occurrence of gate 10010 is gate 10009 which has gate type AND. The three gates above are of the same gate type therefore gate 10010 can be pushed up to gate 10000 and removed as an input from gate 10009. Now considering the second occurrence of gate 10010. This is an input to AND gate, the gate above is also an AND gate, therefore this gate can be pushed up to gate 10000. However the gate 10000 already has an input gate 10010. This means that gate 10010 is just deleted as an input to gate 10011. The tree after these changes is shown in figure 6.20. There is only one appearance of R2 in the structure now and therefore the Common-input Push-up and Elimination techniques are not necessary.

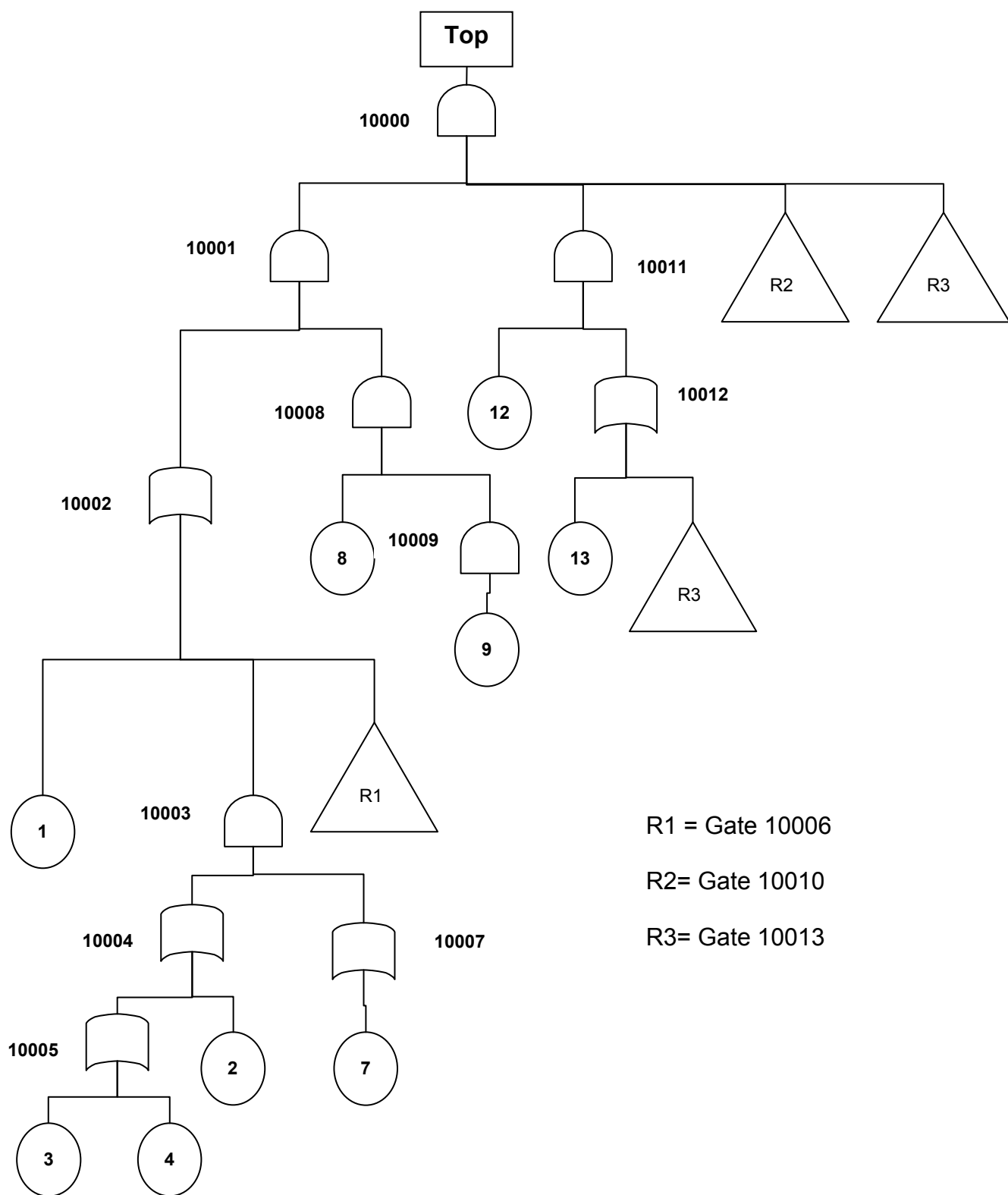


Figure 6.20: Fault tree after a push up of gate 10010.

Now consider the next and final repeated gate 10013 (R3). Gate 10013 appears twice in the tree under gates 10000 and 10012.

#### **Top intersection gate for repeated gate 3 (gate 10013).**

The top intersection gate of the two appearance of gate 10013 (R3) is the top gate 10000.

#### **Push-up 4**

Consider first the appearance of gate 10013 which is an input to the OR gate 10012. The gate above is gate 10011 which is an AND gate therefore the gate cannot be pushed up. Now considering the second appearance which is already an input to the top gate in the fault tree therefore this too cannot be pushed up any more.

#### **Common-input Push-up 2**

There is no structure pattern in the tree of similar to that shown in figure 6.3. Therefore no common-input Push-up can be done.

#### **Elimination**

The Elimination technique simplifies structure patterns of the form shown in figures 6.4 and 6.5 and described in section 6.2.3. Processing the technique requires the identification of pairs of primary and secondary gates as defined in that section. The implementation of the programming for performing this technique is straightforward. It searches for the pairs of primary and secondary gates and removes the appropriate gate depending on if the primary and secondary gate types are the same or different. However there is one complication which needs consideration because of how the data of the fault tree was stored in the arrays. All the gates are only defined once even if the gate is repeated. For a gate which is repeated and contains a potential secondary gate branching from beneath it (this gate is referred to as gate x) then depending on which type of parent gate of gate x that is being considered

there may exist a primary AND gate, primary OR gate or not one at all. Modifying the potential secondary gate for a primary gate above it may be incorrect when considering another parent gate of gate x. For example in figure 6.21 there is a fault tree shown that has a repeated gate (referred to as GR), which is going to be considered. Gate GR is an input to gates G1, G5 and G7. There are also other repeated gates in the tree. Gate G1 is an input to gates G2 and G4. Gate G4 is an input to gates G5 and G6. So gate G1 is a secondary gate and G5 is a primary OR gate and G7 is a primary AND gate for gate GR. This information is obtained by starting with the secondary gate G1 and tracing up the tree searching for primary gates. There are three possible paths. The first path contains gates G1, G2, G3 and the top gate this path does not contain any primary gate. The second path contains gates G1, G4 and G5 which is an OR primary gate. Therefore elimination can be carried out by removing gate GR as a input to gate G1, but this will be incorrect for the first path and incorrect for the third path which leads to a primary AND gate G7. The solution to overcome this problem is to first obtain all of the possible paths from the secondary gate. If the paths lead to different primary gate types or no primary gate at all, then the gates of the paths that belong to more than one path with different primary gate outcomes are copied. These copied gates have their labels modified and are then replaced with the original gate for one of the paths.

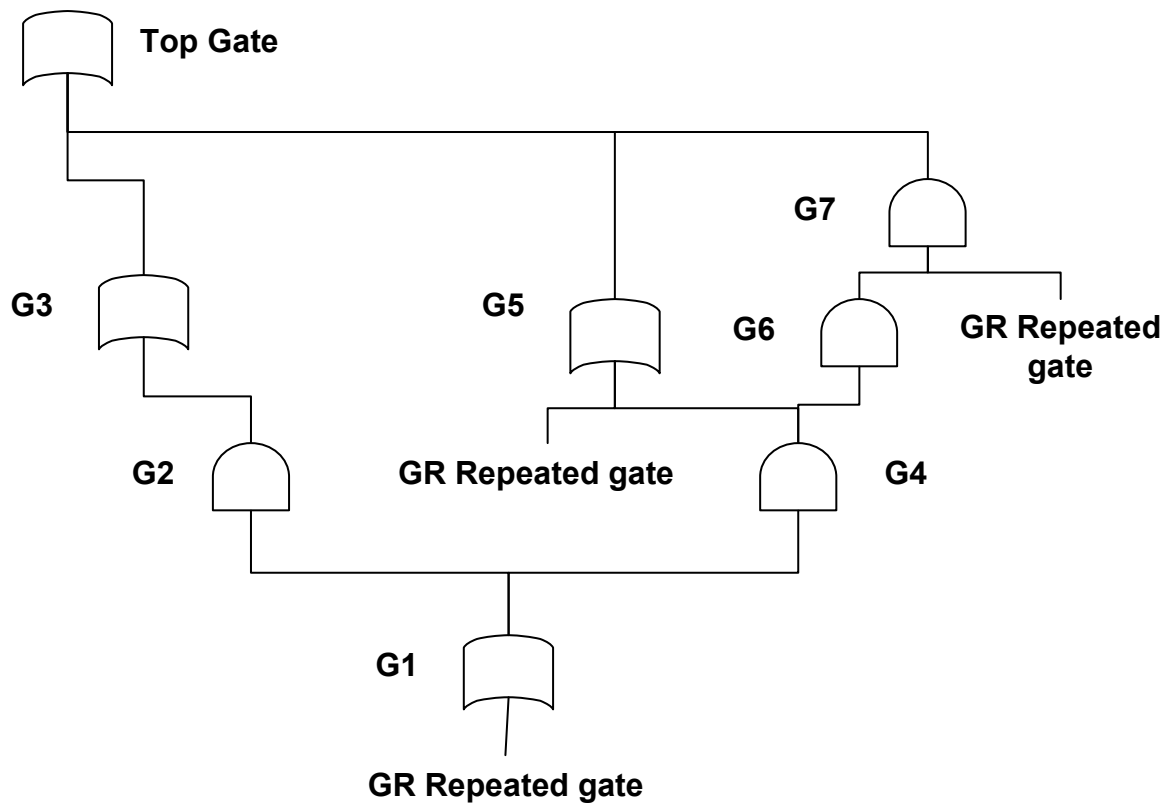


Figure 6.21: Example of a fault tree which is going to have elimination performed on it

Since the main example being worked through does not feature the special cases of the Elimination procedure, its procedure and coding is explained by application to the fault tree shown in figure 6.21. The secondary gate, labelled G1, has as an input the repeated gate GR. First the paths to primary gates or the top gate if a primary gate is not encountered are obtained. The code scans up from the secondary gate G1 to establish all the possible paths and records them in an array. There are three such paths. The first path is gates G1, G2, G3 and the top gate. This path does not encounter any primary gates. The second path is gates G1, G4 and G5 which is a primary OR gate. The third path includes gates G1, G4, G6, and G7, which is a primary AND gate. The gates in the paths are labelled as shown on the fault tree in figure 6.22. The first path is labelled as P<sub>TOP</sub>, the second path is labelled as P<sub>OR</sub> and the third path is labelled as P<sub>AND</sub>. Now that the paths have been obtained the building and rearranging of the gates can start.

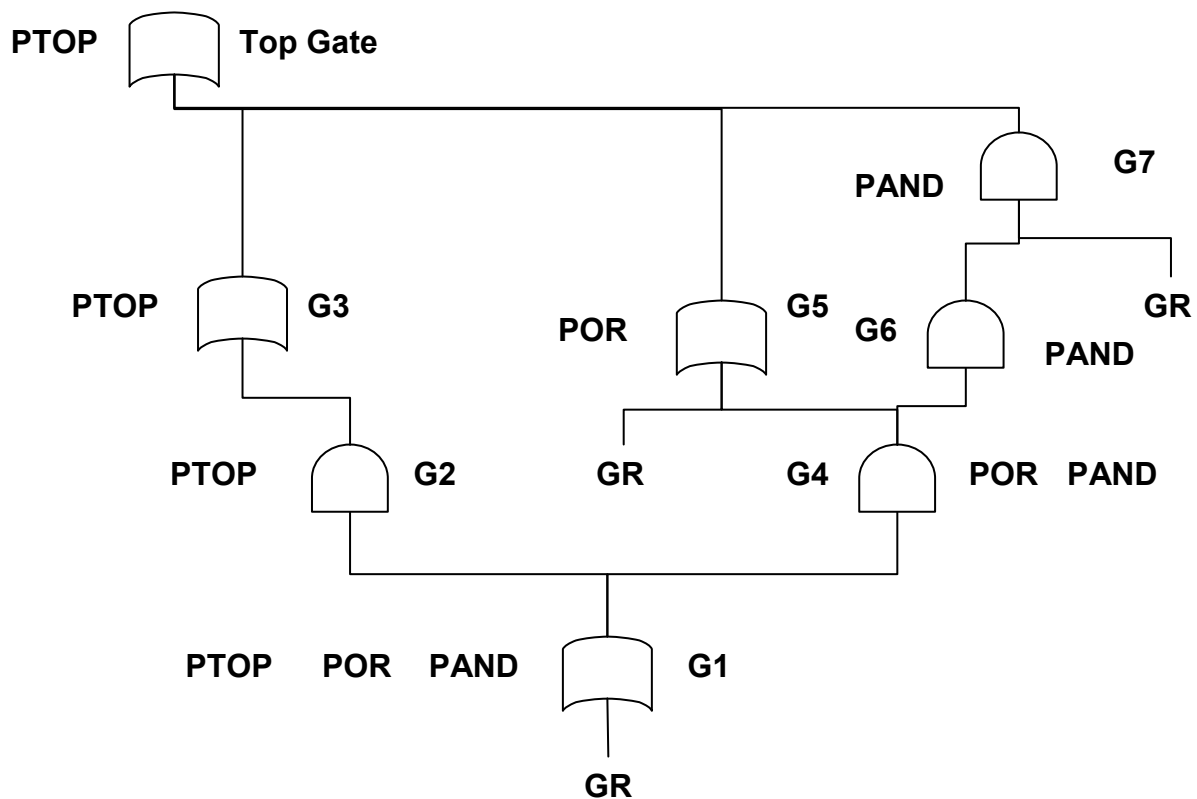


Figure 6.22: Example of a fault tree for elimination which included the paths on it

First the paths which lead to a primary gate of opposite gate type to the secondary gate are considered. Since the secondary gate G1 is an OR type then the paths that lead to a primary AND gate are considered first. The code scans up path PAND. It misses gate G1 and goes on to gate G4 because gate G4 will be the first to be modified. First the gate in the path is checked to see if it exists in other paths. Gate G4 also exists in path POR therefore gate G4 must be copied and modified and replaced by the copied gate for path PAND. The copied gate is referred to as gate N1. Now gate N1 is modified for the path by removing the secondary gate G1 as an input. The next gate in the path to be considered is gate G6. Gate G6 does not exist in any other paths therefore it can be just modified by replacing the input of gate G4 with new modified gate N1. Now the path that leads to a primary AND gate does not overlap with any other path. The changes to the fault tree are shown in figure 6.23.

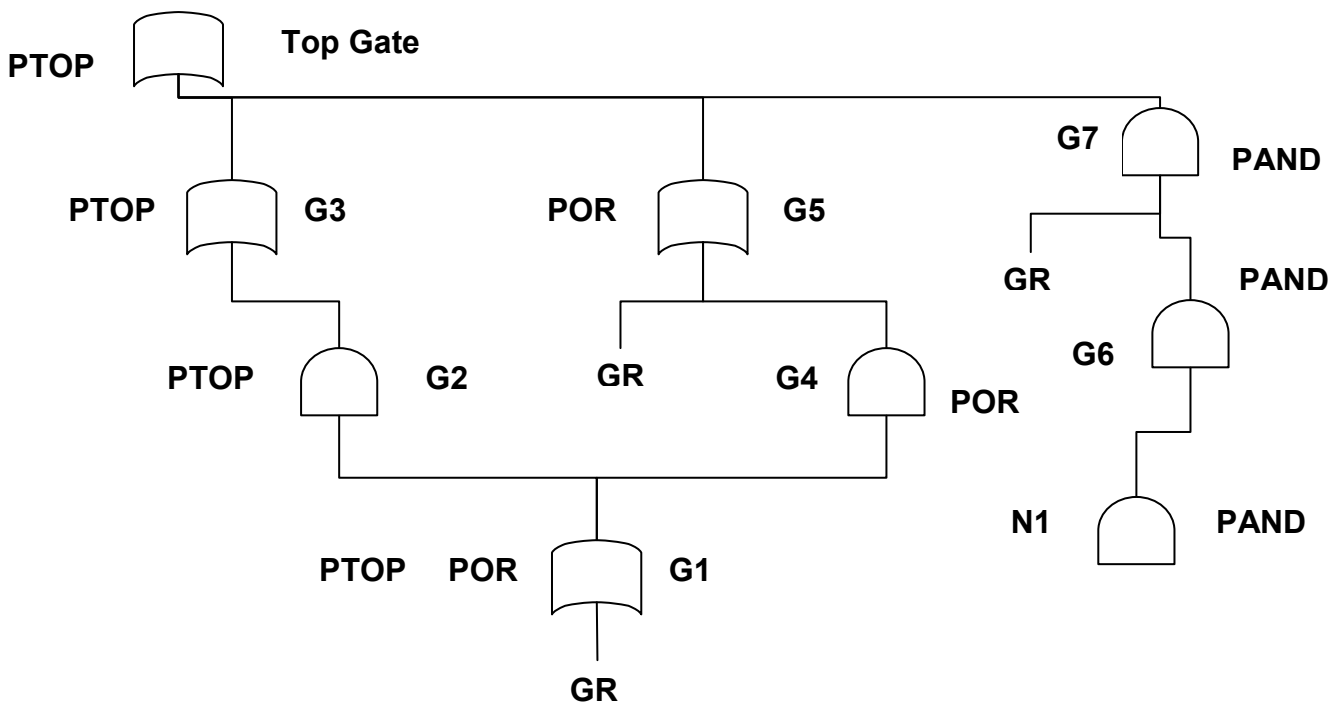


Figure 6.23: Example of a fault tree after first elimination

Next the code scans along the path which leads to primary gates of the same gate type of the secondary gate. Since the secondary gate is an OR type therefore the code scans up path POR. Starting with gate G1 since this will be the first gate to be modified. Gate G1 also belongs to the path PTOP therefore it is copied and modified. The copied gate is referred to as gate N2 and it is modified by removing the repeated gate GR from the inputs. The next gate considered in the path is gate G4. Gate G4 does not belong to any other paths so it is just modified by replacing the input of gate G1 by the new modified gate N2. Since none of the paths which lead to different outcomes now intersect with each other, all of the possible eliminations are performed. The fault tree changes are shown in figure 6.24.

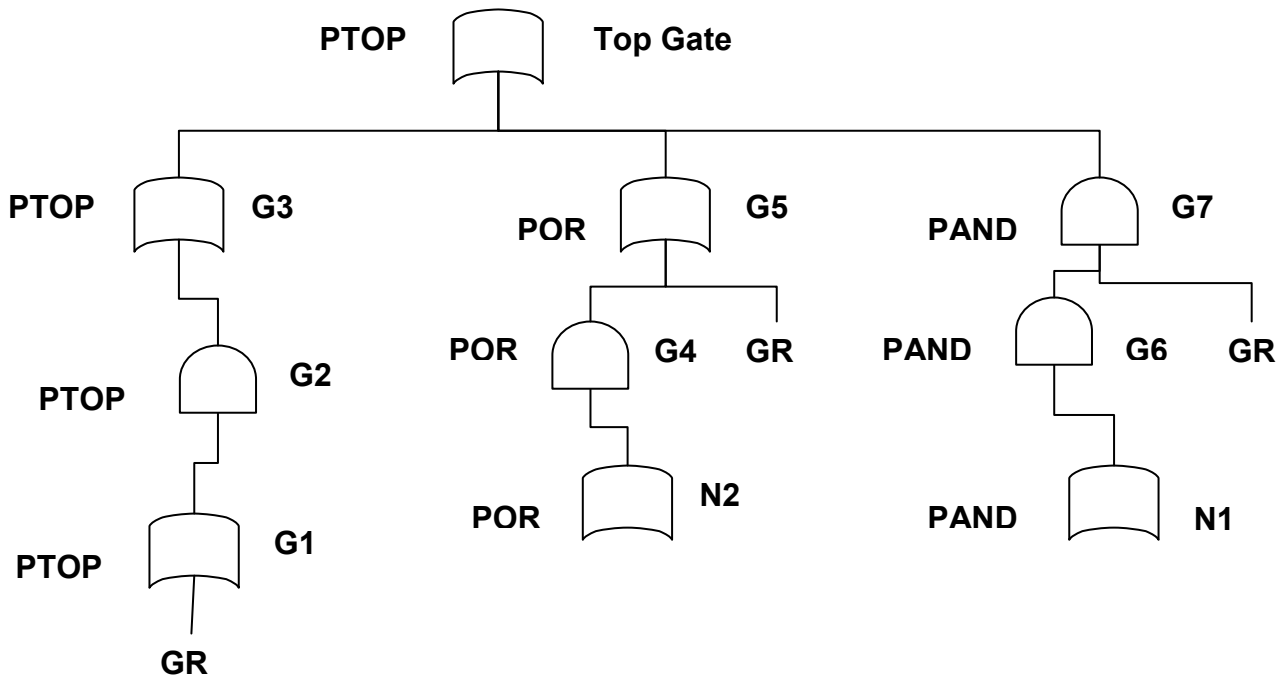


Figure 6.24: Example of a fault tree after second elimination

Considering again (figure 6.20) the main example, for the elimination process to be applied it needs at least two occurrences of a gate to be considered. Since gates 10006 (R1) and 10010 (R2) only appear once there is no elimination possible. Considering the final repeated gate 10013 (R3), this appears twice in the tree under gates 10012 and 10000.

### Elimination 1

The code considers all the possible secondary gates of gate 10013 one by one. The first secondary gate is gate 10012. The code scans up the fault tree from gate 10012 searching for primary gates and recording the paths to them. All the gates above gate 10012 only occur once in the tree therefore there is only one path which leads to a primary AND gate. The path consists of gates 10012, 10011 and the top gate. Now that the paths have been recorded the second part of the procedure can be performed which copies, modifies and



replaces gates. Since there is only one path there are no overlaps of gates listed between difference paths and therefore none of the gates will have to be copied which makes this elimination very simple. The code starts with the gate 10011 in the path and skips gate 10012 since this will be the first gate to be modified because the primary and secondary gates types are different. Since gate 10011 does not belong to another path it is modified by removing the secondary gates 10012 from its input. The resulting changes of the tree are shown in figure 6.25. The code moves on to the next secondly gate of gate 10013 which is the top gate. Since the gate 10013 only appears once in the tree there is no primary gate to pair up with therefore there is no more possible elimination for gate 10013.

### **Apply the restructuring techniques to the repeated events**

The same restructuring techniques push-up, common push up and elimination are also applied to the repeated events. In the example there are only two repeated events 10 and 11 and they do not have any possible moves from the three techniques therefore the tree is not altered.

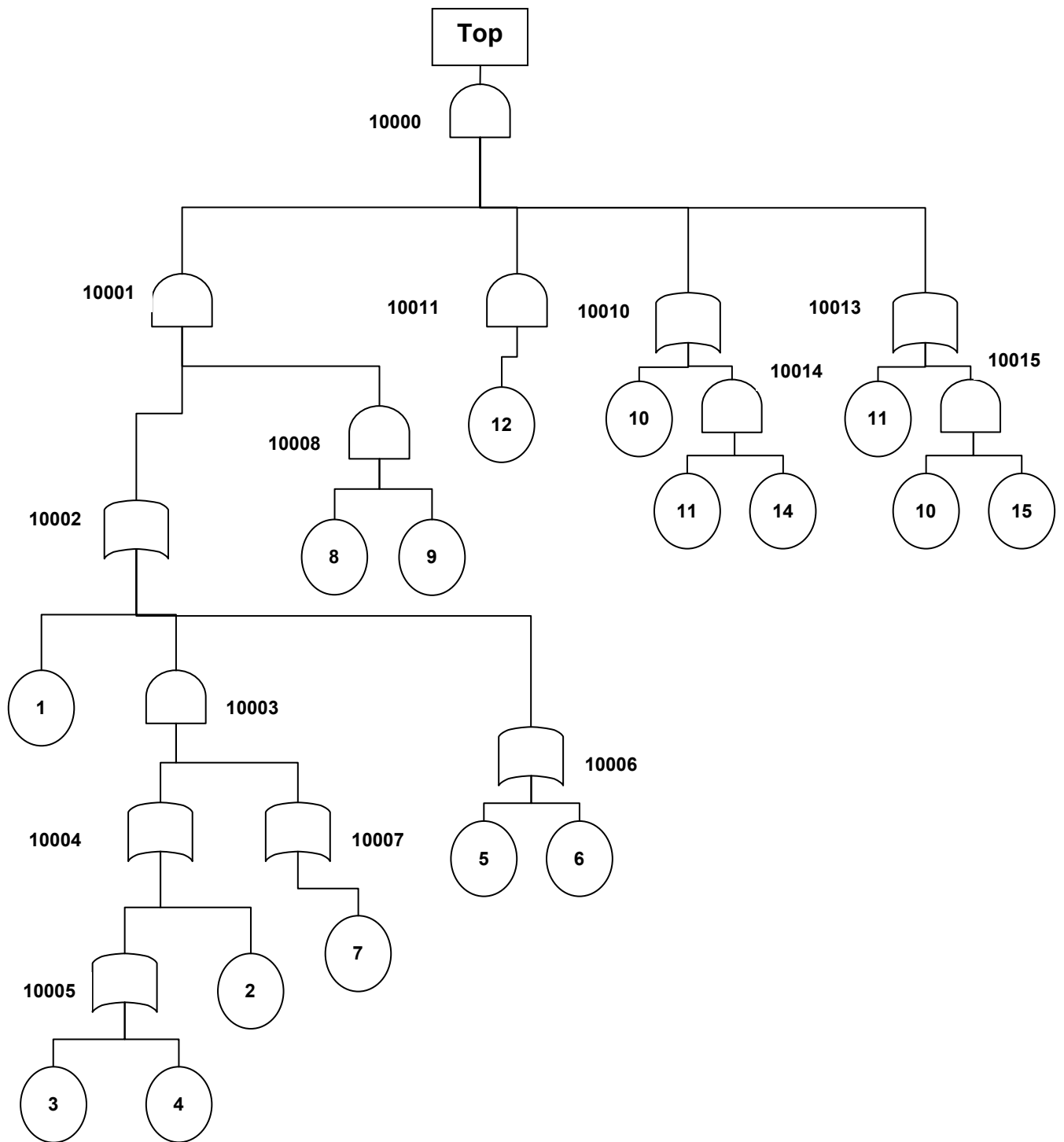


Figure 6.25: Fault file after all the possible push-up, common-input Push-up and elimination being preformed

The next stage of processing in the code deletes any gate from the tree with just one input and that input is pushed up to the parent gate of the gate which has been deleted. In the example shown in figure 6.25 there are two gates with a single input which will be deleted: gates 10007 and 10011. So the event 12 which is an input to gate 10011 becomes an input to gate 10000. Similarly event 7 is pushed up to gate 10003.

## Identifying modules

The code identifies modules by going through every repeated gate and event and tracing up the tree from their location to the highest level intersection gate of the particular repeated gate or event. Every gate that is traced through has an array associated with it which records all the repeated gates and events that get traced through it. Returning to the example shown in figure 6.25, there are no repeated gates in this tree and just two repeated events 10 and 11. The code considers event 10 first which appears twice in the tree as inputs to gates 10010 and 10015 and has a top intersection point at gate 10000. The code traces event 10 from its two appearances to the top intersection gate it passes on the first path through gate 10010, and the second path through gates 10013 and 10015 which are prevented from being modules. The event 11 appears as inputs to the gates 10014 and 10013 which obtain the top intersection gate 10000. The code traces event 11 from its two appearances to the top intersection gate it passes through gate 10013 on the first path, and gates 10010 and 10014 on the second path which are prevented from being modules. This information is recorded and shown in the array below. In the first one-dimensional array in element 10 which relates to gate 10010 is the value 2 which means that two repeated gates or events which occur elsewhere in the tree prevent it from being a module. In the two-dimensional array in column 10 are the values 10 and 11 which are the events preventing gate 10010 from being a module. A similar pattern of information is seen in the values in column 13 representing gate 10013. The arrays also show that gates 10014 and 10015 have one repeated event which prevents it from being a module which are events 10 and 11 respectively. The code can now identify which gates are modules and which ones are not. If the gate does not have any repeated gates

or events preventing it from being a module then the elements of the one-dimensional array entry relating to the gate will be equal to zero. Otherwise, if it is not equal to zero then it has repeated gates or events preventing it from being a module.

Therefore the array shows gates 10010, 10013, 10014 and 10015 are not modules and every other gate is.

One-dimension array where each entry corresponds to a gate in the fault tree and contains the number of repeated gates and events that prevent it from being a module.

0	0	0	0	0	0	0	0	0	2	0	0	2	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Figure 6.26: One-dimension array

Two-dimension array contains those repeated gates and events prevent any gate it from being a module.

									10			10	11	10
									11			11		

Figure 6.27: Two-dimension array

## Factorization

The Factorization procedure searches for two non-module gates, which always occur as inputs to gates which are all of the same type, and when these gates are combined they form a module. Therefore these two gates are dependent but independent of the rest of the tree and so their combination forms an independent sub tree. For this procedure to be performed information is used from the arrays above which contain every gate and a list of which gates and events, appearing on branches beneath it, prevents it from being a module.

Having formed the above arrays, the code scans through all the possible non-module gate pairs. The gates in the pairs are referred to as gate 1 and gate 2. If gate 1 and gate 2 are always inputs to the same gates and all of those gates are of the same gate type then the factorization procedure can continue. Otherwise they cannot be combined to form a module and the next gate pair is considered. Now the second stage of this procedure tests if gate 1 and gate 2 were combined would they be a module. If the combination is a module then any gate or event that branches off under either of gates 1 or 2 would only appear under gates 1 or 2. Therefore the code obtains information about the gates and events beneath gates 1 and 2. The code scans beneath gates 1 and 2 and records all the repeated gates and events beneath them both in an array which is referred to as list. Every gate and event in the list is then examined by going through every appearance of the gate or event and tracing up the tree. The code traces up the tree to see which gates is encountered first out of gate 1, gate 2 and the top gate. If for any trace the top gate is encountered before gates 1 and 2 then gates 1 and 2 cannot be combined to form a module. However if all the traces encounter gates 1 and 2 before the top gate then they can be combined to form a module. If gates 1 and 2 can be combined to form a module then the code creates a new gate with the same gate type as the parents of gates 1 and 2 and adds gate 1 and gate 2 as inputs. This new gate replaces gates 1 and 2 everywhere they appeared in the fault tree.

In the example shown in figure 6.25, the code only finds one pair of gates 10010 and 10013 which are non-modules and always occur together. The list of repeated gates and events below these two gates contains only events 10 and 11. These events in the list are traced up from every appearance of the tree to see if they encounter the top gate or gates 10010, 10013 first. Both events from both of their locations encounter gates 10010 or 10013 first before the top gate. Therefore gates 10010 and 10013 can be combined to form a module. The code created a new gate referred to as 10016 which of gate type AND (since gates 10000 is an AND type). The new gate has as inputs gates 10010 and 10013 and is itself an input to gate 10000 which replaces gate 1 (10010) and gate 2 (10013). These final changes to the tree are shown in figure 6.28.

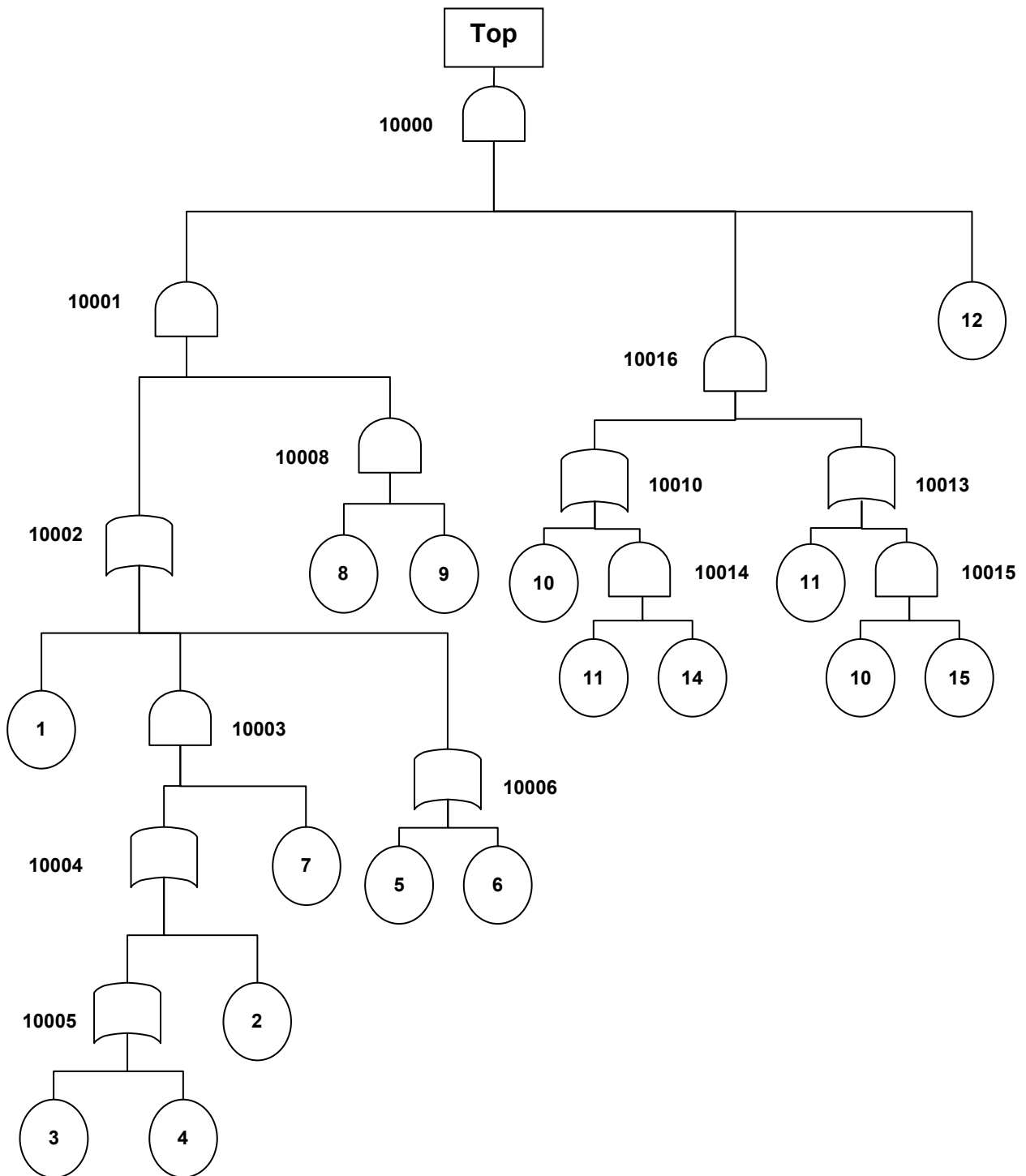


Figure 6.28: fault tree after the Factorization process

After the factorization the code deletes any gate from the tree with just one input and that input is pushed up to the parent gate of the gate which has been deleted. Even those which have already been done before the factorization procedure as a special case which creates a gate with only one input. If two gates were factored that were the only input of their parent gate then after factorization the parent gate would only have one input this is shown in figure 6.29.

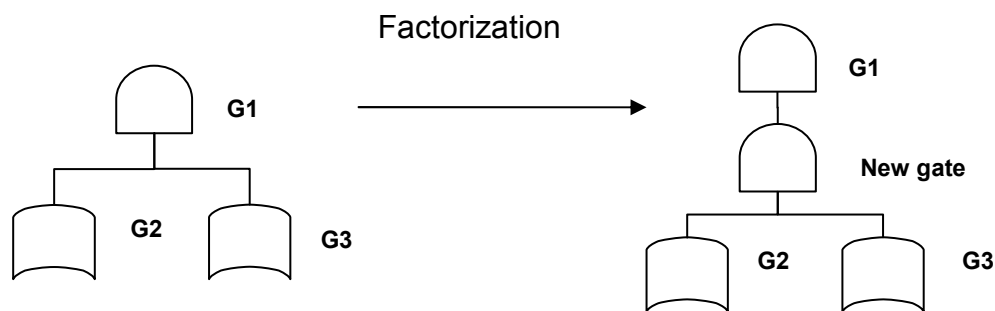


Figure 6.29: Special case of factorization that creates an unnecessary gate

## 6.4 Results

### 6.4.1 The test mission example

In the previous chapter the fault trees for phases of a UAV mission, shown in date format in Appendix B under mission set 9, were created with top events representing catastrophic failure or mission failure. All the fault trees are put together in one data file. This is done because the tree phases share a lot of common branches. The number of distinct gates in all the phases is 980 and the distinct events 1007. The majority of the fault trees developed for the phases are only slightly smaller than a fault tree for the entire mission, since there is a lot of common sub-trees between the phases. Some of the gates are

repeated through the mission several times because of common aircraft functions and the energy sources required for them.

#### **6.4.2 The effects of restructuring and taking out modules on the trees**

All the phase fault trees are restructured by the method discussed in this chapter and all the possible modules are taken out. Tables 6.3 and 6.4 show the number of distinct gates and events for the catastrophic and mission failure fault trees and tables 6.5 and 6.6 show them after the restructuring and modules are taken out. They are significantly reduced, table 6.7 and 6.8 shows the percent reduction of the number of distinct gates and events. The percentage reduction in the number of distinct gates ranges from 80.1% to 96.3% with an average of 90.8%. The percentage reduction in the number of distinct events ranges from 76.7% to 93.2% with an average of 87.5%. This has significantly reduced the size of the fault trees. The total number of modules taken out of all the fault trees is 106. This successful simplification is down to the common functions and energy sources required between the phases that occur many times throughout the fault tree. These common sub-sections which spread throughout the fault trees at different levels pervert many higher level gates from being modules. When the restructuring techniques are applied common sub-sections are brought closer together that changes a large number of non-module gates to module gate. Since the common sub-sections that prevented them from being modules have been pushed past it. Also contributing to the saving of time was the restructuring and factorising techniques applied to the particular Dc and Ac power sub-section which share common components and are the most complex parts of the fault trees. The restructuring brought them together and the factorization made it so they could be combined to form a module therefore the BDD for it was built offline saving online time.



Mission Phase	Catastrophic Failure Fault Tree	
	Number of Distinct gates	Number of Distinct Events
Start up (1)	51	60
Taxi out (3)	51	60
Take off (5)	682	805
Climb (7)	582	692
Cruise (9)	582	691
Decent (11)	705	847
Land (13)	689	781
Taxi in (15)	51	60
Shutdown (17)	51	60

Table 6.3: Characteristics of the catastrophic phase failure before restructuring and modules are taken out

Mission Phase	Mission Failure Fault Tree	
	Number of Distinct gates	Number of Distinct Events
Start up (2)	699	841
Taxi out (4)	556	659
Take off (6)	668	791
Climb (8)	719	862
Cruise (10)	697	838
Decent (12)	705	847
Land (14)	812	937
Taxi in (16)	333	384
Shutdown (18)	52	61

Table 6.4: Characteristics of the mission phase failure before restructuring and modules are taken out

Mission Phases	Catastrophic Failure Fault Tree			
	Number of Distinct gates	Number of Distinct Events	Number of Nodes used Making Phase BDD	Number of Nodes in the phase BDD
Start up (1)	10	15	116	31
Taxi out (3)	10	15	116	31
Take off (5)	32	66	871	229
Climb (7)	35	58	1359	425
Cruise (9)	35	58	1358	425
Decent (11)	35	61	1431	449
Land (13)	30	61	638	218
Taxi in (15)	10	15	116	31
Shutdown (17)	10	15	110	31

Table 6.5: Characteristics of the catastrophe phase failures after restructuring and module are taken out

Mission Phases	Mission Failure Fault Tree			
	Number of Distinct gates	Number of Distinct Events	Number of Nodes used Making Phase BDD	Number of Nodes in the phase BDD
Start up (2)	30	60	654	232
Taxi out (4)	30	56	565	213
Take off (6)	32	65	860	228
Climb (8)	35	62	1435	450
Cruise (10)	35	61	1430	449
Decent (12)	35	61	1430	449
Land (14)	30	64	698	230
Taxi in (16)	23	33	346	160
Shutdown (18)	10	15	127	31

Table 6.6: Characteristics of the mission phase failures after restructuring and module are taken out

Mission Phases	Catastrophic Failure Fault Tree	
	Percentage Reduction in number of gates	Percentage Reduction in number of events
Start up (1)	80.1	75.0
Taxi out (3)	80.1	75.0
Take off (5)	95.3	91.8
Climb (7)	93.9	91.6
Cruise (9)	93.9	91.6
Decent (11)	95.0	92.8
Land (13)	95.6	92.2
Taxi in (15)	80.1	75.0
Shutdown (17)	80.1	75.0

Table 6.7: Percent Reductions of the catastrophe failure phases after restructuring and module are taken out

Mission Phases	Mission Failure Fault Tree	
	Percentage Reduction in number of gates	Percentage Reduction in number of events
Start up (2)	95.7	92.9
Taxi out (4)	94.6	91.5
Take off (6)	95.2	91.8
Climb (8)	95.1	92.8
Cruise (10)	95.0	92.7
Decent (12)	95.0	92.8
Land (14)	96.3	93.2
Taxi in (16)	93.1	91.4
Shutdown (18)	80.8	76.7

Table 6.8: Percentage Reductions of the mission failure phases after restructuring and module are taken out

The number of BDD nodes used is also shown in the tables 6.5 and 6.6. However these values are not exactly the value relating to the final BDD, since the BDD is created by continually generating new BDD branches which replace the old branches. The old branches are retained within the count since they may be used again when looking-up the previous computations to avoid repetition and also to delete them would take up time, it is more efficient for them to be left. The exact number of nodes in the phase BDDs are shown in the last column of tables 6.5 and 6.6. The number of nodes in the phase BDD is less than number of nodes used making the phase BDDs.

### **6.4.3 The times of the mission calculation**

The results of the times of the mission calculation are shown in table 6.9 for catastrophe failure, and table 6.10 for mission failure. The first column represents the mission in terms of the phases identified by, the phase numbers taken from table 6.9 and table 6.10. The second column is the online time of calculation, once the mission starts. The third column is the offline calculation, all the calculation that can be done before the mission starts. The offline calculation is similar for all the missions as shown in the table it is usually about 6 seconds. The forth column is the number of BDD nodes used for the whole calculation. This is not the number of nodes in the final BDD, as was discussed earlier in the chapter. These online times are very efficient, since big trees are calculated with a real time analysis of under 30 seconds. This is far superior to the ordinary method without restructuring the tree. The ordinary method from chapter 4 section 6 attempted to do just one of these phases (cruise phase) and ran for about 2 hour 34 minutes before running out of memory. Even if it had enough memory the calculation time would exceed 2 hour 34 minute. When the ordinary code inputs all the possible phases and tries to build all the module BDDs, at the offline stage, it run for 10 min and run out of memory. So without restructuring the ordinary code cannot build some of the modules. However this does not apply to all the phases. Phases start up, taxi out and shutdown for catastrophe failure and the shutdown mission

failure. These phases can be calculated as many times as needed by the conventional method. Since these phases are all ground phases almost all of the tree is just the sub-tree that represents the event of the engines being on fire and can be taken out as a module, which just consists of one gate and two events.

Mission	Online times in Seconds	Offline time in Seconds	Number of Nodes
1,3,5,7,9	0.50	6.34	18433
1,3,5,7,9,11,13	2.39	6.31	37729
1,3,5,7,9,11,13,15	2.61	6.38	43684
1,3,5,7,9,11,13,15,17 1,3	3.38	6.33	61771
1,3,5,7,9,11,13,15,17 1,3,5,7	4.45	6.38	74731
1,3,5,7,9,11,13,15,17 1,3,5,7,9	5.83	6.38	83361
1,3,5,7,9,11,13,15,17 1,3,5,7,9,11	8.97	6.33	96994
1,3,5,7,9,11,13,15,17 1,3,5,7,9,11,13,15,17	10.97	6.30	115055
1,3,5,7,9,11,13,15,17 1,3,5,7,9,11,13,15,17 1,3,5,7,9,11,13,15,17	24.97	6.30	180438

Table 6.9: Results of the missions made up from the catastrophe failure phase tree

Mission	Online times in Seconds	Offline time in Seconds	Number of Nodes
2,4,6,8,10,12	2.14	6.31	36552
2,4,6,8,10,12,14,16,18	4.16	6.33	59401
2,4,6,8,10,12,14,16,18 2,4,6,8,10,12,14,16,18	15.50	6.32	137461
2,4,6,8,10,12,14,16,18 2,4,6,8,10,12,14,16,18 2,4,6,8,10,12,14,16,18	35.61	8.13	215521

Table 6.10: Results of the missions made up from the missions failure phase tree

#### 6.4.5 Conclusion and further work

The results are very efficient. This is down to considering the nature of the phase tree structure. Since there are a lot of common aircraft functions between the phases. These functions are generally only dependent on each other due to power sources and sensor information (avionics). The method developed restructures the tree so sub trees of power sources and sensor information can be pulled out of the functions and stop them preventing other gates from being modules. By doing this makes the functions independent from each other, which enables simplification of the tree.

## **Chapter 7: Updating the Phased Mission Analysis with mission reconfiguration**

### **7.1 Introduction**

While a mission is being performed the mission objective may change as a result of failure diagnosed or rerouting due to emerging threats. This alters the phases in the remainder of the mission. A new analysis needs to be done for what is effectively a new mission to calculate the probability of the mission failure given that the mission has already successfully completed up to the point where the change was made. This is practical in a real mission situation since the mission objective could change throughout the mission, potentially several times. The calculations of the analysis need to be performed in the fastest time possible since it is an online time analysis used to aid the mission planning. Since the analysis is being updated from the old mission, it can be performed by reusing parts of the old mission analysis to analyse the new mission; this can potentially significantly reduce the time of the analysis. This chapter presents and demonstrates an extension to the current phased mission analysis method, shown in chapter 4. The analysis can then be updated when the mission profile changes. The method is tested on some further phases that represent different mission objectives and are explained below.

### **7.2 Reconfiguration**

Throughout the duration of a mission, the UAV must self-adapt to changes happening in the environment, mission objectives and the aircraft operational state. The mission is reconfigured by considering the changing factors and optimizing to produce best chance of the mission being completed successfully. The mission planning process can be split into three stages: mission generation, mission analysis and mission reconfiguration. A diagram of this procedure is shown in figure 7.1 [28]. The procedure starts with generating a mission plan from the information provide on the mission objectives. The mission is analysed to predict the phase and mission reliability. This is updated when the mission progresses from information provided by the fault diagnostic systems onboard and changing weather conditions or emerging threats. The results are used to make a decision to either reconfigure or to continue

the mission. If the mission is to be reconfigured this is carried out to optimize the reliability of the mission by changing the remaining phases of the mission.

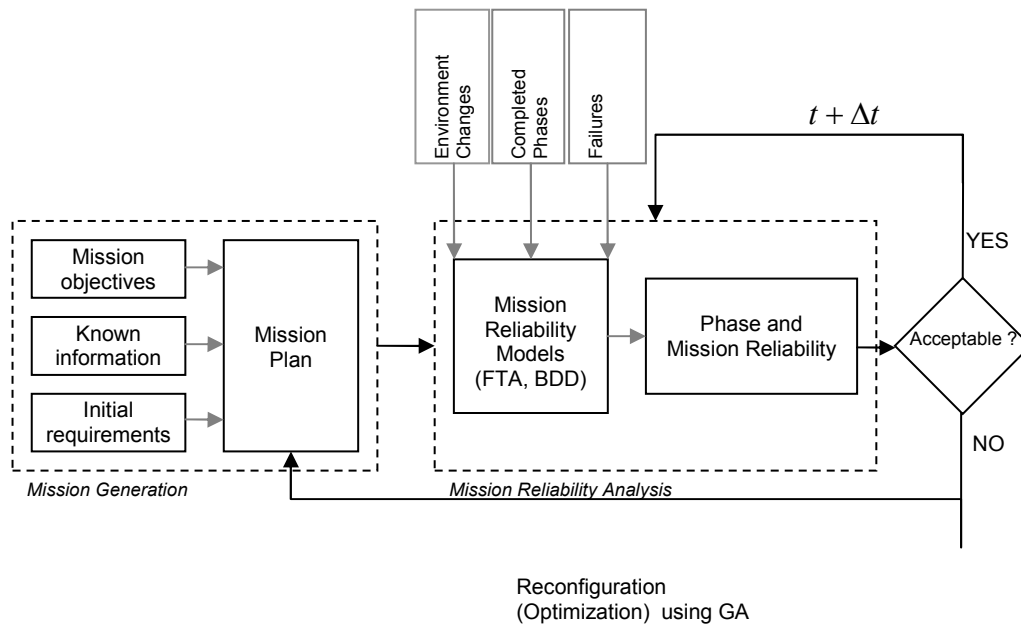


Figure 7.1: The onboard decision making strategy

### 7.3 The phase tasks

Phase tasks are presented in this section which can be used to construct practical UAV phased missions. This phase and mission information was obtained from work done by Samuel Chew [26]. The task phases for the UAV mission are anti-submarine warfare (ASW), anti-surface warfare (ASUW) and search and rescue (SAR). These task phases require a combination of the sub-systems to function as follows:

- Tactical Command System (TCS) – interacts with the sub-system to provide the crew with all the information to compete the mission. It interfaces with the crew through the use of keyboard, mouse, programmable entry panels and multi-function displays.
- Defensive Aids Sub-system (DASS) - this sub-system detects threats to the aircraft, evaluates the size and type of threats and then transmits signals to other subsystems.



- Flight Management System (FMS) - calculates the position and other navigational information for the aircraft.
- Magnetic Anomaly Detector (MAD) – detects fluctuations in the earth’s magnetic field which could indicate the presence of a submarine.
- Stores – the objects that are stored on the aircraft such as weapons.
- Radar/Identification of Friend or Foe (Radar/IFF) – detects the type of items and detected. For example surface vessels may be detected then classified and tracked.
- Electronic Support Measures (ESM) – detects emission of electromagnetic data and obtains information about the emitter such as its position.
- Electro-Optical Surveillance/ Detection (EOSDS) – obtains day and night images while the aircraft is airborne.

The fault trees that represent failure of these phase mission tasks are shown in appendix E.

## 7.4 The method

The mission unreliability calculation process for an updated mission is an extension of the method from chapter 4 section 4.4 (**Method 2**). It is explained by means of an example analysis. Given that there are four possible phases to make up a mission and their fault trees containing three different events are shown in figure 7.2.

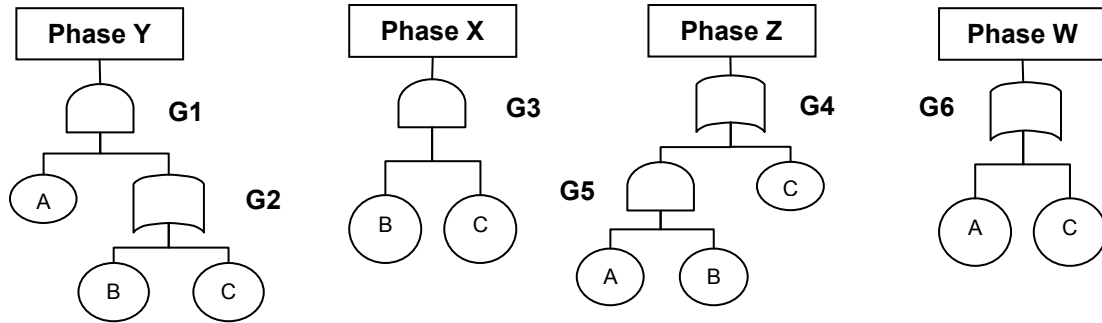


Figure 7.2: Fault trees of phases Y,X,Z and W

#### 7.4.1 Analysis of Original Mission

Suppose the original mission is made up of three phases in the order of Z, Y and X. For analysis the fault trees are first converted to BDDs (variable order  $A < B < C$ ) and the variables converted to phase variables, as shown in figure 7.3.

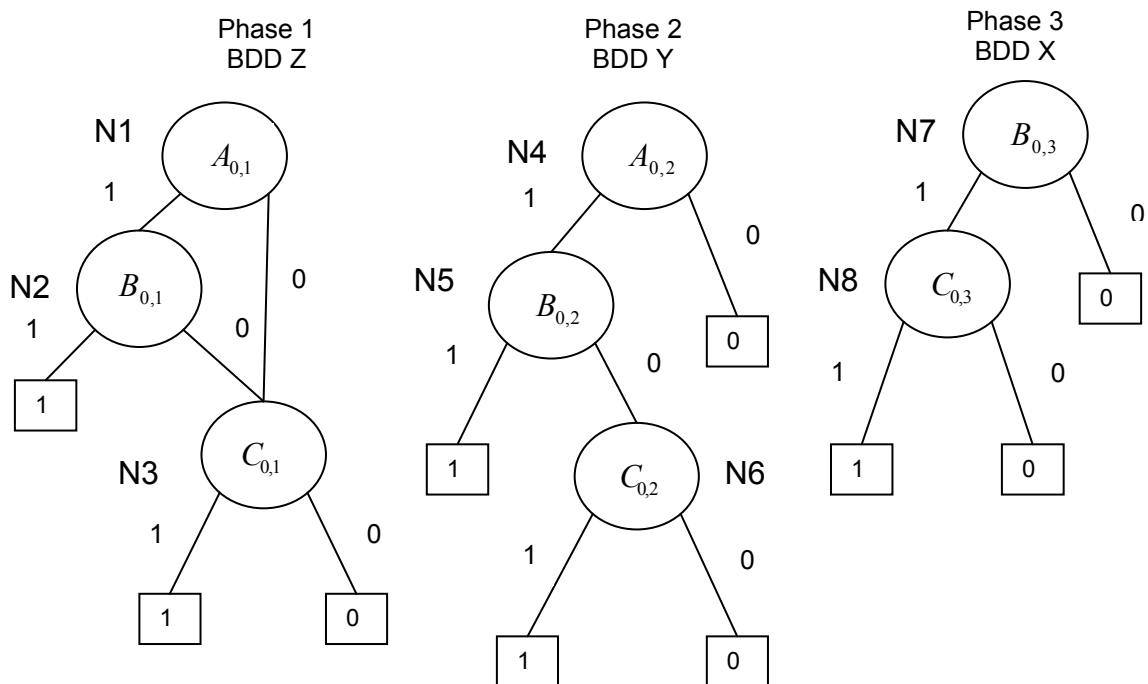


Figure 7.3: Phases BDDs for phases Z, Y and X

Now **Method 2** from chapter 4, section 4.4 is used for calculating the phase unreliability probabilities of the original mission plan, the calculations are shown below:

## For Phase 1

Probability of node 2:

$$\Pr( N_2 = 1 ) = 1 + (1 - \Pr( B_{0,1} = 1 )) \cdot (\Pr( C_{0,1} = 1 ) - 1)$$

Probability of node 1:

$$\Pr( N_1 = 1 ) = \Pr( N_2 = 1 ) + (1 - \Pr( A_{0,1} = 1 )) \cdot (\Pr( C_{0,1} = 1 ) - \Pr( N_2 = 1 ))$$

Probability of failure in phase 1

$$\Pr( P_1 = 1 ) = \Pr( N_1 = 1 )$$

## For Phase 2

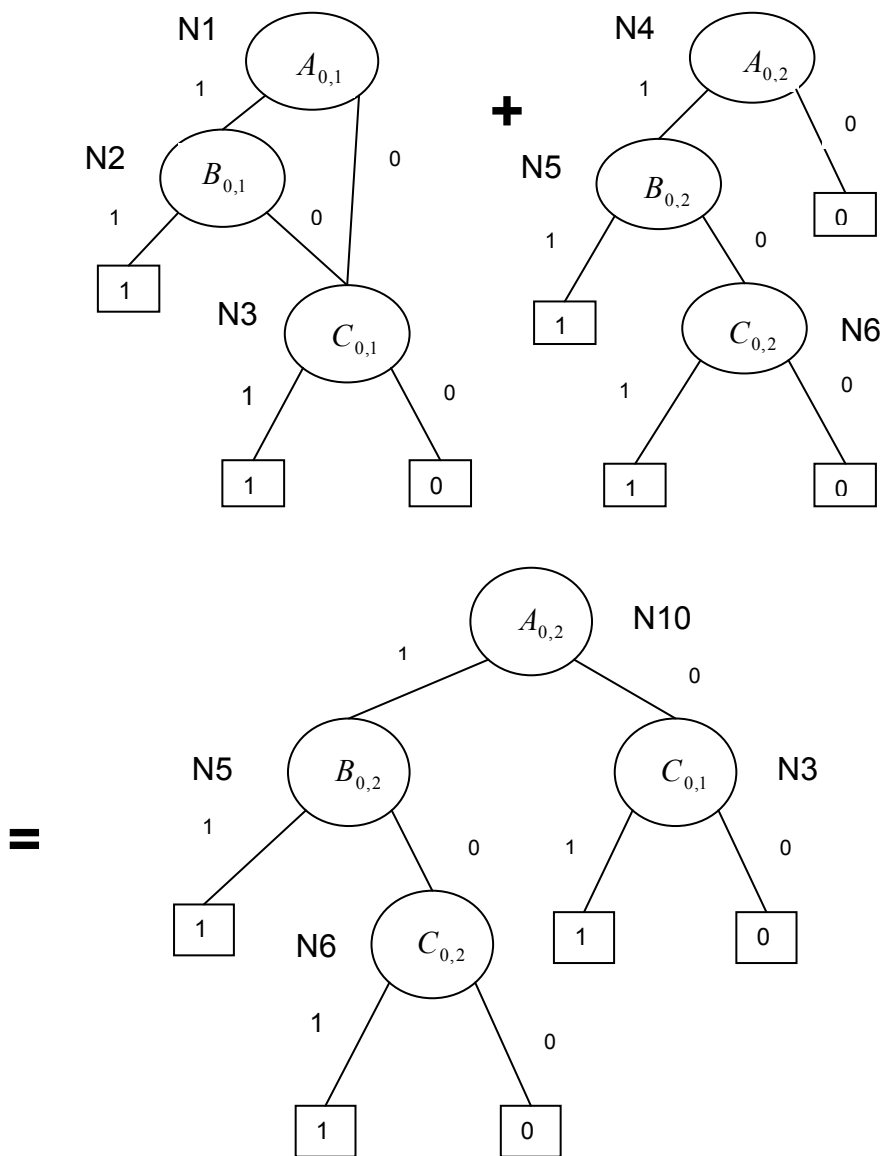


Figure 7.4: Phase 1 OR Phase 2 BDDs

The BDD for failure at the end of phase 2 is obtained as shown in figure 7.4.

Probability of node 5:

$$\Pr( N_5 = 1 ) = 1 + (1 - \Pr( B_{0,2} = 1 )) \cdot (\Pr( C_{0,2} = 1 ) - 1)$$

Probability of node 10:

$$\Pr( N_{10} = 1 ) = \Pr( N_5 = 1 ) + (1 - \Pr( A_{0,2} = 1 )) \cdot (\Pr( C_{0,1} = 1 ) - \Pr( N_5 = 1 ))$$

Probability of failure in phase 1 or Phase 2:

$$\Pr( (P_1 + P_2) = 1 ) = \Pr( N_{10} = 1 )$$

Probability of failure occurs in 2:

$$\Pr( \overline{P_1} P_2 = 1 ) = \Pr( N_{10} = 1 ) - \Pr( N_1 = 1 )$$

### For Phase 3

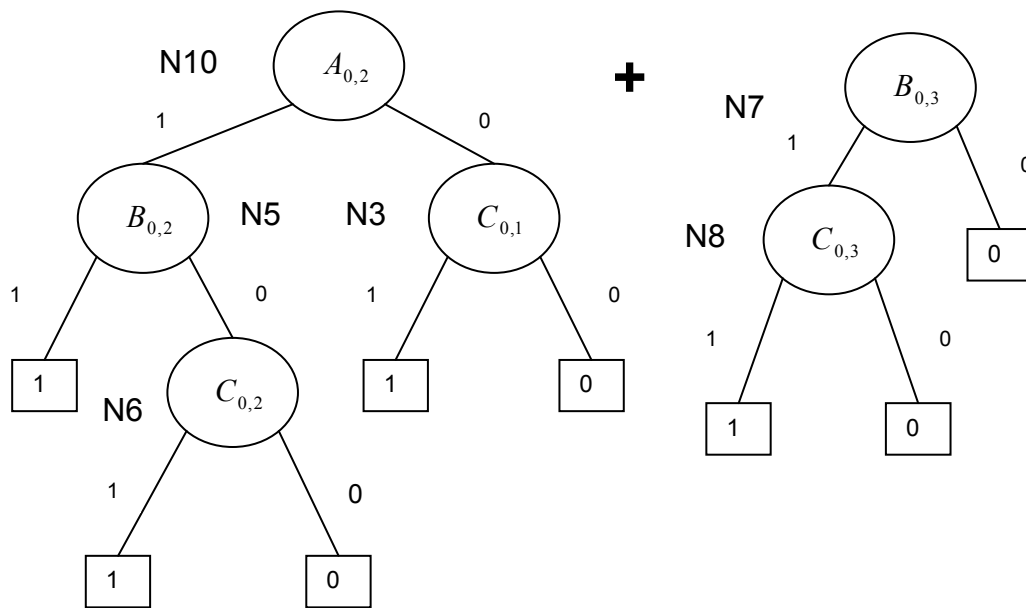


Figure 7.5.a: (Phase 1 OR Phase 2) OR Phase 3 BDDs combined

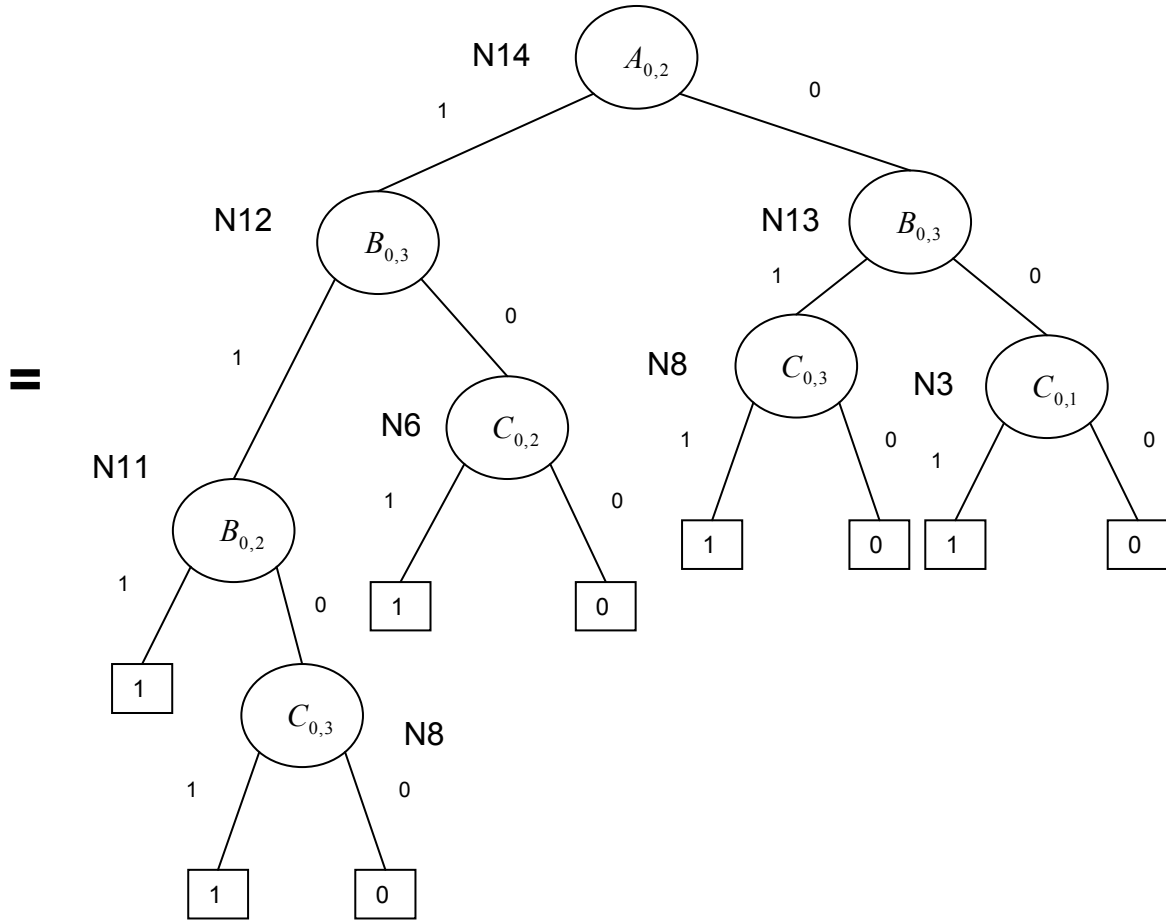


Figure 7.5.b: (Phase 1 OR Phase 2) OR Phase 3 BDDs combined

The BDD representing mission failure at the end of phase 3 is developed as shown in figure 7.5 a and b.

Probability of node 11:

$$\Pr(N_{11} = 1) = 1 + (1 - \Pr(B_{0,3} = 1)) \cdot (\Pr(C_{0,3} = 1) - 1)$$

Probability of node 12:

$$\Pr(N_{12} = 1) = \Pr(N_{11} = 1) + (1 - \Pr(B_{0,3} = 1)) \cdot (\Pr(C_{0,2} = 1) - \Pr(N_{11} = 1))$$

Probability of node 13:

$$\Pr(N_{13} = 1) = \Pr(C_{0,3} = 1) + (1 - \Pr(B_{0,3} = 1)) \cdot (\Pr(C_{0,1} = 1) - \Pr(C_{0,3} = 1))$$

Probability of node 14:

$$\Pr(N_{14} = 1) = \Pr(N_{12} = 1) + (1 - \Pr(A_{0,2} = 1)) \cdot (\Pr(N_{13} = 1) - \Pr(N_{12} = 1))$$

Probability of failure in phase 1 or Phase 2 or Phase 3

$$\Pr((P_1 + P_2 + P_3) = 1) = \Pr(N_{14} = 1)$$

Probability of failure occurs in 3:

$$\Pr(\overline{P_1} \overline{P_2} P_3 = 1) = \Pr(N_{14} = 1) - \Pr(N_{10} = 1)$$

#### 7.4.2 Analysis of Reconfigured Mission

The method analyses the reconfigured mission, given that the original mission was successful up to a particular phase, to produce the failure of future phases. The method reduces its online time of analysis by reusing information obtained from the analysis of the original mission. This can be achieved because the method builds a sequence of phase BDDs by continually combining the BDD established to the current phase with that of the next phase BDD and then quantifying the BDD at each step for the failure probability. Then each probability of failure from mission start to the end of a phase is subtracted from the previous one to obtain the phase failure probabilities. This is demonstrated in the example above. The reconfigured mission will be the same as the original mission to the point where the reconfiguration is instigated. Therefore the new analysis can continue from the point in the sequence where the mission has already been successful instead of starting from the beginning. The phase failure probabilities are substituted in the formula in 7.1 to obtain the conditional probabilities that failure occurred after it was successful up to a certain point in the mission. This equation is from reference [27], where n is the total number of phases in the mission, k is the phase that the reconfigured mission failure occurs and j is the phase that the mission has been successful up to. The previous example is used again to demonstrate the process and is shown below.

$$Q_{k/j} = \frac{Q_k}{1 - \sum_{i=1}^j Q_i} \quad (7.1)$$

$Q_i$  is the probability that mission failure occurs in phase i.

$Q_{k/j}$  is the probability that mission failure occurs in phases k given that the mission has been successful up to phase j.

To consider mission reconfiguration suppose that the mission successfully completes phases 1 and 2 and then something happens so that the phases after phase 2 are replaced with phases W and X. Now phase W is phase 3 and phase X is phase 4. Their fault trees are converted to BDDs and the variables are converted to phase variables as shown in figure 7.6. The method now needs to calculate the probabilities of the mission failure occurring in phases 3 and 4 given that it was successful up to phase 2. Also the phase variable event probabilities are recalculated for phases 3 and 4 since these will have changed due to the reconfiguration from the old mission to the new mission.

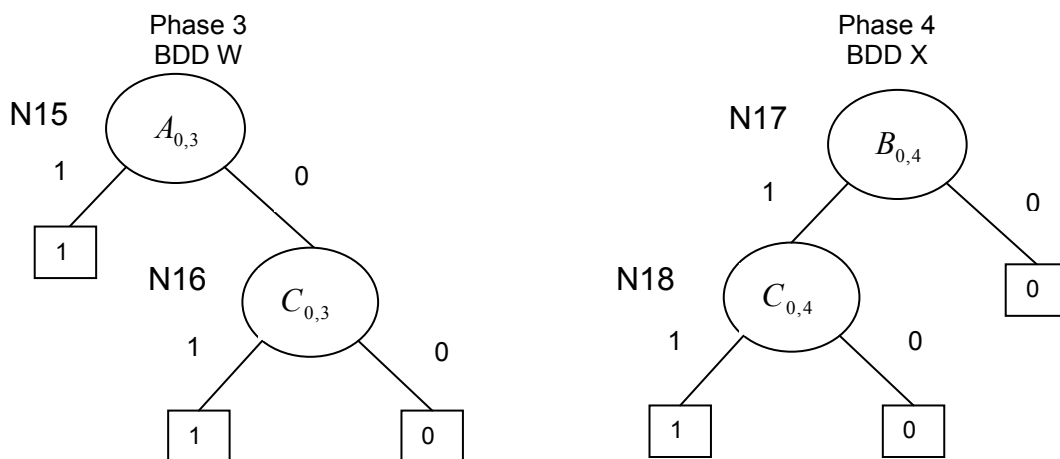


Figure 7.6: Phase 3 and Phase 4 BDDs for the new mission

When creating the BDD that represents failure in phases 1 OR 2 OR 3 for the new mission, the BDD from the previous analysis that represents failure in phases 1 OR 2 shown in figure 7.4 can be reused. The reused BDD is then combined with the BDD that represents the failure causes for phase 3, shown in figure 7.7, by the OR operator. This can be done since the first two phases are the same and the sequence of adding the following phase BDD by an OR operator can be continued. The BDD created for failure in phases 1 OR 2 OR 3 of the new mission is shown in figure 7.7.

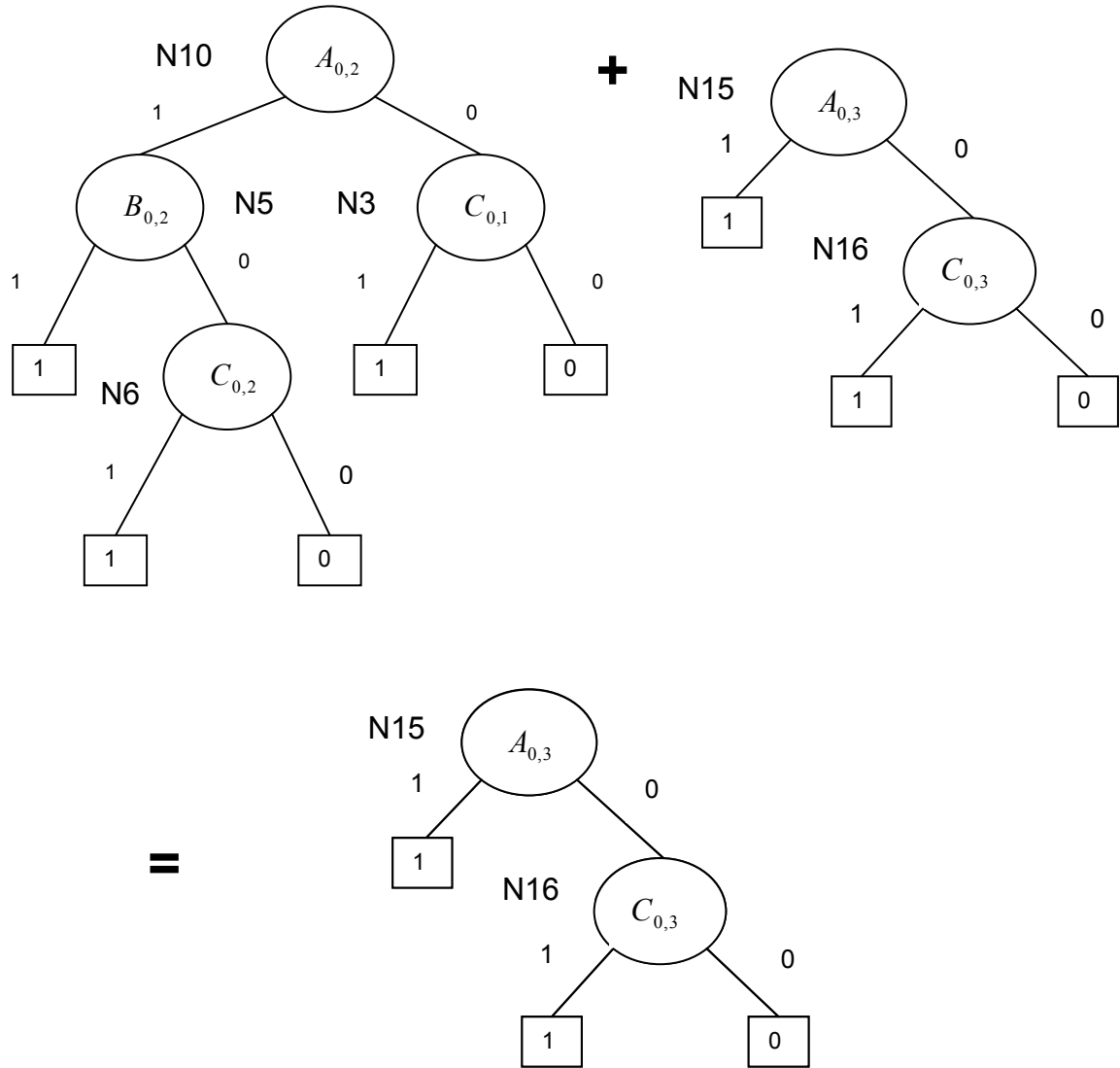


Figure 7.7: (Phase 1 OR Phase 2) OR Phase 3 BDDs combined for new mission

The new BDD for failure in phases 1 OR 2 OR 3 is quantified to establish its failure probability. This probability is subtracted from the probability calculated for the original mission of failure in phases 1 OR 2 to obtain the probability that failure occurs in phase 3. The working for this shown below.

Probability of node 15:

$$\Pr(N_{15} = 1) = 1 + (1 - \Pr(A_{0,3} = 1)) \cdot (\Pr(C_{0,3} = 1) - 1)$$

Probability of failure in phase 1 or Phase 2 or Phase 3

$$\Pr((P_1 + P_2 + P_3) = 1) = \Pr(N_{15} = 1)$$



Probability of failure occurs in 3:

$$\Pr(\overline{P_1} \overline{P_2} P_3 = 1) = \Pr(N_{15} = 1) - \Pr(N_{10} = 1)$$

The analysis continues to develop the BDD for failure in phases 1 OR 2 OR 3 OR 4 of the new mission using the previous BDD that was created and combining this with the BDD for the failure of phase 4.

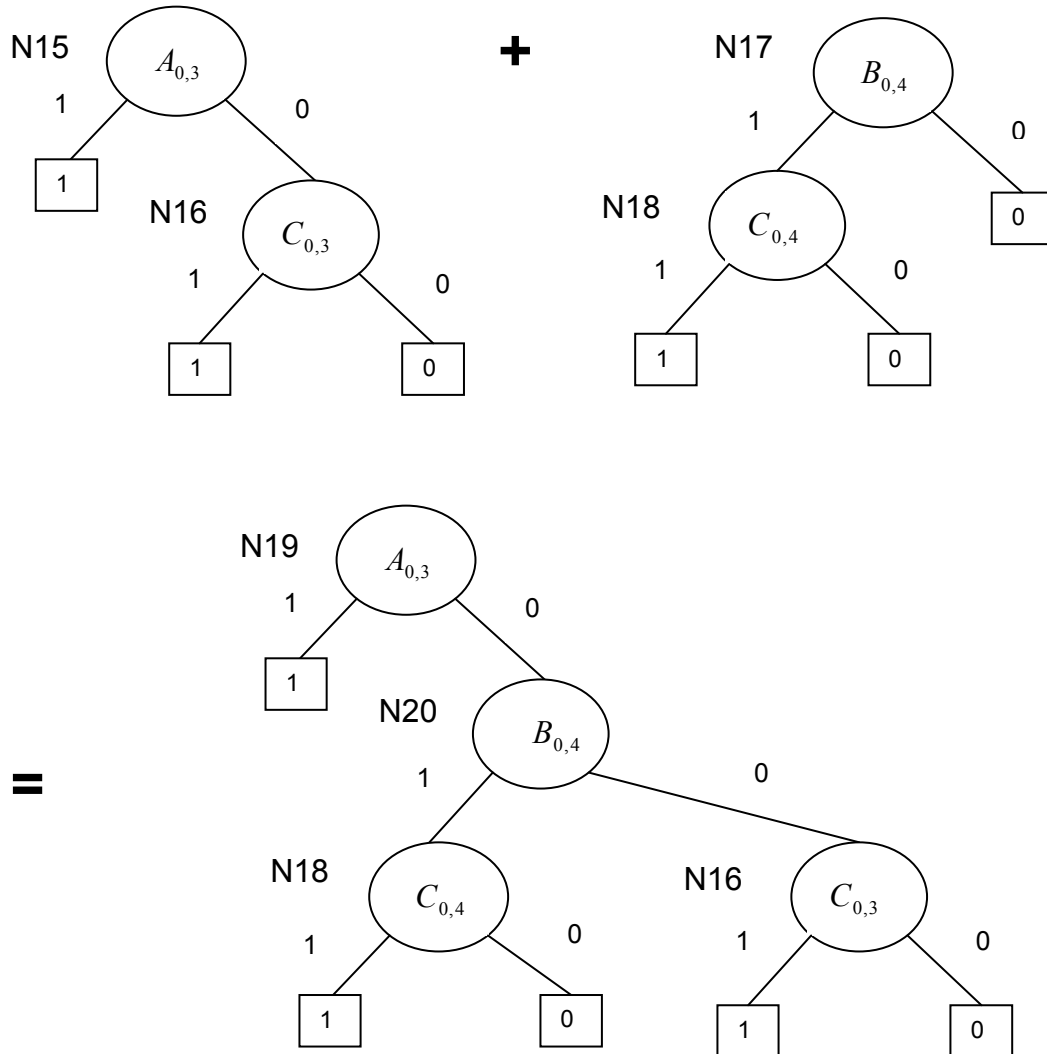


Figure 7.8: (Phase 1 OR Phase 2 OR Phase 3) OR Phase 4 BDDs combined for new mission

Using BDD for failure in phases 1 OR 2 OR 3 OR 4, shown in figure 7.8, the probability of failure in phase 4 is calculated shown below.

Probability of node 20:

$$\Pr( N_{20} = 1 ) = \Pr( C_{0,4} = 1 ) + (1 - \Pr( B_{0,4} = 1 )) \cdot (\Pr( C_{0,3} = 1 ) - \Pr( C_{0,4} = 1 ))$$

Probability of node 19:

$$\Pr( N_{19} = 1 ) = 1 + (1 - \Pr( A_{0,3} = 1 )) \cdot (\Pr( N_{20} = 1 ) - 1)$$

Probability of failure in phase 1 or Phase 2 or Phase 3 or Phase 4

$$\Pr(( P_1 + P_2 + P_3 + P_4 ) = 1) = \Pr( N_{19} = 1)$$

Probability of failure occurs in 3:

$$\Pr(\overline{P}_1 \overline{P}_2 \overline{P}_3 P_4 = 1) = \Pr( N_{19} = 1 ) - \Pr( N_{15} = 1)$$

To calculate the probabilities of phase failure in the new phases of the mission conditional on success up to phase 2, the unconditional probabilities calculated above are substituted into equation 7.1.

## 7.5 Results and discussion

The phases that are selected to test this method are listed in table 7.1. The phases in rows 1 to 5 and 11 to 14 are general UAV mission phases that were used in chapter 6 and their fault trees are shown in appendix B under mission set 8. The phases in rows 6- 10 are specific mission task phases that were discussed earlier and their fault trees are shown in appendix E. Prior to the analysis the fault trees are re-structured to take out modules applying the method from chapter 6. The analysis of the sequence of missions is performed both with and without the updating approach and the online running times are recorded for comparison.

Number	Phases name
1	Start up
2	Taxi out
3	Take off
4	Climb
5	Cruise
6	ASW
7	ASW_ATT
8	ASUW
9	ASUW_ATT
10	SAR
11	Decent
12	Land
13	Taxi in
14	Shutdown

Table 7.1: The phase name

The results are shown in table 7.2. The first column is the number of times the mission has had an updated analysis performed on it. The second column is the number of phases that are successful up to the point where the mission is altered. The third column is the mission configurations in term of the phase index numbers relating to phases in table 7.1. The phase index numbers shown in bold indicates that the phases have already been completed successfully. The forth column shows the online times of the updated analysis being performed and fifth column shows the online times without the updated analysis (ie just **method 2** after restructuring and the modules have been taken out.)

Number of updating	Phase mission successful up to	Mission	Online time with updating method	Online time without updating method
1	0	1,2,3,4,5,11,12,13,14	10.68s	9.26s
2	5	<b>1,2,3,4,5</b> ,6,7,5,11,12,13,14	10.00s	9.92s
3	8	<b>1,2,3,4,5,6,7,5</b> ,8,9,5,11,12,13,14	13.15s	14.44s
4	11	<b>1,2,3,4,5,6,7,5,8,9,5,5</b> ,10,5,10,11,12,13,14	20.83s	22.74s
5	15	<b>1,2,3,4,5,6,7,5,8,9,5,5,10,5,10</b> ,6,7,8,8,9,10,11,12,13,14	17.32s	31.79s
6	20	<b>1,2,3,4,5,6,7,5,8,9,5,5,10,5,10,6,7</b> ,8,8,9,5,8,9,5,11,12,13,14	31.37s	47.81s
7	24	<b>1,2,3,4,5,6,7,5,8,9,5,5,10,5,10,6,7</b> ,8,8,9,5,8,9,5,5,6,8,10,5,11,12,13,14	46.55s	1 min 7.54s
8	29	<b>1,2,3,4,5,6,7,5,8,9,5,5,10,5,10,6,7</b> ,8,8,9,5,8,9,5,5,6,8,10,5,10,5,10,5,6,7,5,11,12,13,14	1min 9.75s	1 min 34.49s
9	37	<b>1,2,3,4,5,6,7,5,8,9,5,5,10,5,10,6,7</b> ,8,8,9,5,8,9,5,5,6,8,10,5,10,5,10,5,6,7,5,5,6,7,8,9,10,5,11,12,13,14	1 min 10.05s	2 min 10.19s
10	42	<b>1,2,3,4,5,6,7,5,8,9,5,5,10,5,10,6,7</b> ,8,8,9,5,8,9,5,5,6,8,10,5,10,5,10,5,6,7,5,5,6,7,8,9,10,5,5,10,5,10,5,6,10,11,12,13,14	1min 18.68s	2min 51.06s
11	49	<b>1,2,3,4,5,6,7,5,8,9,5,5,10,5,10,6,7</b> ,8,8,9,5,8,9,5,5,6,8,10,5,10,5,10,5,6,7,5,5,6,7,8,9,10,5,5,10,5,10,5,6,10,5,6,7,10,11,12,13,14	1min 30.19s	2min 52.72s

Table 7.2: The Online times of the analysis with and without the updating method

The online analysis times for the eleven missions analysed shows that for the first two missions the updated approach is not faster than without, but only less by approximately 10%. As the mission sequence progresses the improvement of the analysis time of the updated approach compared to without it increases. On the last three missions analysed the updated approach is approximately twice as fast. The

reason for these results is that as the mission gets larger there are more computations to reuse which reduces the analysis times. For example the last mission contains 57 phases and the first 49 have been completed successfully. Without the updated method, therefore not using any analysis done before, the analysis would combine the entire 57 phase BDDs together in a sequence. However, when using the updated approach the BDD for the first 49 phases has already been developed from the previous analysis, therefore it can be reused and only the BDDs for the last 8 phases in the sequences have to be newly combined. The approach successfully reduces the number of computations for this practical situation of continually updating the mission analysis. The method will be more successful in reducing the online analysis time for missions with multiple reconfigurations throughout and where the majority of the mission has already been successful. This is practical as real missions can have any reconfigurations throughout its duration. Conversely if a mission has one reconfiguration after its first phase then it will not be as effective.

## **Chapter 8: Parallelization of a PMS Analysis**

### **8.1 Introduction**

Large calculation procedures can be broken down into their sub-calculations. These sub-calculations may depend on each other and therefore have to be performed in a particular ordered sequence. Conversely if the sub-calculations are independent they can be performed simultaneously. The advantage of breaking down the PMS analysis calculation procedure into independent sub-calculations for a real time analysis is that each sub-calculation can be performed on a separate computer which will reduce the online time for obtaining the results. This chapter explains, demonstrates, and tests a method which splits the PMS analysis into independent calculations to be performed in parallel.

The method identifies parts of the set of fault trees from the PMS that can be analysed independently from the rest of the fault trees. These independent parts are referred to as groups. Each group can be treated as a smaller PMS therefore they can be analysed in the same way. The motivation for identifying these groups is that they can be analysed simultaneously, reducing the overall calculation time.

### **8.2 The method**

Assume that a mission of  $n$  phases for which the failure cause for each phase is represented by a fault tree is to be analysed. The steps below identify if the fault trees of the phased mission can be analysed in parallel and explains the procedure.

- 1) All of the top gates of the  $n$  fault trees must be OR types for the procedure to continue. This is restrictive since if at least one top gate is an AND type the method cannot be performed, however the majority of fault trees that represented aircraft mission phases will have an OR type top gate.
- 2) All of the input gates to the top gates are identified and referred to as level one gates.

- 3) For every level one gate all of the repeated gates and events that appear at any level beneath it are identified and recorded.
- 4) For every level one gate all of its repeated gates and events that appear beneath it are traced up from every appearance in the fault trees to locate their gate of intersection, or to another level one gate, which will be recorded. These recorded level one gates that are reached will be logically dependent with the level one gate on which the trace procedure was initiated which means that the two gates have events and gates in common.
- 5) A group of level one gates is formed containing all the level one gates in the group that are logically dependent. This entire group is logically independent of the rest of the level one gates that are not in the group and the rest of the fault trees.
- 6) A group is not further considered if there is any pair of level one gates in the group which belong to the same phase.
- 7) All of the level one gates in a group must be a module of the fault tree to which it belongs. If not then the entire group cannot be considered further.
- 8) Now in the groups that are left, all of the level one gates that they contain are removed from the fault trees. Each group is treated as an individual phased mission. These together with the original phase mission with the groups elements removed are analysed to obtain their probabilities of phase failure.
- 9) The final analysis is performed by combining the probabilities from the calculations for each group and the ordinary phase mission with the group elements removed by the formula given below.

$g_{i,j}$  is the Boolean logic of group i in phase j.

$k_i$  is the Boolean logic of the failure causes of phase i after the group elements have been removed.

The probabilities of group i failure up to the end of each mission phase in turn are calculated in step 8 of the above procedure and shown below:

$$\Pr(g_{i,1})$$

$$\Pr(g_{i,1} + g_{i,2})$$

.

.

$$\Pr(g_{i,1} + g_{i,2} + g_{i,3} + \dots + g_{i,n})$$

$$i = 1, \dots, n$$

The probabilities of failure at the end of each of the  $k_i$  phase after the group elements have been removed from all the phase mission fault trees are also calculated in step 8 of the procedure defined.

$$\Pr(k_1)$$

$$\Pr(k_1 + k_2)$$

.

.

$$\Pr(k_1 + k_2 + \dots + k_n)$$

Now the phases failure probabilities can be expressed in terms of the probabilities generated in step 8 above.



The probability of phase 1 failure can be broken down to all the level one gates in phase 1 and the rest of the logic that is not part of any group.

$$\Pr( p_1 ) = \Pr( g_{1,1} + g_{2,1} + g_{3,1} + \dots + g_{m,1} + k_1 )$$

Applying the inclusion-exclusion expansion equation (2.26), gives:

$$\begin{aligned} & \Pr( g_{1,1} ) + \Pr( g_{2,1} ) + \Pr( g_{3,1} ) + \dots + \Pr( g_{m,1} ) + \Pr( k_1 ) \\ & - \Pr( g_{1,1} \cdot g_{2,1} ) - \Pr( g_{2,1} \cdot g_{3,1} ) - \dots \\ & - (-1)^{m+1} \Pr( g_{1,1} \cdot g_{2,1} \cdot g_{3,1} \cdot \dots \cdot g_{n,1} \cdot k_1 ) \end{aligned}$$

Since the groups are independent, the probability of two logic events occurring together is just the product of their probabilities. This gives the following:

$$\begin{aligned} & = \Pr( g_{1,1} ) + \Pr( g_{2,1} ) + \Pr( g_{3,1} ) + \dots + \Pr( g_{m,1} ) + \Pr( k_1 ) \\ & - \Pr( g_{1,1} ) \cdot \Pr( g_{2,1} ) - \Pr( g_{2,1} ) \cdot \Pr( g_{3,1} ) - \dots \\ & \dots (-1)^{m+1} \Pr( g_{1,1} ) \cdot \Pr( g_{2,1} ) \dots \Pr( g_{m,1} ) \cdot \Pr( k_1 ) \end{aligned} \quad (8.1)$$

All the terms in the expression have already been calculated and therefore just need to be substituted in. The same form of expression is produced for the mission failure at the end of each phase. The working is shown below. The failure probability at the end of phase 2 is.

$$\begin{aligned} & \Pr( p_1 + p_2 ) \\ & = \Pr( g_{1,1} + g_{2,1} + \dots + g_{m,1} + k_1 + g_{1,2} + g_{2,2} + \dots + g_{m,2} + k_2 ) \end{aligned}$$

Re-arranging the terms together with the groups they belong to gives:

$$= \Pr((g_{1,1} + g_{1,2}) + (g_{2,1} + g_{2,2}) + \dots + (g_{m,1} + g_{m,2}) + (k_1 + k_2))$$

Applying the inclusion-exclusion expansion again gives:

$$\begin{aligned} &= \Pr(g_{1,1} + g_{1,2}) + \Pr(g_{2,1} + g_{2,2}) + \dots + \Pr(g_{m,1} + g_{m,2}) + \\ &\Pr(k_1 + k_2) - \Pr((g_{1,1} + g_{1,2}) \cdot (g_{2,1} + g_{2,2})) - \Pr((g_{1,1} + g_{1,2}) \cdot (g_{3,1} + g_{3,2})) \dots \\ &\dots (-1)^{m+1} \Pr((g_{1,1} + g_{1,2}) \cdot (g_{2,1} + g_{2,2}) \dots (g_{m,1} + g_{m,2}) \cdot (k_1 + k_2)) \\ &= \Pr(g_{1,1} + g_{1,2}) + \Pr(g_{2,1} + g_{2,2}) + \dots + \Pr(g_{m,1} + g_{m,2}) + \Pr(k_1 + k_2) \\ &- \Pr(g_{1,1} + g_{1,2}) \cdot \Pr(g_{2,1} + g_{2,2}) - \Pr(g_{1,1} + g_{1,2}) \cdot \Pr(g_{3,1} + g_{3,2}) \dots \\ &(-1)^{m+1} \Pr(g_{1,1} + g_{1,2}) \cdot \Pr(g_{2,1} + g_{2,2}) \dots \Pr(g_{m,1} + g_{m,2}) \cdot \Pr(k_1 + k_2) \quad (8.2) \end{aligned}$$

This process can be repeated for each of the mission phases.

### 8.3 Example

The method is demonstrated by working through an example PMS analysis. The fault trees that represent the failure causes for the phases of the mission are shown in figure 8.1. The working is as follows:

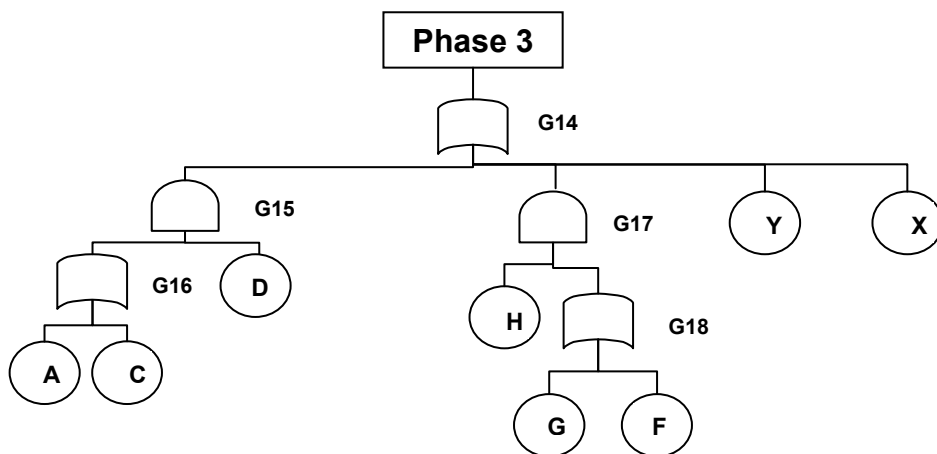
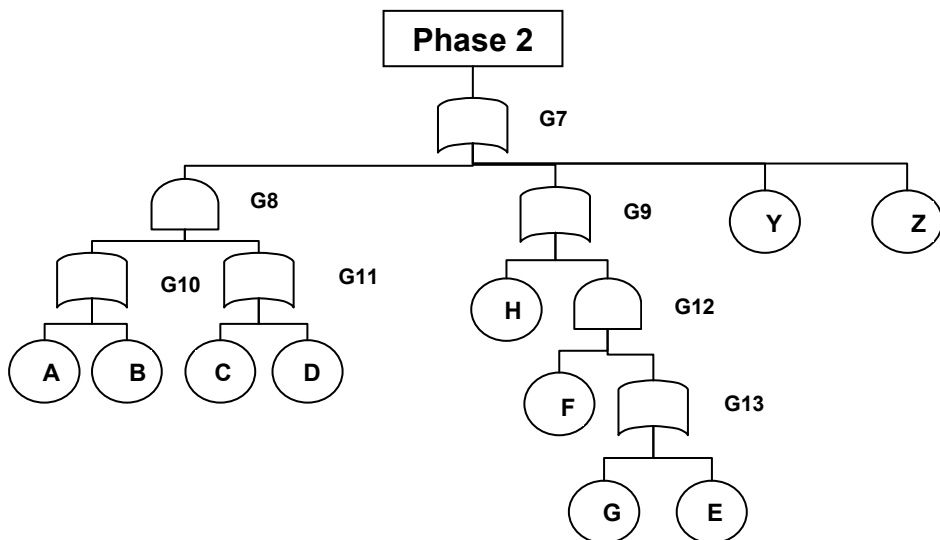
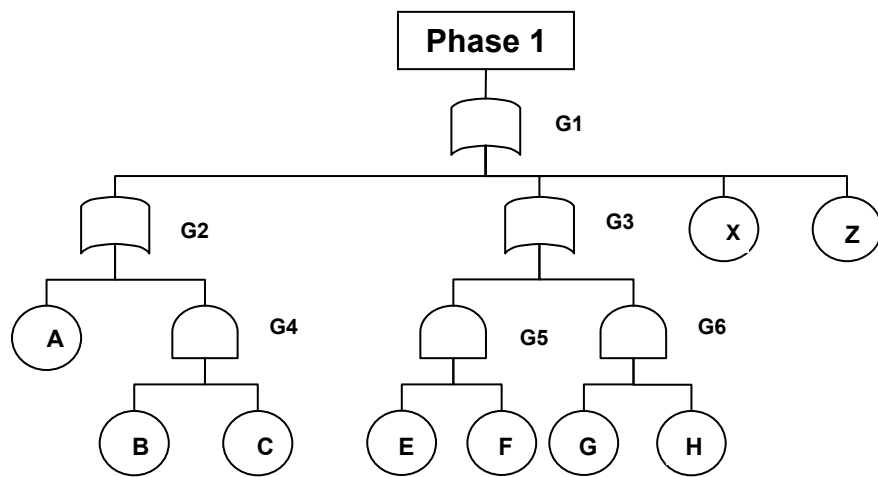


Figure 8.1: Fault trees for the three Phase mission example.

- 1) All the top gates of the fault trees are OR gates therefore the procedure can continue.
- 2) All the input gates to the top gates of the fault trees are recorded and referred to as level one gates and they are shown in the first row of table 8.1.
- 3) Every level one gate has all of its branches scanned and all of the repeated gates and events that are encountered are listed. This information is critical for identifying which level one gates are dependent or independent of each other. For example gate G2 has repeated events A, B, C listed and no repeated gates beneath it. The information for all of the level one gates is shown in the second row of table 8.1.
- 4) This step works out which level one gates are dependent or independent of each other. The level one gates are considered one at a time. For each level one gate all of the repeated gates and events that appear beneath it are examined one by one. Each repeated gate or event has all of its appearances in the mission identified and these are traced up through the tree structure until the top points of intersection are reached. This could be another level one gate then in this case it is recorded as dependent. For example consider the level one gate G2 with repeated events that appear beneath it A, B and C. The appearances of event A are located first. Event A appears three times as inputs to gates G2, G10 and G16 and the top points of intersection are the top gates G1, G7 and G14 since the paths traced do not intersection each other before these gates. When the paths are traced up from gates G10 and G16 they reach the level one gates G8 and G15. These are therefore in the same dependence group as gate G2 which is recorded. The tracing up procedure is also performed for the appearance of events B and C and they obtain the same result of reaching gates G10 and G16. Now the other level one gates are considered and the results of their dependents are shown in the third row of table 8.
- 5) Now it is identified which level one gates are dependent, the groups are formed to create independent mission elements. A group is defined by a set

of level one gates which are dependent but independent of the rest of the level one gates that are not in the set. These are obtained by going through a procedure using the information shown in table 8.1. Starting with the first level one gate G2, assign it to group 1 and assign its level one gates, that it is dependent on (G8 and G15), to group 1 as well. Next all other level one gates that are dependent on gates G8 and G15 are also assigned to group 1. All of the possible level one gates that can go into group 1 are listed and the procedure will keep on repeating. So group 1 now contains the gates G2, G8 and G15. The procedure moves on to the next level one gate that does not belong to a group yet. Gate G3 is considered and the same procedure is performed to form group 2 which consists of the level one gates G3, G9 and G17. Now all the level one gates belong to a group so the procedure stops.

- 6) Groups 1 and 2 are then checked to establish if they contain only one level one gate belonging to any phase. Group 1 has gate G2 belonging to phase 1, gate G8 from phase 2, and gate G15 from phase 3. Group 2 has exactly one gate belonging to each of phases 2 and 3. Therefore the required condition for step 6 is satisfied and the method moves on to step 7. This is necessary because if there is more than one level one phase gate appearing in the same group then they cannot be extracted to form an independent mission.
- 7) Each level one gate for groups 1 and 2 must be a module of the particular phase it belongs to (independent from the rest of the phase fault tree it appears in). Gates G2 and G3 are modules in the phase 1 fault tree, also gates G8 and G9 are modules for phase 2, and gates G15 and G17 for phase 3. This is necessary because if the level one gate is not a module of the phase then the group that it belongs to cannot be extracted to form an independent mission
- 8) All the groups are extracted from the phase fault trees shown in figure 8.1 and each group forms its own phased mission, this shown in figure 8.2.

Level one gate	G2	G3	G8	G9	G15	G17
Repeated gates and events	A B C	E F G H	A B C D	H F G E	A C D	H G F
Dependent level one gates	G8 G15	G9 G17	G2 G15	G3 G17	G2 G8	G3 G9
The group belonging to	Group 1	Group 2	Group 1	Group 2	Group 1	Group 2
The phase belonging to	Phase 1	Phase 1	Phase 2	Phase 2	Phase 3	Phase 3

Table 8.1: information of the level one gates

- 9) All of the phased missions, the groups: and the phase fault trees after the groups are extracted, are analysed by the standard method explained in chapter 4 to yield the probability of failure in every 'phase' or any of the pervious phases. These results are substituted in the equations (8.1) and (8.2) to obtain the probability of failure for the entire phased mission.

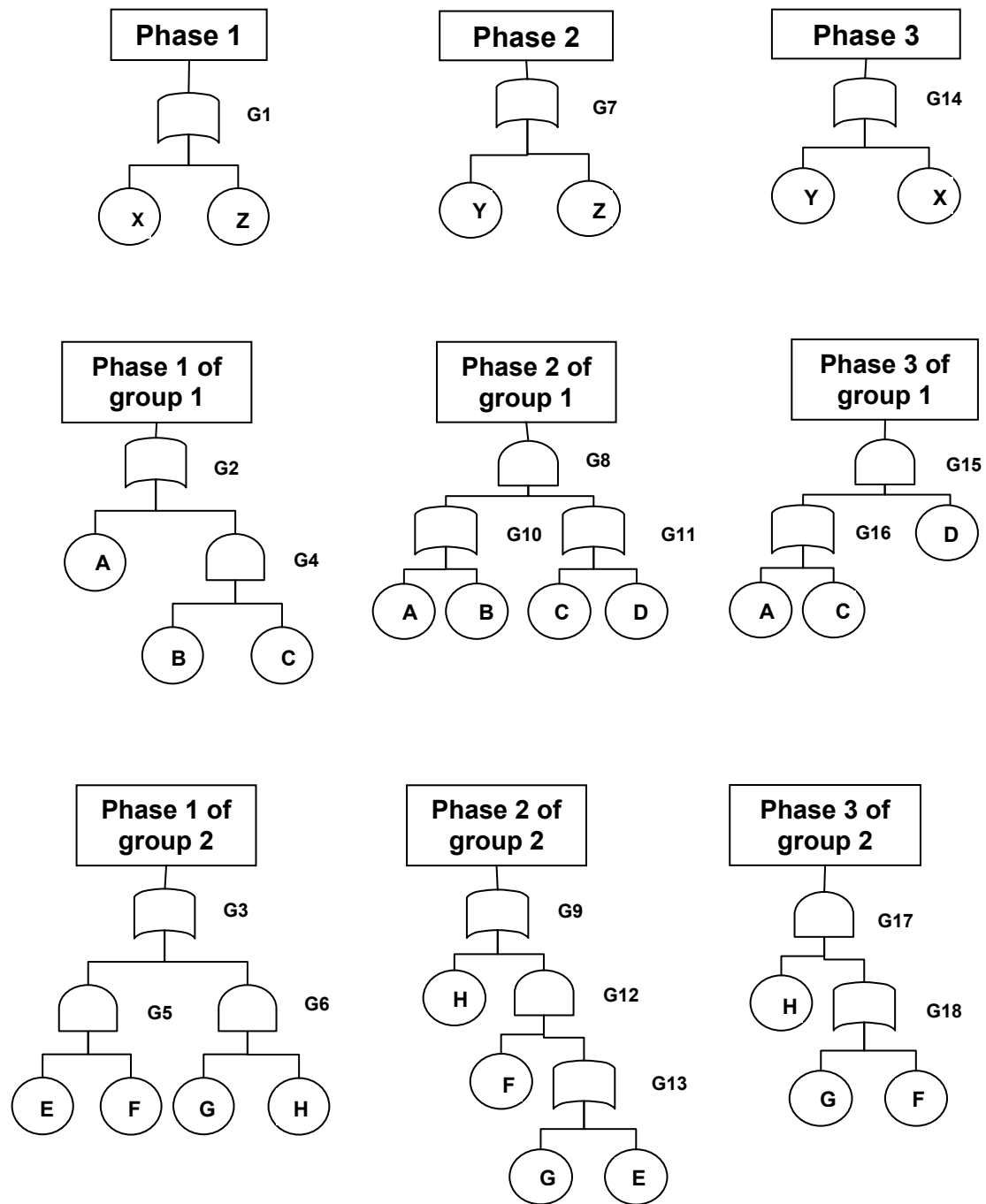


Figure 8.2: The groups of the fault trees for Phase mission

## 8.4 Fault trees structure of practical UAV mission

An example of a UAV mission where cause of failure is represented in fault trees shown in figure 8.3. The phases in the mission require specified sub-systems to function. A sub-system is likely to be required in more than one phase such as the hydraulic system and the fuel system. In the example the hydraulic and fuel systems cannot be taken out as modules because the failure criteria are different in each phase, as shown by its index. However the parallelization approach discussed can take out the hydraulic and fuel system as independent groups that can be treated and analysed by the same phased mission methods.

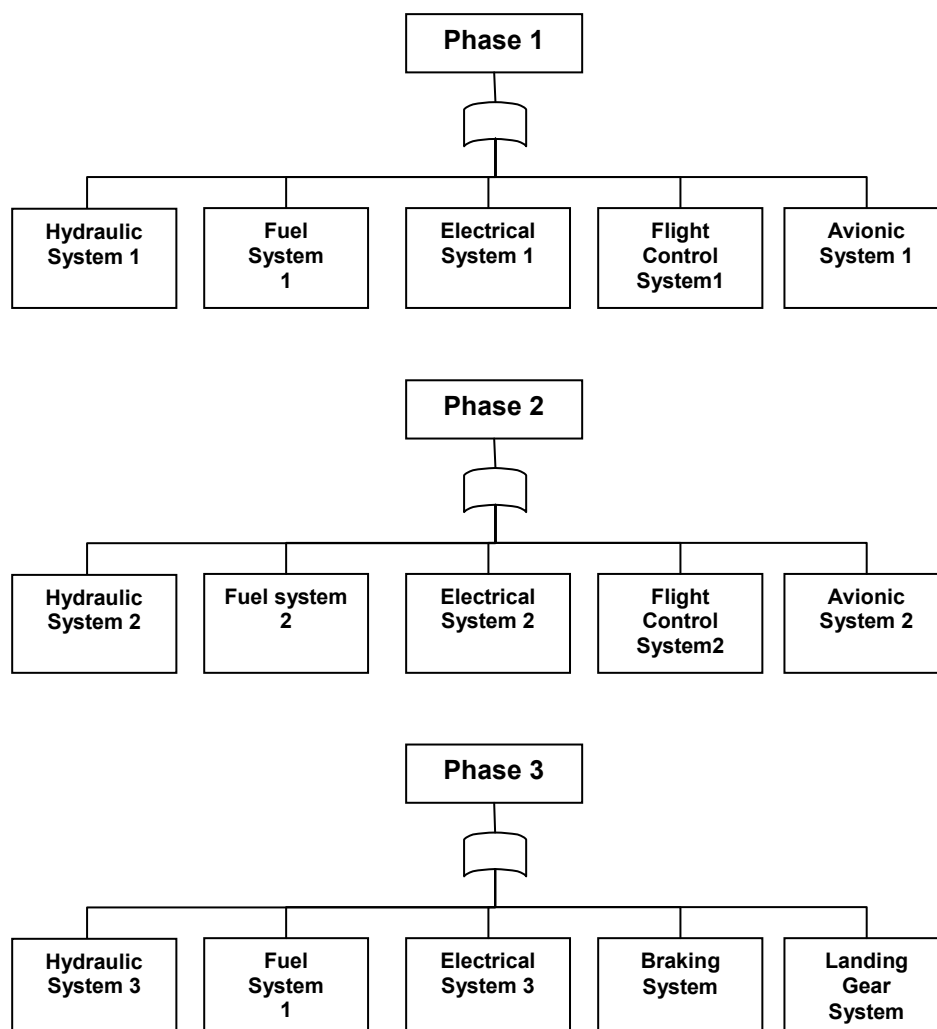


Figure 8.3: Fault trees structure of practical UAV mission



## 8.5 Results and Testing

The parallelization method is designed to reduce the online time required to perform a UAV phased mission analysis. For example consider if a common aircraft function is required in several phases, the part of the fault tree developing the causes of its failure can be taken out as a module of the fault tree for the whole mission, (as carried out in the methods chapters 4 and 6). Now consider a further layer of complexity if the sub-systems failure criteria may differ in the different phases. This case is practical for a real aircraft mission since a sub-system may not perform the same task in all of its phase. For example the fuel system may have different failure criteria in the take off phase compared to the cruise phase.

To test the method the UAV mission phases considered in chapter 7 and shown in appendix E section 1 (listed and numbered in table 8.2) are modified by changing some of the sub-systems for the mission task phases ASW, ASW\_ATT, ASUW, ASUW and SAR so they have different failure criteria in these phases. These changes are shown in appendix E section 2. The parallelization method is applied to the Phased Mission Analysis method from chapter 6 which restructures the fault trees to optimize the number of modules that can be taken out and is the best method produced for a UAV PMS analysis. Parallelization is applied to the method once the fault trees have been restructured and modules have been taken out, and after the method continues to quantify the phase failure probabilities. The parallelization procedure along with the restructuring of the fault trees and module extraction can be done off line.

Number	Phases name
1	Start up
2	Taxi out
3	Take off
4	Climb
5	Cruise
6	ASW
7	ASW_ATT
8	ASUW
9	ASU_ATT
10	SAR
11	Decent
12	Land
13	Taxi in
14	Shutdown

Table 8.2: The UAV mission phases

Table 8.3 contains the results. The second column lists the mission configuration in terms of the phase numbers that are shown in table 8.2. The third column lists the online times for the method when parallelization is applied, there is a list of times which are the online calculation times of the individual parallelization tasks that the method was separated into. The maximum of these times is the one contained in brackets. This time is important since the maximum time is the overall online time that these tasks can be carried out in, because all of these calculations are performed simultaneously. The fourth column lists the online times without the parallelization been applied.

Mission Number	Mission Configuration	Parallelization times	Method time
1	1,2,3,4,5, 6, 11,12,13,14	0.19s,0.20s,0.44s,0.09s,7.94s(7.94s)	1 min 26.3s
2	1,2,3,4,5, 8, 11,12,13,14	0.19s,0.19s,0.45s,0.09s,8.39s(8.39s)	1 min 26.06s
3	1,2,3,4,5, 10 ,11,12,13,14	0.19s,0.17s,0.11s,0.08s,7.44s(7.44s)	25.25s
4	1,2,3,4,5 ,6,7, 11,12,13,14	0.29s,0.28s,2.69s,0.13s,10.09s(10.09s)	1 min 27.01s
5	1,2,3,4,5 ,8,9 ,11,12,13,14	0.29s,0.19s,0.09s,0.09s,8.55s(8.55s)	1 min 25.5s
6	1,2,3,4,5, 6,7,10, 11,12,13,14	0.38s,0.41s,3.13s,0.11s,12.8s(12.8s)	16 min 6.9s
7	1,2,3,4,5,6,7,8,9,10,11,12,13,14	0.79s,0.73s,4.05s,0.25s,16.77s (16.77s)	20min 3.1s

Table 8.3: Running times of Parallelization and Method

The results in table 8.3 shows a significant improvement of the online time of the analysis when performed using the parallelization method. The online times for the parallelization calculations range from 7 to 17 seconds, while without parallel calculation ranges from 25 seconds to 20 minutes. Missions 1 - 5 take approximately 1 minute without parallelization and approximately 10 seconds with parallelization. These missions contain 10 to 11 phases, however only 1 or 2 mission phases numbered 6 to 10, phases 6 to 10 are these whose characteristics are most suited to parallelization. So increasing the number of times these phases are contained in the mission provides the biggest improvements due to the parallelization approach. For mission 7, the online times with parallelization takes just over 10 seconds and without over 15 minutes which is a dramatic difference.

One of the reasons for the success of the parallelization approach is that the calculation is divided over several computers. However this is not the main reason for the dramatic improvement since the sum of the parallelization analysis times do not approximately add up to the online calculation time without the parallelization approach. The main reason for the significant improvement is the reduction in the complexity of the analysis. For example with the analysis performed without using parallelization, all of the BDDs are built together and contain all of the different

groups and the rest of the mission left over when parallelized. Building the BDDs together dramatically increases the complexity, since every time a new element node is computed on the BDD, the procedure has to go through the BDD until it reaches the position where it has a greater order value than the nodes beneath it. Also the number of the nodes recorded increases which is a big issue for efficiency since every time one computation of a node is performed the function scans through the entire list of nodes. As investigated in chapter 5 when a parallelization approach is chosen the BDDs of the different groups are built separately and stored in different lists.

## **8.6 Summary**

The parallelization is successful in reducing the complexity of the PMS analysis and negligible time is invested for the performance of the method. However a condition for the parallelization to obtain its full benefits in a practical application is to have several computers available however when the method was tested it was on the same computer with the individual calculation times measured. The main factor in how successful the parallelization is in minimizing the online time is due to the group sections taken out of the fault trees this reducing the complexity of the analysis. The method requires some conditions and characteristics of the fault trees, which at first might seem quite restrictive. However these conditions and characteristics are very common throughout real UAV mission phase fault trees. For example the majority of mission phase fault trees have a OR top gate. Also independent parts (groups) occurring throughout the set of phase fault trees will appear in practical, complex, sub-systems which are required throughout a mission. Therefore this method of parallelization is very relevant and will reduce the time of online analysis of real UAV phased mission.

## Chapter 9: Conclusions and Future Work

### 9.1 Summary of Work

Autonomous systems such as UAVs perform what are known as phased missions. By calculation of the probabilities that failure will occur during the phases and over the entire mission it is possible to use this information to make decisions on the future of the mission. These probabilities will need to be updated as the mission progresses and phases are successfully completed and also when changes in the situation, such as the weather conditions or component failure, occurs. To contribute to the decision making process the analysis must be performed online in the fastest possible time. However previous research has shown that phased mission analysis can be very computationally intensive.

The aim of this project is to investigate the efficiency of existing PMS analysis methods and to develop new methods that will be faster and therefore able to be applied to online analysis. In order to achieve a fast analysis, advantage can be taken of the characteristics of a UAV mission plan and not aimed at coping with generic failure logic forms. The cause of phase failure is represented by fault trees and an assumption is made that all the components of the UAV are non-repairable for the mission duration.

First a review was carried out on all the existing PMS methods, reported in chapter 3. There was a strong trend that the methods that converted their fault trees to BDDs for the analysis were much more efficient. Therefore in chapter 4 new BDD based methods for PMS analysis were developed and compared. The Prescott et al method [24] was used as a starting benchmark by running some test example missions and recording the online and offline times. The method was efficient as it converts the phase failure cause fault trees to BDDs before the mission configuration is known. Also all the components are treated as phase independent for building the phase failure BDDs, which means they are built instantly by connecting the previous success phase BDDs to the root node of the BDD that represents the failure cause of the current phase. However the next stage of the analysis is to trace through the

phase failure BDDs to calculate the phase failure probabilities which proves to be computationally intensive. This method is improved since every phase unreliability can be obtained by tracing through the final phase failure BDD. This alters the approach by Prescott and is referred to as method 1. Method 1 reduces the number of computations compared to the Prescott et al method by not needing to trace through all of the phase failure BDDs before the final phase. The results show that for online time method 1 was reduced up to 20%. However for larger problems the time taken to evaluate the final phase failure BDD becomes significant and reduces the impact on the overall online time, to approximately only 1% improvement.

Since tracing through the BDDs is very computationally intensive a new method is developed which is an extension of the Trivedi method [13]. The new method calculates the unreliability of the individual phases as well as the overall mission, and is referred to as Method 2. The calculation procedure is also modified for efficiency by rearranging the phase unreliability Boolean formulas so that the phase failure BDD can be obtained by adding to previous phase failure BDDs. The Trivedi method deals with the phase dependencies whilst building the BDD and during the node evaluating stage for the probabilities. Each BDD node only has to be evaluated once which improves on the previous method which traces through the nodes many times. Methods 1 and 2 were tested for speed on mission fault tree examples which have been randomly generated or are related to a UAV mission. The results showed that Method 2 is significantly faster than Method 1 for the UAV mission due to the logic structures which result from real applications. However for the randomly generated fault trees mission Method 2 does not outperform Method 1 since the random fault trees do not contain many common branches. To improve the speed of Method 2, modularisation was applied to take out independent parts of the fault trees that can be analysed separately. This significantly reduces the size of the overall mission fault trees and is referred to as Method 3. These modules can appear many times in UAV mission fault trees since many phases require the same function (such as flaps) or the power system (such as the fuel system). Method 3 applies the linear-time algorithm [24] to a PMS to identify the modules. Method 3 was also tested on the two sets of mission fault trees for a comparison against Method 2. The online time analysis for the randomly generated fault trees were generally not improved by the modularisation approach because they did not have many modules. On the mission

for a UAV operation the improvement was significant for Method 3 compared to Method 2. For missions with more, larger and complex modules, the online time for the biggest mission was reduced from approximately 4 minutes to 1 second. This is down to taking out large complex subsystems such as the DC and AC power supplies.

In chapter 5 the impact of the recursive functions on the analysis time of a PMS are investigated. The recursive function [2], that is applied in building the BDDs, and is modified for PMS analysis, is focussed on since it can be very computationally intensive. There are two stages in the function which can grow significantly. The first searches through all the nodes to establish if the node that has been created already exists. The second stage looks up to see if the computation of the two nodes has already been performed. The first stage which searches for the existence of nodes was tested on 86 missions by running the analysis with and without the search and by an alternative search that is a C++ map. The online and offline times were recorded along with, the number of nodes created and the number of successful searches achieved. The results show that not having a search was a lot faster for the mission consisting of randomly generated phase fault trees since every time the function was called it did not scan through the entire list of nodes already created. However when tested on the PMS for a UAV the search was a lot quicker since there were a lot of successful searches due to the common structure between the phase fault trees. Therefore time was saved by not repeating computations and the option of not having a search approach performed badly because the number of nodes created became very large and in some cases exceeded the memory capacity. Out of the two searches the normal search performed faster for the mission with the randomly generated fault trees and the C++ map search performed faster for the UAV mission. This shows that if a search is used then different searches may be suited to different fault tree structures. Also the recursive function used by Rauzy looks up to establish if a computation of two nodes has already been carried out. The lookup procedure does not take a lot of time to do but requires a lot of memory. The computation lookup was investigated by running the analysis with and without the lookup and recording: the online times, number of nodes created, and number of successful look ups. All results show that the lookup procedure is used a lot for all examples and plays a big part in the speed of the analysis.

Taking out modules from the PMS and analysing them separately has shown to be very effective for UAV mission analysis. The phase fault trees from UAV missions have characteristics that allow a lot of modules to be extracted. In chapter 6 a method is developed that restructures the fault trees to take out the maximum number of modules for a PMS analysis. It was established that in the fault trees there is a relationship between the position of repeated gates and events and the number of independent modules that can be extracted. The method restructures the fault trees by considering the repeated events and gates one at a time. All of the appearances of a repeated event or gate are brought closer together in the tree structure by operations such as push-up, common-input push-up and elimination. The method is tested on the full scale UAV mission. These phase fault trees contain 980 distinct gates and 1007 distinct events. The number of distinct gates and events are recorded before and after the fault trees are restructured and modules are taken out. The number of distinct gates was reduced on average by 90.8% and distinct events by 87.5%. Also the online times are recorded for the method with and without restructuring and the modules taken out. All the online times for taking out modules were less than 30 seconds and when modules were not taken out the method failed to obtain an answer. Even just trying to analysis one of these phases the code would run for 2 hours and runs out of memory. The method is very successful in analysing a large scale problem UAV PMS in a short amount of time. The success is down to significantly reducing the fault tree complexity by maximising the number of modules that can be extracted. This is very effective on the UAV mission structure phase fault trees because they contain lots of common functions and energy supply sources.

In chapter 7 the PMS method presented in chapter 6 was extended to consider the practical situation where the UAV mission needs to be reconfigured partway through. Reconfiguration can be due to a failure diagnosed or rerouting required to avoid emerging threats. The method updates the analysis of the PMS for the new reconfigured mission in an efficient way by reusing phase BDDs from the original mission. The probabilities of phase failure given that it was successful up to a point in the mission are predicted. The method was tested on missions to perform tasks such as anti-submarine warfare (ASW), anti-surface warfare (ASUW) and search and rescue (SAR). The missions were analysed reusing the previous phases



BDDs and also without this approach. The online analysis times were recorded for a comparison. The results show that as the mission progresses and the number of completed phases grows the greater the benefit of the new approach.

An additional parallelization technique that adds to the efficiency of the running of the online analysis is developed in chapter 8. This enables the analysis to be performed faster by using several processors.

## **9.2 Conclusions**

The aim of this research was to develop new methods that analyse the phase unreliabilities of a real UAV mission in the shortest possible time. The following conclusions are made:

- PMS based on the BDD technique are the most efficient for obtaining the phase unreliabilities in the quickest time.
- Method 1 improved the PMS method for building the phase BDDs failure enabling the failure causes for each phase to be obtained offline. However when the method quantifies the phase BDDs for larger problems the time it takes grows significantly.
- Method 2, which is an extension of the Trivedi method, obtains the phase failure probabilities as well as the entire mission unreliability in a more efficient way. This method is more effective than method 1 for UAV missions. Since it deals with the component dependencies between phases whilst creating the phase BDDs.
- Modularization significantly reduces the complexity of UAV mission online analysis since there are many modules due to common functions and power supplies throughout the phases.

- Restructuring techniques can be applied to the phase fault trees offline to maximise the number of modules for a UAV mission. The extraction of these modules from the phase fault trees simplifies the analysis dramatically.
- Search lookup techniques for recursive functions in the analysis only have positive impact on reducing the online analysis if there are many repeated computations which are commonly performed when analysing the UAV missions. However where the analysis does not have many repeated computations such as in randomly generated missionstructures lookup techniques will have the adverse effect of increasing online analysis. In addition to this, different search techniques have a different affect on the speed of the analysis.
- The parallelization method significantly reduces the online time analysis for UAV missions.

### **9.3 Further Work**

The result of this research leads to the possibility of further areas of investigation. Potential directions are discussed in the following section.

#### **9.3.1 Factorization**

In chapter 6 a technique was used in the PMS method which factorizes a pair of gates in the fault trees that always occur together and are just dependent on each other, so the pair can be combined and taken out as a module. An extension to this technique could combine and factorize a set of gates that only depend on each other. For example three, four or even more gates could be factorized together.

#### **9.3.2 Optimum BDD Ordering Schemes for PMS**

A lot of research has given attention to the optimum BDDnode ordering scheme. This has proven to have a big impact on the efficiency of the analysis. Little research

has been done on optimum nodeordering schemes for a PMS. Different ordering schemes can be effective for different fault tree structures. Therefore research could be conducted to produce an ordering scheme for maximizing the efficiency phase fault trees for a UAV mission.

### **9.3.3 Multi Platform Missions**

Research has been carried out for the analysis of multi-platform missions. In this thesis the focus of the analysis was on a single UAV mission. The research could be extended to consider a set of autonomous vehicles (such as UAV's, ground vehicles etc) that work together to achieve a common mission. First the methods developed in this thesis could be applied to a multi-platform mission analysis. Secondly the efficiency of the calculations can be investigated. For example one area where the analysis time could be reduced is that the UAVs may have common subsystems, such as communications, or might be exactly the same vehicle type. Therefore parts of analysis on one UAV may be reused in the analysis of another.

### **9.3.4 Dependency**

The methods developed in this thesis assume that component failures are independent from each other. However in practical situations some of the UAV component failure states may have dependencies between each other. Methods have already been developed for the single phase case analysing systems with component dependencies. The application of these methods could be investigated and developed for the PMS case.

### **9.3.5 Multi Component States**

Throughout this research it has been assumed that all the components are in one of two states, failed or working. However, a component could exist in one of several failure or partial failure. Methods could be developed to analysis this situation for a PMS.

### **9.3.6 Importance Measure**

Work has been conducted to establish importance measures for PMS. Measures relate components and minimal cut sets and quantify the amount they contribute to the failure of the mission or phase. An importance measure could be developed to indicate how a UAV sub-system contributes to the failure of the mission or to a phase.

## Reference

- [1] J.D. Andrews, T.R. Moss, "Reliability and Risk Assessment", Second Edition, Professional Engineering Publishing Limited, 2002.
- [2] W.E Vesely, 'A Time-Dependent Methodology for Fault Tree Evaluation', nuclear Design and Engineering, vol. 13, 1970, pp 337-360.
- [3] A.Rauzy, "New Algorithms for Fault", Reliability Engineering and System Safety, vol 40, 1993, pp203-211.
- [4] R.M. Sinnamon, J.D. Andrews, "Improved Efficiency in Qualitative Fault Tree Analysis", Quality and Reliability Engineering Int, vol 13, no. 5, 1997, pp293-298.
- [5] R.M. Sinnamon, J.D. Andrew, "Improved Accuracy in Quantitative Fault Tree Analysis", Quality and Reliability Engineering Int, vol 13, no.5, 1997, pp285-292.
- [6] J.D.Esary, H. Ziehms, "Reliability of Phased Missions", Reliability and Fault-Tree Analysis, Society for Industrial Applied Mathematics, Phila 1975, pp 213-236, 1974 September.
- [7] G.R Burdick, J.B.Fussell, D.M.Rasmuson, J.R.Wilson, "Phased Mission Analysis: A Review of New Developments and an Application", IEEE trans. Reliability, vol R-26, 1977 Apr, pp 43-49
- [8] D.F.Montague, J.B.Fussell, "A Methodogy for Calculating the Expected Number of Failure of a System Undergoing a Phased Mission", Nuclear Science and Engineering, vol 74, 1980, pp 199-209
- [9] Xue Dazhi, Wang Xiaozhong, " A Practical Approach for Phased Mission Analysis", Reliability Engineering and System Safety, vol 25, 1989, pp 333-347.
- [10] T.Kohda, M. Wada, K. Inoue, "A Simple Method for Phased Mission Analysis", Reliability Engineering and System Safety, vol 45, 1994, pp299-309
- [11] A.K.Somani, K.S.Trivedi, "Boolean Algebraic Methods for Phased-Mission System Analysis", in Proceedings of Sigmetrics, 1994 May, pp 98-107.
- [12] Y.Ma, K.S.Trivedi, " An algorithm for reliability of phased-mission systems", Reliability Engineering and System and Safety, vol 66, 1999, pp157-170.
- [13] R A La Band, J D Andrews, " Phased Mission Modelling using Fault Tree Analysis", Department of Aeronautical and Automotive Engineering. Loughborough, Leicestershire, UK.

- [14] X.Zang, H.Sun, and K.S.Trivedi, "A BDD-based algorithm for reliability analysis of phased-mission systems," IEEE Transactions on Reliability, vol. 48, pp. 50-60, March 1999.
- [15] L.Xing, J.B.Dugan, "Analysis of Generalized Phased-Mission Systems Reliability, Performance, and Sensitivity" IEEE Transactions on Reliability, vol. 51, pp. 199-211, June 2002.
- [16] L.Xing, J.B.Dugan, "Comments on PMS BDD Generation in " A BDD-Based Algorithm for Reliability Analysis of Phased-Mission Systems"" IEEE Transactions on Reliability, vol. 53, pp. 169-173, June 2004.
- [17] L.Xing, J.B.Dugan, "A Separable Ternary Decision Diagram Based Analysis of Generalized Phased-Mission Reliability", IEEE Transactions on Reliability, vol. 53, pp. 174-184, June 2004.
- [18] R. La Band, "Systems Reliability for Phased Missions", Doctor of Philosophy of Loughborough University, March 2005.
- [19] L.Xing, "Reliability Evaluation of Phased-Mission System with Imperfect Fault Coverage and Common-Cause Failures", IEEE Transactions on Reliability, vol. 56, pp. 58-67, June 2007.
- [20] Z.Tang, J.B.Dugan, "BDD-Based Reliability Analysis of Phased-Mission Systems with Multimode Failures", IEEE Transactions on Reliability, vol. 55, pp. 350-360, June 2006.
- [21] G.Vyaite, S.Dunnett, J.Anderws, "Cause-consequence analysis of non-repairable phased mission", Reliability Engineering and System Safety, vol 91,pp 398-406, 2006
- [22] J.K.Vaurio, "Fault tree analysis of phased mission system with repairable and non-repairable component", Reliability Engineering and System Safety, vol 74,pp 169-180, 2001
- [23] Y.Ou,J.B.Dugan, "Modular Solution of Dynamic Multi-Phase Systems", IEEE Transactions on Reliability, vol. 53, pp. 499-508, December 2004.
- [24] D.R.Prescott, R.Remenyte, S.Reed, J.D.Andrewes,"A Fast Analysis Method for Phased Mission Using the Binary Decision Diagram Method", Department of Aeronautical and Automotive Engineering, Loughborough University.
- [25] Y.Dutuit, A.Rauzy "A Linear-Time Algorithm to Find Modules of Fault Trees" IEEE Transactions on Reliability, vol. 45, pp. 422-425, September 2002.

[26] samuel crew thesis ,”Systems Reliability Modelling for Phases Missions with Maintenance Free Operating Periods 2009” Department of Aeronautical and Automotive Engineering, Loughborough University

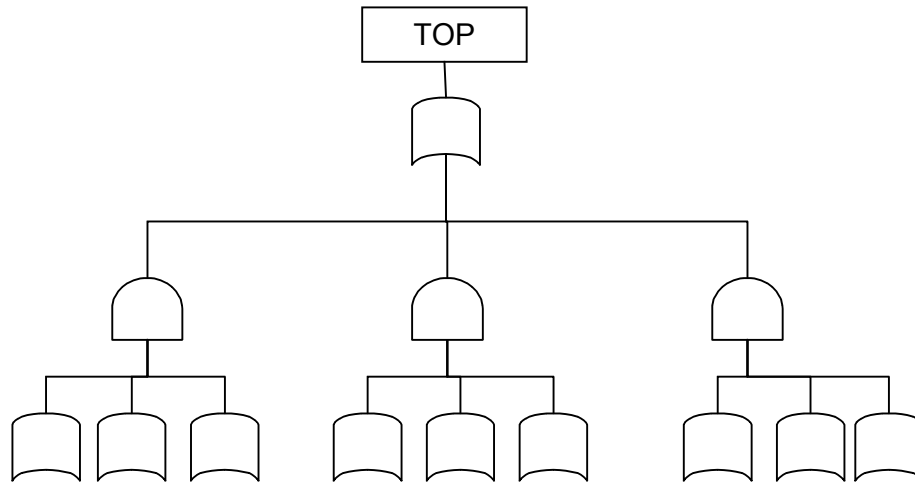
[27] D R Prescott “Multipaltion phased mission reliability modelling for mission planning” Department of Aeronautical and Automotive Engineering, Loughborough University

[28] K.Brazensite, W.H.Chen, J.D. Andrews “Reliability Based Mission reconfiguration for UAVs using Genetic Algorithm Approach” Department of Aeronautical and Automotive Engineering, Loughborough University

## Appendix A: Mission data of random generated phase fault trees

### Mission set 1

Each gate has 3 events inputs



Possible phases Fault tree	No gate (OR,AND)	No events Including repeated events	No events not including repeated events	No common Phase Events not including repeated events
1	13 (10,3)	33	27	23
2	13 (10,3)	33	30	19
3	13 (10,3)	33	29	22
4	13 (10,3)	33	31`	20
5	13 (10,3)	33	32	18
6	13 (10,3)	33	31	23
7	13 (10,3)	33	32	24
8	13 (10,3)	33	31	27



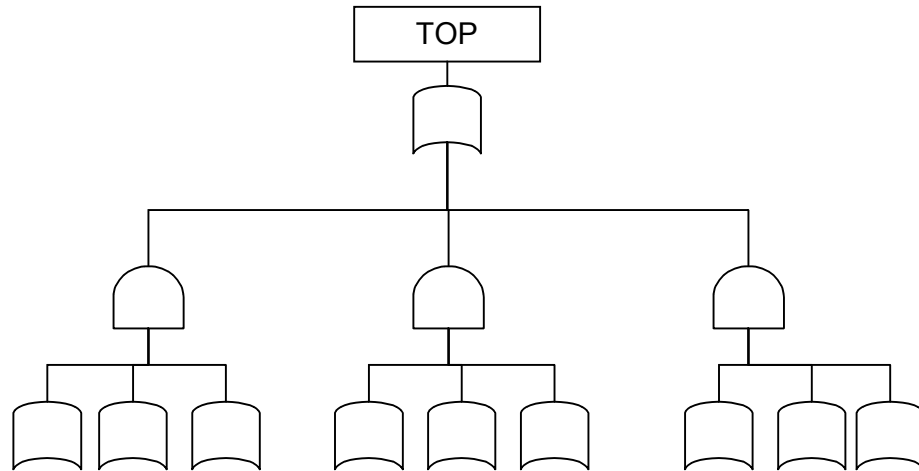
Number of modules that are in all the possible phase	48
Number of gate not repeated in all the possible phase	103
Number of event not repeated in all the possible phase	201

<p>TOP1 1 3 3 G1 G2 G3 JeJe BjJBjJ BdjBdj  G1 2 3 3 G4 G5 G6 AA CC BB  G4 1 0 3 DdDd JcJc BdbBdb  G5 1 0 3 BafBaf DeDe BheBhe  G6 1 0 3 GgGg DgDg BgcBgc  G2 2 3 3 G7 G8 G9 BafBaf BgcBgc BjdBjd  G7 1 0 3 BfbBfb BgeBge BbhBbh  G8 1 0 3 II AA BbjBbj  G9 1 0 3 BfbBfb BbcBbc BgaBga  G3 2 3 3 G10 G11 G12 BjfBjf BfhBfh BfbBfb  G10 1 0 3 BdBd BjaBja BaiBai  G11 1 0 3 GgGg BeaBea DhDh  G12 1 0 3 BciBci BihBih IfIf</p>	<p>TOP2 1 3 3 G13 G14 G15 BddBdd BjBj BfdBfd  G13 2 3 3 G16 G17 G18 BeBe DcDc BcbBcb  G16 1 0 3 IhIh FcFc Iele  G17 1 0 3 FiFi BehBeh BjJBjJ  G18 1 0 3 BB BajBaj BiaBia  G14 2 3 3 G19 G20 G21 HeHe BdhBdh BbjBbj  G19 1 0 3 Iblb FdFd BjJBjJ  G20 1 0 3 BfiBfi Iblb CdCd  G21 1 0 3 EeEe BcfBcf BiiBii  G15 2 3 3 G22 G23 G24 BdBd AA FeFe  G22 1 0 3 BbjBbj DhDh DD  G23 1 0 3 BaeBae BcgBcg JijI  G24 1 0 3 BifBif BffBff FfFf</p>
<p>TOP3 1 3 3 G25 G26 G27 BjeBje HaHa BciBci  G25 2 3 3 G28 G29 G30 BceBce BB HfHf  G28 1 0 3 EE BbdBbd BheBhe  G29 1 0 3 BddBdd IfIf CdCd  G30 1 0 3 BigBig BhgBhg BggBgg  G26 2 3 3 G31 G32 G33 BhcBhc BfhBfh  BggBgg  G31 1 0 3 JeJe BgbBgb BcjBcj  G32 1 0 3 DfDf EeEe BbiBbi  G33 1 0 3 BbgBbg FjFj FfFf  G27 2 3 3 G34 G35 G36 BgiBgi BefBef BhgBhg  G34 1 0 3 FF IfIf BjhBjh  G35 1 0 3 BejBej DdDd DcDc  G36 1 0 3 GgGg BheBhe BahBah</p>	<p>TOP4 1 3 3 G37 G38 G39 Ii BehBeh BheBhe  G37 2 3 3 G40 G41 G42 BeiBei IdId BbaBba  G40 1 0 3 BjgBjg BagBag DcDc  G41 1 0 3 BbdBbd GjGj BahBah  G42 1 0 3 GdGd HiHi HeHe  G38 2 3 3 G43 G44 G45 BhaBha CjCj DeDe  G43 1 0 3 BdiBdi GaGa EjEj  G44 1 0 3 BdBd BdgBdg DaDa  G45 1 0 3 JdJd EfEf DhDh  G39 2 3 3 G46 G47 G48 FF HeHe JcJc  G46 1 0 3 JJ HaHa BbhBbh  G47 1 0 3 BghBgh CaCa JbJb  G48 1 0 3 BhjBhj BchBch BjBjBj</p>

<p>TOP5 1 3 3 G49 G50 G51 BfBf BefBef GaGa  G49 2 3 3 G52 G53 G54 BcjBcj DcDc BbBb  G52 1 0 3 BdhBdh HdHd BiaBia  G53 1 0 3 BbeBbe EiEi HeHe  G54 1 0 3 FbFb EaEa FF  G50 2 3 3 G55 G56 G57 BdaBda BdiBdi JcJc  G55 1 0 3 IjIj BbbBbb BccBcc  G56 1 0 3 HbHb BcfBcf BfaBfa  G57 1 0 3 CiCi EfEf DhDh  G51 2 3 3 G58 G59 G60 BgiBgi EgEg BibBib  G58 1 0 3 BebBeb DD BiiBii  G59 1 0 3 lala GbGb CfCf  G60 1 0 3 BaaBaa FF BggBgg</p>	<p>TOP6 1 3 3 G61 G62 G63 BcdBcd BdgBdg DeDe  G61 2 3 3 G64 G65 G66 JcJc FcFc IfIf  G64 1 0 3 BejBej DfDf BhBh  G65 1 0 3 BghBgh BdeBde BgBg  G66 1 0 3 BbcBbc BbiBbi BggBgg  G62 2 3 3 G67 G68 G69 CcCc Ii CfCf  G67 1 0 3 CjCj HfHf BecBec  G68 1 0 3 DgDg CaCa BhbBhb  G69 1 0 3 BagBag BfbBfb BdjBdj  G63 2 3 3 G70 G71 G72 DdDd JiJi BfeBfe  G70 1 0 3 BbgBbg JeJe BgBg  G71 1 0 3 lili BhhBhh IjIj  G72 1 0 3 DgDg BheBhe BihBih</p>
<p>TOP7 1 3 3 G73 G74 G75 BfaBfa BgcBgc  GaGa  G73 2 3 3 G76 G77 G78 BgaBga BefBef  BdbBdb  G76 1 0 3 DfDf BheBhe CdCd  G77 1 0 3 EeEe DhDh Iglg  G78 1 0 3 BfjBfj HiHi JiJi  G74 2 3 3 G79 G80 G81 BceBce BjBj  BahBah  G79 1 0 3 EdEd BbeBbe Iblb  G80 1 0 3 BbfBbf JfJf JjJj  G81 1 0 3 FdFd BdcBdc BdjBdj  G75 2 3 3 G82 G83 G84 BgjBgj EiEi GeGe  G82 1 0 3 BcdBcd IjIj EcEc  G83 1 0 3 JdJd BdjBdj BfdBfd  G84 1 0 3 BeiBei JbJb IfIf</p>	<p>TOP8 1 3 3 G85 G86 G87 BgdBgd Iele BgeBge  G85 2 3 3 G88 G89 G90 BagBag BfhBfh BbbBbb  G88 1 0 3 BheBhe BjjBjj BdbBdb  G89 1 0 3 DcDc BbBb BcaBca  G90 1 0 3 BafBaf BdcBdc BiaBia  G86 2 3 3 G91 G92 G93 BbfBbf DD BedBed  G91 1 0 3 BaeBae BjBj EbEb  G92 1 0 3 AA BbdBbd BdjBdj  G93 1 0 3 BdgBdg BeaBea BfaBfa  G87 2 3 3 G94 G95 G96 BfbBfb JiJi DcDc  G94 1 0 3 BdiBdi BffBff BiiBii  G95 1 0 3 lala DdDd BjbBjb  G96 1 0 3 BdjBdj BfeBfe EjEj</p>

## Mission set 2

Each gate has 0- 3 events inputs



Possible phases Fault tree	No gate (OR,AND)	No events Including repeated events	No events not including repeated events	No common Phase events
1	13(10,3)	30	28	12
2	13(10,3)	32	31	15
3	13(10,3)	31	29	16
4	13(10,3)	29	29	13
5	13(10,3)	32	31	12
6	13(10,3)	33	31	23

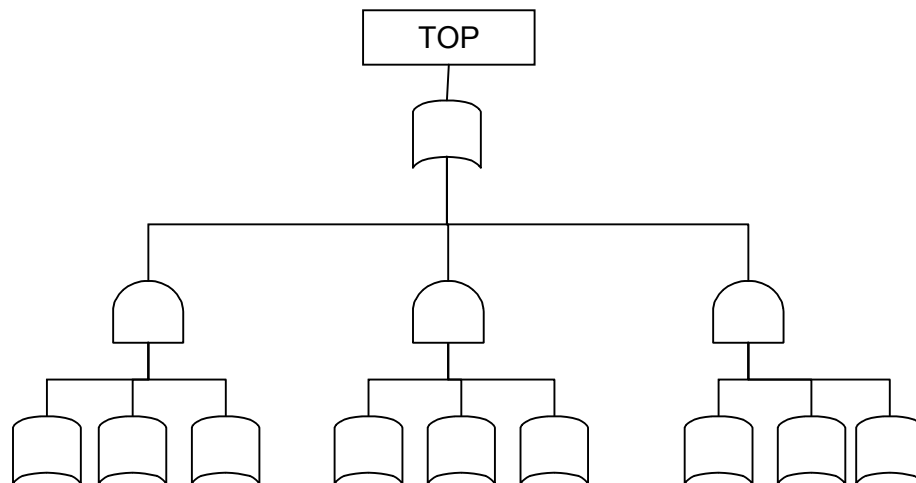
Number of modules that are in all the possible phase	50
Number of gate not repeated in all the possible phase	72
Number of event not repeated in all the possible phase	175

<p>TOP1 1 3 1 G1 G2 G3 JeJe</p> <p>G1 2 3 3 G4 G5 G6 AA CC BB</p> <p>G4 1 0 2 JcJc BdbBdb</p> <p>G5 1 0 3 BafBaf DeDe BheBhe</p> <p>G6 1 0 3 GgGg DgDg BgcBgc</p> <p>G2 2 3 0 G7 G8 G9</p> <p>G7 1 0 3 BfbBfb BgeBge BbhBbh</p> <p>G8 1 0 2 Il BbjBbj</p> <p>G9 1 0 3 BfbBfb BbcBbc BgaBga</p> <p>G3 2 3 3 G10 G11 G12 BjfBjf BfhBfh BfbBfb</p> <p>G10 1 0 2 BjaBja BaiBai</p> <p>G11 1 0 2 BeaBea DhDh</p> <p>G12 1 0 3 BciBci BihBih Iflf</p>	<p>TOP2 1 3 1 G13 G14 G15 BddBdd</p> <p>G13 2 3 3 G16 G17 G18 BeBe DcDc BcbBcb</p> <p>G16 1 0 2 IhIh FcFc</p> <p>G17 1 0 2 FiFi BehBeh</p> <p>G18 1 0 3 BB BajBaj BiaBia</p> <p>G14 2 3 1 G19 G20 G21 HeHe</p> <p>G19 1 0 3 lblb FdFd BjJBjj</p> <p>G20 1 0 3 BfiBfi lblb CdCd</p> <p>G21 1 0 3 EeEe BcfBcf BiiBii</p> <p>G15 2 3 3 G22 G23 G24 BdDd AA FeFe</p> <p>G22 1 0 2 BbjBbj DhDh</p> <p>G23 1 0 3 BaeBae BcgBcg JijI</p> <p>G24 1 0 3 BifBif BffBff FfFf</p>
<p>TOP3 1 3 2 G25 G26 G27 BjeBje HaHa</p> <p>G25 2 3 1 G28 G29 G30 BceBce</p> <p>G28 1 0 2 EE BbdBbd</p> <p>G29 1 0 2 BddBdd Iflf</p> <p>G30 1 0 3 BigBig BhgBhg BggBgg</p> <p>G26 2 3 0 G31 G32 G33</p> <p>G31 1 0 3 JeJe BgbBgb BcjBcj</p> <p>G32 1 0 3 DfDf EeEe BbiBbi</p> <p>G33 1 0 3 BbgBbg FjFj FfFf</p> <p>G27 2 3 3 G34 G35 G36 BgiBgi BefBef</p> <p>BhgBhg</p> <p>G34 1 0 3 FF Iflf BjhBjh</p> <p>G35 1 0 3 BejBej DdDd DcDc</p> <p>G36 1 0 2 GgGg BahBah</p>	<p>TOP4 1 3 1 G37 G38 G39 BehBeh</p> <p>G37 2 3 2 G40 G41 G42 BeiBei IdId</p> <p>G40 1 0 3 BjbBjb BagBag DcDc</p> <p>G41 1 0 2 BbdBbd GjGj</p> <p>G42 1 0 2 GdGd HeHe</p> <p>G38 2 3 3 G43 G44 G45 BhaBha CjCj DeDe</p> <p>G43 1 0 3 BdiBdi GaGa EjEj</p> <p>G44 1 0 2 BdDd DaDa</p> <p>G45 1 0 3 JdJd EfEf DhDh</p> <p>G39 2 3 1 G46 G47 G48 JcJc</p> <p>G46 1 0 3 JJ HaHa BbhBbh</p> <p>G47 1 0 2 BghBgh JbJb</p> <p>G48 1 0 2 BchBch BjBjBj</p>

TOP5 1 3 3 G49 G50 G51 BfBf BefBef GaGa	TOP6 1 3 1 G61 G62 G63 JiJi
G49 2 3 3 G52 G53 G54 BcjBcj DcDc BbBb	G61 2 3 3 G64 G65 G66 JcJc FcFc IfIf
G52 1 0 3 BdhBdh HdHd BiaBia	G64 1 0 2 BejBej DfDf
G53 1 0 3 BbeBbe EiEi HeHe	G65 1 0 3 BghBgh BdeBde BgBg
G54 1 0 3 FbFb EaEa FF	G66 1 0 3 BbcBbc BbiBbi BggBgg
G50 2 3 1 G55 G56 G57 BdaBda	G62 2 3 2 G67 G68 G69 CcCc II
G55 1 0 3 IjIj BbbBbb BccBcc	G67 1 0 3 CjCj HfHf BecBec
G56 1 0 2 HbHb BcfBcf	G68 1 0 3 DgDg CaCa BhbBhb
G57 1 0 3 CiCi EfEf DhDh	G69 1 0 2 BagBag BdjBdj
G51 2 3 0 G58 G59 G60	G63 2 3 3 G70 G71 G72 DdDd JiJi BfeBfe
G58 1 0 3 BebBeb DD BiiBii	G70 1 0 3 BbgBbg JeJe BgBg
G59 1 0 3 lala GbGb CfCf	G71 1 0 3 lili BhhBhh IjIj
G60 1 0 2 BaaBaa FF	G72 1 0 2 DgDg BheBhe

### Mission set 3

Each gate has 0- 3 events inputs



Possible phases Fault tree	No gate (OR,AND)	No events Including repeated events	No events not including repeated events	No common Phase Events not including repeated events
1	13(10,3)	31	22	8
2	13(10,3)	32	22	7
3	13(10,3)	31	26	9
4	13(10,3)	29	18	5
5	13(10,3)	33	22	10
6	13(10,3)	33	23	15

Number of modules that are in all the possible phase	6
Number of gate not repeated in all the possible phase	72
Number of event not repeated in all the possible phase	107 (101)

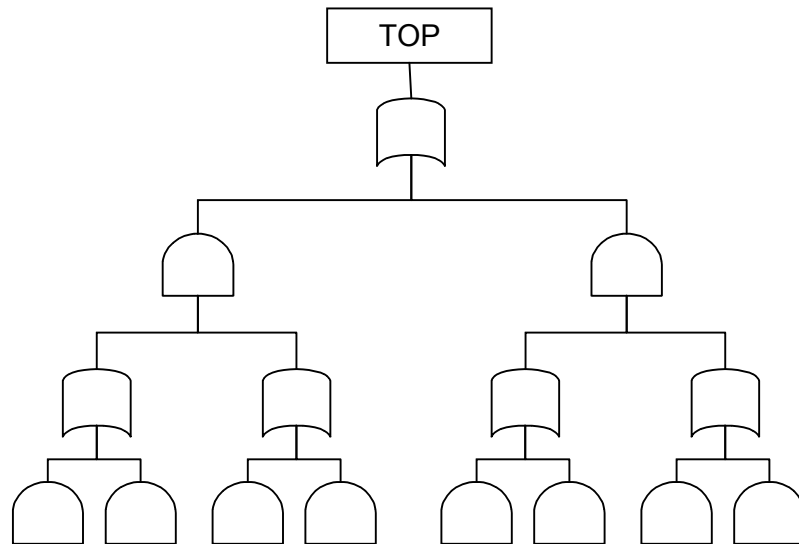
TOP1 1 3 1 G1 G2 G3 BB G1 2 3 3 G4 G5 G6 AA CC BB G4 1 0 2 JcJc BdbBdb G5 1 0 3 BafBaf DeDe BheBhe G6 1 0 3 GgGg DgDg BgcBgc G2 2 3 0 G7 G8 G9 G7 1 0 3 BfbBfb BdbBdb CC G8 1 0 2 II BbjBbj G9 1 0 3 DgDg BbcBbc BgaBga G3 2 3 3 G10 G11 G12 DeDe BfhBfh II G10 1 0 2 BjaBja DgDg G11 1 0 2 BeaBea DhDh G12 1 0 3 AA BihBih Ifif	TOP2 1 3 1 G13 G14 G15 FeFe G13 2 3 3 G16 G17 G18 BeBe FffFf BcbBcb G16 1 0 2 IhIh Iblb G17 1 0 2 FiFi BehBeh G18 1 0 3 BB Iblb BiaBia G14 2 3 1 G19 G20 G21 BehBeh G19 1 0 3 Iblb FdFd BjjBjj G20 1 0 3 BfiBfi Iblb BeBe G21 1 0 3 BB BcfBcf BiiBii G15 2 3 3 G22 G23 G24 BB AA FeFe G22 1 0 2 BbjBbj BehBeh G23 1 0 3 BaeBae BcgBcg JijJi G24 1 0 3 DhDh BffBff FffFf
--	---

<p>TOP3 1 3 2 G25 G26 G27 BjeBje HaHa</p> <p>G25 2 3 1 G28 G29 G30 BceBce</p> <p>G28 1 0 2 EE BbdBbd</p> <p>G29 1 0 2 BddBdd Iflf</p> <p>G30 1 0 3 BigBig BhgBhg BbdBbd</p> <p>G26 2 3 0 G31 G32 G33</p> <p>G31 1 0 3 JeJe BgbBgb DcDc</p> <p>G32 1 0 3 DcDc EeEe BbiBbi</p> <p>G33 1 0 3 BbgBbg FjFj EE</p> <p>G27 2 3 3 G34 G35 G36 BgiBgi BefBef Iflf</p> <p>G34 1 0 3 FF Iflf BjhBjh</p> <p>G35 1 0 3 BejBej DdDd DcDc</p> <p>G36 1 0 2 DfDf BahBah</p>	<p>TOP4 1 3 1 G37 G38 G39 BehBeh</p> <p>G37 2 3 2 G40 G41 G42 BeiBei DaDa</p> <p>G40 1 0 3 Bj BagBag BjbBjb</p> <p>G41 1 0 2 DaDa GjGj</p> <p>G42 1 0 2 JbJb HaHa</p> <p>G38 2 3 3 G43 G44 G45 BagBag CjCj Bj</p> <p>G43 1 0 3 BdiBdi GaGa EjEj</p> <p>G44 1 0 2 HeHe DaDa</p> <p>G45 1 0 3 JdJd EfEf DhDh</p> <p>G39 2 3 1 G46 G47 G48 EjEj</p> <p>G46 1 0 3 JJ HaHa CjCj</p> <p>G47 1 0 2 DaDa JbJb</p> <p>G48 1 0 2 CjCj BjbBjb</p>
---	--

<p>TOP5 1 3 3 G49 G50 G51 BfBf BefBef GaGa</p> <p>G49 2 3 3 G52 G53 G54 BcjBcj DcDc BbBb</p> <p>G52 1 0 3 BdhBdh GaGa BiaBia</p> <p>G53 1 0 3 FbFb EiEi DhDh</p> <p>G54 1 0 3 FbFb lala GaGa</p> <p>G50 2 3 1 G55 G56 G57 BdaBda</p> <p>G55 1 0 3 IjIj BbbBbb BccBcc</p> <p>G56 1 0 2 HbHb BcfBcf</p> <p>G57 1 0 3 BdhBdh EfEf BbBb</p> <p>G51 2 3 0 G58 G59 G60</p> <p>G58 1 0 3 DhDh DD DhDh</p> <p>G59 1 0 3 lala GbGb GaGa</p> <p>G60 1 0 2 BaaBaa BbBb</p>	<p>TOP6 1 3 1 G61 G62 G63 JiJi</p> <p>G61 2 3 3 G64 G65 G66 JcJc FcFc Iflf</p> <p>G64 1 0 2 BejBej BhhBhh</p> <p>G65 1 0 3 BdjBdj BdeBde BgBg</p> <p>G66 1 0 3 BejBej JiJi BggBgg</p> <p>G62 2 3 2 G67 G68 G69 CcCc II</p> <p>G67 1 0 3 CcCc HfHf JiJi</p> <p>G68 1 0 3 DgDg HfHf BheBhe</p> <p>G69 1 0 2 BagBag BdjBdj</p> <p>G63 2 3 3 G70 G71 G72 DdDd JiJi lili</p> <p>G70 1 0 3 BbgBbg JeJe BgBg</p> <p>G71 1 0 3 lili BhhBhh IjIj</p> <p>G72 1 0 2 DdDd BejBej</p>
---	--

## Mission set 4

Each gate has 2 events inputs



Possible phases Fault tree	No gate (OR,AND)	No events Including repeated events	No events not Including repeated events	No common Phase Events not including repeated events
1	15(5,10)	30	22	10
2	15(5,10)	30	19	12
3	15(5,10)	30	21	14
4	15(5,10)	30	22	12
5	15(5,10)	30	24	18

Number of modules that are in all the possible phase	4
Number of gate not repeated in all the possible phase	56
Number of event not repeated in all the possible phase	40 (37)



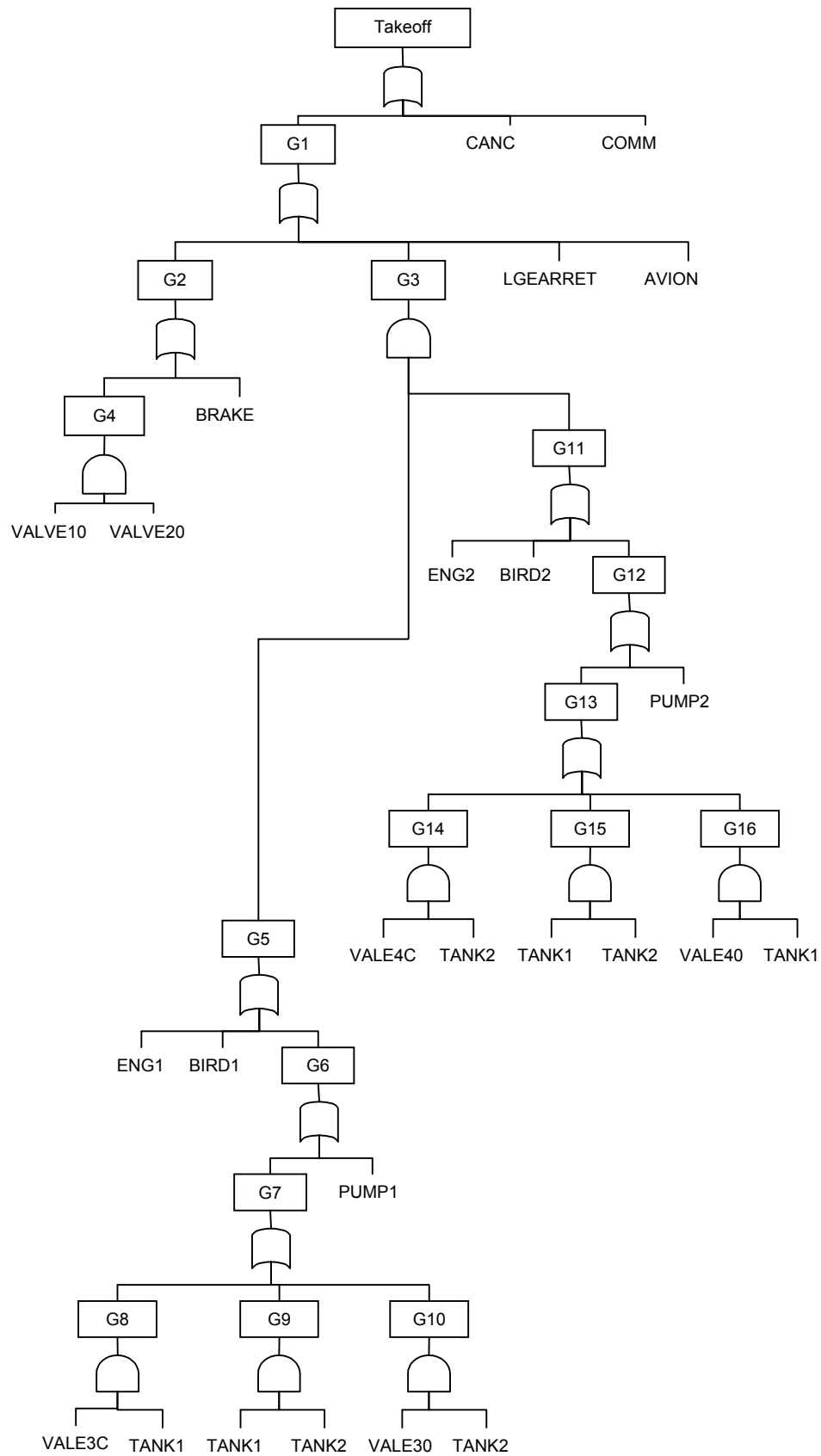
<p>TOP1 1 2 2 G1 G2 II BaBa  G1 2 2 2 G3 G4 AA GG  G3 1 2 2 G5 G6 HH CC  G5 2 0 2 CbCb FF  G6 2 0 2 BiBi BcBc  G4 1 2 2 G7 G8 BdBd BgBg  G7 2 0 2 BeBe CfCf  G8 2 0 2 CC BjBj  G2 2 2 2 G9 G10 BeBe BbBb  G9 1 2 2 G11 G12 CgCg EE  G11 2 0 2 BfBf CdCd  G12 2 0 2 CfCf BcBc  G10 1 2 2 G13 G14 FF BcBc  G13 2 0 2 CbCb BfBf  G14 2 0 2 BdBd BgBg</p>	<p>TOP2 1 2 2 G15 G16 II ChCh  G15 2 2 2 G17 G18 CC EE  G17 1 2 2 G19 G20 CfCf BcBc  G19 2 0 2 CcCc CeCe  G20 2 0 2 CaCa BiBi  G18 1 2 2 G21 G22 BdBd CbCb  G21 2 0 2 CgCg CfCf  G22 2 0 2 JJ DD  G16 2 2 2 G23 G24 CfCf CeCe  G23 1 2 2 G25 G26 CiCi BiBi  G25 2 0 2 BeBe CaCa  G26 2 0 2 JJ EE  G24 1 2 2 G27 G28 BdBd CgCg  G27 2 0 2 BfBf BcBc  G28 2 0 2 CaCa DD</p>
<p>TOP3 1 2 2 G29 G30 BiBi BcBc  G29 2 2 2 G31 G32 CC BgBg  G31 1 2 2 G33 G34 BdBd CaCa  G33 2 0 2 JJ GG  G34 2 0 2 BhBh DD  G32 1 2 2 G35 G36 CjCj BeBe  G35 2 0 2 CcCc II  G36 2 0 2 BB BaBa  G30 2 2 2 G37 G38 CC JJ  G37 1 2 2 G39 G40 BeBe CeCe  G39 2 0 2 CiCi CdCd  G40 2 0 2 II CaCa  G38 1 2 2 G41 G42 CbCb BB  G41 2 0 2 DD CdCd  G42 2 0 2 CfCf GG</p>	<p>TOP4 1 2 2 G43 G44 BdBd CbCb  G43 2 2 2 G45 G46 BeBe EE  G45 1 2 2 G47 G48 CiCi BaBa  G47 2 0 2 CcCc JJ  G48 2 0 2 CgCg BbBb  G46 1 2 2 G49 G50 ChCh GG  G49 2 0 2 HH BfBf  G50 2 0 2 BhBh BaBa  G44 2 2 2 G51 G52 BfBf CfCf  G51 1 2 2 G53 G54 JJ BaBa  G53 2 0 2 CcCc CdCd  G54 2 0 2 AA CeCe  G52 1 2 2 G55 G56 CcCc ChCh  G55 2 0 2 BB BbBb  G56 2 0 2 HH CdCd</p>

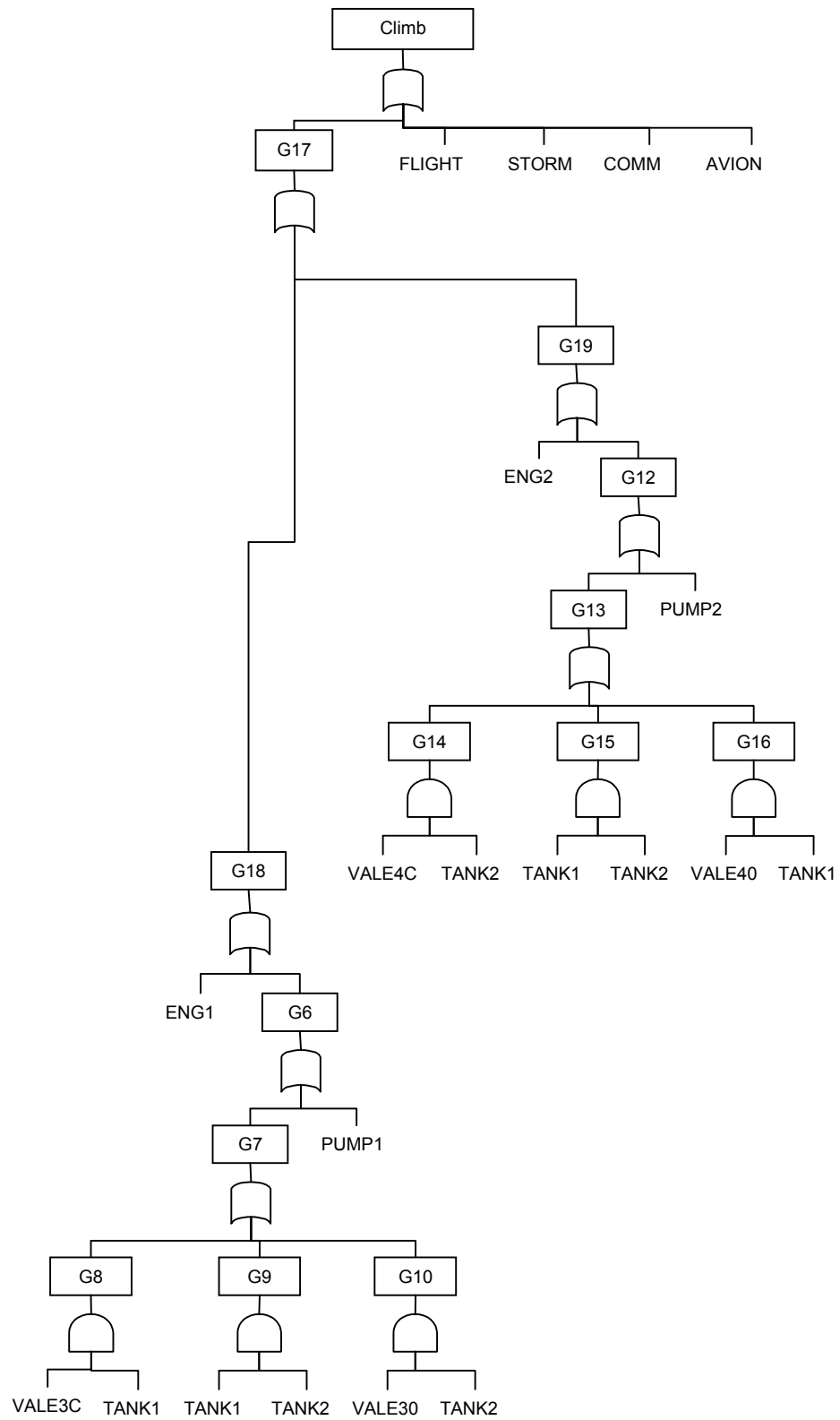
```
TOP 1 2 2 G43 G44 jam GG
G43 2 2 2 G45 G46 bb EE
G45 1 2 2 G47 G48 CiCi BaBa
G47 2 0 2 BfBf BcBc
G48 2 0 2 chch olol
G46 1 2 2 G49 G50 CiCi GG
G49 2 0 2 HH BfBf
G50 2 0 2 qwqw BaBa
G44 2 2 2 G51 G52 BfBf CfCf
G51 1 2 2 G53 G54 ii erer
G53 2 0 2 CcCc CdCd
G54 2 0 2 AA CeCe
G52 1 2 2 G55 G56 CcCc ChCh
G55 2 0 2 BB DD
G56 2 0 2 HH caca
ENDOFTREE
```

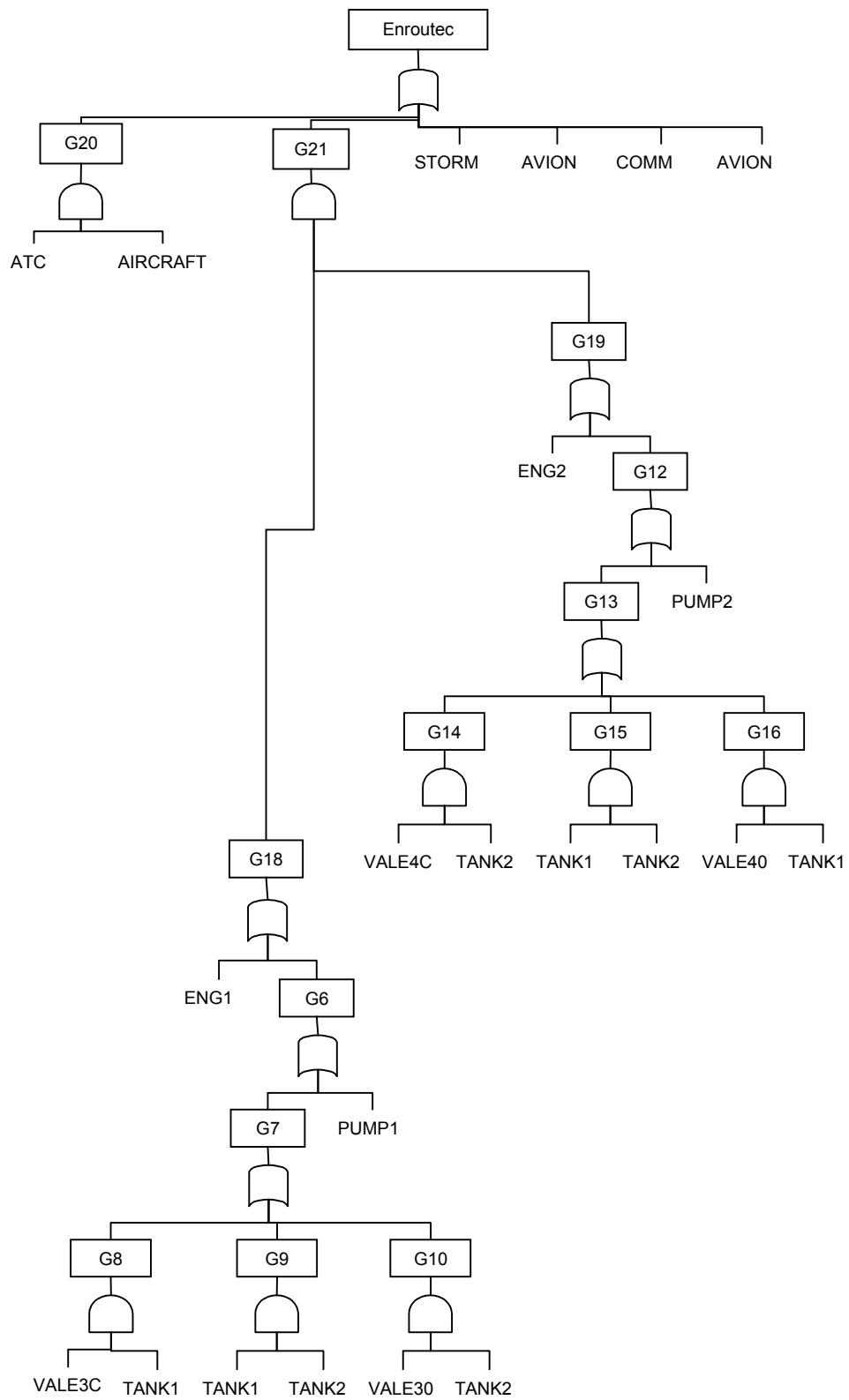
## Appendix B: Mission data of UAV generated phase fault trees

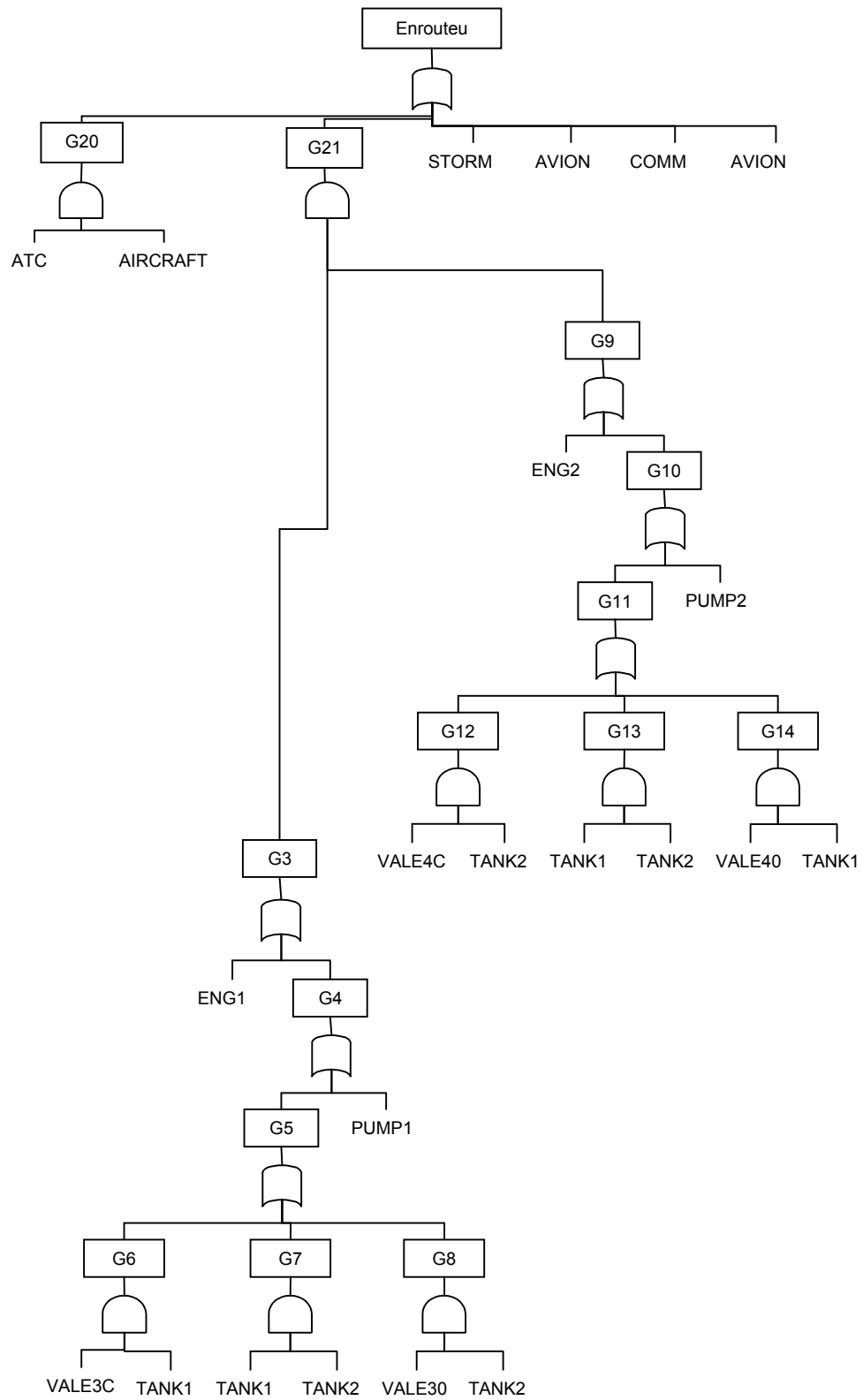
### Mission set 5

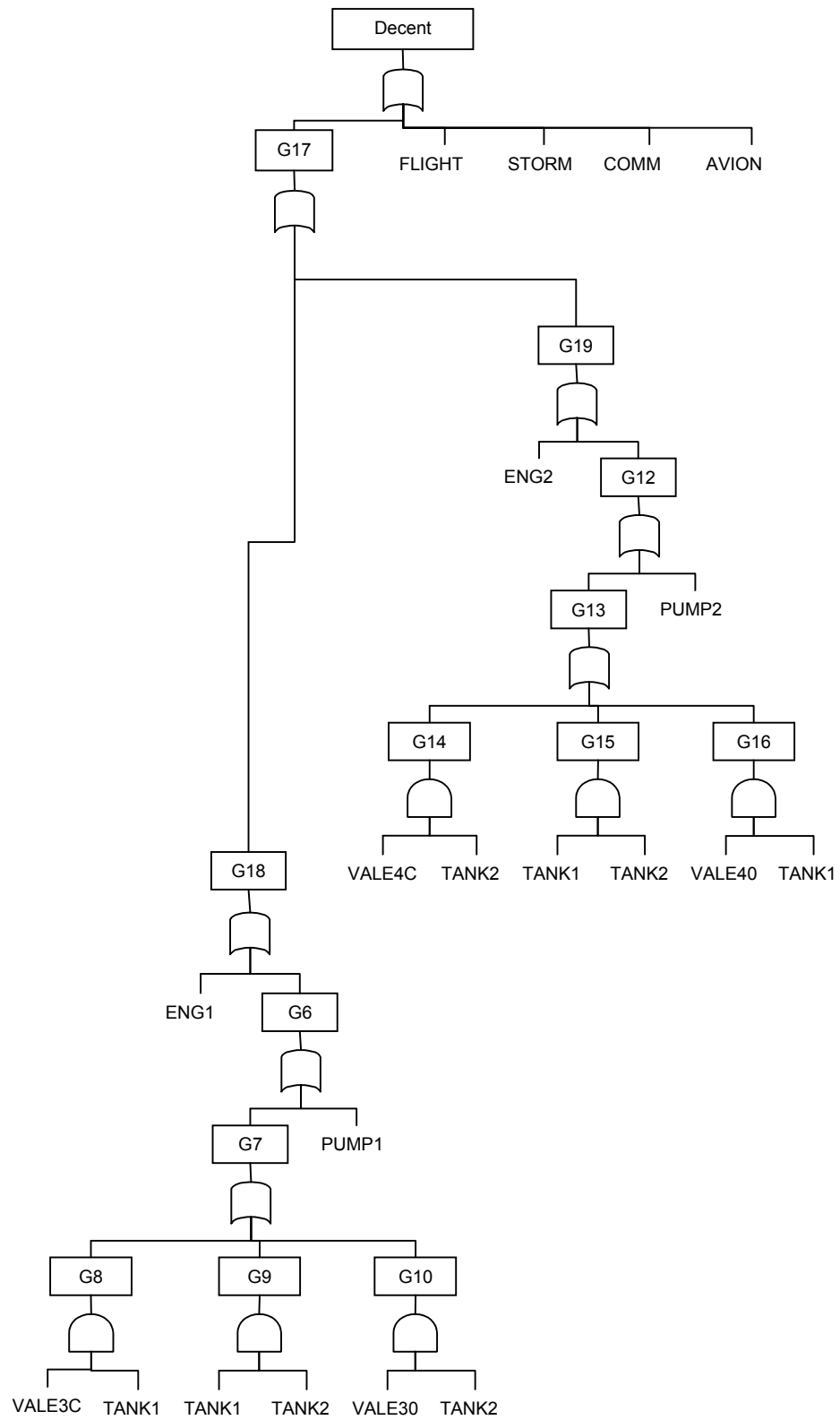
Possible phases Fault tree	No gate (OR,AND)	No events Including repeated events	No events not Including repeated events
1(takeoff)	17 (9,8)	25	19
2(Climb)	14 (8,6)	20	14
3(enrouteC)	15 (7,8)	22	16
4(enrouteU)	15 (7,8)	22	16
5(decent)	14 (8,6)	20	14
6(Land)	17 (9,8)	26	20



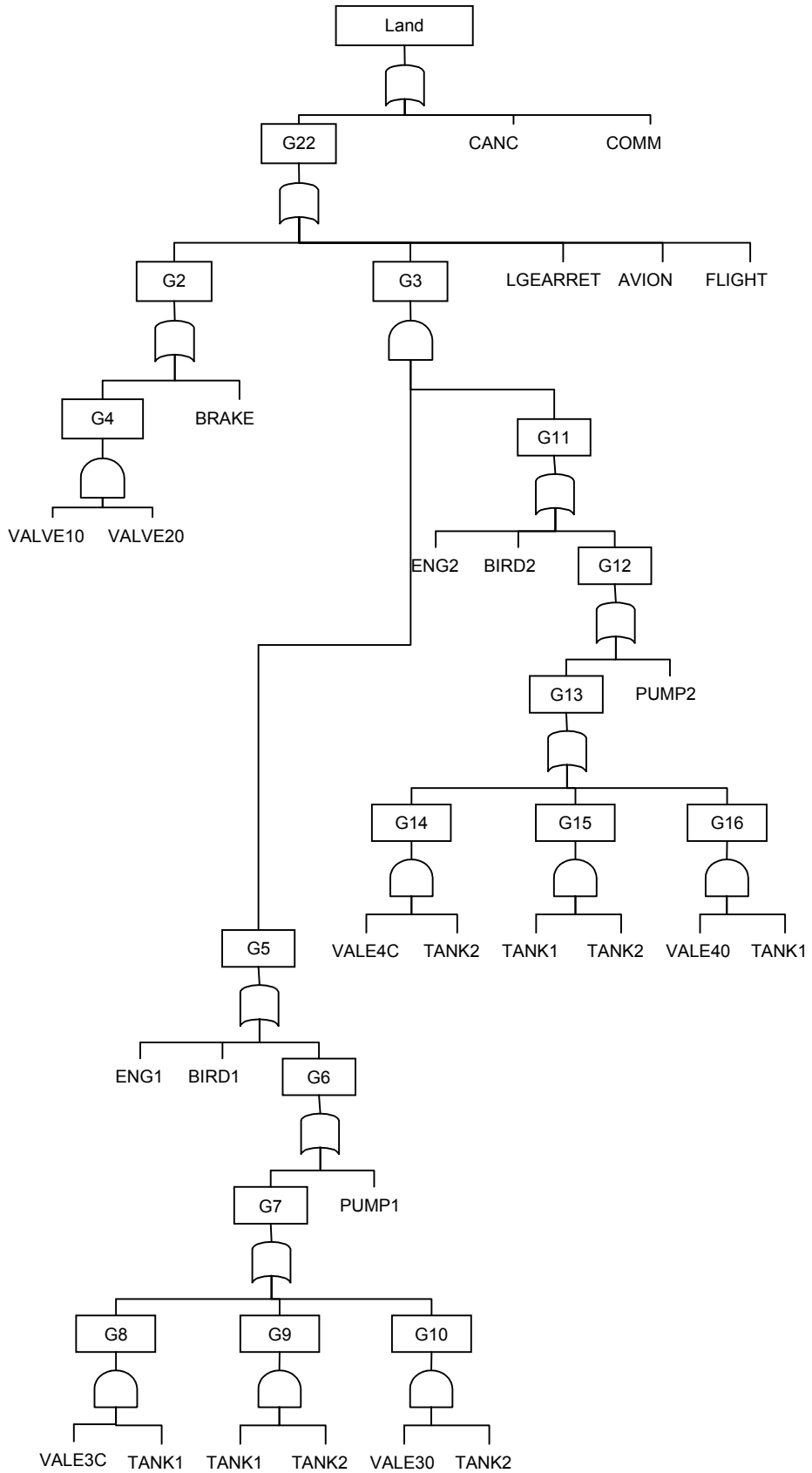












All the UAV fault trees have been constructed from the UAV subsystems which are shown in diagram with descriptions in appendix F.

### Mission small UAV mission

Possible phases Fault tree	No gate (OR,AND)	No events Including repeated events	No events not Including repeated events
1 SM_STARTUP	10(6,4)	9	9
2 SM_TAXIOUT	10(6,4)	9	9
3 SM_TAKEOFF	65(49,16)	92	77
4 SM_CLIMB	98(73,25)	161	117
5 SM_CRUSIE	76(60,16)	130	97
6 SM_DECENT	76(60,16)	130	97
7 SM_LAND	91(76,15)	121	95
8 SM_TAXIIN	10(6,4)	9	9
9 SM_SHUTDOWN	10(6,4)	9	9

SM\_STARTUP 1 1 1 PGATE5343 BATTDAM  
PGATE5343 1 2 0 ENG1FIRE ENG2FIRE

SM\_TAXIOUT 1 1 1 PGATE5373 BATTDAM  
PGATE5373 1 2 0 ENG1FIRE ENG2FIRE

SM\_TAKEOFF 1 2 0 PGATE485 PGATE487  
PGATE485 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE487 1 5 0 PGATE3 THRUST\_1OR2 ENG1FIRE ENG2FIRE NOTRETH  
PGATE3 1 2 0 AILERONS SLATS

SM\_CLIMB 1 2 0 PGATE490 PGATE491  
PGATE490 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE491 1 6 0 PGATE802 ENG1FIRE ENG2FIRE NOTRETH NOTUNDCARR THRUST\_1&2  
PGATE802 1 3 0 SLATS SPOILERS RUDDERS

SM\_CLIMB 1 2 0 PGATE490 PGATE491  
PGATE490 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE491 1 6 0 PGATE802 ENG1FIRE ENG2FIRE NOTRETH NOTUNDCARR THRUST\_1&2  
PGATE802 1 3 0 SLATS SPOILERS RUDDERS

SM\_DECENT 1 2 0 PGATE539 PGATE540  
PGATE539 1 0 2 BADWEATER BATTDAM  
PGATE540 1 5 0 PGATE5052 NOTUNDCARR THRUST\_1&2 ENG1FIRE ENG2FIRE  
PGATE5052 1 2 0 ELEVATORS SPOILERS

SM\_LAND 1 2 0 PGATE506 PGATE507  
PGATE506 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE507 1 6 1 PGATE511 ENG1FIRE ENG2FIRE UNDERCARRIAGEF1 THRUST\_1OR2 BRAKING\_F  
RE\_THRUST\_F  
PGATE511 1 3 0 ELEVATORS AILERONS RUDDERS

SM\_TAXIIN 1 1 1 PGATE5133 BATTDAM

PGATE5133 1 2 0 ENG1FIRE ENG2FIRE

SM\_SHUTDOWN 1 1 1 PGATE5163 BATTDAM  
PGATE5163 1 2 0 ENG1FIRE ENG2FIRE

ELEVATORS 1 1 1 HTOP1 ELEVATOR  
HTOP1 2 2 0 HGATE1 HGATE2  
HGATE1 1 1 1 HY\_LIQ\_LE\_ACT1 EL\_ACT1  
HY\_LIQ\_LE\_ACT1 1 2 0 HGATE68 HYDRAULICFAIL  
HGATE68 1 1 2 DCPOWER ServoE1 POSTION\_SENSOR  
HGATE2 1 1 1 HY\_LIQ\_LE\_ACT2 EL\_ACT2  
HY\_LIQ\_LE\_ACT2 1 2 0 HGATE75 HYDRAULICFAIL  
HGATE75 1 1 2 DCPOWER ServoE2 POSTION\_SENSOR

AILERONS 1 1 1 HGATE601 AILERON  
HGATE601 2 2 0 HGATE602 HGATE603  
HGATE602 1 1 1 HY\_LIQ\_LA\_ACT1 AI\_ACT1  
HY\_LIQ\_LA\_ACT1 1 2 0 HGATE770 HYDRAULICFAIL  
HGATE770 1 1 2 DCPOWER ServoAI1 POSTION\_SENSOR\_LA  
HGATE603 1 1 1 HY\_LIQ\_LA\_ACT2 AI\_ACT2  
HY\_LIQ\_LA\_ACT2 1 2 0 HGATE775 HYDRAULICFAIL  
HGATE775 1 1 2 DCPOWER ServoAI2 POSTION\_SENSOR\_LA

SLATS 1 1 1 SHTOP1 SLAT  
SHTOP1 2 2 0 SHGATE1 SHGATE2  
SHGATE1 1 1 1 HY\_LIQ\_S\_ACT1 S\_ACT1  
HY\_LIQ\_S\_ACT1 1 2 0 SHGATE68 HYDRAULICFAIL  
SHGATE68 1 1 2 DCPOWER ServoS1 SPOSTION\_SENSOR  
SHGATE2 1 1 1 HY\_LIQ\_S\_ACT2 S\_ACT2  
HY\_LIQ\_S\_ACT2 1 2 0 SHGATE75 HYDRAULICFAIL  
SHGATE75 1 1 2 DCPOWER ServoS2 SPOSTION\_SENSOR

SLATS 1 1 1 SHTOP1 SLAT  
SHTOP1 2 2 0 SHGATE1 SHGATE2  
SHGATE1 1 1 1 HY\_LIQ\_S\_ACT1 S\_ACT1  
HY\_LIQ\_S\_ACT1 1 2 0 SHGATE68 HYDRAULICFAIL  
SHGATE68 1 1 2 DCPOWER ServoS1 SPOSTION\_SENSOR  
SHGATE2 1 1 1 HY\_LIQ\_S\_ACT2 S\_ACT2  
HY\_LIQ\_S\_ACT2 1 2 0 SHGATE75 HYDRAULICFAIL  
SHGATE75 1 1 2 DCPOWER ServoS2 SPOSTION\_SENSOR

SPOILERS 1 1 1 SPHTOP1 SPOILER  
SPHTOP1 2 2 0 SPHGATE1 SPHGATE2  
SPHGATE1 1 1 1 HY\_LIQ\_SP\_ACT1 SP\_ACT1  
HY\_LIQ\_SP\_ACT1 1 2 0 SPHGATE68 HYDRAULICFAIL  
SPHGATE68 1 1 2 DCPOWER ServoSP1 SPPOSTION\_SENSOR  
SPHGATE2 1 1 1 HY\_LIQ\_SP\_ACT2 SP\_ACT2  
HY\_LIQ\_SP\_ACT2 1 2 0 SPHGATE75 HYDRAULICFAIL  
SPHGATE75 1 1 2 DCPOWER ServoSP2 SPPOSTION\_SENSOR

RUDDERS 1 1 1 HGATE621 RUDDER  
HGATE621 2 2 0 HGATE622 HGATE623  
HGATE622 1 1 1 HY\_LIQ\_RU\_ACT1 R\_ACT1  
HY\_LIQ\_RU\_ACT1 1 2 0 HGATE787 HYDRAULICFAIL  
HGATE787 1 1 2 DCPOWER ServoR1 POSTION\_SENSOR\_R  
HGATE623 1 1 1 HY\_LIQ\_RU\_ACT2 R\_ACT2  
HY\_LIQ\_RU\_ACT2 1 2 0 HGATE792 HYDRAULICFAIL  
HGATE792 1 1 2 DCPOWER ServoR2 POSTION\_SENSOR\_R

UNDERCARRIAGEF1 1 1 1 FOS\_UNLOCK1 UNCLOC1UNLOC  
FOS\_UNLOCK1 2 2 0 HGATE336 HY\_LIQ\_UNLOC1\_A1  
HGATE336 1 0 3 LA1L1STUCK LA1L1LEAK LA1L1RUPT  
HY\_LIQ\_UNLOC1\_A1 1 2 0 HGATE398 HGATE685  
HGATE398 2 2 0 UNSEQVALF1 UNSEQVALF2  
UNSEQVALF1 1 2 1 DCPOWER HYDRAULICFAIL LSEQ1VF  
UNSEQVALF2 1 2 1 DCPOWER HYDRAULICFAIL LSEQ2VF  
HGATE685 1 1 1 DCPOWER LSEV1L1P1  
HGATE334 1 2 0 HGATE338 HY\_LIQ\_UNLOC1\_A2  
HGATE338 1 0 3 LA2L1STUCK LA2L1LEAK LA2L1RUPT

HY\_LIQ\_UNLOC1\_A2 1 2 0 HGATE398 HGATE686  
HGATE686 1 1 1 DCPower LSEV2L1P1

BRAKING\_F 1 2 1 HGATE720 HY\_LIQ\_BRS L\_WHEEL\_F  
HGATE720 2 0 2 L\_BKE\_1F L\_BKE\_2F  
HY\_LIQ\_BRS 2 2 0 HGATE722 HGATE723  
HGATE722 1 2 0 LEFTANTI-SKID HY\_LIQ\_FR\_BCV1  
LEFTANTI-SKID 1 3 0 DCPower HGATE724 SIG\_F\_ANTSS  
HGATE724 1 0 2 BRLSVOMD BRLSVOLECK  
SIG\_F\_ANTSS 1 2 0 HGATE732 HGATE733  
HGATE732 2 2 0 HGATE734 HGATE735  
HGATE734 1 0 2 BRASU1NO BRASU1WR  
HGATE735 1 0 2 BRASU2NO BRASU2WR  
HGATE733 2 2 0 HGATE736 HGATE737  
HGATE736 1 0 2 BRTRS1NO BRTRS1WR  
HGATE737 1 0 2 BRTRS2NO BRTRS2WR  
HY\_LIQ\_FR\_BCV1 1 3 0 HGATE738 DCPower HYDRAULICFAIL  
HGATE738 1 0 2 BRBCV1MD BRBCV1LEAK  
HGATE723 1 2 0 RIGHTANTI-SKID HY\_LIQ\_FR\_BCV2  
RIGHTANTI-SKID 1 3 0 DCPower HGATE729 SIG\_F\_ANTSS  
HGATE729 1 0 2 BRRSVOMD BRRSVOLECK  
HY\_LIQ\_FR\_BCV2 1 3 0 HGATE739 DCPower HYDRAULICFAIL  
HGATE739 1 0 2 BRBCV2MD BRBCV2LEAK

RE\_THRUST\_F 1 2 0 FGATE114 FGATE115  
FGATE114 1 2 0 L\_RTDOOR THRUSTL  
L\_RTDOOR 1 1 1 FGATE116 L\_RTDOORS  
FGATE116 2 2 0 FGATE117 FGATE118  
FGATE117 1 2 0 FGATE120 HY\_LIQ\_LCDR\_ACT1  
FGATE120 1 0 3 HA1LDRSTUCK HA1LDRLECK HA1LDRRUPT  
HY\_LIQ\_LCDR\_ACT1 1 2 0 FGATE126 HYDRAULICFAIL  
FGATE126 1 1 1 DCPower RTSEVLDR\_1  
FGATE118 1 2 0 FGATE122 HY\_LIQ\_LCDR\_ACT2  
FGATE122 1 0 3 HA2LDRSTUCK HA2LDRLECK HA2LDRRUPT  
HY\_LIQ\_LCDR\_ACT2 1 2 0 FGATE129 HYDRAULICFAIL  
FGATE129 1 1 1 DCPower RTSEVLDR\_2  
FGATE115 1 1 1 R\_RTDOOR THRUSTRR  
R\_RTDOOR 1 1 1 FGATE132 R\_RTDOORS  
FGATE132 2 2 0 FGATE133 FGATE134  
FGATE133 1 2 0 FGATE136 HY\_LIQ\_RCDR\_ACT1  
FGATE136 1 0 3 HA1RDRSTUCK HA1RDRLECK HA1RDRRUPT  
HY\_LIQ\_RCDR\_ACT1 1 2 0 FGATE141 HYDRAULICFAIL  
FGATE141 1 1 1 DCPower RTSEVRDR\_1

FGATE134 1 2 0 FGATE138 HY\_LIQ\_RCDR\_ACT2 FGATE138 1 0 3 HA2RDRSTUCK HA2RDRLECK HA2RDRRUPT  
HY\_LIQ\_RCDR\_ACT2 1 2 0 FGATE143 HYDRAULICFAIL  
FGATE143 1 1 1 DCPower RTSEVRDR\_2

ENG1FIRE 2 1 1 FUELFLOWHIL NOTCLENFLOW  
FUELFLOWHIL 1 1 1 FGATE171 FLACECHI  
FGATE171 2 1 1 FGATE173 Fevent172  
FGATE173 1 0 2 FLACPUHI FRACPUHI

ENG2FIRE 2 1 1 FUELFLOWHIR NOTCRENGFLOW  
FUELFLOWHIR 1 1 1 FGATE174 FRACECHI  
FGATE174 2 1 1 FGATE173\_2 Fevent175  
FGATE173\_2 1 0 2 FLACPUHI\_2 FRACPUHI\_2

NOTRETH 1 2 0 FGATE145 FGATE146  
FGATE145 1 2 1 RETRLACTS FGATE155 RTLCSTUCK  
RETRLACTS 1 3 0 FGATE149 FGATE150 FGATE151  
FGATE149 2 0 2 RETHLACT1ST RETHLACT2ST  
FGATE150 2 0 2 RETHLACT1ST RETHLACT3ST  
FGATE151 2 0 2 RETHLACT2ST RETHLACT3ST  
FGATE155 1 2 0 FGATE156 FGATE157  
FGATE156 2 0 3 RETHSIGNL&BU1 RETHSIGNL&BU2 RETHSIGNL&BU3  
FGATE157 2 0 3 RETHSIGNFMS1 RETHSIGNFMS2 RETHSIGNFMS3  
FGATE146 1 2 1 RETRRACTS FGATE155 RTRCSTUCK  
RETRRACTS 1 3 0 FGATE152 FGATE153 FGATE154  
FGATE152 2 0 2 RETHRACT1ST RETHRACT2ST  
FGATE153 2 0 2 RETHRACT1ST RETHRACT3ST  
FGATE154 2 0 2 RETHRACT2ST RETHRACT3ST

NOTUNDCARR 1 2 0 HGATE741 HGATE742

HGATE741 1 2 0 HGATE743 HGATE744  
 HGATE743 2 0 3 NOTUNDSIGL&B1 NOTUNDSIGL&B2 NOTUNDSIGL&B3  
 HGATE744 2 0 3 NOTUNDSIGFMS1 NOTUNDSIGFMS2 NOTUNDSIGFMS3  
 HGATE742 2 4 0 NOTUNLOCK1 NOTDOORDN NOTUNLOCK3 NOTWHEELDN  
 NOTUNLOCK1 1 1 1 HGATE745 NOTUNDUNLOCK1  
 HGATE745 2 3 0 HGATE746 HGATE747 HGATE748  
 HGATE746 1 0 4 NOTUNDA1LOC1 NOTUNDA2LOC1 NOTUNDS1LOC1 NOTUNDS2LOC1  
 HGATE747 1 0 4 NOTUNDA1LOC1 NOTUNDA3LOC1 NOTUNDS1LOC1 NOTUNDS3LOC1  
 HGATE748 1 0 4 NOTUNDA2LOC1 NOTUNDA3LOC1 NOTUNDS2LOC1 NOTUNDS3LOC1  
 NOTDOORDN 1 1 1 HGATE753 NOTUNDDOOR  
 HGATE753 2 3 0 HGATE754 HGATE755 HGATE756  
 HGATE754 1 0 4 NOTUNDA1DOOR NOTUNDA2DOOR NOTUNDS1DOOR NOTUNDS2DOOR  
 HGATE755 1 0 4 NOTUNDA1DOOR NOTUNDA3DOOR NOTUNDS1DOOR NOTUNDS3DOOR  
 HGATE756 1 0 4 NOTUNDA2DOOR NOTUNDA3DOOR NOTUNDS2DOOR NOTUNDS3DOOR  
 NOTUNLOCK3 1 1 1 HGATE749 NOTUNDUNLOCK3  
 HGATE749 2 3 0 HGATE750 HGATE751 HGATE752  
 HGATE750 1 0 4 NOTUNDA1LOC3 NOTUNDA2LOC3 NOTUNDS1LOC3 NOTUNDS2LOC3  
 HGATE751 1 0 4 NOTUNDA1LOC3 NOTUNDA3LOC3 NOTUNDS1LOC3 NOTUNDS3LOC3  
 HGATE752 1 0 4 NOTUNDA2LOC3 NOTUNDA3LOC3 NOTUNDS2LOC3 NOTUNDS3LOC3  
 NOTWHEELDN 1 1 1 HGATE757 NOTUNDWHEEL  
 HGATE757 2 3 0 HGATE758 HGATE759 HGATE760  
 HGATE758 1 0 4 NOTUNDA1WHEEL NOTUNDA2WHEEL NOTUNDS1WHEEL NOTUNDS2WHEEL  
 HGATE759 1 0 4 NOTUNDA1WHEEL NOTUNDA3WHEEL NOTUNDS1WHEEL NOTUNDS3WHEEL  
 HGATE760 1 0 4 NOTUNDA2WHEEL NOTUNDA3WHEEL NOTUNDS2WHEEL NOTUNDS3WHEEL

THRUST\_10R2 1 1 1 THRUSTL THRUSTR

THRUST\_1&2 121 1 THRUSTL THRUSTR  
 THRUSTL 1 4 0 DCPOWER ENGINEL FUELENGL ACPOWER  
 ENGINEL 1 2 0 FGATE80 FGATE81  
 FGATE80 1 0 6 FENG1COM FENG1CH FENG1TUR FENG1EX FENG1SHA FENG1FAN  
 FGATE81 1 1 2 TOLEFTHP FLSPNOZ FLHPCOCK  
 TOLEFTHP 1 1 1 FGATE108 FLVALF  
 FGATE108 1 0 4 FLACECPX FLACECPIN FLACECPSLOW FLACECPNO  
 FUELENGL 1 1 1 FGATE37 FLLPCCLOSE  
 FGATE37 1 2 0 FGATE38 LL\_FUEL  
 FGATE38 1 1 1 LTEMSENSERS FLHEATF  
 LTEMSENSERS 2 2 0 FGATE40 FGATE41  
 FGATE40 1 0 2 FLTEMSEN1NO FLTEMSEN1WR  
 FGATE41 1 0 2 FLTEMSEN2NO FLTEMSEN2WR  
 LL\_FUEL 1 1 2 FGATE54 FPIP1 FLNRVF  
 FGATE54 1 2 0 FGATE55 FROM\_LTANK  
 FGATE55 1 0 4 FLACPUNO FLACPUSLOW FLACPUSIND FLACPUSEXD  
 FROM\_LTANK 1 2 0 FGATE58 L\_AIRPRESS  
 FGATE58 1 0 3 FLTANMD FLTANRUP FLTANLEAK  
 L\_AIRPRESS 1 2 0 FGATE60 FGATE61  
 FGATE60 2 2 0 FGATE63 FGATE64  
 FGATE63 1 0 2 FLAPS1NO FLAPS1WR  
 FGATE64 1 0 2 FLAPS2NO FLAPS2WR  
 FGATE61 1 0 2 FLVENTIN FLVENTOUT

## Mission 7

### Mission medium UAV mission

Possible phases Fault tree	No gate (OR,AND)	No events Including repeated events	No events not Including repeated events
1 SM_STARTUP	10(6,4)	9	9
2 SM_TAXIOUT	10(6,4)	9	9
3 SM_TAKEOFF	222(144,48)	298	254
4 SM_CLIMB	225(168,57)	363	294
5 SM_CRUSIE	203(155,48)	336	274
6 SM_DECENT	203(155,48)	336	274
7 SM_LAND	218(171,47)	312	272
8 SM_TAXIIN	10(6,4)	9	9
9 SM_SHUTDOWN	10(6,4)	9	9

HYDRAULICFAIL 2 3 0 HYSUBSYS1 HYSUBSYS2 HYSUBSYS3  
 HYSUBSYS1 1 3 0 HGATE17 HGATE18 HY1PUMPS  
 HGATE17 1 0 3 H1PIP1 H1PRVOPEN H1NRV  
 HGATE18 1 2 0 HGATE20 HGATE25  
 HGATE20 1 2 0 HGATE23 LOW\_TO\_RES1  
 HGATE23 1 0 4 H1RLEAK H1RMECH H1RAERO H1RLIQ  
 LOW\_TO\_RES1 1 1 1 HGATE127 H1REPI  
 HGATE127 1 2 1 HGATE28 HGATE128 HY1HEEXV  
 HGATE28 1 0 5 H1HBLOCK H1HNOFLUID H1HNOPRESS H1HINT H1HLEV  
 HGATE128 2 2 0 HGATE129 HGATE130  
 HGATE129 1 0 2 HY1ITEMS1NO HY1ITEMS1WR  
 HGATE130 1 0 2 HY1ITEMS2NO HY1ITEMS2WR  
 HGATE25 2 2 0 HGATE19 H1BYPASS  
 HGATE19 1 0 2 H1FOB H1FGAP  
 H1BYPASS 1 1 1 HGATE120 HY1BYPASSBLO  
 HGATE120 1 1 1 HGATE121 HY1BYVAL  
 HGATE121 2 2 0 HGATE123 HGATE124  
 HGATE123 1 0 2 HY1FBSENNO1 HY1FBSENWR1  
 HGATE124 1 0 2 HY1FBSENNO2 HY1FBSENWR2  
 HY1PUMPS 2 3 0 HGATE21 HGATE27 H1ACELPUMP  
 HGATE21 1 2 0 HGATE16 H1MECPUMPCON  
 HGATE16 1 0 5 H1MPUMP H1MPLOSPEED H1MPINT H1MPEXT H1MPBLOCK  
 H1MECPUMPCON 1 3 0 HY\_UMS\_AVIONIC HGATE22 HGATE141  
 HY\_UMS\_AVIONIC 1 0 4 HYDR\_UNIT FMS UMS DATA\_BUS  
 HGATE22 1 0 2 H1LGBXNO H1LGBXLOW  
 HGATE141 2 2 0 HGATE109 HGATE110  
 HGATE109 1 0 2 H1PSENNO1 H1PSENWR1  
 HGATE110 1 0 2 H1PSENNO2 H1PSENWR2  
 HGATE27 1 0 4 H1ALOPRESS H1AINT H1ADISCH H1ASUPPLY  
 H1ACELPUMP 1 2 0 HGATE113 H1ACPUMPCON  
 HGATE113 1 0 5 H1ACPPUMP H1ACPLOSPEED H1ACPINT H1ACPEXT H1ACPBLOCK  
 H1ACPUMPCON 1 2 0 HY\_UMS\_AVIONIC HGATE141  
 HYSUBSYS2 1 3 0 HGATE31 HGATE32 HY2PUMPS  
 HGATE31 1 0 3 H2PIP1 H2PRVOPEN H2NRV

HGATE32 1 2 0 HGATE34 HGATE35  
 HGATE34 1 2 0 HGATE184 LOW\_TO\_RES2  
 HGATE184 1 0 4 H2RLEAK H2RMECH H2RAERO H2RLIQ  
 LOW\_TO\_RES2 1 1 1 HGATE186 H2REPIP  
 HGATE186 1 3 1 HGATE187 HY\_UMS\_AVIONIC HGATE188 HY2HEEXV  
 HGATE187 1 0 5 H2HBLOCK H2HNOFLUID H2HNOPRESS H2HINT H2HLEV  
 HGATE188 2 2 0 HGATE190 HGATE191  
 HGATE190 1 0 2 HY2TEMS1NO HY2TEMS1WR  
 HGATE191 1 0 2 HY2TEMS2NO HY2TEMS2WR  
 HGATE35 2 2 0 HGATE192 H2BYPASS  
 HGATE192 1 0 2 H2FOB H2FGAP  
 H2BYPASS 1 1 1 HGATE194 HY2BYPASSBLO  
 HGATE194 1 2 1 HGATE195 HY\_UMS\_AVIONIC HY2BYVAL  
 HGATE195 2 2 0 HGATE196 HGATE197  
 HGATE196 1 0 2 HY2FBSENNO1 HY2FBSENWR1  
 HGATE197 1 0 2 HY2FBSENNO2 HY2FBSENWR2  
 HY2PUMPS 2 3 0 HGATE37 H2ACELPUMP1 H2ACELPUMP2  
 HGATE37 1 0 4 H2ALOPRESS H2AINT H2ADISCH H2ASUPLY  
 H2ACELPUMP1 1 2 0 HGATE168 H2ACPUMP1CON  
 HGATE168 1 0 5 H2ACP1PUMP H2ACP1LOSPEED H2ACP1INT H2ACP1EXT H2ACP1BLOCK  
 H2ACPUMP1CON 1 2 0 HY\_UMS\_AVIONIC HGATE173  
 HGATE173 2 2 0 HGATE174 HGATE175  
 HGATE174 1 0 2 H2PSENNO1 H2PSENWR1  
 HGATE175 1 0 2 H2PSENNO2 H2PSENWR2  
 H2ACELPUMP1 1 2 0 HGATE168 H2ACPUMP1CON  
 HGATE168 1 0 5 H2ACP1PUMP H2ACP1LOSPEED H2ACP1INT H2ACP1EXT H2ACP1BLOCK  
 H2ACPUMP1CON 1 2 0 HY\_UMS\_AVIONIC HGATE173  
 HGATE173 2 2 0 HGATE174 HGATE175  
 HGATE174 1 0 2 H2PSENNO1 H2PSENWR1  
 HGATE175 1 0 2 H2PSENNO2 H2PSENWR2  
 H2ACELPUMP2 1 2 0 HGATE177 H2ACPUMP2CON  
 HGATE177 1 0 5 H2ACP2PUMP H2ACP2LOSPEED H2ACP2INT H2ACP2EXT H2ACP2BLOCK  
 H2ACPUMP2CON 1 2 0 HY\_UMS\_AVIONIC HGATE173  
 HYSUBSYS3 1 3 0 HGATE45 HGATE46 HY3PUMPS  
 HGATE45 1 0 3 H3PIP1 H3PRVOPEN H3NRV  
 HGATE46 1 2 0 HGATE48 HGATE49  
 HGATE48 1 2 0 HGATE50 LOW\_TO\_RES3  
 HGATE50 1 0 4 H3RLEAK H3RMECH H3RAERO H3RLIQ  
 LOW\_TO\_RES3 1 1 1 HGATE162 H3REPIP  
 HGATE162 1 3 1 HGATE52 HY\_UMS\_AVIONIC HGATE163 HY3HEEXV  
 HGATE52 1 0 5 H3HBLOCK H3HNOFLUID H3HNOPRESS H3HINT H3HLEV  
 HGATE163 2 2 0 HGATE164 HGATE165  
 HGATE164 1 0 2 HY3TEMS1NO HY3TEMS1WR  
 HGATE165 1 0 2 HY3TEMS2NO HY3TEMS2WR  
 HGATE49 2 2 0 HGATE53 H3BYPASS  
 HGATE53 1 0 2 H3FOB H3FGAP  
 H3BYPASS 1 1 1 HGATE156 HY3BYPASSBLO  
 HGATE156 1 2 1 HGATE157 HY\_UMS\_AVIONIC HY3BYVAL  
 HGATE157 2 2 0 HGATE158 HGATE159  
 HGATE158 1 0 2 HY3FBSENNO1 HY3FBSENWR1  
 HGATE159 1 0 2 HY3FBSENNO2 HY3FBSENWR2  
 HY3PUMPS 2 3 0 HGATE55 HGATE132 HGATE146  
 HGATE55 1 0 4 H3ALOPRESS H3AINT H3ADISCH H3ASUPLY  
 HGATE132 1 2 0 HGATE56 H3MECPUMP1CON  
 HGATE56 1 0 5 H3MPUMP H3MPLOSPEED H3MPINT H3MPEXT H3MPBLOCK  
 H3MECPUMP1CON 1 3 0 HY\_UMS\_AVIONIC HGATE136 HGATE143  
 HGATE136 1 0 2 H1RGBOXNO H1RGBOXLOW  
 HGATE143 2 2 0 HGATE144 HGATE145  
 HGATE144 1 0 2 H3PSENNO1 H3PSENWR1  
 HGATE145 1 0 2 H3PSENNO2 H3PSENWR2  
 HGATE146 1 2 0 HGATE147 H3ACPUMPCON  
 HGATE147 1 0 5 H3ACPUMP H3PACLOSPEED H3ACPINT H3ACPEXT H3ACPBLOCK  
 H3ACPUMPCON 1 2 0 HY\_UMS\_AVIONIC HGATE143

DCPower 1 2 0 GATEDC2 GATEDC3  
 GATEDC2 2 3 0 GATEDC4 GATEDC5 GATEDC17  
 GATEDC17 1 3 3 GATEDC18 GATEDC19 GATEDC20 DC10 DC11 DC12  
 GATEDC18 2 0 3 DC1 DC2 DC3  
 GATEDC19 2 0 3 DC4 DC5 DC6  
 GATEDC20 2 0 3 DC7 DC8 DC9  
 GATEDC4 2 0 3 DC1 DC2 DC3  
 GATEDC5 1 0 3 DC4 DC5 DC6  
 GATEDC3 2 2 0 GATEDC6 GATEDC7  
 GATEDC6 2 2 0 GATEDC8 GATEDC9

GATEDC8 1 1 3 GATEDC13 DC4 DC7 DC8  
 GATEDC13 2 3 3 GATEDC14 GATEDC15 GATEDC16 DC10 DC11 DC12  
 GATEDC14 1 0 3 DC1 DC2 DC3  
 GATEDC15 1 0 3 DC4 DC5 DC6  
 GATEDC16 1 0 3 DC7 DC8 DC9  
 GATEDC9 1 0 3 DC3 DC9 DC10  
 GATEDC7 2 2 2 GATEDC10 GATEDC11 DC12 DC13  
 GATEDC10 1 0 2 DC11 DC12  
 GATEDC11 1 0 3 DC1 DC11 DC4

ACPOWER 1 2 0 GATEAC2 GATEAC3  
 GATEAC2 2 2 0 GATEAC4 GATEAC5  
 GATEAC4 2 0 3 AC1 AC2 AC3  
 GATEAC5 1 0 3 AC4 AC5 AC6  
 GATEAC3 2 2 0 GATEAC6 GATEAC7  
 GATEAC6 2 2 0 GATEAC8 GATEAC9  
 GATEAC8 1 0 3 AC4 AC7 AC8  
 GATEAC9 1 0 3 AC3 AC9 AC10  
 GATEAC7 2 2 2 GATEAC10 GATEAC11 AC12 AC13  
 GATEAC10 1 0 2 AC11 AC12  
 GATEAC11 1 0 3 AC1 AC11 AC4

## Mission 8

### Mission Large UAV mission

Possible phases Fault tree	No gate (OR,AND)	No events Including repeated events	No events not Including repeated events
1 SM_STARTUP	10(6,4)	9	9
2 SM_TAXIOUT	10(6,4)	9	9
3 SM_TAKEOFF	277(192,55)	387	356
4 SM_CLIMB	280(216,64)	452	396
5 SM_CRUSIE	258(203,55)	425	376
6 SM_DECENT	258(203,55)	425	376
7 SM_LAND	273(219,54)	401	374
8 SM_TAXIIN	10(6,4)	9	9
9 SM_SHUTDOWN	10(6,4)	9	9

DCPOWER 2 3 0 LEFT\_FCDC CEN\_FCDC RIGHT\_FCDC  
 LEFT\_FCDC 1 3 0 EGATE4 LEFT\_PWRS EGATE93  
 EGATE4 1 0 4 ELLBUSNO ELLBUSLOW ELLBUSMD ELLBUSOH  
 LEFT\_PWRS 2 4 0 LVDC\_BUS L\_PMG HOT\_BBUS CEN\_PWRS2  
 LVDC\_BUS 1 2 0 EGATE9 EGATE10  
 EGATE9 1 0 4 ELLVDCNO ELLVDCLOW ELLVDCMD ELLVDCOV  
 EGATE10 2 2 0 L\_TRU RVDC\_BUS2



L\_TRU 1 1 1 EGATE13 EGATE14  
 EGATE13 1 0 4 ELLTRUNO ELLTRUMD ELLTRUWR ELLTRUOV  
 EGATE14 2 2 0 LAC\_XTRBUS GEN\_RAT  
 LAC\_XTRBUS 1 2 0 EGATE17 EGATE18  
 EGATE17 1 0 4 ELLACXTRBNO ELLACXTRBLOW ELLACXTRBMD ELLACXTRBOV  
 EGATE18 2 4 0 L\_LGEN L\_APUGEN L\_RGEN L\_BKUPGEN  
 L\_LGEN 1 2 0 EGATE24 EGATE25  
 EGATE24 1 0 3 EL\_AVIONIC CPU ELLGCBO  
 EGATE25 1 0 5 ELL\_MGENMD ELL\_MGENNO ELL\_MGENLOW ELL\_MGENPU ELL\_MGENOV  
 L\_APUGEN 1 3 0 EGATE26 EGATE27 EGATE47  
 EGATE26 1 0 3 EL\_AVIONIC CPU ELAPBOPEN  
 EGATE27 1 0 5 ELAPUMD ELAPUNO ELAPULOW ELAPUPU ELAPUOV  
 EGATE47 1 0 3 EL\_AVIONIC CPU ELLBTBOPEN  
 L\_RGEN 1 4 0 EGATE28 EGATE29 EGATE48 EGATE49  
 EGATE28 1 0 3 EL\_AVIONIC CPU EL\_RGCBOPEN  
 EGATE29 1 0 5 ELRMGENMD ELRMGENNO ELRMGENLOW ELRMGENPU ELRMGENOV  
 EGATE48 1 0 3 EL\_AVIONIC CPU ELRBTBOPEN  
 EGATE49 1 0 3 EL\_AVIONIC CPU ELLBTBOPEN  
 L\_BKUPGEN 1 2 0 EGATE30 EGATE31  
 EGATE30 1 0 3 EL\_AVIONIC CPU ELLBUSTCOPEN  
 EGATE31 1 1 1 EGATE33 ELVSCFFAIL  
 EGATE33 2 2 0 EGATE34 EGATE35  
 EGATE34 1 0 5 ELLBUGENMD ELLBUGENNO ELLBUGENLOW ELLBUGENPU ELLBUGENOV  
 EGATE35 1 0 5 ELRBUGENMD ELRBUGENNO ELRBUGENLOW ELRBUGENPU ELRBUGENOV  
 GEN\_RAT 1 0 6 EL\_RATOV EL\_RATNO EL\_RATMD EL\_RATNOE EL\_RATLOW EL\_RATPU  
 RVDC\_BUS2 1 2 0 EGATE55 EGATE56  
 EGATE55 1 0 4 ELRVDCBUSNO ELRVDCBUSLOW ELRVDCBUSMD ELRVDCBUSOV  
 EGATE56 1 2 0 EGATE57 EGATE58  
 EGATE57 2 2 0 GEN\_RAT RAC\_XTRBUS  
 RAC\_XTRBUS 1 2 0 EGATE79 EGATE36  
 EGATE79 2 6 0 RAC\_XTRBUS2 LAC\_XTRBUS2 R\_LGEN R\_BKUPGEN R\_APUGEN R\_RGEN  
 RAC\_XTRBUS2 1 4 0 EGATE36 EGATE37 EGATE30 EGATE46  
 EGATE37 2 4 0 R\_LGEN R\_BKUPGEN R\_APUGEN R\_RGEN  
 LAC\_XTRBUS2 1 4 0 EGATE120 EGATE30 EGATE46 EGATE17  
 EGATE120 2 4 0 L\_LGEN L\_APUGEN L\_RGEN L\_BKUPGEN  
 EGATE17 1 0 4 ELLACXTRBNO ELLACXTRBLOW ELLACXTRBMD ELLACXTRBOV  
 EGATE46 1 0 3 EL\_AVIONIC CPU ELRBUSTOPEN  
 R\_LGEN 1 4 0 EGATE25 EGATE24 EGATE48 EGATE49  
 R\_BKUPGEN 1 2 0 EGATE46 EGATE31  
 R\_APUGEN 1 3 0 EGATE26 EGATE48 EGATE27  
 R\_RGEN 1 2 0 EGATE29 EGATE28  
 EGATE36 1 0 4 ELRACXTRBNO ELRACXTRBLOW ELRACXTRBMD ELRACXTRBOV  
 EGATE58 1 0 4 ELTRU2NO ELTRU2MD ELTRU2CON ELTRU2OV  
 L\_PMG 1 0 5 ELLPMGMD ELLPMGNO ELLPMGLOW ELLPMGPU ELLPMGOV  
 HOT\_BBUS 1 2 0 EGATE64 EGATE65  
 EGATE64 1 0 4 ELHOTBATNO ELHOTBATLOW ELHOTBATMD ELHOTBATOV  
 EGATE65 2 2 0 BATTBUS MAINBATT  
 BATTBUS 1 0 4 ELBATBUSNO ELBATBUSLOW ELBATBUSMD ELBATBUSOV  
 MAINBATT 1 0 4 ELMANBATNO ELMANBATLOW ELMANBATMD ELMANBA  
 CEN\_PWRS2 2 3 0 L\_PMG HOT\_BBUS R\_PMG  
 R\_PMG 1 0 5 ELRPMGMD ELRPMGNO ELRPMGLOW ELRPMGPU ELRPMGOV  
 EGATE93 1 0 2 EL\_AVIONIC ELLCBROPEN  
 CEN\_FCDC 1 3 0 EGATE68 EGATE94 CEN\_PWRS  
 EGATE68 1 0 4 ELCBUSNO ELCBUSLOW ELCBUSMD ELCBUSOV  
 EGATE94 1 0 2 EL\_AVIONIC ELCCBROPEN  
 CEN\_PWRS 2 4 0 L\_PMG HOT\_BBUS R\_PMG LEFT\_PWRS2  
 R\_PMG 1 0 5 ELRPMGMD ELRPMGNO ELRPMGLOW ELRPMGPU ELRPMGOV  
 LEFT\_PWRS2 2 3 0 LVDC\_BUS L\_PMG HOT\_BBUS  
 RIGHT\_FCDC 1 3 0 RIGHT\_PWRS EGATE90 EGATE95  
 RIGHT\_PWRS 2 2 0 R\_PMG RVDC\_BUS  
 RVDC\_BUS 1 2 0 EGATE55 EGATE92  
 EGATE92 2 2 0 EGATE77 LVDC\_BUS2  
 EGATE77 1 2 0 EGATE58 EGATE112  
 EGATE112 2 2 0 RAC\_XTRBUS GEN\_RAT  
 LVDC\_BUS2 1 2 0 EGATE9 L\_TRU  
 EGATE90 1 0 4 ELRBUSNO ELRBUSLOW ELRBUSMD ELRBUSOV  
 EGATE95 1 0 2 EL\_AVIONIC ELRCBROPE

## Mission 9

### Full scale UAV mission

C\_STARTUP 1 1 1 PGATE534 BATTDAM  
M\_STARTUP 1 3 0 PGATE539 PGATE540 ICE  
C\_TAXIOUT 1 1 1 PGATE537 BATTDAM  
M\_TAXIOUT 1 3 0 PGATE542 PGATE543 ICE  
C\_TAKEOFF 1 3 0 PGATE485 PGATE487 PGATE486  
M\_TAKEOFF 1 3 0 PGATE518 PGATE519 PGATE520  
C\_CLIMB 1 3 0 PGATE490 PGATE491 PGATE492  
M\_CLIMB 1 3 0 PGATE521 PGATE522 PGATE523  
C\_CRUISE 1 3 0 PGATE495 PGATE496 PGATE497  
M\_CRUISE 1 3 0 PGATE524 PGATE525 PGATE526  
C\_DECENT 1 3 0 PGATE500 PGATE501 PGATE502  
M\_DECENT 1 3 0 PGATE527 PGATE528 PGATE529  
C\_LAND 1 3 0 PGATE506 PGATE507 PGATE508  
M\_LAND 1 3 0 PGATE530 PGATE531 PGATE532  
C\_TAXIIN 1 1 1 PGATE513 BATTDAM  
M\_TAXIIN 1 3 0 PGATE545 PGATE546 ICE  
C\_SHUTDOWN 1 1 1 PGATE516 BATTDAM  
M\_SHUTDOWN 1 2 0 PGATE548 PGATE549  
PGATE3 1 5 0 AILERONS ELEVATORS RUDDER FLAPS SLATS  
BRAKEMER 2 1 1 BRAKING\_F OBJRUNWAY  
PGATE9 1 3 0 ELEVATORS AILERONS RUDDER  
PGATE17 1 3 0 ELEVATORS AILERONS RUDDER  
PGATE65 1 2 0 SLATS FLAPS  
PGATE72 1 5 0 ELEVATORS AILERONS RUDDER FLAPS SLATS  
ENG1FIRE 2 2 0 FUELFLOWHIL NOTCRENGFLOW  
ENG2FIRE 2 2 0 FUELFLOWHIR NOTCRENGFLOW  
PGATE485 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE486 1 3 0 BRAKEMER ICE COLLISION  
PGATE487 1 5 0 PGATE3 THRUST\_1OR2 ENG1FIRE ENG2FIRE NOTRETH  
AILERONS 1 2 0 L\_AILERON R\_AILERON  
ELEVATORS 1 2 0 L\_ELEVATOR R\_ELEVATOR  
RUDDER 1 1 1 HGATE621 RUDDER  
FLAPS 1 2 0 L\_FLAP R\_FLAP  
SLATS 1 2 0 L\_SLAT R\_SLAT  
THRUST\_1OR2 1 2 0 THRUSTL THRUSTR  
PGATE490 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE491 1 5 0 PGATE9 THRUST\_1&2 NOTUNDCARR ENG1FIRE ENG2FIRE  
PGATE492 1 2 0 ICE COLLISION  
THRUST\_1&2 2 2 0 THRUSTL THRUSTR  
NOTUNDCARR 1 2 0 HGATE741 HGATE742  
PGATE495 1 0 2 BADWEATER BATTDAM  
PGATE496 1 5 0 PGATE17 THRUST\_1&2 NOTUNDCARR ENG1FIRE ENG2FIRE  
PGATE497 1 2 0 ICE COLLISION  
PGATE500 1 0 2 BADWEATER BATTDAM  
PGATE501 1 5 0 PGATE505 THRUST\_1&2 NOTUNDCARR ENG1FIRE ENG2FIRE  
PGATE502 1 2 0 ICE COLLISION  
PGATE505 1 6 0 ELEVATORS AILERONS RUDDER FLAPS SLATS SPOILERS  
SPOILERS 1 2 0 L\_SPOILER R\_SPOILER  
PGATE506 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE507 1 6 0 PGATE511 THRUST\_1&2 UNDERCARRIAGEF1 ENG1FIRE ENG2FIRE RE\_THRUST\_F  
PGATE508 1 2 1 ICE COLLISION OBJRUNWAY  
PGATE511 1 3 0 ELEVATORS AILERONS RUDDER  
UNDERCARRIAGEF1 1 6 0 HGATE318 HGATE319 HGATE320 HGATE321 HGATE322 HGATE323  
PGATE513 1 2 0 ENG1FIRE ENG2FIRE  
PGATE516 1 2 0 ENG1FIRE ENG2FIRE  
PGATE518 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE519 1 5 0 PGATE72 THRUST\_1OR2 ENG1FIRE ENG2FIRE UNDERCARRIAGEF2  
PGATE520 1 3 0 ICE COLLISION BRAKEMER  
PGATE521 1 6 0 PGATE505 NOTUNDCARR THRUST\_1&2 ENG1FIRE ENG2FIRE NOTRETH  
PGATE522 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE523 1 2 0 ICE COLLISION  
PGATE524 1 0 2 BADWEATER BATTDAM  
PGATE525 1 5 0 PGATE505 NOTUNDCARR THRUST\_1&2 ENG1FIRE ENG2FIRE  
PGATE526 1 2 0 ICE COLLISION  
PGATE527 1 0 2 BADWEATER BATTDAM  
PGATE528 1 5 0 PGATE505 NOTUNDCARR THRUST\_1&2 ENG1FIRE ENG2FIRE  
PGATE529 1 2 0 ICE COLLISION  
PGATE530 1 0 3 BADWEATER BATTDAM BIEDSTRIKE  
PGATE531 1 6 0 PGATE505 UNDERCARRIAGEF1 THRUST\_1&2 ENG1FIRE ENG2FIRE RE\_THRUST\_F

PGATE532 1 2 1 ICE COLLISION OBJRUNWAY  
 PGATE534 1 2 0 ENG1FIRE ENG2FIRE  
 PGATE537 1 2 0 ENG1FIRE ENG2FIRE  
 PGATE539 1 0 2 BADWEATER BATTDAM  
 PGATE540 1 5 0 PGATE505 THRUST\_1OR2 NOTUNDCARR ENG1FIRE ENG2FIRE  
 PGATE542 1 0 2 BADWEATER BATTDAM  
 PGATE543 1 5 0 PGATE65 THRUST\_1OR2 NOTUNDCARR ENG1FIRE ENG2FIRE  
 PGATE545 1 0 2 BADWEATER BATTDAM  
 PGATE546 1 4 0 THRUST\_1&2 NOTUNDCARR ENG1FIRE ENG2FIRE  
 PGATE548 1 0 2 BADWEATER BATTDAM  
 PGATE549 1 2 0 ENG1FIRE ENG2FIRE  
 ICE 2 1 1 PGATE552 ICEWEATER  
 PGATE552 1 1 2 DCPOWER ANTIICEF ICESENSORS  
 DCPOWER 1 2 0 EGATE82 PWR\_HIGHDC  
 COLLISION 1 3 0 PGATE553 PGATE554 PGATE555  
 PGATE553 2 1 1 NAVINFO MOUNTON  
 PGATE554 2 1 1 PGATE556 OTHERAIRCRAFT  
 PGATE555 2 0 2 GROUND GPWS  
 NAVINFO 1 2 0 HGATE238 HGATE239  
 PGATE556 2 0 2 ATC TCAS  
 BRAKING\_F 1 2 0 HGATE718 HGATE719  
 HTOP1 2 3 0 HGATE1 HGATE2 HGATE3  
 HGATE1 1 2 0 HGATE4 HY\_LIQ\_LE\_ACT1  
 HGATE2 1 2 0 HGATE6 HY\_LIQ\_LE\_ACT2  
 HGATE3 1 2 0 HGATE8 HY\_LIQ\_LE\_ACT3  
 HGATE4 1 0 3 HA1LESTUCK HA1LELEAK HA1LERUPT  
 HY\_LIQ\_LE\_ACT1 1 2 0 HGATE68 HYDRAULICFAIL  
 HGATE6 1 0 3 HA2LESTUCK HA2LELEAK HA2LERUPT  
 HY\_LIQ\_LE\_ACT2 1 2 0 HGATE75 HYDRAULICFAIL  
 HGATE8 1 0 3 HA3LESTUCK HA3LELEAK HA3LERUPT  
 HY\_LIQ\_LE\_ACT3 1 2 0 HGATE82 HYDRAULICFAIL  
 HGATE10 1 0 4 HLESVO1MD HLESVO1INSIGN HLESVO1LECK HLESVO1WISIGN  
 HY\_PRE\_LOW 2 3 0 HYSUBSYS1 HYSUBSYS2 HYSUBSYS3  
 HYSUBSYS1 1 3 0 HGATE17 HGATE18 HY1PUMPS  
 HGATE16 1 0 5 H1MPUMP H1MPLOSPEED H1MPINT H1MPEXT H1MPBLOCK  
 HGATE17 1 0 3 H1PIP1 H1PRVOPEN H1NRV  
 HGATE18 1 2 0 HGATE20 HGATE25  
 HGATE19 1 0 2 H1FOB H1FGAP  
 HGATE20 1 2 0 HGATE23 LOW\_TO\_RES1  
 HGATE21 1 2 0 HGATE16 H1MECPUMPCON  
 HGATE22 1 0 2 H1LGBXNO H1LGBXLOW  
 HGATE23 1 0 4 H1RLEAK H1RMECH H1RAERO H1RLIQ  
 LOW\_TO\_RES1 1 1 1 HGATE127 H1REPIP  
 HGATE25 2 2 0 HGATE19 H1BYPASS  
 HY1PUMPS 2 3 0 HGATE21 HGATE27 H1ACELPUMP  
 HGATE27 1 0 4 H1ALOPRESS H1AINT H1ADISCH H1ASUPPLY  
 HGATE28 1 0 5 H1HBLOCK H1HNOFLUID H1HNOPRESS H1HINT H1HLEV  
 HYSUBSYS2 1 3 0 HGATE31 HGATE32 HY2PUMPS  
 HGATE31 1 0 3 H2PIP1 H2PRVOPEN H2NRV  
 HGATE32 1 2 0 HGATE34 HGATE35  
 HY2PUMPS 2 3 0 HGATE37 H2ACELPUMP1 H2ACELPUMP2  
 HGATE34 1 2 0 HGATE184 LOW\_TO\_RES2  
 HGATE35 2 2 0 HGATE192 H2BYPASS  
 HGATE37 1 0 4 H2ALOPRESS H2AINT H2ADISCH H2ASUPPLY  
 HYSUBSYS3 1 3 0 HGATE45 HGATE46 HY3PUMPS  
 HGATE45 1 0 3 H3PIP1 H3PRVOPEN H3NRV  
 HGATE46 1 2 0 HGATE48 HGATE49  
 HY3PUMPS 2 3 0 HGATE55 HGATE132 HGATE146  
 HGATE48 1 2 0 HGATE50 LOW\_TO\_RES3  
 HGATE49 2 2 0 HGATE53 H3BYPASS  
 HGATE50 1 0 4 H3RLEAK H3RMECH H3RAERO H3RLIQ  
 LOW\_TO\_RES3 1 1 1 HGATE162 H3REPIP  
 HGATE52 1 0 5 H3HBLOCK H3HNOFLUID H3HNOPRESS H3HINT H3HLEV  
 HGATE53 1 0 2 H3FOB H3FGAP  
 HGATE55 1 0 4 H3ALOPRESS H3AINT H3ADISCH H3ASUPPLY  
 HGATE56 1 0 5 H3MPUMP H3MPLOSPEED H3MPINT H3MPEXT H3MPBLOCK  
 HGATE58 1 0 4 HLESVO2MD HLESVO2INSIGN HLESVO2LECK HLESVO2OSIGN  
 HGATE63 1 0 4 HLESVO3MD HLESVO3INSIGN HLESVO3LECK HLESVO3OSIGN  
 HGATE68 1 4 0 HGATE10 POSTION\_SENSOR FCAVIONIC DCPOWER  
 POSTION\_SENSOR 2 2 0 HGATE70 HGATE71  
 HGATE70 1 0 2 HLEPOSSSEN1NO HLEPOSSSENWO1  
 HGATE71 1 0 2 HLEPOSSSEN2NO HLEPOSSSENWO2  
 HGATE75 1 4 0 HGATE58 POSTION\_SENSOR FCAVIONIC DCPOWER  
 HGATE82 1 4 0 HGATE63 POSTION\_SENSOR FCAVIONIC DCPOWER  
 HY\_PRE\_HIGH 1 2 0 HY\_PRE\_HIGH1 HY\_PRE\_HIGH2

H1ACELPUMP 1 2 0 HGATE113 H1ACPUMPCON  
H1MECPUMPCON 1 4 0 HY\_UMS\_AVIONIC HGATE93 HGATE108 DCPower  
HY\_UMS\_AVIONIC 1 4 0 HYDR\_UNIT FMS UMS DATA\_BUS  
HGATE93 1 2 0 HGATE22 THRUSTL  
HYDR\_UNIT 2 3 0 HGATE97 HGATE98 HGATE99  
HGATE97 1 0 2 HSUIO1 HSUIWR1  
HGATE98 1 0 2 HSUIO2 HSUIWR2  
HGATE99 1 0 2 HSUIO3 HSUIWR3  
FMS 2 3 0 HGATE102 HGATE103 HGATE104  
UMS 2 3 0 HGATE105 HGATE106 HGATE107  
HGATE102 1 0 2 FMSNO1 HSFMSWR1  
HGATE103 1 0 2 FMSNO2 HSFMSWR2  
HGATE104 1 0 2 FMSNO3 HSFMSWR3  
HGATE105 1 0 2 UMSNO1 HSUMSWR1  
HGATE106 1 0 2 UMSNO2 HSUMSWR2  
HGATE107 1 0 2 UMSNO3 HSUMSWR3  
HGATE108 1 2 0 HGATE141 DCPower  
HGATE109 1 0 2 H1PSENNO1 H1PSENWR1  
HGATE110 1 0 2 H1PSENNO2 H1PSENWR2  
HGATE113 1 0 5 H1ACPPUMP H1ACPLOSPEED H1ACPINT H1ACPEXT H1ACPBLOCK  
H1ACPUMPCON 1 4 0 HY\_UMS\_AVIONIC ACPOWER HGATE108 DCPower  
LAND&BRAKEAVIONIC 1 4 0 HGATE267 DATA\_BUS NAVINFO FMS  
ACPOWER 1 2 0 EGATE96 PWR\_HIGHAC  
H1BYPASS 1 1 1 HGATE120 HY1BYPASSBLO  
HGATE120 1 3 1 HGATE121 HY\_UMS\_AVIONIC DCPower HY1BYVAL  
HGATE121 2 2 0 HGATE123 HGATE124  
HGATE123 1 0 2 HY1FBSENNO1 HY1FBSENWR1  
HGATE124 1 0 2 HY1FBSENNO2 HY1FBSENWR2  
HGATE127 1 4 1 HGATE28 HY\_UMS\_AVIONIC HGATE128 DCPower HY1HEEXV  
HGATE128 2 2 0 HGATE129 HGATE130  
HGATE129 1 0 2 HY1TEMS1NO HY1TEMS1WR  
HGATE130 1 0 2 HY1TEMS2NO HY1TEMS2WR  
HGATE132 1 2 0 HGATE56 H3MECPUMP1CON  
H3MECPUMP1CON 1 4 0 HY\_UMS\_AVIONIC HGATE134 HGATE135 DCPower  
HGATE134 1 2 0 HGATE136 THRUSTR  
HGATE135 1 2 0 HGATE143 DCPower  
HGATE136 1 0 2 H1RGBOXNO H1RGBOXLOW  
HGATE141 2 2 0 HGATE109 HGATE110  
HGATE143 2 2 0 HGATE144 HGATE145  
HGATE144 1 0 2 H3PSENNO1 H3PSENWR1  
HGATE145 1 0 2 H3PSENNO2 H3PSENWR2  
HGATE146 1 2 0 HGATE147 H3ACPUMPCO  
HGATE147 1 0 5 H3ACPUMP H3PACLOSPEED H3ACPINT H3ACPEXT H3ACPBLOCK  
H3ACPUMPCON 1 4 0 HY\_UMS\_AVIONIC HGATE135 ACPOWER DCPower  
H3BYPASS 1 1 1 HGATE156 HY3BYPASSBLO  
HGATE156 1 3 1 HGATE157 HY\_UMS\_AVIONIC DCPower HY3BYVAL  
HGATE157 2 2 0 HGATE158 HGATE159  
HGATE158 1 0 2 HY3FBSENNO1 HY3FBSENWR1  
HGATE159 1 0 2 HY3FBSENNO2 HY3FBSENWR2  
HGATE162 1 4 1 HGATE52 HY\_UMS\_AVIONIC HGATE163 DCPower HY3HEEXV  
HGATE163 2 2 0 HGATE164 HGATE165  
HGATE164 1 0 2 HY3TEMS1NO HY3TES1WR  
HGATE165 1 0 2 HY3TEMS2NO HY3TEMS2WR  
H2ACELPUMP 1 1 2 0 HGATE168 H2ACPUMP1CON  
HGATE168 1 0 5 H2ACP1PUMP H2ACP1LOSPEED H2ACP1INT H2ACP1EXT H2ACP1BLOCK  
H2ACPUMP1CON 1 4 0 HY\_UMS\_AVIONIC HGATE171 ACPOWER DCPower  
HGATE171 1 2 0 HGATE173 DCPower  
HGATE173 2 2 0 HGATE174 HGATE175  
HGATE174 1 0 2 H2PSENNO1 H2PSENWR1  
HGATE175 1 0 2 H2PSENNO2 H2PSENWR2  
H2ACELPUMP 2 1 2 0 HGATE177 H2ACPUMP2CON  
HGATE177 1 0 5 H2ACP2PUMP H2ACP2LOSPEED H2ACP2INT H2ACP2EXT H2ACP2BLOCK  
H2ACPUMP2CON 1 4 0 HY\_UMS\_AVIONIC HGATE71 ACPOWER DCPower  
HGATE184 1 0 4 H2RLEAK H2RMECH H2RAERO H2RLIQ  
LOW\_TO\_RES 2 1 1 1 HGATE186 H2REPIP  
HGATE186 1 4 1 HGATE187 HY\_UMS\_AVIONIC HGATE188 DCPower HY2HEEXV  
HGATE187 1 0 5 H2HBLOCK H2HNOFLUID H2HNOPRESS H2HINT H2HLEV  
HGATE188 2 2 0 HGATE190 HGATE191  
HGATE190 1 0 2 HY2TEMS1NO HY2TEMS1WR  
HGATE191 1 0 2 HY2TEMS2NO HY2TEMS2WR  
HGATE192 1 0 2 H2FOB H2FGAP  
H2BYPASS 1 1 1 HGATE194 HY2BYPASSBLO  
HGATE194 1 3 1 HGATE195 HY\_UMS\_AVIONIC DCPower HY2BYVAL  
HGATE195 2 2 0 HGATE196 HGATE197  
HGATE196 1 0 2 HY2FBSENNO1 HY2FBSENWR1

HGATE197 1 0 2 HY2FBSENNO2 HY2FBSENWR2  
 FCAVIONIC 1 6 0 ACT\_DR\_COM FCU ADC DATA\_BUS NAVINFO FMS  
 ACT\_DR\_COM 2 3 0 HGATE202 HGATE203 HGATE204  
 HGATE202 1 0 2 AVADC1NO AVADC1WR  
 HGATE203 1 0 2 AVADC2NO AVADC2WR  
 HGATE204 1 0 2 AVADC3NO AVADC3WR  
 FCU 2 3 0 HGATE206 HGATE207 HGATE208  
 HGATE206 1 0 2 AVFCU1NO AVFCU1WR  
 HGATE207 1 0 2 AVFCU2NO AVFCU2WR  
 HGATE208 1 0 2 AVFCU3NO AVFCU3WR  
 ADC 2 3 0 HGATE214 HGATE215 HGATE216  
 HGATE214 1 2 0 HGATE217 AIR1SENSORS  
 HGATE215 1 2 0 HGATE218 AIR1SENSORS  
 HGATE216 1 2 0 HGATE219 AIR1SENSORS  
 HGATE217 1 0 2 AVAIRDC1NO AVAIRDC1WR  
 HGATE218 1 0 2 AVAIRDC2NO AVAIRDC2WR  
 HGATE219 1 0 2 AVAIRDC3NO AVAIRDC3WR  
 AIR1SENSORS 1 3 0 HGATE221 TOTALAIR AIRTEM  
 HGATE221 2 3 0 HGATE224 HGATE225 HGATE226  
 TOTALAIR 2 3 0 HGATE227 HGATE228 HGATE229  
 AIRTEM 2 3 0 HGATE230 HGATE231 HGATE232  
 HGATE224 1 0 2 AVSAS1NO AVSAS1WR  
 HGATE225 1 0 2 AVSAS2NO AVSAS2WR  
 HGATE226 1 0 2 AVSAS3NO AVSAS3WR  
 HGATE227 1 0 2 AVTAS1NO AVTAS1WR  
 HGATE228 1 0 2 AVTAS2NO AVTAS2WR  
 HGATE229 1 0 2 AVTAS3NO AVTAS3WR  
 HGATE230 1 0 2 AVTEAS1NO AVTEAS1WR  
 HGATE231 1 0 2 AVTEAS2NO AVTEAS2WR  
 HGATE232 1 0 2 AVTEAS3NO AVTEAS3WR  
 DATA\_BUS 2 3 0 HGATE234 HGATE235 HGATE236  
 HGATE234 1 0 2 AVDBUS1NO AVDBUS1WR  
 HGATE235 1 0 2 AVDBUS2NO AVDBUS2WR  
 HGATE236 1 0 2 AVDBUS3NO AVDBUS3WR  
 HGATE238 2 4 0 IRS GPS VOR DMO  
 HGATE239 2 2 0 HGATE244 HGATE245  
 IRS 2 2 0 HGATE246 HGATE247  
 GPS 2 2 0 HGATE248 HGATE249  
 VOR 2 2 0 HGATE250 HGATE251  
 DMO 2 2 0 HGATE252 HGATE253  
 HGATE244 2 2 0 HGATE254 HGATE255  
 HGATE245 2 2 0 HGATE256 HGATE257  
 HGATE246 1 0 2 AVIRS1NOPOS AVIRS1WRPOS  
 HGATE247 1 0 2 AVIRS2NOPOS AVIRS2WRPOS  
 HGATE248 1 0 2 AVGPS1NOPOS AVGPS1WRPOS  
 HGATE249 1 0 2 AVGPS2NOPOS AVGPS2WRPOS  
 HGATE250 1 0 2 AVVOR1NO AVVOR1WR  
 HGATE251 1 0 2 AVVOR2NO AVVOR2WR  
 HGATE252 1 0 2 AVDMO1NO AVDMO1WR  
 HGATE253 1 0 2 AVDMO2NO AVDMO2WR  
 HGATE254 1 0 2 AVIRS1NOATT AVIRS1WRATT  
 HGATE255 1 0 2 AVIRS2NOATT AVIRS2WRATT  
 HGATE256 1 0 2 AVGPS1NOATT AVGPS1WRATT  
 HGATE257 1 0 2 AVGPS2NOATT AVGPS2WRATT  
 HGATE267 2 3 0 HGATE268 HGATE269 HGATE270  
 HGATE268 1 0 2 AVLNB1NO AVLNB1WR  
 HGATE269 1 0 2 AVLNB2NO AVLNB2WR  
 HGATE270 1 0 2 AVLNB3NO AVLNB3WR  
 L\_ELEVATOR 1 1 1 HTOP1 L\_ELEVATOR  
 R\_ELEVATOR 1 1 1 HGATE462 R\_ELEVATOR  
 L\_AILERON 1 1 1 HGATE601 L\_AILERON  
 R\_AILERON 1 1 1 HGATE610 R\_AILERON  
 L\_SPOILER 1 1 1 HGATE630 L\_SPOILER  
 R\_SPOILER 1 1 1 HGATE639 R\_SPOILER  
 L\_FLAP 1 1 1 HGATE648 L\_FLAP  
 R\_FLAP 1 1 1 HGATE657 R\_FLAP  
 L\_SLAT 1 1 1 HGATE664 L\_SLAT  
 R\_SLAT 1 1 1 HGATE673 R\_SLAT  
 HY\_PRE\_HIGH1 1 3 0 HGATE289 HGATE290 HGATE291  
 HY\_PRE\_HIGH2 2 3 0 HGATE292 HGATE293 HGATE294  
 HGATE289 2 1 1 HY1\_HIGH\_PRE H1COFFVOPEN  
 HGATE290 2 1 1 HY2\_HIGH\_PRE H2COFFVOPEN  
 HGATE291 2 1 1 HY3\_HIGH\_PRE H3COFFVOPEN  
 HGATE292 1 1 1 HY1\_HIGH\_PRE H1COFFVCLOSE

HGATE293 1 1 1 HY2\_HIGH\_PRE H2COFFVCLOSE  
 HGATE294 1 1 1 HY3\_HIGH\_PRE H3COFFVCLOSE  
 HY1\_HIGH\_PRE 2 1 1 HGATE295 H1PRVCLOSE  
 HY2\_HIGH\_PRE 2 1 1 HGATE305 H2PRVCLOSE  
 HY3\_HIGH\_PRE 2 1 1 HGATE312 H3PRVCLOSE  
 HGATE295 1 2 0 HGATE296 HGATE297  
 HGATE296 1 1 1 HGATE300 H1MPHISPEED  
 HGATE297 1 1 1 HGATE303 H1ACHISPEED  
 HGATE300 1 2 1 HY\_UMS\_AVIONIC HGATE302 OV\_PWR\_LEN  
 HGATE302 2 0 2 H1PSENWR1 H1PSENWR2  
 HGATE303 1 2 0 HY\_UMS\_AVIONIC HGATE302  
 HGATE305 1 2 0 HGATE306 HGATE307  
 HGATE306 1 1 1 HGATE308 H2AC1HISPEED  
 HGATE307 1 1 1 HGATE309 H2AC2HISPEED  
 HGATE308 1 2 0 HY\_UMS\_AVIONIC HGATE310  
 HGATE309 1 2 0 HY\_UMS\_AVIONIC HGATE310  
 HGATE310 2 0 2 H2PSENWR1 H2PSENWR2  
 HGATE312 1 2 0 HGATE313 HGATE314  
 HGATE313 1 1 1 HGATE315 H3MPHISPEED  
 HGATE314 1 1 1 HGATE316 H3ACHISPEED  
 HGATE315 1 2 1 HY\_UMS\_AVIONIC HGATE317 OV\_PWR\_RENG  
 HGATE316 1 2 0 HY\_UMS\_AVIONIC HGATE317  
 HGATE317 2 0 2 H3PSENWR1 H3PSENWR2  
 HGATE318 1 1 1 FOS\_UNLOCK1 UNCLOC1UNLOC  
 HGATE319 1 1 1 FOS\_DOORDN UNCDAMDR  
 HGATE320 1 1 1 FOS\_LOCK2 UNCLOC2LOC  
 HGATE321 1 1 1 FOS\_UNLOCK3 UNCLOC3UNLOC  
 HGATE322 1 1 1 FOS\_UNCDR UNLEGFAIL  
 HGATE323 1 1 1 FOS\_LOCK4 UNCLOC4LOC  
 FOS\_UNLOCK1 2 3 0 HGATE333 HGATE334 HGATE335  
 FOS\_DOORDN 2 3 0 HGATE378 HGATE379 HGATE380  
 FOS\_LOCK2 2 3 0 HGATE351 HGATE352 HGATE353  
 FOS\_UNLOCK3 2 3 0 HGATE360 HGATE361 HGATE362  
 FOS\_UNCDR 2 3 0 HGATE387 HGATE388 HGATE389  
 FOS\_LOCK4 2 3 0 HGATE369 HGATE370 HGATE371  
 UNDERCARRIAGEF2 1 6 0 HGATE325 HGATE326 HGATE327 HGATE328 HGATE329 HGATE330  
 HGATE325 1 1 1 FOS\_UNLOCK4 UNCLOC4UNLOC  
 HGATE326 1 1 1 FOS\_UNCUP UNLEGFAIL  
 HGATE327 1 1 1 FOS\_LOCK3 UNCLOC3LOC  
 HGATE328 1 1 1 FOS\_UNLOCK2 UNCLOC2UNLOC  
 HGATE329 1 1 1 FOS\_DOORUP UNCDAMDR  
 HGATE330 1 1 1 FOS\_LOCK1 UNCLOC1LOC  
 FOS\_UNLOCK4 2 3 0 HGATE429 HGATE430 HGATE431  
 FOS\_LOCK3 2 3 0 HGATE441 HGATE442 HGATE443  
 FOS\_UNLOCK2 2 3 0 HGATE447 HGATE448 HGATE449  
 FOS\_DOORUP 2 3 0 HGATE342 HGATE343 HGATE344  
 FOS\_LOCK1 2 3 0 HGATE453 HGATE454 HGATE455  
 FOS\_UNCUP 2 3 0 HGATE435 HGATE436 HGATE437  
 HGATE333 1 2 0 HGATE336 HY\_LIQ\_UNLOC1\_A1  
 HGATE334 1 2 0 HGATE338 HY\_LIQ\_UNLOC1\_A2  
 HGATE335 1 2 0 HGATE340 HY\_LIQ\_UNLOC1\_A3  
 HGATE336 1 0 3 LA1L1STUCK LA1L1LEAK LA1L1RUPT  
 HY\_LIQ\_UNLOC1\_A1 1 2 0 HGATE398 HGATE685  
 HGATE338 1 0 3 LA2L1STUCK LA2L1LEAK LA2L1RUPT  
 HY\_LIQ\_UNLOC1\_A2 1 2 0 HGATE398 HGATE686  
 HGATE340 1 0 3 LA3L1STUCK LA3L1LEAK LA3L1RUPT  
 HY\_LIQ\_UNLOC1\_A3 1 2 0 HGATE398 HGATE687  
 HGATE342 1 2 0 HGATE381 HY\_LIQ\_DRUP\_A1  
 HGATE343 1 2 0 HGATE383 HY\_LIQ\_DRUP\_A2  
 HGATE344 1 2 0 HGATE385 HY\_LIQ\_DRUP\_A3  
 HGATE351 1 2 0 HGATE354 HY\_LIQ\_LOC2\_A1  
 HGATE352 1 2 0 HGATE356 HY\_LIQ\_LOC2\_A2  
 HGATE353 1 2 0 HGATE358 HY\_LIQ\_LOC2\_A3  
 HGATE354 1 0 3 LA1L2STUCK LA1L2LEAK LA1L2RUPT  
 HY\_LIQ\_LOC2\_A1 1 2 0 HGATE398 HGATE694  
 HGATE356 1 0 3 LA2L2STUCK LA2L2LEAK LA2L2RUPT  
 HY\_LIQ\_LOC2\_A2 1 2 0 HGATE398 HGATE695  
 HGATE358 1 0 3 LA3L2STUCK LA3L2LEAK LA3L2RUPT  
 HY\_LIQ\_LOC2\_A3 1 2 0 HGATE398 HGATE696  
 HGATE360 1 2 0 HGATE363 HY\_LIQ\_UNLOC3\_A1  
 HGATE361 1 2 0 HGATE365 HY\_LIQ\_UNLOC3\_A2  
 HGATE362 1 2 0 HGATE367 HY\_LIQ\_UNLOC3\_A3  
 HGATE363 1 0 3 LA1L3STUCK LA1L3LEAK LA1L3RUPT  
 HY\_LIQ\_UNLOC3\_A1 1 2 0 HGATE398 HGATE703  
 HGATE365 1 0 3 LA2L3STUCK LA2L3LEAK LA2L3RUPT

HY\_LIQ\_UNLOC3\_A2 1 2 0 HGATE398 HGATE704  
 HGATE367 1 0 3 LA3L3STUCK LA3L3LEAK LA3L3RUPT  
 HY\_LIQ\_UNLOC3\_A3 1 2 0 HGATE398 HGATE705  
 HGATE369 1 2 0 HGATE372 HY\_LIQ\_LOC4\_A1  
 HGATE370 1 2 0 HGATE374 HY\_LIQ\_LOC4\_A2  
 HGATE371 1 2 0 HGATE376 HY\_LIQ\_LOC4\_A3  
 HGATE372 1 0 3 LA1L4STUCK LA1L4LEAK LA1L4RUPT  
 HY\_LIQ\_LOC4\_A1 1 2 0 HGATE398 HGATE712  
 HGATE374 1 0 3 LA2L4STUCK LA2L4LEAK LA2L4RUPT  
 HY\_LIQ\_LOC4\_A2 1 2 0 HGATE398 HGATE713  
 HGATE376 1 0 3 LA3L4STUCK LA3L4LEAK LA3L4RUPT  
 HY\_LIQ\_LOC4\_A3 1 2 0 HGATE398 HGATE714  
 HGATE378 1 2 0 HGATE381 HY\_LIQ\_DRDN\_A1  
 HGATE379 1 2 0 HGATE383 HY\_LIQ\_DRDN\_A2  
 HGATE380 1 2 0 HGATE385 HY\_LIQ\_DRDN\_A3  
 HGATE381 1 0 3 LA1DRSTUCK LA1DRLEAK LA1DRRUPT  
 HY\_LIQ\_DRDN\_A1 1 2 0 HGATE398 HGATE688  
 HGATE383 1 0 3 LA2DRSTUCK LA2DRLEAK LA2DRRUPT  
 HY\_LIQ\_DRDN\_A2 1 2 0 HGATE398 HGATE689  
 HGATE385 1 0 3 LA3DRSTUCK LA3DRLEAK LA3DRRUPT  
 HY\_LIQ\_DRDN\_A3 1 2 0 HGATE398 HGATE690  
 HGATE387 1 2 0 HGATE390 HY\_LIQ\_UNCDR\_A1  
 HGATE388 1 2 0 HGATE392 HY\_LIQ\_UNCDR\_A2  
 HGATE389 1 2 0 HGATE394 HY\_LIQ\_UNCDR\_A3  
 HGATE390 1 0 3 LA1UNCSTUCK LA1UNCLEAK LA1UNCRUPT  
 HY\_LIQ\_UNCDR\_A1 1 2 0 HGATE398 HGATE706  
 HGATE392 1 0 3 LA2UNCSTUCK LA2UNCLEAK LA2UNCRUPT  
 HY\_LIQ\_UNCDR\_A2 1 2 0 HGATE398 HGATE707  
 HGATE394 1 0 3 LA3UNCSTUCK LA3UNCLEAK LA3UNCRUPT  
 HY\_LIQ\_UNCDR\_A3 1 2 0 HGATE398 HGATE708  
 LAND\_SIGNAL 1 5 0 NAVINFO DATA\_BUS LAND\_BRA\_UNI HGATE416 FMS  
 HGATE398 2 2 0 UNSEQVALF1 UNSEQVALF2  
 UNSEQVALF1 1 3 1 HYDRAULICFAIL LAND\_SIGNAL DCPOWER LSEQ1VF  
 UNSEQVALF2 1 3 1 HYDRAULICFAIL LAND\_SIGNAL DCPOWER LSEQ2VF  
 HYDRAULICFAIL 1 2 0 HY\_PRE\_HIGH HY\_PRE\_LOW  
 LAND\_BRA\_UNI 2 3 0 HGATE413 HGATE414 HGATE415  
 HGATE413 1 0 2 AVLANUNI1NO AVLANUNI1WR  
 HGATE414 1 0 2 AVLANUNI2NO AVLANUNI2WR  
 HGATE415 1 0 2 AVLANUNI3NO AVLANUNI3WR  
 HGATE416 2 2 0 HGATE417 HGATE418  
 HGATE417 1 0 2 AVUNDSSEN1NO AVUNDSSEN1WR  
 HGATE418 1 0 2 AVUNDSSEN2NO AVUNDSSEN2WR  
 HGATE429 1 2 0 HGATE372 HY\_LIQ\_UNLOC4\_A1  
 HGATE430 1 2 0 HGATE374 HY\_LIQ\_UNLOC4\_A2  
 HGATE431 1 2 0 HGATE376 HY\_LIQ\_UNLOC4\_A3  
 HY\_LIQ\_UNLOC4\_A1 1 2 0 HGATE398 HGATE715  
 HY\_LIQ\_UNLOC4\_A2 1 2 0 HGATE398 HGATE716  
 HY\_LIQ\_UNLOC4\_A3 1 2 0 HGATE398 HGATE717  
 HGATE435 1 2 0 HGATE390 HY\_LIQ\_UNCUP\_A1  
 HGATE436 1 2 0 HGATE392 HY\_LIQ\_UNCUP\_A2  
 HGATE437 1 2 0 HGATE394 HY\_LIQ\_UNCUP\_A3  
 HY\_LIQ\_UNCUP\_A1 1 2 0 HGATE398 HGATE709  
 HY\_LIQ\_UNCUP\_A2 1 2 0 HGATE398 HGATE710  
 HY\_LIQ\_UNCUP\_A3 1 2 0 HGATE398 HGATE711  
 HGATE441 1 2 0 HGATE363 HY\_LIQ\_LOC3\_A1  
 HGATE442 1 2 0 HGATE365 HY\_LIQ\_LOC3\_A2  
 HGATE443 1 2 0 HGATE367 HY\_LIQ\_LOC3\_A3  
 HY\_LIQ\_LOC3\_A1 1 2 0 HGATE398 HGATE700  
 HY\_LIQ\_LOC3\_A2 1 2 0 HGATE398 HGATE701  
 HY\_LIQ\_LOC3\_A3 1 2 0 HGATE398 HGATE702  
 HGATE447 1 2 0 HGATE354 HY\_LIQ\_UNLOC2\_A1  
 HGATE448 1 2 0 HGATE356 HY\_LIQ\_UNLOC2\_A2  
 HGATE449 1 2 0 HGATE358 HY\_LIQ\_UNLOC2\_A3  
 HY\_LIQ\_UNLOC2\_A1 1 2 0 HGATE398 HGATE697  
 HY\_LIQ\_UNLOC2\_A2 1 2 0 HGATE398 HGATE698  
 HY\_LIQ\_UNLOC2\_A3 1 2 0 HGATE398 HGATE699  
 HGATE453 1 2 0 HGATE336 HY\_LIQ\_LOC1\_A1  
 HGATE454 1 2 0 HGATE338 HY\_LIQ\_LOC1\_A2  
 HGATE455 1 2 0 HGATE340 HY\_LIQ\_LOC1\_A3  
 HY\_LIQ\_LOC1\_A1 1 2 0 HGATE398 HGATE682  
 HY\_LIQ\_LOC1\_A2 1 2 0 HGATE398 HGATE683  
 HY\_LIQ\_LOC1\_A3 1 2 0 HGATE398 HGATE684  
 HY\_LIQ\_DRUP\_A1 1 2 0 HGATE398 HGATE691  
 HY\_LIQ\_DRUP\_A2 1 2 0 HGATE398 HGATE692  
 HY\_LIQ\_DRUP\_A3 1 2 0 HGATE398 HGATE693

HGATE462 2 3 0 HGATE600 HGATE464 HGATE465  
 HGATE600 1 2 0 HGATE466 HY\_LIQ\_RE\_ACT1  
 HGATE464 1 2 0 HGATE468 HY\_LIQ\_RE\_ACT2  
 HGATE465 1 2 0 HGATE470 HY\_LIQ\_RE\_ACT3  
 HGATE466 1 0 3 HA1RESTUCK HA1RELEAK HA1RERUPT  
 HY\_LIQ\_RE\_ACT1 1 2 0 HGATE761 HYDRAULICFAIL  
 HGATE468 1 0 3 HA2RESTUCK HA2RELEAK HA2RERUPT  
 HY\_LIQ\_RE\_ACT2 1 2 0 HGATE766 HYDRAULICFAIL  
 HGATE470 1 0 3 HA3RESTUCK HA3RELEAK HA3RERUPT  
 HY\_LIQ\_RE\_ACT3 1 2 0 HGATE768 HYDRAULICFAIL  
 HGATE601 2 3 0 HGATE602 HGATE603 HGATE604  
 HGATE602 1 2 0 HGATE605 HY\_LIQ\_LA\_ACT1  
 HGATE603 1 2 0 HGATE607 HY\_LIQ\_LA\_ACT2  
 HGATE604 1 2 0 HGATE609 HY\_LIQ\_LA\_ACT3  
 HGATE605 1 0 3 HA1LASTUCK HA1LLEAK HA1LARUPT  
 HY\_LIQ\_LA\_ACT1 1 2 0 HGATE770 HYDRAULICFAIL  
 HGATE607 1 0 3 HA2LASTUCK HA2LLEAK HA2LARUPT  
 HY\_LIQ\_LA\_ACT2 1 2 0 HGATE775 HYDRAULICFAIL  
 HGATE609 1 0 3 HA3LASTUCK HA3LLEAK HA3LARUPT  
 HY\_LIQ\_LA\_ACT3 1 2 0 HGATE777 HYDRAULICFAIL  
 HGATE610 2 3 0 HGATE611 HGATE612 HGATE613  
 HGATE611 1 2 0 HGATE614 HY\_LIQ\_RA\_ACT1  
 HGATE612 1 2 0 HGATE616 HY\_LIQ\_RA\_ACT2  
 HGATE613 1 2 0 HGATE618 HY\_LIQ\_RA\_ACT  
 HGATE614 1 0 3 HA1RASTUCK HA1RALEAK HA1RARUPT  
 HY\_LIQ\_RA\_ACT1 1 2 0 HGATE779 HYDRAULICFAIL  
 HGATE616 1 0 3 HA2RASTUCK HA2RALEAK HA2RARUPT  
 HY\_LIQ\_RA\_ACT2 1 2 0 HGATE783 HYDRAULICFAIL  
 HGATE618 1 0 3 HA3RASTUCK HA3RALEAK HA3RARUPT  
 HY\_LIQ\_RA\_ACT3 1 2 0 HGATE785 HYDRAULICFAIL  
 HGATE621 2 3 0 HGATE622 HGATE623 HGATE624  
 HGATE622 1 2 0 HGATE625 HY\_LIQ\_RU\_ACT1  
 HGATE623 1 2 0 HGATE627 HY\_LIQ\_RU\_ACT2  
 HGATE624 1 2 0 HGATE629 HY\_LIQ\_RU\_ACT3  
 HGATE625 1 0 3 HA1RUSTUCK HA1RULEAK HA1RURUPT  
 HY\_LIQ\_RU\_ACT1 1 2 0 HGATE787 HYDRAULICFAIL  
 HGATE627 1 0 3 HA2RUSTUCK HA2RULEAK HA2RURUPT  
 HY\_LIQ\_RU\_ACT2 1 2 0 HGATE792 HYDRAULICFAIL  
 HGATE629 1 0 3 HA3RUSTUCK HA3RULEAK HA3RURUPT  
 HY\_LIQ\_RU\_ACT3 1 2 0 HGATE794 HYDRAULICFAIL  
 HGATE630 2 3 0 HGATE631 HGATE632 HGATE633  
 HGATE631 1 2 0 HGATE634 HY\_LIQ\_LS\_ACT1  
 HGATE632 1 2 0 HGATE636 HY\_LIQ\_LS\_ACT2  
 HGATE633 1 2 0 HGATE638 HY\_LIQ\_LS\_ACT3  
 HGATE634 1 0 3 HA1LSSTUCK HA1LSLEAK HA1LSRUPT  
 HY\_LIQ\_LS\_ACT1 1 2 0 HGATE796 HYDRAULICFAIL  
 HGATE636 1 0 3 HA2LSSTUCK HA2LSLEAK HA2LSRUPT  
 HY\_LIQ\_LS\_ACT2 1 2 0 HGATE801 HYDRAULICFAIL  
 HGATE638 1 0 3 HA3LSSTUCK HA3LSLEAK HA3LSRUPT  
 HY\_LIQ\_LS\_ACT3 1 2 0 HGATE803 HYDRAULICFAIL  
 HGATE639 2 3 0 HGATE640 HGATE641 HGATE642  
 HGATE640 1 2 0 HGATE643 HY\_LIQ\_RS\_ACT1  
 HGATE641 1 2 0 HGATE645 HY\_LIQ\_RS\_ACT2  
 HGATE642 1 2 0 HGATE647 HY\_LIQ\_RS\_ACT3  
 HGATE643 1 0 3 HA1RSSTUCK HA1RSLEAK HA1RSRUPT  
 HY\_LIQ\_RS\_ACT1 1 2 0 HGATE805 HYDRAULICFAIL  
 HGATE645 1 0 3 HA2RSSTUCK HA2RSLEAK HA2RSRUPT  
 HY\_LIQ\_RS\_ACT2 1 2 0 HGATE809 HYDRAULICFAIL  
 HGATE647 1 0 3 HA3RSSTUCK HA3RSLEAK HA3RSRUPT  
 HY\_LIQ\_RS\_ACT3 1 2 0 HGATE811 HYDRAULICFAIL  
 HGATE648 2 3 0 HGATE649 HGATE650 HGATE651  
 HGATE649 1 2 0 HGATE652 HY\_LIQ\_LF\_ACT1  
 HGATE650 1 2 0 HGATE654 HY\_LIQ\_LF\_ACT2  
 HGATE651 1 2 0 HGATE656 HY\_LIQ\_LF\_ACT3  
 HGATE652 1 0 3 HA1LFSTUCK HA1LFLEAK HA1LFRUPT  
 HY\_LIQ\_LF\_ACT1 1 2 0 HGATE813 HYDRAULICFAIL  
 HGATE654 1 0 3 HA2LFSTUCK HA2LFLEAK HA2LFRUPT  
 HY\_LIQ\_LF\_ACT2 1 2 0 HGATE818 HYDRAULICFAIL  
 HGATE656 1 0 3 HA3LFSTUCK HA3LFLEAK HA3LFRUPT  
 HY\_LIQ\_LF\_ACT3 1 2 0 HGATE820 HYDRAULICFAIL  
 HGATE657 2 3 0 HGATE658 HGATE659 HGATE660  
 HGATE658 1 2 0 HGATE661 HY\_LIQ\_RF\_ACT1  
 HGATE659 1 2 0 HGATE662 HY\_LIQ\_RF\_ACT2  
 HGATE660 1 2 0 HGATE663 HY\_LIQ\_RF\_ACT3



HGATE661 1 0 3 HA1RFSTUCK HA1RFLEAK HA1RFRUPT  
 HGATE662 1 0 3 HA2RFSTUCK HA2RFLEAK HA2RFRUPT  
 HGATE663 1 0 3 HA3RFSTUCK HA3RFLEAK HA3RFRUPT  
 HY\_LIQ\_RF\_ACT1 1 2 0 HGATE822 HYDRAULICFAIL  
 HY\_LIQ\_RF\_ACT2 1 2 0 HGATE826 HYDRAULICFAIL  
 HY\_LIQ\_RF\_ACT3 1 2 0 HGATE828 HYDRAULICFAIL  
 HGATE664 2 3 0 HGATE665 HGATE666 HGATE667  
 HGATE665 1 2 0 HGATE668 HY\_LIQ\_LSL\_ACT1  
 HGATE666 1 2 0 HGATE670 HY\_LIQ\_LSL\_ACT2  
 HGATE667 1 2 0 HGATE672 HY\_LIQ\_LSL\_ACT3  
 HGATE668 1 0 3 HA1LSLSTUCK HA1LSLLEAK HA1LSLRUPT  
 HY\_LIQ\_LSL\_ACT1 1 2 0 HGATE830 HYDRAULICFAIL  
 HGATE670 1 0 3 HA2LSLSTUCK HA2LSLLEAK HA2LSLRUPT  
 HY\_LIQ\_LSL\_ACT2 1 2 0 HGATE834 HYDRAULICFAIL  
 HGATE672 1 0 3 HA3LSLSTUCK HA3LSLLEAK HA3LSLRUPT  
 HY\_LIQ\_LSL\_ACT3 1 2 0 HGATE836 HYDRAULICFAIL  
 HGATE673 2 3 0 HGATE674 HGATE675 HGATE676  
 HGATE674 1 2 0 HGATE677 HY\_LIQ\_RSL\_ACT1  
 HGATE675 1 2 0 HGATE679 HY\_LIQ\_RSL\_ACT2  
 HGATE676 1 2 0 HGATE681 HY\_LIQ\_RSL\_ACT3  
 HGATE677 1 0 3 HA1RSLSTUCK HA1RSLLEAK HA1RSLRUPT  
 HY\_LIQ\_RSL\_ACT1 1 2 0 HGATE838 HYDRAULICFAIL  
 HGATE679 1 0 3 HA2RSLSTUCK HA2RSLLEAK HA2RSLRUPT  
 HY\_LIQ\_RSL\_ACT2 1 2 0 HGATE842 HYDRAULICFAIL  
 HGATE681 1 0 3 HA3RSLSTUCK HA3RSLLEAK HA3RSLRUPT  
 HY\_LIQ\_RSL\_ACT3 1 2 0 HGATE844 HYDRAULICFAIL  
 HGATE682 1 2 1 LAND\_SIGNAL DCPOWER LSEV1L1P2  
 HGATE683 1 2 1 LAND\_SIGNAL DCPOWER LSEV2L1P2  
 HGATE684 1 2 1 LAND\_SIGNAL DCPOWER LSEV3L1P2  
 HGATE685 1 2 1 LAND\_SIGNAL DCPOWER LSEV1L1P1  
 HGATE686 1 2 1 LAND\_SIGNAL DCPOWER LSEV2L1P1  
 HGATE687 1 2 1 LAND\_SIGNAL DCPOWER LSEV3L1P1  
 HGATE688 1 2 1 LAND\_SIGNAL DCPOWER LSEV1DRP1  
 HGATE689 1 2 1 LAND\_SIGNAL DCPOWER LSEV2DRP1  
 HGATE690 1 2 1 LAND\_SIGNAL DCPOWER LSEV3DRP1  
 HGATE691 1 2 1 LAND\_SIGNAL DCPOWER LSEV1DRP2  
 HGATE692 1 2 1 LAND\_SIGNAL DCPOWER LSEV2DRP2  
 HGATE693 1 2 1 LAND\_SIGNAL DCPOWER LSEV3DRP2  
 HGATE694 1 2 1 LAND\_SIGNAL DCPOWER LSEV1L2P1  
 HGATE695 1 2 1 LAND\_SIGNAL DCPOWER LSEV2L2P1  
 HGATE696 1 2 1 LAND\_SIGNAL DCPOWER LSEV3L2P1  
 HGATE697 1 2 1 LAND\_SIGNAL DCPOWER LSEV1L2P2  
 HGATE698 1 2 1 LAND\_SIGNAL DCPOWER LSEV2L2P2  
 HGATE699 1 2 1 LAND\_SIGNAL DCPOWER LSEV3L2P2  
 HGATE700 1 2 1 LAND\_SIGNAL DCPOWER LSEV1L3P2  
 HGATE701 1 2 1 LAND\_SIGNAL DCPOWER LSEV2L3P2  
 HGATE702 1 2 1 LAND\_SIGNAL DCPOWER LSEV3L3P2  
 HGATE703 1 2 1 LAND\_SIGNAL DCPOWER LSEV1L3P1  
 HGATE704 1 2 1 LAND\_SIGNAL DCPOWER LSEV2L3P1  
 HGATE705 1 2 1 LAND\_SIGNAL DCPOWER LSEV3L3P1  
 HGATE706 1 2 1 LAND\_SIGNAL DCPOWER LSEV1UNCP1  
 HGATE707 1 2 1 LAND\_SIGNAL DCPOWER LSEV2UNCP1  
 HGATE708 1 2 1 LAND\_SIGNAL DCPOWER LSEV3UNCP1  
 HGATE709 1 2 1 LAND\_SIGNAL DCPOWER LSEV1UNCP2  
 HGATE710 1 2 1 LAND\_SIGNAL DCPOWER LSEV2UNCP2  
 HGATE711 1 2 1 LAND\_SIGNAL DCPOWER LSEV3UNCP2  
 HGATE712 1 2 1 LAND\_SIGNAL DCPOWER LSEV1L4P1  
 HGATE713 1 2 1 LAND\_SIGNAL DCPOWER LSEV2L4P1  
 HGATE714 1 2 1 LAND\_SIGNAL DCPOWER LSEV3L4P1  
 HGATE715 1 2 1 LAND\_SIGNAL DCPOWER LSEV1L4P2  
 HGATE716 1 2 1 LAND\_SIGNAL DCPOWER LSEV2L4P2  
 HGATE717 1 2 1 LAND\_SIGNAL DCPOWER LSEV3L4P2  
 HGATE718 1 2 1 HGATE720 HY\_LIQ\_BRS L\_WHEEL\_F  
 HGATE719 1 2 1 HGATE721 HY\_LIQ\_BRS R\_WHEEL\_F  
 HGATE720 2 0 2 L\_BKE\_1F L\_BKE\_2F  
 HGATE721 2 0 2 R\_BKE\_1F R\_BKE\_2F  
 HY\_LIQ\_BRS 2 2 0 HGATE722 HGATE723  
 HGATE722 1 2 0 LEFTANTI-SKID HY\_LIQ\_FR\_BCV1  
 HGATE723 1 2 0 RIGHTANTI-SKID HY\_LIQ\_FR\_BCV2  
 HGATE724 1 0 2 BRLSVOMD BRLSVOLECK  
 LEFTANTI-SKID 1 4 0 HGATE724 SIG\_F\_ANTSS LAND&BRAKEAVIONIC DCPOWER  
 SIG\_F\_ANTSS 1 2 0 HGATE732 HGATE733  
 HY\_LIQ\_FR\_BCV1 1 4 0 HGATE738 LAND&BRAKEAVIONIC HYDRAULICFAIL DCPOWER  
 RIGHTANTI-SKID 1 4 0 HGATE729 SIG\_F\_ANTSS LAND&BRAKEAVIONIC DCPOWER

HGATE729 1 0 2 BRRSVOMD BRRSVOLEC  
 HGATE732 2 2 0 HGATE734 HGATE735  
 HGATE733 2 2 0 HGATE736 HGATE737  
 HGATE734 1 0 2 BRASU1NO BRASU1WR  
 HGATE735 1 0 2 BRASU2NO BRASU2WR  
 HGATE736 1 0 2 BRTRS1NO BRTRS1WR  
 HGATE737 1 0 2 BRTRS2NO BRTRS2WR  
 HY\_LIQ\_FR\_BCV2 1 4 0 HGATE739 LAND&BRAKEAVIONIC HYDRAULICFAIL DCPOWER  
 HGATE738 1 0 2 BRBCV1MD BRBCV1LEAK  
 HGATE739 1 0 2 BRBCV2MD BRBCV2LEAK  
 HGATE741 1 2 0 HGATE743 HGATE744  
 HGATE742 2 4 0 NOTUNLOCK1 NOTDOORDN NOTUNLOCK3 NOTWHEELDN  
 HGATE743 2 0 3 NOTUNDSIGL&B1 NOTUNDSIGL&B2 NOTUNDSIGL&B3  
 HGATE744 2 0 3 NOTUNDSIGFMS1 NOTUNDSIGFMS2 NOTUNDSIGFMS3  
 NOTUNLOCK1 1 1 1 HGATE745 NOTUNDUNLOCK1  
 NOTDOORDN 1 1 1 HGATE753 NOTUNDDOOR  
 NOTUNLOCK3 1 1 1 HGATE749 NOTUNDUNLOCK3  
 NOTWHEELDN 1 1 1 HGATE757 NOTUNDWHEEL  
 HGATE745 2 3 0 HGATE746 HGATE747 HGATE748  
 HGATE746 1 0 4 NOTUNDA1LOC1 NOTUNDA2LOC1 NOTUNDS1LOC1 NOTUNDS2LOC1  
 HGATE747 1 0 4 NOTUNDA1LOC1 NOTUNDA3LOC1 NOTUNDS1LOC1 NOTUNDS3LOC1  
 HGATE748 1 0 4 NOTUNDA2LOC1 NOTUNDA3LOC1 NOTUNDS2LOC1 NOTUNDS3LOC1  
 HGATE749 2 3 0 HGATE750 HGATE751 HGATE752  
 HGATE750 1 0 4 NOTUNDA1LOC3 NOTUNDA2LOC3 NOTUNDS1LOC3 NOTUNDS2LOC3  
 HGATE751 1 0 4 NOTUNDA1LOC3 NOTUNDA3LOC3 NOTUNDS1LOC3 NOTUNDS3LOC3  
 HGATE752 1 0 4 NOTUNDA2LOC3 NOTUNDA3LOC3 NOTUNDS2LOC3 NOTUNDS3LOC3  
 HGATE753 2 3 0 HGATE754 HGATE755 HGATE756  
 HGATE754 1 0 4 NOTUNDA1DOOR NOTUNDA2DOOR NOTUNDS1DOOR NOTUNDS2DOOR  
 HGATE755 1 0 4 NOTUNDA1DOOR NOTUNDA3DOOR NOTUNDS1DOOR NOTUNDS3DOOR  
 HGATE756 1 0 4 NOTUNDA2DOOR NOTUNDA3DOOR NOTUNDS2DOOR NOTUNDS3DOOR  
 HGATE757 2 3 0 HGATE758 HGATE759 HGATE760  
 HGATE758 1 0 4 NOTUNDA1WHEEL NOTUNDA2WHEEL NOTUNDS1WHEEL NOTUNDS2WHEEL  
 HGATE759 1 0 4 NOTUNDA1WHEEL NOTUNDA3WHEEL NOTUNDS1WHEEL NOTUNDS3WHEEL  
 HGATE760 1 0 4 NOTUNDA2WHEEL NOTUNDA3WHEEL NOTUNDS2WHEEL NOTUNDS3WHEEL  
 HGATE761 1 4 0 HGATE762 POSTION\_SENSOR\_RE FCAVIONIC DCPOWER  
 HGATE762 1 0 4 HRESVO1MD HRESVO1INSIGN HRESVO1LECK HRESVO1WISIGN  
 POSTION\_SENSOR\_RE 2 2 0 HGATE764 HGATE765  
 HGATE764 1 0 2 HREPOSSSEN1NO HREPOSSSENWO1  
 HGATE765 1 0 2 HREPOSSSEN2NO HREPOSSSENWO2  
 HGATE766 1 4 0 HGATE767 POSTION\_SENSOR\_RE FCAVIONIC DCPOWER  
 HGATE767 1 0 4 HRESVO2MD HRESVO2INSIGN HRESVO2LECK HRESVO2WISIGN  
 HGATE768 1 4 0 HGATE769 POSTION\_SENSOR\_RE FCAVIONIC DCPOWER  
 HGATE769 1 0 4 HRESVO3MD HRESVO3INSIGN HRESVO3LECK HRESVO3WISIGN  
 HGATE770 1 4 0 HGATE771 POSTION\_SENSOR\_LA FCAVIONIC DCPOWER  
 HGATE771 1 0 4 HLASVO1MD HLASVO1INSIGN HLASVO1LECK HLASVO1WISIGN  
 POSTION\_SENSOR\_LA 2 2 0 HGATE773 HGATE774  
 HGATE773 1 0 2 HLAPOSSSEN1NO HLAPOSSSENWO1  
 HGATE774 1 0 2 HLAPOSSSEN2NO HLAPOSSSENWO2  
 HGATE775 1 4 0 HGATE776 POSTION\_SENSOR\_LA FCAVIONIC DCPOWER  
 HGATE776 1 0 4 HLASVO2MD HLASVO2INSIGN HLASVO2LECK HLASVO2WISIGN  
 HGATE777 1 4 0 HGATE778 POSTION\_SENSOR\_LA FCAVIONIC DCPOWER  
 HGATE778 1 0 4 HLASVO3MD HLASVO3INSIGN HLASVO3LECK HLASVO3WISIGN  
 HGATE779 1 4 0 HGATE780 POSTION\_SENSOR\_RA FCAVIONIC DCPOWER  
 HGATE780 1 0 4 HRASVO1MD HRASVO1INSIGN HRASVO1LECK HRASVO1WISIGN  
 POSTION\_SENSOR\_RA 2 2 0 HGATE781 HGATE782  
 HGATE781 1 0 2 HRAPOSSSEN1NO HRAPOSSSENWO1  
 HGATE782 1 0 2 HRAPOSSSEN2NO HRAPOSSSENWO2  
 HGATE783 1 4 0 HGATE784 POSTION\_SENSOR\_RA FCAVIONIC DCPOWER  
 HGATE784 1 0 4 HRASVO2MD HRASVO2INSIGN HRASVO2LECK HRASVO2WISIGN  
 HGATE785 1 4 0 HGATE786 POSTION\_SENSOR\_RA FCAVIONIC DCPOWER  
 HGATE786 1 0 4 HRASVO3MD HRASVO3INSIGN HRASVO3LECK HRASVO3WISIGN  
 HGATE787 1 4 0 HGATE788 POSTION\_SENSOR\_R FCAVIONIC DCPOWER  
 HGATE788 1 0 4 HRSVO1MD HRSVO1INSIGN HRSVO1LECK HRSVO1WISIGN  
 POSTION\_SENSOR\_R 2 2 0 HGATE790 HGATE791  
 HGATE790 1 0 2 HRPOSSSEN1NO HRPOSSSENWO1  
 HGATE791 1 0 2 HRPOSSSEN2NO HRPOSSSENWO2  
 HGATE792 1 4 0 HGATE793 POSTION\_SENSOR\_R FCAVIONIC DCPOWER  
 HGATE793 1 0 4 HRSVO2MD HRSVO2INSIGN HRSVO2LECK HRSVO2WISIGN  
 HGATE794 1 4 0 HGATE795 POSTION\_SENSOR\_R FCAVIONIC DCPOWER  
 HGATE795 1 0 4 HRSVO3MD HRSVO3INSIGN HRSVO3LECK HRSVO3WISIGN  
 HGATE796 1 4 0 HGATE797 POSTION\_SENSOR\_LSP FCAVIONIC DCPOWER  
 HGATE797 1 0 4 HLSPVO1MD HLSPVO1INSIGN HLSPVO1LECK HLSPVO1WISIGN  
 POSTION\_SENSOR\_LSP 2 2 0 HGATE799 HGATE800  
 HGATE799 1 0 2 HLSPPOSSSEN1NO HLSPPOSSSENWO1

HGATE800 1 0 2 HSPPOSSEN2NO HSPPOSSENWO2  
 HGATE801 1 4 0 HGATE802 POSTION\_SENSOR\_LSP FCAVIONIC DCPOWER  
 HGATE802 1 0 4 HSPVO2MD HSPVO2INSIGN HSPVO2LECK HSPSVO2WISIGN  
 HGATE803 1 4 0 HGATE804 POSTION\_SENSOR\_LSP FCAVIONIC DCPOWER  
 HGATE804 1 0 4 HSPVO3MD HSPVO3INSIGN HSPVO3LECK HSPSVO3WISIGN  
 HGATE805 1 4 0 HGATE806 POSTION\_SENSOR\_RSP FCAVIONIC DCPOWER  
 HGATE806 1 0 4 HSPVO1MD HSPVO1INSIGN HSPVO1LECK HSPSVO1WISIGN  
 POSTION\_SENSOR\_RSP 2 2 0 HGATE807 HGATE808  
 HGATE807 1 0 2 HSPPOSSEN1NO HSPPOSSENWO1  
 HGATE808 1 0 2 HSPPOSSEN2NO HSPPOSSENWO2  
 HGATE809 1 4 0 HGATE810 POSTION\_SENSOR\_RSP CAVIONIC DCPOWER  
 HGATE810 1 0 4 HSPVO2MD HSPVO2INSIGN HSPVO2LECK HSPSVO2WISIGN  
 HGATE811 1 4 0 HGATE812 POSTION\_SENSOR\_RSP FCAVIONIC DCPOWER  
 HGATE812 1 0 4 HSPVO3MD HSPVO3INSIGN HSPVO3LECK HSPSVO3WISIGN  
 HGATE813 1 4 0 HGATE814 POSTION\_SENSOR\_LF FCAVIONIC DCPOWER  
 HGATE814 1 0 4 HFSVO1MD HFSVO1INSIGN HFSVO1WISIGN  
 POSTION\_SENSOR\_LF 2 2 0 HGATE816 HGATE81  
 HGATE816 1 0 2 HFPPOSSEN1NO HFPPOSSENWO1  
 HGATE817 1 0 2 HFPPOSSEN2NO HFPPOSSENWO2  
 HGATE818 1 4 0 HGATE819 POSTION\_SENSOR\_LF FCAVIONIC DCPOWER  
 HGATE819 1 0 4 HFSVO2MD HFSVO2INSIGN HFSVO2LECK HFSVO2WISIGN  
 HGATE820 1 4 0 HGATE821 POSTION\_SENSOR\_LF FCAVIONIC DCPOWER  
 HGATE821 1 0 4 HFSVO3MD HFSVO3INSIGN HFSVO3LECK HFSVO3WISIGN  
 HGATE822 1 4 0 HGATE823 POSTION\_SENSOR\_RF FCAVIONIC DCPOWER  
 HGATE823 1 0 4 HFSVO1MD HFSVO1INSIGN HFSVO1WISIGN  
 POSTION\_SENSOR\_RF 2 2 0 HGATE824 HGATE825  
 HGATE824 1 0 2 HRFPOSSEN1NO HRFPOSSENWO1  
 HGATE825 1 0 2 HRFPOSSEN2NO HRFPOSSENWO2  
 HGATE826 1 4 0 HGATE827 POSTION\_SENSOR\_RF FCAVIONIC DCPOWER  
 HGATE827 1 0 4 HFSVO2MD HFSVO2INSIGN HFSVO2WISIGN  
 HGATE828 1 4 0 HGATE829 POSTION\_SENSOR\_RF FCAVIONIC DCPOWER  
 HGATE829 1 0 4 HFSVO3MD HFSVO3INSIGN HFSVO3WISIGN  
 HGATE830 1 4 0 HGATE831 POSTION\_SENSOR\_LSL FCAVIONIC DCPOWER  
 HGATE831 1 0 4 HLSLVO1MD HLSLVO1INSIGN HLSLVO1WISIGN  
 POSTION\_SENSOR\_LSL 2 2 0 HGATE832 HGATE833  
 HGATE832 1 0 2 HLSLPOSSEN1NO HLSLPOSSENWO1  
 HGATE833 1 0 2 HLSLPOSSEN2NO HLSLPOSSENWO2  
 HGATE834 1 4 0 HGATE835 POSTION\_SENSOR\_LSL FCAVIONIC DCPOWER  
 HGATE835 1 0 4 HLSLVO2MD HLSLVO2INSIGN HLSLVO2WISIGN  
 HGATE836 1 4 0 HGATE837 POSTION\_SENSOR\_LSL FCAVIONIC DCPOWER  
 HGATE837 1 0 4 HLSLVO3MD HLSLVO3INSIGN HLSLVO3WISIGN  
 HGATE838 1 4 0 HGATE839 POSTION\_SENSOR\_RSL FCAVIONIC DCPOWER  
 HGATE839 1 0 4 HRSLSVO1MD HRSLSVO1INSIGN HRSLSVO1WISIGN  
 POSTION\_SENSOR\_RSL 2 2 0 HGATE840 HGATE841  
 HGATE840 1 0 2 HRSLPOSSEN1NO HRSLPOSSENW1  
 HGATE841 1 0 2 HRSLPOSSEN2NO HRSLPOSSENWO2  
 HGATE842 1 4 0 HGATE843 POSTION\_SENSOR\_RSL FCAVIONIC DCPOWER  
 HGATE843 1 0 4 HRSLSVO2MD HRSLSVO2INSIGN HRSLSVO2WISIGN  
 HGATE844 1 4 0 HGATE845 POSTION\_SENSOR\_RSL FCAVIONIC DCPOWER  
 HGATE845 1 0 4 HRSLSVO3MD HRSLSVO3INSIGN HRSLSVO3WISIGN  
 PWR\_HIGHAC 1 3 0 GATE204 GATE205 GATE206  
 THRUSTL 1 2 0 ENGINEL FUELENGL  
 THRUSTR 1 2 0 ENGINER FUELENGR  
 ENGINEL 1 2 0 FGATE80 FGATE81  
 FUELENGL 1 1 1 FGATE37 FLLPCCLOSE  
 ENGINER 1 2 0 FGATE109 FGATE110  
 FUELENGR 1 1 1 FGATE44 FRLPCCLOSE  
 F\_AVIONIC 1 4 0 UMS FMS DATA\_BUS FAULUNIT  
 FAULUNIT 2 3 0 FGATE34 FGATE35 FGATE36  
 FGATE34 1 0 2 FFU1NO FFU1WR  
 FGATE35 1 0 2 FFU2NO FFU2WR  
 FGATE36 1 0 2 FFU3NO FFU3WR  
 FGATE37 1 2 0 FGATE38 FGATE43  
 FGATE38 1 3 0 LTEMSENSERS F\_AVIONIC FGATE42  
 LTEMSENSERS 2 2 0 FGATE40 FGATE41  
 FGATE40 1 1 2 DCPOWER FLTEMSEN1NO FLTEMSEN1WR  
 FGATE41 1 1 2 DCPOWER FLTEMSEN2NO FLTEMSEN2WR  
 FGATE42 1 1 1 ACPOWER FLHEATF  
 FGATE43 2 2 0 LL\_FUEL RL\_FUEL  
 LL\_FUEL 1 1 2 FGATE54 FPIP1 FLNRVF  
 RL\_FUEL 1 3 1 FGATE74 FGATE75 RR\_FUEL FPIPCRO  
 FGATE44 1 2 0 FGATE45 FGATE46  
 FGATE45 1 4 0 RTEMSSENSERS F\_AVIONIC FGATE48 DCPOWER  
 FGATE46 2 2 0 RR\_FUEL LR\_FUEL  
 RTEMSSENSERS 2 2 0 FGATE49 FGATE50

FGATE48 1 1 1 ACPOWER FRHEATF  
 FGATE49 1 1 2 DCPOWER FRTESEN1NO FRTESEN1WR  
 FGATE50 1 1 2 DCPOWER FRTESEN2NO FRTESEN2WR  
 RR\_FUEL 1 1 2 FGATE65 FPIP2 FRNRVF  
 LR\_FUEL 1 3 1 FGATE79 FGATE75 LL\_FUEL FPIPCRO  
 FGATE54 1 2 0 FGATE55 FROM\_LTANK  
 FGATE55 1 1 4 ACPOWER FLACPUNO FLACPUSLOW FLACPUSIND FLACPUSEXD  
 FROM\_LTANK 1 2 0 FGATE58 L\_AIRPRESS  
 FGATE58 1 0 3 FLTANMD FLTANRUP FLTANLEAK  
 L\_AIRPRESS 1 3 0 FGATE60 FGATE61 F\_AVIONIC  
 FGATE60 2 2 0 FGATE63 FGATE64  
 FGATE61 1 1 2 DCPOWER FLVENTIN FLVENTOUT  
 FGATE63 1 1 2 DCPOWER FLAPS1NO FLAPS1WR  
 FGATE64 1 1 2 DCPOWER FLAPS2NO FLAPS2WR  
 FGATE65 1 2 0 FGATE66 FROM\_RTANK  
 FGATE66 1 1 4 ACPOWER FRACPUNO FRACPUSLOW FRACPUSIND FRACPUSEXD  
 FROM\_RTANK 1 2 0 FGATE68 R\_AIRPRESS  
 FGATE68 1 0 3 FRTANMD FRTANRUP FRTANLEAK  
 R\_AIRPRESS 1 3 0 FGATE70 FGATE71 F\_AVIONIC  
 FGATE70 2 2 0 FGATE72 FGATE73  
 FGATE71 1 1 2 DCPOWER FRVENTIN FRVENTOUT  
 FGATE72 1 1 2 DCPOWER FRAPS1NO FRAPS1WR  
 FGATE73 1 1 2 DCPOWER FRAPS2NO FRAPS2WR  
 FGATE74 2 0 2 FLPRESEN1 FLPRESEN2  
 FGATE75 1 2 0 FGATE78 F\_AVIONIC  
 FGATE78 1 0 3 FCROVID FCROVED FCROVCL  
 FGATE79 2 0 2 FRPRESEN1 FRPRESEN2  
 RE\_THRUST\_F 1 2 0 FGATE114 FGATE115  
 FGATE80 1 0 6 FENG1COM FENG1CH FENG1TUR FENG1EX FENG1SHA FENG1FAN  
 FGATE81 1 1 2 TOLEFTHP FLSPNOZ FLHPCOCK  
 TOLEFTHP 1 2 0 FGATE84 FGATE108  
 FGATE84 1 2 1 EC\_AVIONIC ECSSENSORSL FLVALF  
 EC\_AVIONIC 1 6 0 DATA\_BUS FMS ADC NAVINFO FADEC TMS  
 FADEC 2 3 0 FGATE90 FGATE91 FGATE92  
 FGATE90 1 0 2 FFADEC1NO FFADEC1WR  
 FGATE91 1 0 2 FFADEC2NO FFADEC2WR  
 FGATE92 1 0 2 FFADEC3NO FFADEC3WR  
 TMS 2 3 0 FGATE95 FGATE96 FGATE97  
 FGATE95 1 0 2 FTMS1NO FTMS1WR  
 FGATE96 1 0 2 FTMS2NO FTMS2WR  
 FGATE97 1 0 2 FTMS3NO FTMS3W  
 ECSSENSORSL 1 3 0 FGATE99 FGATE100 FGATE101  
 FGATE99 2 2 0 FGATE102 FGATE103  
 FGATE100 2 2 0 FGATE104 FGATE105  
 FGATE101 2 2 0 FGATE106 FGATE17  
 FGATE102 1 0 2 FTESEN1NOL FTESEN1WRL  
 FGATE103 1 0 2 FTESEN2NOL FTESEN2WRL  
 FGATE104 1 0 2 FPRSEN1NOL FPRSEN1WRL  
 FGATE105 1 0 2 FPRSEN2NOL FPRSEN2WRL  
 FGATE106 1 0 2 FSPSEN1NOL FSPSEN1WRL  
 FGATE107 1 0 2 FSPSEN2NOL FSPSEN2WRL  
 FGATE108 1 1 4 ACPOWER FLACECPX FLACECPIN FLACECPSLOW FLACECPNO  
 FGATE109 1 0 6 FENG2COM FENG2CH FENG2TUR FENG2EX FENG2SHA FENG2FAN  
 FGATE110 1 1 2 TORIGHTHP FRSPNOZ FRHPCOCK  
 TORIGHTHP 1 2 0 FGATE112 FGATE113  
 FGATE112 1 2 1 EC\_AVIONIC ECSSENSORSR FRVALF  
 FGATE113 1 1 4 ACPOWER FRACECPX FRACECPIN FRACECPSLOW FRACECPNO  
 FGATE114 1 2 0 L\_RTDOOR THRUSTL  
 FGATE115 1 2 0 R\_RTDOOR THRUSTR  
 L\_RTDOOR 1 1 1 FGATE116 L\_RTDOOR  
 R\_RTDOOR 1 1 1 FGATE132 R\_RTDOOR  
 FGATE116 2 3 0 FGATE117 FGATE118 FGATE119  
 FGATE117 1 2 0 FGATE120 HY\_LIQ\_LCDR\_ACT1  
 FGATE118 1 2 0 FGATE122 HY\_LIQ\_LCDR\_ACT2  
 FGATE119 1 2 0 FGATE124 HY\_LIQ\_LCDR\_ACT3  
 FGATE120 1 0 3 HA1LDRSTUCK HA1LDRLECK HA1LDRRUPT  
 HY\_LIQ\_LCDR\_ACT1 1 2 0 FGATE126 HYDRAULICFAIL  
 FGATE122 1 0 3 HA2LDRSTUCK HA2LDRLECK HA2LDRRUPT  
 HY\_LIQ\_LCDR\_ACT2 1 2 0 FGATE129 HYDRAULICFAIL  
 FGATE124 1 0 3 HA3LDRSTUCK HA3LDRLECK HA3LDRRUPT  
 HY\_LIQ\_LCDR\_ACT3 1 2 0 FGATE131 HYDRAULICFAIL  
 FGATE126 1 2 1 DCPOWER LAND&BRAKEAVIONIC RTSEVLDR\_1  
 FGATE129 1 2 1 DCPOWER LAND&BRAKEAVIONIC RTSEVLDR\_2  
 FGATE131 1 2 1 DCPOWER LAND&BRAKEAVIONIC RTSEVLDR\_3  
 FGATE132 2 3 0 FGATE133 FGATE134 FGATE135

FGATE133 1 2 0 FGATE136 HY\_LIQ\_RCDR\_ACT1  
 FGATE134 1 2 0 FGATE138 HY\_LIQ\_RCDR\_ACT2  
 FGATE135 1 2 0 FGATE140 HY\_LIQ\_RCDR\_ACT3  
 FGATE136 1 0 3 HA1RDRSTUCK HA1RDRLECK HA1RDRRUPT  
 HY\_LIQ\_RCDR\_ACT1 1 2 0 FGATE141 HYDRAULICFAIL  
 FGATE138 1 0 3 HA2RDRSTUCK HA2RDRLECK HA2RDRRUPT  
 HY\_LIQ\_RCDR\_ACT2 1 2 0 FGATE143 HYDRAULICFAIL  
 FGATE140 1 0 3 HA3RDRSTUCK HA3RDRLECK HA3RDRRUPT  
 HY\_LIQ\_RCDR\_ACT3 1 2 0 FGATE144 HYDRAULICFAIL  
 FGATE141 1 2 1 DCPOWER LAND&BRAKEAVIONIC RTSEVRDR\_1  
 FGATE143 1 2 1 DCPOWER LAND&BRAKEAVIONIC RTSEVRDR\_2  
 FGATE144 1 2 1 DCPOWER LAND&BRAKEAVIONIC RTSEVRDR\_3  
 NOTRETH 1 2 0 FGATE145 FGATE146  
 FGATE145 1 2 1 RETRLACTS FGATE155 RTLCSTUCK  
 FGATE146 1 2 1 RETRRACTS FGATE155 RTRCSTUCK  
 RETRLACTS 1 3 0 FGATE149 FGATE150 FGATE151  
 RETRRACTS 1 3 0 FGATE152 FGATE153 FGATE154  
 FGATE149 2 0 2 RETHLACT1ST RETHLACT2ST  
 FGATE150 2 0 2 RETHLACT1ST RETHLACT3ST  
 FGATE151 2 0 2 RETHLACT2ST RETHLACT3ST  
 FGATE152 2 0 2 RETHRACT1ST RETHRACT2ST  
 FGATE153 2 0 2 RETHRACT1ST RETHRACT3ST  
 FGATE154 2 0 2 RETHRACT2ST RETHRACT3ST  
 FGATE155 1 2 0 FGATE156 FGATE157  
 FGATE156 2 0 3 RETHSIGNL&BU1 RETHSIGNL&BU2 RETHSIGNL&BU3  
 FGATE157 2 0 3 RETHSIGNFMS1 RETHSIGNFMS2 RETHSIGNFMS3  
 ECSENSORSR 1 3 0 FGATE158 FGATE159 FGATE160  
 FGATE158 2 2 0 FGATE161 FGATE162  
 FGATE159 2 2 0 FGATE163 FGATE164  
 FGATE160 2 2 0 FGATE165 FGATE166  
 FGATE161 1 0 2 FTEMSSEN1NOR FTEMSSEN1WRR  
 FGATE162 1 0 2 FTEMSSEN2NOR FTEMSSEN2WRR  
 FGATE163 1 0 2 FPRSEN1NOR FPRSEN1WRR  
 FGATE164 1 0 2 FPRSEN2NOR FPRSEN2WRR  
 FGATE165 1 0 2 FSPSEN1NOR FSPSEN1WRR  
 FGATE166 1 0 2 FSPSEN2NOR FSPSEN2WRR  
 FUELFLOWHIL 1 1 1 FGATE171 FLACECHI  
 NOTCLENFLOW 1 2 1 ECSENSORSR F\_AVIONIC FLECVOPEN  
 FUELFLOWHIR 1 1 1 FGATE174 FRACECHI  
 NOTCRENFLOW 1 2 1 ECSENSORSR F\_AVIONIC FRECVOPEN  
 FGATE171 2 2 0 FGATE172 FGATE173  
 FGATE172 1 2 1 FGATE74 F\_AVIONIC FLLPCOPEN  
 FGATE173 1 0 2 FLACPUHI FRACPUHI  
 FGATE174 2 2 0 FGATE175 FGATE173  
 FGATE175 1 2 1 FGATE79 F\_AVIONIC FLRPCOPEN  
 LEFT\_FCDC 1 3 0 EGATE4 LEFT\_PWRS EGATE93  
 CEN\_FCDC 1 3 0 EGATE68 CEN\_PWRS EGATE94  
 RIGHT\_FCDC 1 3 0 RIGHT\_PWRS EGATE90 EGATE95  
 EGATE4 1 0 4 ELLBUSNO ELLBUSLOW ELLBUSMD ELLBUSOH  
 LEFT\_PWRS 2 4 0 LVDC\_BUS L\_PMG HOT\_BBUS CEN\_PWRS2  
 LVDC\_BUS 1 2 0 EGATE9 EGATE10  
 L\_PMG 1 0 5 ELLPMGMD ELLPMGNO ELLPMGLOW ELLPMGPU ELLPMGOV  
 HOT\_BBUS 1 2 0 EGATE64 EGATE65  
 EGATE9 1 0 4 ELLVDCNO ELLVDCLOW ELLVDCMD ELLVDCOV  
 EGATE10 2 2 0 L\_TRU RVDC\_BUS2  
 L\_TRU 1 2 0 EGATE13 EGATE14  
 RVDC\_BUS2 1 2 0 EGATE55 EGATE56  
 EGATE13 1 0 4 ELLTRUNO ELLTRUMD ELLTRUWR ELLTRUOV  
 EGATE14 2 2 0 LAC\_XTRBUS GEN\_RAT  
 LAC\_XTRBUS 1 2 0 EGATE17 EGATE18  
 GEN\_RAT 1 0 6 EL\_RATOV EL\_RATNO EL\_RATMD EL\_RATNOE EL\_RATLOW EL\_RATPU  
 EGATE17 1 0 4 ELLACXTRBNO ELLACXTRBLOW ELLACXTRBMD ELLACXTRBOV  
 EGATE18 2 5 0 L\_LGEN L\_APUGEN L\_RGEN L\_BKUPGEN RAC\_XTRBUS2.  
 L\_LGEN 1 2 0 EGATE24 EGATE25  
 L\_APUGEN 1 3 0 EGATE26 EGATE27 EGATE47  
 L\_RGEN 1 4 0 EGATE28 EGATE29 EGATE48 EGATE49  
 L\_BKUPGEN 1 2 0 EGATE30 EGATE31  
 RAC\_XTRBUS2 1 4 0 EGATE36 EGATE37 EGATE30 EGATE46  
 EGATE24 1 2 1 EL\_AVIONIC CPU ELLGCBO  
 EGATE25 1 0 5 ELL\_MGENMD ELL\_MGENNO ELL\_MGENLOW ELL\_MGENPU ELL\_MGENOV  
 EGATE26 1 2 1 EL\_AVIONIC CPU ELAPBOPN  
 EGATE27 1 0 5 ELAPUMD ELAPUNO ELAPULOW ELAPUPU ELAPUOV  
 EGATE28 1 2 1 EL\_AVIONIC CPU EL\_RGCBOPEN  
 EGATE29 1 0 5 ELRMGENMD ELRMGENNO ELRMGENLOW ELRMGENPU ELRMGENOV  
 EGATE30 1 2 1 EL\_AVIONIC CPU ELLBUSTCOPEN

EGATE31 1 1 1 EGATE33 ELVSCFFAIL

EGATE33 2 2 0 EGATE34 EGATE35  
EGATE34 1 0 5 ELLBUGENMD ELLBUGENNO ELLBUGENLOW ELLBUGENPU ELLBUGENOV  
EGATE35 1 0 5 ELRBUGENMD ELRBUGENNO ELRBUGENLOW ELRBUGENPU ELRBUGENOV  
EGATE36 1 0 4 ELRACXTRBNO ELRACXTRBLOW ELRACXTRBMD ELRACXTRBOV  
EGATE37 2 4 0 R\_LGEN R\_BKUPGEN R\_APUGEN R\_RGEN  
R\_LGEN 1 4 0 EGATE25 EGATE24 EGATE48 EGATE49  
R\_BKUPGEN 1 2 0 EGATE46 EGATE31  
R\_APUGEN 1 3 0 EGATE26 EGATE48 EGATE27  
R\_RGEN 1 2 0 EGATE29 EGATE28  
EGATE46 1 2 1 EL\_AVIONIC CPU ELRBUSTOPEN  
EGATE47 1 2 1 EL\_AVIONIC CPU ELLBTBOPEN  
EGATE48 1 2 1 EL\_AVIONIC CPU ELRBTBOPEN  
EGATE49 1 2 1 EL\_AVIONIC CPU ELLBTBOPEN  
EGATE55 1 0 4 ELRVDCBUSNO ELRVDCBUSLOW ELRVDCBUSMD ELRVDCBUSOV  
EGATE56 1 2 0 EGATE57 EGATE58  
EGATE57 2 2 0 GEN\_RAT RAC\_XTRBUS  
EGATE58 1 0 4 ELTRU2NO ELTRU2MD ELTRU2CON ELTRU2OV  
EGATE64 1 0 4 ELHOTBATNO ELHOTBATLOW ELHOTBATMD ELHOTBATOV  
EGATE65 2 2 0 BATTBUS MAINBATT  
BATTBUS 1 0 4 ELBATBUSNO ELBATBUSLOW ELBATBUSMD ELBATBUSOV  
MAINBATT 1 0 4 ELMANBATNO ELMANBATLOW ELMANBATMD ELMANBATOV  
EGATE68 1 0 4 ELCBUSNO ELCBUSLOW ELCBUSMD ELCBUSOV  
CEN\_PWRS 2 4 0 L\_PMG HOT\_BBUS R\_PMG LEFT\_PWRS2  
R\_PMG 1 0 5 ELRPMGMD ELRPMGNO ELRPMGLOW ELRPMGPU ELRPMGOV  
CEN\_PWRS2 2 3 0 L\_PMG HOT\_BBUS R\_PMG  
LEFT\_PWRS2 2 3 0 LVDC\_BUS L\_PMG HOT\_BBUS  
RIGHT\_PWRS 2 2 0 R\_PMG RVDC\_BUS  
EGATE92 2 2 0 EGATE77 LVDC\_BUS2  
EGATE77 1 2 0 EGATE58 EGATE112  
LVDC\_BUS2 1 2 0 EGATE9 L\_TRU  
RAC\_XTRBUS 1 2 0 EGATE79 EGATE36  
EGATE79 2 5 0 LAC\_XTRBUS2 R\_LGEN R\_BKUPGEN R\_APUGEN R\_RGEN  
LAC\_XTRBUS2 1 4 0 EGATE120 EGATE30 EGATE46 EGATE17  
EGATE82 2 3 0 LEFT\_FCDC CEN\_FCDC RIGHT\_FCDC  
PWR\_HIGHDC 1 3 0 GATE1 GATE2 GATE3  
EL\_AVIONIC 1 4 0 FMS UMS DATA\_BUS EL\_UNIT  
EL\_UNIT 2 3 0 EGATE87 EGATE88 EGATE89  
EGATE87 1 0 2 AVELUN1NO AVELUN1WR  
EGATE88 1 0 2 AVELUN2NO AVELUN2WR  
EGATE89 1 0 2 AVELUN3NO AVELUN3WR  
EGATE90 1 0 4 ELRBUSNO ELRBUSLOW ELRBUSMD ELRBUSOH  
RVDC\_BUS 1 2 0 EGATE55 EGATE92  
EGATE93 1 1 1 EL\_AVIONIC ELLCBROPEN  
EGATE94 1 1 1 EL\_AVIONIC ELCCBROPEN  
EGATE95 1 1 1 EL\_AVIONIC ELRCBROPEN  
EGATE96 2 3 0 LACMAINBUS RACMAINBUS SBACMAINBUS  
LACMAINBUS 1 2 0 EGATE99 EGATE103  
RACMAINBUS 1 2 0 EGATE100 EGATE104  
SBACMAINBUS 1 2 0 EGATE101 ACSTBY  
EGATE99 1 0 4 ELLACBUSNO ELLACBUSLOW ELLACBUSMD ELLACBUSOV  
EGATE100 1 0 4 ELRACBUSNO ELRACBUSLOW ELRACBUSMD ELRACBUSOV  
EGATE101 1 0 4 ELSBACBUSNO ELSBACBUSLOW ELSBACBUSMD ELSBACBUSOV  
ACSTBY 2 2 0 LAC\_XTRBUS EGATE109  
EGATE103 2 3 0 L\_LGEN L\_RGEN L\_APUGEN  
EGATE104 2 3 0 R\_LGEN R\_RGEN R\_APUGEN  
EGATE109 1 2 0 EGATE110 BATTERYBUSAC  
EGATE110 1 0 4 ELINVERNO ELINVERMD ELINVERWR ELINVEROV  
BATTERYBUSAC 1 3 0 MAINBATT EGATE64 BATTBUS  
EGATE112 2 2 0 RAC\_XTRBUS GEN\_RAT  
CPU 2 3 0 EGATE117 EGATE118 EGATE119  
EGATE117 1 0 2 ELCPU1NO ELCPU1WR  
EGATE118 1 0 2 ELCPU2NO ELCPU2WR  
EGATE119 1 0 2 ELCPU3NO ELCPU3WR  
EGATE120 2 4 0 L\_LGEN L\_APUGEN L\_RGEN L\_BKUPGEN  
GATE1 2 2 0 L\_PWRS\_HIGH GATE56  
GATE2 2 2 0 C\_PWRS\_HIGH GATE57  
GATE3 2 2 0 R\_PWRS\_HIGH GATE58  
L\_PWRS\_HIGH 1 2 2 LVDCBUSHI CPWRSHI ELLPMGHI ELMBATTHI  
LVDCBUSHI 1 2 0 R\_TRU GATE11  
CPWRSHI 1 0 2 ELLPMGHI ELRPMGHI  
GATE11 1 1 1 LACXTRBUSHI ELRATHI  
LACXTRBUSHI 1 1 1 GATE14 ELACLXTRUWR  
GATE14 1 5 0 L\_LMAINGENHI L\_APUHI L\_RMAINGENHI L\_BKUPGENHI RACXTRBUSHI2

L\_LMAINGENHI 2 1 1 GATE20 ELLMGENHI  
 L\_APUHI 2 2 1 GATE22 GATE23 ELAPUGENHI  
 L\_RMAINGENHI 2 3 1 GATE24 GATE22 GATE59 ELRMGENHI  
 L\_BKUPGENHI 2 2 0 GATE26 GATE61  
 RACXTRBUSHI 2 3 0 GATE60 GATE26 GATE41  
 GATE20 1 2 1 EL\_AVIONIC CPU ELLGCBBCLOSE  
 GATE22 1 2 1 EL\_AVIONIC CPU ELLBTBCLOSE  
 GATE23 1 2 1 EL\_AVIONIC CPU ELAPBCLOSE  
 GATE24 1 2 1 EL\_AVIONIC CPU ELRBTBCLOSE  
 GATE26 1 2 1 EL\_AVIONIC CPU ELLDCBUSCC  
 R\_LMAINGENHI 2 3 1 GATE20 GATE22 GATE24 ELLMGENHI  
 R\_LAPUGENHI 2 2 1 GATE23 GATE24 ELAPUGENHI  
 R\_BKUPGENHI 2 2 0 GATE41 GATE61  
 GATE33 1 0 2 ELLBUPGENHI ELRBUPGENHI  
 GATE41 1 2 1 EL\_AVIONIC CPU ELRDCBUSCC  
 R\_TRU 1 1 1 GATE43 ELTRUWR  
 GATE43 1 1 1 RACXTRBUSHI ELRATHI  
 RACXTRBUSHI 1 1 1 GATE45 ELACRXTRUWR  
 GATE45 1 5 0 R\_APUHI R\_RMAINGENHI R\_BKUPGENHI LACXTRBUSHI2 R\_LMAINGENHI  
 R\_APUHI 2 2 1 GATE23 GATE24 ELAPUGENHI  
 R\_RMAINGENHI 2 1 1 GATE59 ELRMGENHI  
 C\_PWRS\_HIGH 1 1 3 GATE54 ELRPMGHI ELLPMGHI ELMBATTHI  
 R\_PWRS\_HIGH 1 1 1 RVDCBUSHI ELRPMGHI  
 GATE54 1 1 1 LVDCBUSHI ELLPMGHI  
 RVDCBUSHI 1 2 0 R\_TRU GATE11  
 GATE56 1 1 1 EL\_AVIONIC ELLCBCLOSE  
 GATE57 1 1 1 EL\_AVIONIC ELCCBCLOSE  
 GATE58 1 1 1 EL\_AVIONIC ELRCBCLOSE  
 GATE59 1 2 1 EL\_AVIONIC CPU ELRGCBBCLOSE  
 GATE60 1 4 1 R\_LMAINGENHI R\_LAPUGENHI R\_BKUPGENHI R\_RMAINGENHI ELACRXTRUWR  
 GATE61 1 1 1 GATE33 ELVSCFWR  
 LACXTRBUSHI 2 2 3 0 GATE62 GATE26 GATE41  
 GATE62 1 4 1 L\_LMAINGENHI L\_APUHI L\_RMAINGENHI L\_BKUPGENHI ELACLXTRUWR  
 GATE201 1 3 0 L\_LMAINGENHI L\_APUHI L\_RMAINGENHI  
 GATE202 1 3 0 R\_APUHI R\_RMAINGENHI R\_LMAINGENHI  
 GATE203 1 1 1 LACXTRBUSHI ELMBATTHI  
 GATE204 2 2 0 GATE201 GATE207  
 GATE205 2 2 0 GATE202 GATE208  
 GATE206 2 2 0 GATE203 GATE209  
 GATE207 1 1 1 EL\_AVIONIC EVENT1  
 GATE208 1 1 1 EL\_AVIONIC ELRCBACCLOSE  
 GATE209 1 1 1 EL\_AVIONIC ELSCBACCLOSE

### Appendix C: Results for search lookup technique on Method 3

Mission Set	Configuration	Standard search	Map search	No search	Number Of Line Created For no search	Number Of finds	Number Of Line created
1	1,2,4	0.53s (1.19s)	1.03s (4.14s)	0.09s (1.08s)	15692 1162	224 5	15411 1157
1	2,1,5	0.84s (1.19s)	1.55s (6.5s)	0.23s (1.17s)	17477 2372	0 5	17419 2367
1	1,4,5	1.02s (1.83s)	1.40s (4.28s)	0.13s (1.28s)	21817 1298	4 5	21762 1293
1	1,2,7	1.05s (1.30s)	1.98s (17.41s)	0.16s (1.28s)	22284 1865	264 5	21907 1860
1	1,2,3	1.97s (1.31s)	2.03s (4.63s)	0.17s (1.45s)	30264 2368	0 149	29693 2219
1	4,3,1	2.23s (1.41s)	2.55s (4.72s)	0.16s (1.38s)	35832 1770	80 53	32083 1717
1	2,4,6	3.52s (2.03s)	2.88s (4.70s)	0.27s (2.06s)	42382 1409	36 0	42046 1409
1	2,3,5	29.61s (2.53s)	16.11s (7.84s)	3.11s (6.28s)	379223 3863	250609 144	94520 3719
1	2,3,5,1	33.13s (2.52s)	17.16s (7.81s)	3.14s (6.42s)	389120 4003	250609 149	102320 3854
1	1,2,4,6	35.51s (3.38s)	9.38s (5.94s)	0.72s (3.02s)	135711 1921	152 5	133895 1916
1	5,6,1,2	1min 35.85s (7.88s)	37.19s (6.99s)	15.94s (4.26s)	206310 3070	1340 5	200663 3065
1	6,2,3	1min 50.29s (4.44s)	19.54s (7.27s)	1.75s (4.47s)	228640 2894	305 176	223907 2718
1	2,3,4,1	4min 16.04s (7.86s)	32.27s (9.25s)	2.73s (6.39s)	384174 3038	2592 149	352279 2889
1	1,2,3,4	5min 13.8s (9.38s)	34.13s (10.27s)	3.22s (7.11s)	422471 3038	2624 149	390599 2889
1	2,3,5,1,4	13min 55.62s (12.91s)	1min 33.55s (1min 28.99s)	1min 50.29s (50.6s)	3383800 4702	253461 149	626151 4553
1	3,4,8	19min 45.63s (14.49s)	1min 19.31s (17.5s)	6.31s (16.19s)	807787 1790	96 32	721423 1758



1	2,3,6,1	24min 17.06s (15.62s)	1min 33.0s (16.28s)	7.79s (15.49s)	826404 3294	11628 185	807757 3109
1	1,2,7,3	29min 31.61s (17.17s)	1min 41.71s (19.47s)	8.00s (16.14s)	940304 3481	1560 149	906290 3323
1	2,4,6,5	42min 56.05s (20.27s)	1min 48.76s (20.79s)	11.89s (26.20s)	1126521 3531	1512 0	1122339 3531
1	2,4,6,5,1	1hour 14min 10.45s (24.09s)	2min 21.09s (24.9s)	18.23s (26.11s)	1377530 3832	5564 5	1367647 3827
1	1,2,3,4,5	1 hour 19 min 56.24s (63.71s)	9min 20.42s (22.11s)	M	M	3701500 149	1214248 4553
1	3,1,6,4,2	3hour 1min 45.49s (37.08s)	4min 31.87s (39.36s)	36.26s (34.54s)	2332759 3979	39496 185	2204664 3794
1	5,1,4,2,6	1 hour 53min 26.2s (31.64s)	3min 52.5s (31.36s)	29.84s (32.29s)	1793590 3832	894 5	1791414 3827
1	1,2,4,6,5	1 hour 14min 50s (25.84s)	2min 46.48s (26.01s)	32.75s (25.14s)	1468707 3832	1592 5	1462482 3827

Mission Set	Configuration	Standard search	Map search	No search	Number Of Line Created For no search	Number Of finds	Number Of Line created
2	1,2,5	0.44s (2.56s)	0.81s (2.52s)	0.11s (2.41s)	12560 1158	0 0	12560 1158
2	1,2,3	0.86s (2.58s)	1.25s (2.58s)	0.16s (2.55s)	18446 1415	72 0	18374 1415
2	1,2,5,6	8.55s (3.69s)	5.03s (4.61s)	0.50s (3.47s)	63799 1363	1296 0	62503 1363
2	1,4,5,6	16.69s (3.95s)	6.61s (3.64s)	0.67s (3.91s)	92920 (991)	0 0	92920 991
2	3,2,5,6	1min 8.47s (4.88s)	16.97s (4.81s)	8.48s (8.75s)	428579 1826	24208 0	175843 1826
2	3,4,6,1	1min 28.56s (5.17s)	24.55s (5.20s)	10.64s (5.74s)	233949 1615	20 16	199723 1501
2	1,2,3,4	2min 31.73s (6.48s)	21.97s (6.58s)	2.06s (6.98s)	314609 1962	900 16	268475 1848
2	3,4,5,6	2min 50.59s (6.52s)	29.09s (6.50s)	34.17s (1 min 0.03s)	713595 1493	28380 8	276505 1443
2	1,2,3,4,5	M	M	M	M	M	M
2	1,2,5,6,3	23min 21.74s (15.45s)	1min 29.07s (14.34s)	10.08s (16.33s)	860170 2356	56256 0	784714 2356

Mission Set	Configuration	Standard search	Map search	No search	Number Of Line Created For no search	Number Of finds	Number Of Line created
3	1,2,3	0.01s (1.51s)	0.09s (1.50s)	0.02s (1.97s)	1460 488	0 7	1428 480
3	1,2,3,4	0.03s (1.52s)	0.22s (1.52s)	0.03s (1.97s)	2714 666	28 16	2590 643
3	1,2,3,4,5	0.11s (1.86s)	0.44s (1.63s)	0.08s (2.05s)	6898 918	748 18	5797 893
3	1,2,3,4,5,6	1.92s (1.89s)	1.97s (1.87s)	0.28s (2.38s)	33729 1103	748 32	27215 1062
3	1,2,3,4,5,6 1,2	2.40s (1.94s)	2.38s (1.94s)	0.36s (2.48s)	44499 1103	1872 32	32533 1062
3	1,2,3,4,5,6 1,2,3	6.63s (2.16s)	4.61s (2.14s)	1.58s (3.01s)	80596 1103	6876 32	50871 1062
3	1,2,3,4,5,6 1,2,3,4	18.44 (2.61s)	10.97s (2.59s)	5.13s (5.53s)	288219 1103	83376 32	89821 1062
3	1,2,3,4,5,6 1,2,3,4,5	33.05s (2.95s)	17.52s (3.00s)	46.19s (11.47s)	777157 1103	163521 32	115378 1062
3	1,2,3,4,5,6 1,2,3,4,5,6	48.31s (3.22s)	23.91s (3.22s)	M	M	242630 32	136698 1062
3	1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6	2min 29.75s (4.48s)	50.16s (4.52s)	M	M	484512 32	246181 1062
3	1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6	4min 50.89s (5.73s)	4min 56.78s (5.95s)	M	M	484512 32	246181 1062
4	1,2,3	0.17s (1.73s)	0.64s (1.73s)	0.08s (1.75s)	8212 1042	918 46	6670 977
4	1,2,3,4	2.03s (2.00s)	2.13s (1.97s)	0.44s (2.20s)	44450 1374	6652 51	27751 1303
4	1,2,3,4,5	32.31s (3.17s)	13.47s (3.16s)	2.63s (5.58s)	324259 1673	73857 65	125694 1549
4	1,2,3,4,5 1,2,3	3min 23.43s (5.38s)	49.0s (5.4s)	3min 29.7s (37.8s)	2828124 1673	447064 65	307983 1549
4	1,2,3,4,5 1,2,3,4,5	8min 0.94s (7.89s)	1min 26.59s (7.28s)	M	M	877622 65	468030 1549
4	1,2,3,4,5 1,2,3,4,5	23min 54s	2min 39.12s	M	M	1681387 65	810366 1549

	1,2,3,4,5	(11.41s)	(11.44s)				
5	1,2,3,4,5,6	0.03s (1.66s)	0.13s (1.66s)	0.02s (1.66s)	1941 239	129 4	1466 237
5	1,4,3,2,3,5 6,1	0.05s (1.66s)	0.16s (1.66s)	0.03s (1.67s)	2776 239	183 4	1993 237
5	1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6	0.13s (1.72s)	0.55s (1.72s)	0.22s (1.88s)	16987 239	493 4	5116 237
5	1,5,4,6,2,1 3,2,3,4,5,3 1,3,5,6,3,3 2,1,2,3,	0.17s (1.75s)	0.63s (1.73s)	0.34s (2.03s)	26931 239	656 4	6507 237
5	1,5,4,6,2,1 3,2,3,4,5,3 1,3,5,6,3,3 2,1,2,3,1,2 3,4	0.22s (1.75s)	0.58s (1.77s)	0.52s (2.2s)	43093 239	825 4	7765 237
5	1,5,4,6,2,1 3,2,3,4,5,3 1,3,5,6,3,3 2,1,2,3,1,2 3,4,2,3,4,5 3,5,6,3	0.29s (1.76s)	1.00s (1.78s)	1.09s (2.63s)	75787 239	1015 4	10225 237
6	1,2	0.001s (4.33s)	0.06s (4.36s)	0.02s (4.27s)	867 433	1 83	866 433
6	1,7	0.02s (4.28s)	0.09s (4.36s)	0.02s (4.28s)	1504 927	143 163	1281 847
6	3,5	0.001s (4.31s)	0.08s (4.31s)	0.02s (4.28s)	1153 595	4 83	1149 595
6	3,4	0.016s (4.27s)	0.08s (4.34s)	0.001s (4.25s)	1227 651	32 83	1193 651
6	3,7	0.016s (4.28s)	0.16s (4.36s)	0.001s (4.31s)	1789 1084	9 163	1698 1004
6	5,4	0.001s (4.28s)	0.08s (4.31s)	0.02s (4.26s)	1175 613	8 84	1163 611
6	6,7	0.01s (4.25s)	0.14s (4.36s)	0.001s (4.23s)	1691 1002	3 163	1606 922
6	4,7	0.01s (4.28s)	0.19s (4.33s)	0.02s (4.28s)	1886 1129	3 163	1801 1049
6	1,2,8	0.02s (4.31s)	0.08s (4.33s)	0.01s (4.26s)	1301 433	2 83	1299 433
6	1,2,3	0.02s (4.27s)	0.08s (4.27s)	0.01s (4.26s)	1413 517	28 83	1385 517
6	1,3,6	0.01s (5.06s)	0.09s (4.41s)	0.01s (4.25s)	1653 613	30 83	1623 613
6	1,3,5,4	0.02s (4.33s)	0.13s (4.33s)	0.01s (4.29s)	2343 669	41 84	2296 667
6	3,6,4,7	0.03s (4.33s)	0.16s (4.34s)	0.02s (4.36s)	3234 1112	26 169	3104 1030

6	3,5,6,4,7	0.03s (4.50s)	0.28s (4.39s)	0.03s (4.38s)	4000 1129	39 164	3836 1047
6	1,2,3,4,5,6 7	0.05s (4.47s)	0.31s (4.48s)	0.05s (4.36s)	4901 1130	67 164	4709 1048
6	1,2,3,4,5,6 7,8,9	0.06s (4.48s)	0.39s (4.59s)	0.05s (4.45s)	7601 1130	93 164	7126 1048
6	1,2,3,4,5,6 7,8,9,1,2,3,4, 5,6,7,8,9	0.31s (4.97s)	1.56s (4.95s)	0.14s (4.91s)	21104 1130	409 164	18117 1048
6	1,2,3,4,5,6 7,8,9,1,2,3,4, 5,6,7,8,9,1,2 ,3,4,5,6,7,8,9	0.73s (5.50s)	2.59s (6.13s)	0.29s (5.59s)	39968 1130	725 164	29108 1048
7	1,2,3,4,5,6,7 8,9	0.09s (5.56s)	0.55s (5.68s)	0.06s (5.68s)	11586 1741	93 663	10169 1382
7	1,2,3,4,5,6,7 8,9,1,2,3,4,5, 6,7,8,9	0.45s (6.92s)	1.61s (6.69s)	0.17s (7.24s)	31271 1741	524 663	24481 1382
7	1,2,3,4,5,6,7 8,9,1,2,3,4,5 ,6,7,8,9,1,2,3 ,4,5,6,7,8,9	1.09s (7.61s)	3.39s (7.72s)	0.41s (10.59s)	57851 1741	955 663	38793 1382
8	1,2,3,4,5,6,7 8,9	0.11s (7.44s)	0.64s (7.80s)	0.08s (7.48s)	13390 1956	94 1569	11429 1491
8	1,2,3,4,5,6,7 8,9,1,2,3,4,5	0.27s (8.17s)	0.92s (8.04s)	0.14s (8.11s)	24684 1956	128 1569	19738 1491
8	1,2,3,4,5,6,7 8,9,1,2,3,4,5, 6,7,8,9	0.48s (8.77s)	1.70s (8.79s)	0.23s (9.31s)	36994 1956	471 1569	26545 1491
8	1,2,3,4,5,6,7 8,9,1,2,3,4,5, 6,7,8,9,1,2,3, 4,5	0.77s (9.29s)	3.83s (9.49s)	0.44s (12.58s)	53286 1956	505 1569	34854 1491
8	1,2,3,4,5,6,7 8,9,1,2,3,4,5, 6,7,8,9,1,2,3,4, 5,6,7,8,9	1.08s (9.50s)	2.86s (9.77s)	0.64s (11.28s)	75595 1956	848 1569	42661 1491

#### Appendix D: Results for search lookup technique on Method 2

Mission Set	Configuration	Standard search	Map search	No search	Number Of Line Created For no search	Number Of finds	Number Of Line created
1	1,2,4	0.29s (0.27s)	0.61s (0.22s)	0.05s (0.20s)	11280 1501	116 5	11107 1496
1	2,1,5	0.08s (0.17s)	0.34s (0.14s)	0.03s (0.19s)	5858 2440	0 5	5800 2435
1	1,4,5	8.45s (0.85s)	4.66s (0.88s)	0.44s (0.83s)	65648 2849	4 5	65593 2844
1	1,2,7	0.86s (0.29s)	1.16s (0.31s)	0.39s (8.73s)	20051 1943	264 5	19674 1938
1	1,2,3	1.75s (0.41s)	1.72s (0.41s)	0.16s (0.41s)	28331 2340	0 149	28042 2191
1	4,3,1	9.63s (0.88s)	4.81s (0.88s)	0.61s (1.21s)	78765 2750	40 149	67664 2601
1	2,4,6	12.64s (0.98s)	5.23s (0.98s)	0.38s (1.00s)	80415 2443	108 54	78071 2384
1	2,3,5	12.42s (0.78s)	9.19s (0.78s)	2.02s (2.66s)	227977 3910	153221 144	61724 3766
1	2,3,5,1	14.5s (0.88s)	9.56s (0.92s)	2.13s (2.86s)	237914 4043	153221 149	69564 3894
1	1,2,4,6	33.82s (1.55s)	9.52s (1.66s)	0.73s (1.64s)	132815 2576	152 59	130113 2512
1	5,6,1,2	1min 52.24s (2.88s)	18.95s (2.78s)	1.50s (2.92s)	251418 3515	59 1340	233275 3451
1	6,2,3	4min 57.53s (4.39s)	39.16s (4.34s)	3.81s (4.64s)	380924 3282	525 198	372684 3079
1	2,3,4,1	2min 56.2s (3.45s)	23.43s (3.45s)	2.94s (3.58s)	309042 3104	2592 149	291549 2955
1	1,2,3,4	4min 2.15s (3.89s)	28.00s (3.86s)	2.72s (4.02s)	347197 3104	2624 149	329727 2955
1	2,3,5,1,4	3min 34.94s (3.70s)	32.29s (3.69s)	44.5s (21.7s)	1877198 4806	154813 149	319389 4657
1	3,4,8	M	M	M	M	M	M
1	2,3,6,1	24min 56.84s (9.61s)	1min 38.69s (9.58s)	8.56s (9.63s)	826538 3415	11628 203	807845 3207

1	1,2,7,3	20min 33.37s (8.63s)	1min 21.69s (8.59s)	8.59s (8.79s)	757060 3546	1032 149	732934 3397
1	2,4,6,5	1 hour 13min 4.15s (17.12s)	2min 27.47s (17.08s)	20.47s (18.08s)	1562870 4145	324 54	1462239 4086
1	2,4,6,5,1	1 hour 15min 9.51s (17.13s)	2min 30.68s (17.13s)	21.79s (18.33s)	1588174 4278	4340 59	1483079 4214
1	1,2,3,4,5	35min 52.69s (9.36s)	5min 17.22s (9.27s)	2min 49.24s (42.75s)	3732159 4806	2404560 149	813546 4657
1	3,1,6,4,2	3hour 9min 0.27s (9.61s)	5min 14.33s (26.66s)	37.75s (26.24s)	2333394 4179	39496 203	2205206 3971
1	5,1,4,2,6	1 hour 32 min 47.43s (19.36s)	4min 29.9s (19.38s)	15.47s (20.27s)	1706491 4278	474 59	1601383 4214
1	1,2,4,6,5	1 hour 25min 6.99s (20.94s)	2min 56.7s (3min 1.11s)	36.25s (19.94s)	1678822 4278	368 59	1577444 4214
2	1,2,5	0.03s (0.09s)	0.14s (0.11s)	0.03s (0.08s)	2630 1105	0 0	2630 1105
2	1,2,3	0.06s (0.11s)	0.22s (0.11s)	0.03s (0.11s)	4084 1309	12 98	3974 1211
2	1,2,5,6	1.23s (0.38s)	1.34s (0.34s)	0.11s (0.31s)	22880 1269	360 0	22520 1269
2	1,4,5,6	6min (4.89s)	48.95s (4.89s)	8.79s (4.88s)	427868 1834	0 0	427868 1934
2	3,2,5,6	23.37s (1.19s)	8.03s (1.19s)	7.44s (4.42s)	389487 2115	25660 98	99975 2075
2	3,4,6,1	9min 33.1s (5.89s)	1min 0.13s (5.89s)	22.52s (6.92s)	612629 2039	100 98	517637 1941
2	1,2,3,4	14.14s (1.03s)	5.72s (0.98s)	0.53s (1.19s)	93442 2002	444 98	81978 1904
2	3,4,5,6	M	M	M	M	M	M
2	1,2,3,4,5	9min 43.54s (4.94s)	2min 20.51s (4.98s)	46.45s (20.7s)	1832482 2712	1034824 98	430516 2614
2	1,2,5,6,3	1min 35.6s (2.42s)	17.5s (2.44s)	2.16s (2.86s)	240516 2184	12576 98	210994 2086
3	1,2,3	0.01s	0.06s	0.02s	1439	0	1407

		(0.08s)	(0.06s)	(0.06s)	485	7	477
3	1,2,3,4	0.03s (0.09s)	0.13s (0.08s)	0.01s (0.08s)	2591 625	28 16	2476 649
3	1,2,3,4,5	0.09s (0.13s)	0.30s (0.14s)	0.05s (0.13s)	6828 971	748 18	5736 946
3	1,2,3,4,5,6	1.64s (0.36s)	1.66s (0.38s)	0.23s (0.44s)	33558 1100	748 32	27053 1059
3	1,2,3,4,5,6 1,2	2.30s (0.44s)	1.94s (0.42s)	0.28s (0.56s)	44322 110	1872 32	32365 1059
3	1,2,3,4,5,6 1,2,3	3.36s (0.55s)	2.48s (0.5s)	0.75s (0.83s)	68473 110	6876 32	39113 1059
3	1,2,3,4,5,6 1,2,3,4	8.92s (0.78s)	5.56s (0.78s)	3.06s (2.30s)	196353 1100	46566 32	63336 1059
3	1,2,3,4,5,6 1,2,3,4,5	19.63s (1.06s)	10.95s (1.08s)	43.1s (7.92s)	685288 1100	126711 32	88890 1059
3	1,2,3,4,5,6 1,2,3,4,5,6	32.36s (1.50s)	17.09s (1.41s)	M	M	205820 32	110207 1059
3	1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6	1min 31.41s (2.28s)	35.08s (2.30s)	M	M	410892 32	193361 1059
3	1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6	2min 58.9s (3.23s)	54.49s (3.20s)	M	M	615964 32	276515 1059
4	1,2,3	0.14s (0.13s)	0.39s (0.14s)	0.05s (0.16s)	8203 1039	918 46	6661 974
4	1,2,3,4	1.79s (0.38s)	1.83s (0.38s)	0.22s (0.55s)	44439 1372	6652 51	27740 1301
4	1,2,3,4,5	29.14s (1.39s)	10.83s (1.41s)	2.83s (3.41s)	295627 1655	64113 65	119206 1529
4	1,2,3,4,5 1,2,3	4min 38.23s (4.09s)	59.45s (4.11s)	M	M	525733 65	361605 1529
4	1,2,3,4,5 1,2,3,4,5	10min 46.51s (6.16s)	1min 34.33s (6.09s)	M	M	1014887 65	538299 1529
4	1,2,3,4,5 1,2,3,4,5 1,2,3,4,5	33min 41s (11.27s)	3min 14.08s (11.36s)	M	M	1695661 65	957392 1529
5	1,2,3,4,5,6	0.03s (0.06s)	0.08s (0.06s)	0.03s (0.08s)	1735 208	139 2	1277 206
5	1,4,3,2,3,5 6,1	0.05s (0.08s)	0.09s (0.06s)	0.03s (0.08s)	2528 208	203 2	1730 206
5	1,2,3,4,5,6 1,2,3,4,5,6 1,2,3,4,5,6	0.13s (0.09s)	0.31s (0.08s)	0.22s (0.23s)	16472 208	509 2	4547 206



5	1,5,4,6,2,1 3,2,3,4,5,3 1,3,5,6,3,3 2,1,2,3,	0.16s (0.11s)	0.41s (0.11s)	0.45s (0.36s)	26322 208	674 2	5803 206
5	1,5,4,6,2,1 3,2,3,4,5,3 1,3,5,6,3,3 2,1,2,3,1,2 3,4	0.19s (0.14s)	0.66s (0.14s)	0.61s (0.53s)	42587 208	846 2	6930 206
5	1,5,4,6,2,1 3,2,3,4,5,3 1,3,5,6,3,3 2,1,2,3,1,2 3,4,2,3,4,5 3,5,6,3	0.30s (0.16s)	0.66s (0.16s)	0.80s (0.90s)	75034 208	1040 2	9153 206
6	1,2	0.02s (0.09s)	0.05s (0.14s)	0.0001s (0.11s)	756 369	14 0	738 369
6	1,7	0.01s (0.11s)	0.09s (0.13s)	0.02s (0.11s)	1405 1017	15 35	1351 982
6	3,5	0.01s (0.13s)	0.11s (0.19s)	0.02s (0.13s)	1709 973	12 6	1677 967
6	3,4	0.02s (0.14s)	0.11s (0.19s)	0.02s (0.05s)	1760 1004	81 6	1649 998
6	3,7	0.01s (0.13s)	0.13s (0.13s)	0.02s (0.13s)	1932 1189	17 41	1858 1148
6	5,4	0.03s (0.11s)	0.13s (0.14s)	0.02s (0.11s)	2080 1047	63 6	1951 1041
6	6,7	0.01s (0.14s)	0.14s (0.13s)	0.02s (0.11s)	2253 1334	16 35	2196 1299
6	4,7	0.02s (0.13s)	0.16s (0.53s)	0.01s (0.13s)	2478 1431	15 41	2406 1390
6	1,2,8	0.01s (0.34s)	0.08s (0.13s)	0.01s (0.11s)	1156 369	0 28	1107 369
6	1,2,3	0.01s (0.13s)	0.09s (0.11s)	0.02s (0.13s)	1427 639	29 6	1371 633
6	1,3,6	0.03s (0.13s)	0.13s (0.14s)	0.01s (0.19s)	2157 977	27 6	2092 971
6	1,3,5,4	0.03s (0.14s)	0.19s (0.14s)	0.03s (0.20s)	3616 1173	101 6	3363 1167
6	3,6,4,7	0.06s (0.14s)	0.25s (0.16s)	0.03s (0.23s)	5266 1693	112 41	4825 1652
6	3,5,6,4,7	0.06s (0.16s)	0.31s (0.19s)	0.05s (0.20s)	7233 1782	229 41	6192 1741
6	1,2,3,4,5,6 7	0.08s (0.17s)	0.34s (0.20s)	0.05s (0.19s)	8171 1783	258 41	6940 1742
6	1,2,3,4,5,6 7,8,9	0.11s (0.22s)	0.45s (0.22s)	0.08s (0.25s)	12179 1783	353 41	10448 1742
6	1,2,3,4,5,6 7,8,9,1,2,3,4,	0.36s (0.41s)	4.14s (0.41s)	0.31s (0.50s)	36749 1783	41 737	26875 1742

	5,6,7,8,9						
6	1,2,3,4,5,6 7,8,9,1,2,3,4, 5,6,7,8,9,1,2 ,3,4,5,6,7,8,9	0.73s (0.58s)	4.97s (0.56s)	0.97s (0.94s)	73133 1783	1121 41	43302 1742
7	1,2,3,4,5,6,7 8,9	2.44s (1.25s)	5.16s (1.27s)	0.94s (1.84s)	152184 21219	3532 727	96138 19462
7	1,2,3,4,5,6,7 8,9,1,2,3,4,5, 6,7,8,9	9.30s (3.27s)	18.67s (3.28s)	3min 19.9s (19.66s)	1676007 21219	7959 727	274343 19462
7	1,2,3,4,5,6,7 8,9,1,2,3,4,5, 6,7,8,9,1,2,3	11.48s (3.88s)	25.32s (4.50s)	10min 54.8s (32.8s)	2805918 21219	9109 727	333288 19462
7	1,2,3,4,5,6,7 8,9,1,2,3,4,5 ,6,7,8,9,1,2,3 ,4,5,6,7,8,9	20.92s (5.28s)	35.39s (5.30s)	M	M	12386 727	452548 19462
8	1,2,3,4,5,6,7 8,9	27.91s (4.48s)	30.83s (4.48s)	2min 54.12s (24.66s)	2170625 99206	31016 3169	378924 80739
8	1,2,3,4,5,6,7 8,9,1,2	34.35s (6.66s)	43.83s (6.28s)	2min 50.35s (26.95s)	2370747 99206	31084 3169	540602 80739
8	1,2,3,4,5,6,7 8,9,1,2,3	43.87s (7.22s)	44.19s (7.49s)	M	M	80739 3109	623239 80739
8	1,2,3,4,5,6,7 8,9,1,2,3,4,5	1min 1.53s (9.08s)	56.42s (9.11s)	M	M	53545 3169	788520 80739
8	1,2,3,4,5,6,7 8,9,1,2,3,4,5, 6,7,8,9	1min 39.02s (13.00s)	1min 17.44s (12.79s)	M	M	80739 3169	1115531 80739
8	1,2,3,4,5,6,7 8,9,1,2,3,4,5, 6,7,8,9,1,2,3, 4,5	2min 46.73s (17.66s)	1min 42.36s (17.63s)	M	M	97973 3169	1525127 80739
8	1,2,3,4,5,6,7 8,9,1,2,3,4,5, 6,7,8,9,1,2,3,4, 5,6,7,8,9	3min 53.99s (21.36s)	2min 10.63s (21.54s)	M	M	119872 3169	1852138 80739

## Appendix E: Fault trees data for mission task phases ASW, ASUW and SAR

The fault trees data below is for the missions task phases ASW (Aircraft surface war ),ASW\_ATT (Aircraft surface war attack),ASUW ( Aircraft submarine war), ASUW\_ATT ( Aircraft submarine war attack) and SAR ( search and recue ). These are used in chapter 7.

M\_STARTUP 1 11 0 DASS FMS TCS ESM RADAR\_IFF STORES MAD EOSDS PGATE539 PGATE540 ICE

ASW 1 8 0 DASS FMS TCS ESM RADAR\_IFF PGATE524 PGATE540 PGATE526

ASW\_ATT 1 9 0 DASS FMS TCS ESM RADAR\_IFF STORES PGATE524 PGATE540 PGATE526

ASUW 1 7 0 DASS FMS TCS MAD PGATE524 PGATE540 PGATE526

ASUW\_ATT 1 8 0 DASS FMS TCS MAD STORES PGATE524 PGATE540 PGATE526

SAR 1 8 0 DASS FMS TCS EOSDS RADAR\_IFF PGATE524 PGATE540 PGATE526

EOSDS 1 4 6 EOSDSCP DCPOWER ACPOWER DATA\_BUS TUR1 SCU1 PDU1 HGP1 TLU1 PWP1  
EOSDSCP 1 0 2 DPS1 RTS1  
MAD 1 3 5 DCPOWER ACPOWER DATA\_BUS MADCOMPAMP DETECTHEAD VECTMAGNET MADBASE MADCB  
RADAR\_IFF 1 6 6 NAVINFO DCPOWER ACPOWER DATA\_BUS IFFINTGTR RADCOOLING DATAPROC SIGPROC  
REC\_EXC RFTRANS RFCTRL RFSCAN  
IFFINTGTR 1 0 3 IFFUNIT CRYPTOFILL IFFCB  
RADCOOLING 2 0 2 DRADTDCR PRADTDCR  
STORES 1 5 0 SMGMTSYS SONORELSYS DCPOWER ACPOWER DATA\_BUS  
SMGMTSYS 1 2 2 STNCTRLU1 WEAPRELSW SMGMTPROC BBAYDR  
STNCTRLU1 2 0 5 STNCTRLU1 STNCTRLU2 STNCTRLU3 STNCTRLU4 STNCTRLU5  
WEAPRELSW 2 0 4 WRELSW1 WRELSW2 WRELSW3 WRELSW4  
SONORELSYS 2 2 0 SINGLAUNCH TENSHTOT  
SINGLAUNCH 2 0 2 SINGSHOT1 SINGSHOT2  
TENSHTOT 2 0 4 TENSHTOT1 TENSHTOT2 TENSHTOT3 TENSHTOT4

ESM 1 4 1 GSIGPROC DCPOWER ACPOWER DATA\_BUS MSU  
GSIGPROC 1 1 1 SIGDEL ESMPRO  
SIGDEL 1 2 0 SPINCHAN MAINCHAN  
SPINCHAN 1 1 1 PROCFAIL SAU  
PROCFAIL 1 0 2 SRX SFE  
MAINCHAN 1 1 1 ANTCLUST MRX  
ANTCLUST 2 4 0 ACLUST1 ACLUST2 ACLUST3 ACLUST4  
ACLUST1 2 2 1 LBA1 HBA1 FR1  
LBA1 1 1 1 LSPIANT1 LB1  
LSPIANT1 2 0 2 LS1-1 LS1-2  
HBA1 1 1 1 HSPIANT1 HB1  
HSPIANT1 2 0 2 HS1-1 HS1-2  
ACLUST2 2 2 1 LBA2 HBA2 FR2  
LBA2 1 1 1 LSPIANT2 LB2  
LSPIANT2 2 0 2 LS2-1 LS2-2  
HBA2 1 1 1 HSPIANT2 HB2  
HSPIANT2 2 0 2 HS2-1 HS2-2  
ACLUST3 2 2 1 LBA3 HBA3 FR3  
LBA3 1 1 1 LSPIANT3 LB3  
LSPIANT3 2 0 2 LS3-1 LS3-2  
HBA3 1 1 1 HSPIANT3 HB3  
HSPIANT3 2 0 2 HS3-1 HS3-2  
ACLUST4 2 2 1 LBA4 HBA4 FR4  
LBA4 1 1 1 LSPIANT4 LB4  
LSPIANT4 2 0 2 LS4-1 LS4-2  
HBA4 1 1 1 HSPIANT4 HB4  
HSPIANT4 2 0 2 HS4-1 HS4-2

TCS 1 6 0 TCSPROC TCSOPTCSREC DCPOWER ACPOWER DATA\_BUS  
 TCSPROC 1 1 1 TCSOS IOIFU  
 TCSOS 1 1 2 IOPS TCSOSMD TCSOSSCSI  
 IOPS 2 0 3 TAC1PWRSW IOP1 IOP2  
 TCSOP 2 4 0 NORM5WKSTN PILOTWKSTN ACO2WKSTN ORD2WKSTN  
 NORM5WKSTN 2 5 0 WKSTN1 WKSTN2 WKSTN3 WKSTN4 WKSTN5  
 WKSTN1 1 2 3 WK1PEP WK1INPUTS WK1DP WK1PSU WK1CHRD  
 WK1PEP 2 0 2 WK1PEP1 WK1PEP2  
 WK1INPUTS 2 0 4 WK1KEYP WK1KEYB WK1KEYPL WK1ROLBAL  
 WKSTN2 1 2 3 WK2PEP WK2INPUTS WK2DP WK2PSU WK2CHRD  
 WK2PEP 2 0 2 WK2PEP1 WK2PEP2  
 WK2INPUTS 2 0 4 WK2KEYP WK2KEYB WK2KEYPL WK2ROLBAL  
 WKSTN3 1 2 3 WK3PEP WK3INPUTS WK3DP WK3PSU WK3CHRD  
 WK3PEP 2 0 2 WK3PEP1 WK3PEP2  
 WK3INPUTS 2 0 4 WK3KEYP WK3KEYB WK3KEYPL WK3ROLBAL  
 WKSTN4 1 2 3 WK4PEP WK4INPUTS WK4DP WK4PSU WK4CHRD  
 WK4PEP 2 0 2 WK4PEP1 WK4PEP2  
 WK4INPUTS 2 0 4 WK4KEYP WK4KEYB WK4KEYPL WK4ROLBAL  
 WKSTN5 1 2 3 WK5PEP WK5INPUTS WK5DP WK5PSU WK5CHRD  
 WK5PEP 2 0 2 WK5PEP1 WK5PEP2  
 WK5INPUTS 2 0 4 WK5KEYP WK5KEYB WK5KEYPL WK5ROLBAL  
  
 PILOTWKSTN 1 0 4 PILOTTCPILOTDP PILOTTCPIFU PILOTCHRD  
 ACO2WKSTN 2 2 0 ACOWKSTN1 ACOWKSTN2  
 ACOWKSTN1 1 3 1 ACO1PEPS ACO1CHRD ACO1INPUTS ACO1PSU  
 ACO1PEPS 2 0 2 ACO1PEP1 ACO1PEP2  
 ACO1CHRD 2 1 1 SPARECHRD ACO1CHRD  
 SPARECHRD 2 0 2 SPARECHRD1 SPARECHRD2  
 ACO1INPUTS 2 0 4 ACO1KEYP ACO1KEYB ACO1KEYPL ACO1ROLBAL  
 ACOWKSTN2 1 3 1 ACO2PEPS ACO2CHRD ACO2INPUTS ACO2PSU  
 ACO2PEPS 2 0 2 ACO2PEP1 ACO2PEP2  
 ACO2CHRD 2 1 1 SPARECHRD ACO2CHRD  
 ACO2INPUTS 2 0 4 ACO2KEYP ACO2KEYB ACO2KEYPL ACO2ROLBAL  
 ORD2WKSTN 2 2 0 ORDWKSTN1 ORDWKSTN2  
 ORDWKSTN1 1 0 3 ORD1PEP ORD1PSU ORD1SONRS  
 ORDWKSTN2 1 0 3 ORD2PEP ORD2PSU ORD2SONRS  
 TCSREC 2 0 6 VIDINTU HDDR MAGDISK1 MAGDISK2 CBS TXTPRINT

DASS 1 5 1 DASSDET DASSPROT DCPOWER ACPOWER DATA\_BUS DSM  
 DASSDET 1 2 0 RADWARNR MISWARNR  
 RADWARNR 1 2 1 SUPERHET SIGNALREC APR  
 SUPERHET 1 0 2 SUC SUR  
 SIGNALREC 1 3 0 CDBAND MIDHIBAND SIGCONV  
 CDBAND 1 0 2 CBR CBA  
 MIDHIBAND 1 1 1 DASSANT DRR  
 DASSANT 1 2 0 DASSFANT DASSRANT  
 DASSFANT 2 0 2 HA1 HA2  
 DASSRANT 2 0 2 HA3 HA4  
 SIGCONV 2 0 2 DC1 DC2  
 MISWARNR 1 1 1 MISSENS ECU  
 MISSENS 1 2 2 MISLSENS MISRSSENS STP STB  
 MISLSENS 2 0 2 SLN SLB  
 MISRSSENS 2 0 2 SRN SRB  
 DASSPROT 1 2 0 TRDS CFD  
 TRDS 1 1 2 DASSLNCH DEC TQG  
 DASSLNCH 1 0 2 LRA PLC

CFD 1 1 2 CFDFMISS SDU DCU  
 CFDFMISS 2 12 0 DoR1 DR2 DR3 DR4 DR5 DR6 DR7 DR8 DR9 DR10 DR11 DR12  
 DoR1 1 0 4 FMG1M CMG1 TOP1Mm BOT1  
 DR2 1 0 4 FMG2MCMG2 TOP2Mm BOT2  
 DR3 1 0 4 FMG3MCMG3 TOP3Mm BOT3  
 DR4 1 0 4 FMG4MCMG4 TOP4Mm BOT4  
 DR5 1 0 4 FMG5MCMG5 TOP5Mm BOT5  
 DR6 1 0 4 FMG6MCMG6 TOP6Mm BOT6  
 DR7 1 0 4 FMG7MCMG7 TOP7Mm BOT7  
 DR8 1 0 4 FMG8MCMG8 TOP8Mm BOT8  
 DR9 1 0 4 FMG9MCMG9 TOP9Mm BOT9  
 DR10 1 0 4 FMG10M CMG10 TOP10Mm BOT10  
 DR11 1 0 4 FMG11M CMG11 TOP11Mm BOT11  
 DR12 1 0 4 FMG12M CMG12 TOP12Mm BOT12

Below is the fault trees data of the mission tasks phases with dependents. These are used in chapter 8.

M\_STARTUP 1 11 0 DASS\_START FMS TCS ESM\_START RADAR\_IFF STORES MAD EOSDS PGATE539 PGATE540  
ICE  
ASW 1 8 0 DASS\_ASW FMS NORM5WKSTN2 ESM\_ASW RADAR\_IFF PGATE524 PGATE540 PGATE526  
ASW\_ATT 1 9 0 DASS\_ASW FMS NORM5WKSTN2 ESM\_ASW RADAR\_IFF STORES PGATE524 PGATE540  
PGATE526

ASUW 1 7 0 DASS\_ASUW FMS NORM5WKSTN2 MAD PGATE524 PGATE540 PGATE526

ASUW\_ATT 1 7 0 DASS\_ASUW FMS MAD STORES PGATE524 PGATE540 PGATE526

SAR 1 7 0 DASS\_SAR FMS TCS RADAR\_IFF PGATE524 PGATE540 PGATE526  
ESM\_ASW 2 4 0 ESM\_ASWg1 ESM\_ASWg2 ESM\_ASWg3 ESM\_ASWg4

ESM\_ASWg1 1 2 0 ESM\_ASWg5 ESM\_ASWg6  
ESM\_ASWg5 2 2 0 ESM\_ASWg13 ESM\_ASWg14  
ESM\_ASWg13 1 3 0 ESMPRO SRX MSU  
ESM\_ASWg14 1 3 0 MRX SFE SAU  
ESM\_ASWg6 2 2 0 ESM\_ASWg15 ESM\_ASWg16  
ESM\_ASWg15 1 2 0 LSPIANT1 HSPIANT1  
ESM\_ASWg16 1 3 0 FR1 LB1 HB1

ESM\_ASWg2 1 2 0 ESM\_ASWg7 ESM\_ASWg8  
ESM\_ASWg7 2 2 0 ESM\_ASWg17 ESM\_ASWg18  
ESM\_ASWg17 1 3 0 SRX MSU SFE  
ESM\_ASWg18 1 3 0 ESMPRO MRX SAU  
ESM\_ASWg8 2 2 0 ESM\_ASWg19 ESM\_ASWg20  
ESM\_ASWg19 1 2 0 LSPIANT2 HSPIANT2  
ESM\_ASWg20 1 3 0 FR2 LB2 HB2

ESM\_ASWg3 1 2 0 ESM\_ASWg9 ESM\_ASWg10  
ESM\_ASWg9 2 2 0 ESM\_ASWg21 ESM\_ASWg22  
ESM\_ASWg21 1 3 0 SRX SFE ESMPRO  
ESM\_ASWg22 1 3 0 MSU MRX SAU  
ESM\_ASWg10 2 2 0 ESM\_ASWg23 ESM\_ASWg24  
ESM\_ASWg23 1 2 0 LSPIANT3 HSPIANT3  
ESM\_ASWg24 1 3 0 FR3 LB3 HB3

ESM\_ASWg4 1 2 0 ESM\_ASWg11 ESM\_ASWg12  
ESM\_ASWg11 2 2 0 ESM\_ASWg25 ESM\_ASWg26  
ESM\_ASWg25 1 3 0 SFE MRX MSU  
ESM\_ASWg26 1 3 0 SAU SRX ESMPRO  
ESM\_ASWg12 2 2 0 ESM\_ASWg27 ESM\_ASWg28  
ESM\_ASWg27 1 2 0 LSPIANT4 HSPIANT4  
ESM\_ASWg28 1 3 0 FR4 LB4 HB4

ESM\_START 2 2 0 GSIGPROC MSU  
GSIGPROC 1 2 0 SIGDEL ESMPRO  
SIGDEL 1 2 0 SPINCHAN MAINCHAN  
SPINCHAN 1 2 0 PROCFAIL SAU  
PROCFAIL 1 2 0 SRX SFE  
MAINCHAN 1 2 0 ANTCLUST MRX  
ANTCLUST 2 4 0 ACLUST1 ACLUST2 ACLUST3 ACLUST4

ACLUST1 2 3 0 LBA1 HBA1 FR1  
LBA1 1 2 0 LSPIANT1 LB1  
LSPIANT1 2 0 2 LS1-1 LS1-2  
HBA1 1 2 0 HSPIANT1 HB1  
HSPIANT1 2 0 2 HS1-1 HS1-2

ACLUST2 2 3 0 LBA2 HBA2 FR2  
LBA2 1 0 2 LSPIANT2 LB2  
LSPIANT2 2 0 2 LS2-1 LS2-2  
HBA2 1 2 0 HSPIANT2 HB2

HSPIANT2 2 0 2	HS2-1	HS2-2		
ACLUST3	2 3 0	LBA3	HBA3	FR3
LBA3 1 2 0	LSPIANT3	LB3		
LSPIANT3 2 0 2	LS3-1	LS3-2		
HBA3 1 2 0	HSPIANT3	HB3		
HSPIANT3 2 0 2	HS3-1	HS3-2		
ACLUST4	2 3 0	LBA4	HBA4	FR4
LBA4 1 2 0	LSPIANT4	LB4		
LSPIANT4 2 0 2	LS4-1	LS4-2		
HBA4 1 2 0	HSPIANT4	HB4		
HSPIANT4 2 0 2	HS4-1	HS4-2		
MSU 1 0 2	MSUevent1	MSUevent2		
ESMPRO 1 0 2	ESMPROe1	ESMPROe2		
SAU 1 0 2	SAUe1	SAUe2		
SRX 1 0 2	SRXe1	SRXe2		
SFE 1 0 2	SFEe1	SFEe2		
MRX 1 0 2	MRXe1	MRXe2		
FR1 1 0 2	FR1e1	FR1e2		
FR2 1 0 2	FR2e1	FR2e2		
FR3 1 0 2	FR3e1	FR3e2		
FR4 1 0 2	FR4e1	FR4e2		
LB1 1 0 2	LB1e1	LB1e2		
HB1 1 0 2	HB1e1	HB1e2		
LB2 1 0 2	LB2e1	LB2e2		
HB2 1 0 2	HB2e1	HB2e2		
LB3 1 0 2	LB3e1	LB3e2		
HB3 1 0 2	HB3e1	HB3e2		
LB4 1 0 2	LB4e1	LB4e2		
HB4 1 0 2	HB4e1	HB4e2		
TCS 2 3 0	TCSPROC	TCSOPT	TCSREC	
TCSPROC	1 2 0	TCSOS	IOIFU	
TCSOS 1 3 0	IOPS	TCSOSMD	TCSOSSCSI	
IOPS 2 0 3	TAC1PWRSW	IOP1	IOP2	
TCSOP 2 4 0	NORM5WKSTN	PILOTWKSTN	ACO2WKSTN	ORDN2WKSTN
NORM5WKSTN2 2 2 0	4TCSOPg1	4TCSOPg6		
4TCSOPg1 1 2 0	4TCSOPg4	4TCSOPg5		
4TCSOPg4 2 2 0	4TCSOPg10	4TCSOPg11		
4TCSOPg10 1 3 0	WK1DP	WK1PSU	WK1CHRD	
4TCSOPg11 1 2 0	WK1PEP	WK1INPUTS		
4TCSOPg5 2 2 0	4TCSOPg12	4TCSOPg13		
4TCSOPg12 1 2 0	WK2PEP	WK2INPUTS		
4TCSOPg13 1 2 0	WK2DP	WK2PSU		
4TCSOPg6 2 2 0	4TCSOPg14	4TCSOPg15		
4TCSOPg14 1 2 0	WK3PEP	WK3INPUTS		
4TCSOPg15 1 2 0	WK3DP	WK3PSU		
NORM5WKSTN 2 3 0	WKSTN1	WKSTN2	WKSTN3	
WKSTN1 1 5 0	WK1PEP	WK1INPUTS	WK1DP	WK1PSU WK1CHRD
WK1PEP 2 0 2	WK1PEP1	WK1PEP2		
WK1INPUTS 2 0 4	WK1KEYP	WK1KEYB	WK1KEYPL	WK1ROLBAL
WK1DP 1 0 2	WK1DPe1	WK1DPe2		
WK1PSU 1 0 2	WK1PSUe1	WK1PSUe2		
WK1CHRD	1 0 2	WK1CHRDe1	WK1CHRDe2	
WKSTN2 1 5 0	WK2PEP	WK2INPUTS	WK2DP	WK2PSU WK2CHRD
WK2PEP 2 0 2	WK2PEP1	WK2PEP2		
WK2INPUTS 2 0 4	WK2KEYP	WK2KEYB	WK2KEYPL	WK2ROLBAL
WK2DP 1 0 2	WK2DPe1	WK2DPe2		
WK2PSU 1 0 2	WK2PSUe1	WK2PSUe2		
WK2CHRD 1 0 2	WK2CHRDe1	WK2CHRDe2		
WKSTN3 1 5 0	WK3PEP	WK3INPUTS	WK3DP	WK3PSU WK3CHRD
WK3PEP 2 0 2	WK3PEP1	WK3PEP2		
WK3INPUTS 2 0 4	WK3KEYP	WK3KEYB	WK3KEYPL	WK3ROLBAL

WK3DP 1 0 2 WK3DPe1 WK3DPe2  
WK3PSU 1 0 2 WK3DPe1 WK3DPe2  
WK3CHRD 1 0 2 WK3DPe1 WK3DPe2

DASS\_START 2 2 0 DASSDET DASSPROT  
DASSDET 1 2 0 RADWARNR MISWARNR  
RADWARNR 1 2 0 SUPERHET SIGNALREC  
SUPERHET 1 0 2 SUC SUR  
SIGNALREC 1 3 0 CDBAND MIDHIBAND SIGCONV  
CDBAND 1 0 2 CBR CBA  
MIDHIBAND 1 1 1 DASSANT DRR  
DASSANT 1 2 0 DASSFANT DASSRANT

DASSFANT 2 0 2 HA1 HA2  
DASSRANT 2 0 2 HA3 HA4

SIGCONV 2 0 2 DC1 DC2  
MISWARNR 1 1 1 MISSENS ECU  
MISSENS 1 2 2 MISLSENS MISRSENS STP STB  
MISLSENS 2 0 2 SLN SLB  
MISRSENS 2 0 2 SRN SRB

DASSPROT 1 2 0 TRDS CFD

TRDS 1 1 2 DASSLNCH DEC TQG  
DASSLNCH 1 0 2 LRA PLC

CFD 1 1 2 CFDFMISS SDU DCU

CFDFMISS 2 12 0 DoR1 DR2 DR3 DR4 DR5 DR6 DR7 DR8 DR9 DR10 DR11 DR12  
DoR1 1 0 4 FMG1M CMG1 TOP1Mm BOT1  
DR2 1 0 4 FMG2MCMG2 TOP2Mm BOT2  
DR3 1 0 4 FMG3MCMG3 TOP3Mm BOT3  
DR4 1 0 4 FMG4MCMG4 TOP4Mm BOT4  
DR5 1 0 4 FMG5MCMG5 TOP5Mm BOT5  
DR6 1 0 4 FMG6MCMG6 TOP6Mm BOT6  
DR7 1 0 4 FMG7MCMG7 TOP7Mm BOT7  
DR8 1 0 4 FMG8MCMG8 TOP8Mm BOT8  
DR9 1 0 4 FMG9MCMG9 TOP9Mm BOT9  
DR10 1 0 4 FMG10M CMG10 TOP10Mm BOT10  
DR11 1 0 4 FMG11M CMG11 TOP11Mm BOT11  
DR12 1 0 4 FMG12M CMG12 TOP12Mm BOT12

DASS\_ASW 2 3 0 DASS\_G1 DASS\_G2 DASS\_G3  
DASS\_G1 1 2 0 CFD TRDS  
DASS\_G2 1 2 0 MISWARNR SIGCONV  
DASS\_G3 1 3 0 MIDHIBAND CDBAND SUPERHET

DASS\_ASUW 2 3 0 DASS\_G4 DASS\_G5 DASS\_G6  
DASS\_G4 1 3 0 CFD MISWARNR CDBAND  
DASS\_G5 1 2 0 SIGCONV TRDS  
DASS\_G6 1 2 0 MIDHIBAND SUPERHET

DASS\_SAR 2 2 0 DASS\_G7 DASS\_G8  
DASS\_G7 1 4 0 CFD TRDS MISWARNR SIGCONV  
DASS\_G8 1 3 0 MIDHIBAND CDBAND SUPERHET

## **Appendix F: Mission Fault tree structure**

### **F.1 Introduction**

Many Systems such as a UAV aircraft are broken down into sub functions, these are further broken down into subsystems then to particular components. This chapter collects UAV system structure information and with this fault trees for a UAV mission are built.

Up to now in this project only general phase fault trees have been considered. This information of UAV mission fault tree structure will be very important, since the aim of this project is to reduce time calculation of PMS unreliability. Therefore simplify a specific case of PMS will give more options of methods compared to a general case. This chapter will be organized as follows: Deception of the phases of the mission, Sub-systems deception and diagrams, how functions of the UAV are broken down into sub-system.

### **F.2 Phases of the mission**

The UAV mission phases are shown in figure F.1 and described as follows:

Start up – Ground base preparing of the UAV before the flight.

Taxiout – Travailing to the runaway on the ground.

Takeoff – Getting speed high an enough for UAV to be airborne.

Climb – Increasing UAV to the right altitude.

Cruise- Travailing in the air at the same altitude.

Descent- Decreasing UAV to landing altitude.

Landing- Landing the UAV on the runaway safely.

Taxiin -Travailing to the shut-down base

Shut-down – Switching off the UAV.



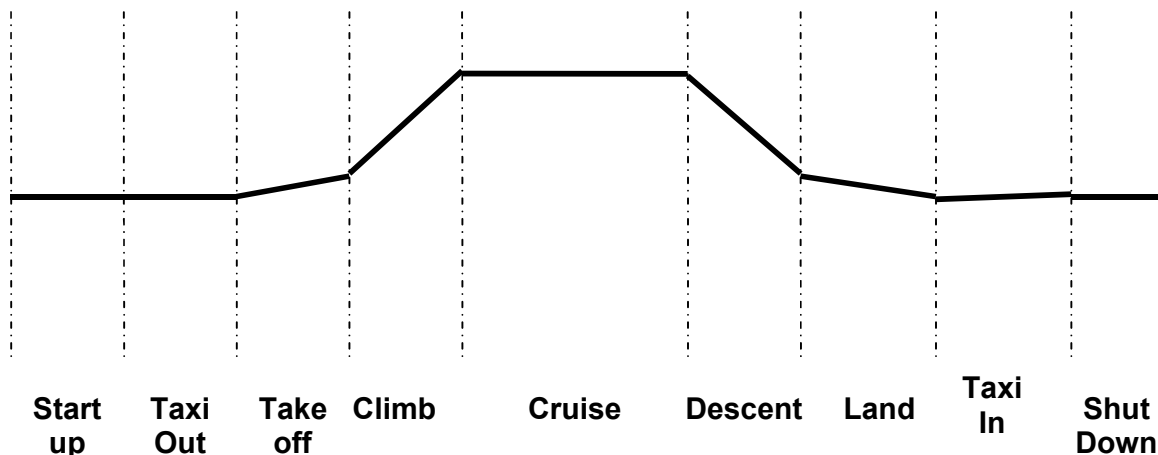


Figure F.1 Phase mission

There are two top events for consideration for phase failure which are mission and catastrophic failure. The top levels of the trees have been categorised into three sections external, internal and external effects which can be protected by an internal system shown in figure F.2. The majority of the tree will be internal effects. The internal effects are majority functions and sub-systems not working properly. The requirements for the functions will vary depending on the phase and if it mission failure or catastrophic failure. Two tables (table F.1 and F.1) have been constructed which listed the functions of the UAV and what is required by them in each phase, table F.1 for mission to be successful and table F.2 for a catastrophic failure not to happen.

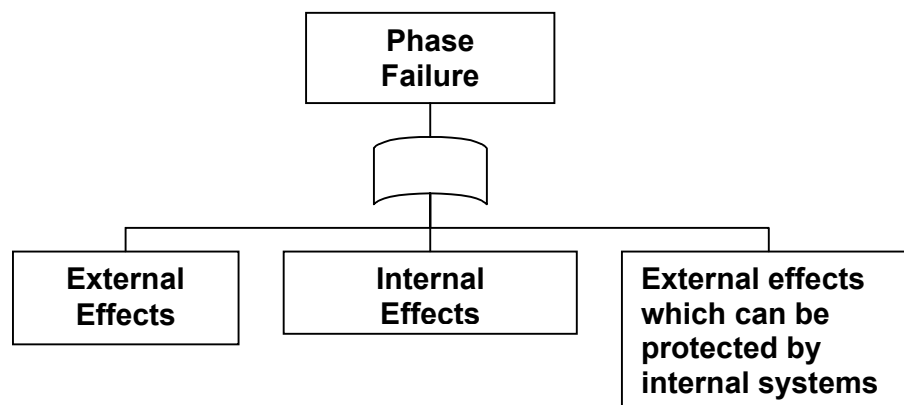


Figure F.2: Top level of the tree

Mission	start-up	taxi-out	takeoff	climb	cruise	descent	Landing	taxi-in	shut-down
Movability (control surface)	Actuators (Func): Elevators Rudder Ailerons Flaps and Slats Spoiler	Actuators (Func): Flaps and Slats	Actuators (Func); Elevators Rudder Ailerons Flaps and Slats	Actuators (Func): Elevators Ailerons Rudder Flaps and Slats Spoiler	Actuators (Func): Elevators Ailerons Rudder Flaps and Slats Spoiler	Actuators (Func): Elevators Ailerons Rudder Flaps and Slats Spoiler	Actuators (Func): Elevators Ailerons Rudder Flaps and Slats Spoiler	N/A	N/A
Thrust (primary power)	Engine 1 & 2 (Func) Not fire	Engine 1 & 2 (Func) Not fire	Engine 1 & 2 (Func) Not fire	Engine 1 or 2 (Func) Not fire	Engine 1 or 2 (Func) Not fire	Engine 1 or 2 (Func) Not fire	Engine 1 or 2 (Func) Not fire	Engine 1 or 2 (Func) Not fire	Engine 1 & 2 (Not fire)
reverse Thrust	(Not Func)	(Not Func)	(Not Func)	(Not Func)	(Not Func)	(Not Func)	(Func)	N/A	N/A
Landing gear	(Not Func)	(Not Func)	(Func)	(Not Func)	(Not Func)	(Not Func)	(Func)	(Not Func)	(Not Func)
Braking	(Func)	(Func)	Function in emergences case	N/A	N/A	N/A	(Func)	(Func)	

Table F.1: Mission success criteria

Catastrophic	start-up	taxi-out	takeoff	climb	cruise	descent	Landing	taxi-in	shut-down
Movability (control surface)	N/A	N/A	Actuators (Func); Elevators Rudder Ailerons Flaps and Slats	Actuators (Func): Elevator Ailerons Rudder	Actuators (Func): Elevator Ailerons Rudder	Actuators (Func): Elevator Ailerons Rudder Flaps and Slats Spoiler	Actuators (Func): Elevator Ailerons Rudder	N/A	N/A
Thrust (primary power)	Engine 1 & 2 (Func) Not fire	Engine 1 & 2 (Func) Not fire	Engine 1 & 2 (Func) Not fire	Engine 1 or 2 (Func) Not fire	Engine 1 or 2 (Func) Not fire	Engine 1 or 2 (Func) Not fire	Engine 1 or 2 (Func) Not fire	Engine 1 & 2 (Not fire)	Engine 1 & 2 (Not fire)
reverse Thrust	N/A	N/A	(Not Func)	(Not Func)	(Not Func)	(Not Func)	(Func)	N/A	N/A
Landing gear	N/A	N/A	N/A	(Not Func)	(Not Func)	(Not Func)	(Func)	N/A	N/A
Braking	N/A	N/	Function in emergencies case	N/A	N/A	N/A	(Func)	N/A	N/A

Table F.2: Not catastrophic criteria

### **F.3Sub-Systems**

Aircraft Functions are broken down into different subsystems which supply power, information to other control subsystem. This section gives a review of several subsystems which consist of a diagram and a description.

#### **F.3.1 Hydraulic system**

Hydraulic system provides a high hydraulic power for many aircraft functions such as flight control surfaces and landing gear. Hydraulic power is produced by using principles of fluid mechanics and is based on the physical characteristics of liquids.

In figure F.3 shown a single hydraulic system .Two pumps provide pressure in the system, which are power by different power sources, AC power and mechanical. This gives redundancy to the system. Two pressure sensors detected if the pressure is to low, if so a signal is send to the 3 hydraulic units (for redundancy). The units do so computations and send a signal to the pumps. Similar loop is made for temperature control which consists of a temperature sensors, header and hydraulic unit process.

Filter is fitted to clean hydraulic fluid from any small foreign bodies and particles. In the case if the filter get block then there is a bypass way. There are sensors which detected if the filter get blocked and then sends a signal to the hydraulic units which send a signal to the bypass valve for it to open. Reservoir is fitted for storing hydraulic liquid in the system. Non-return valve NRV is fitted to make sure the liquid travel in the right direction. In the case if the pressure is too high in the system a pressure relief valve PRV will open and allow liquid to be put back into the reservoir.

Three of these Systems will be place in the UAV for redundancy. The only different between the systems is that one of them will have both pressure pumps power by AC Power. The Mechanical pumps of the other two systems are drive by gearboxes. The first gearbox will be driven by the left engine and the second one by the right engine.

Figure F.4 shows the connection of the three hydraulic systems to the user.

## Hydraulic System

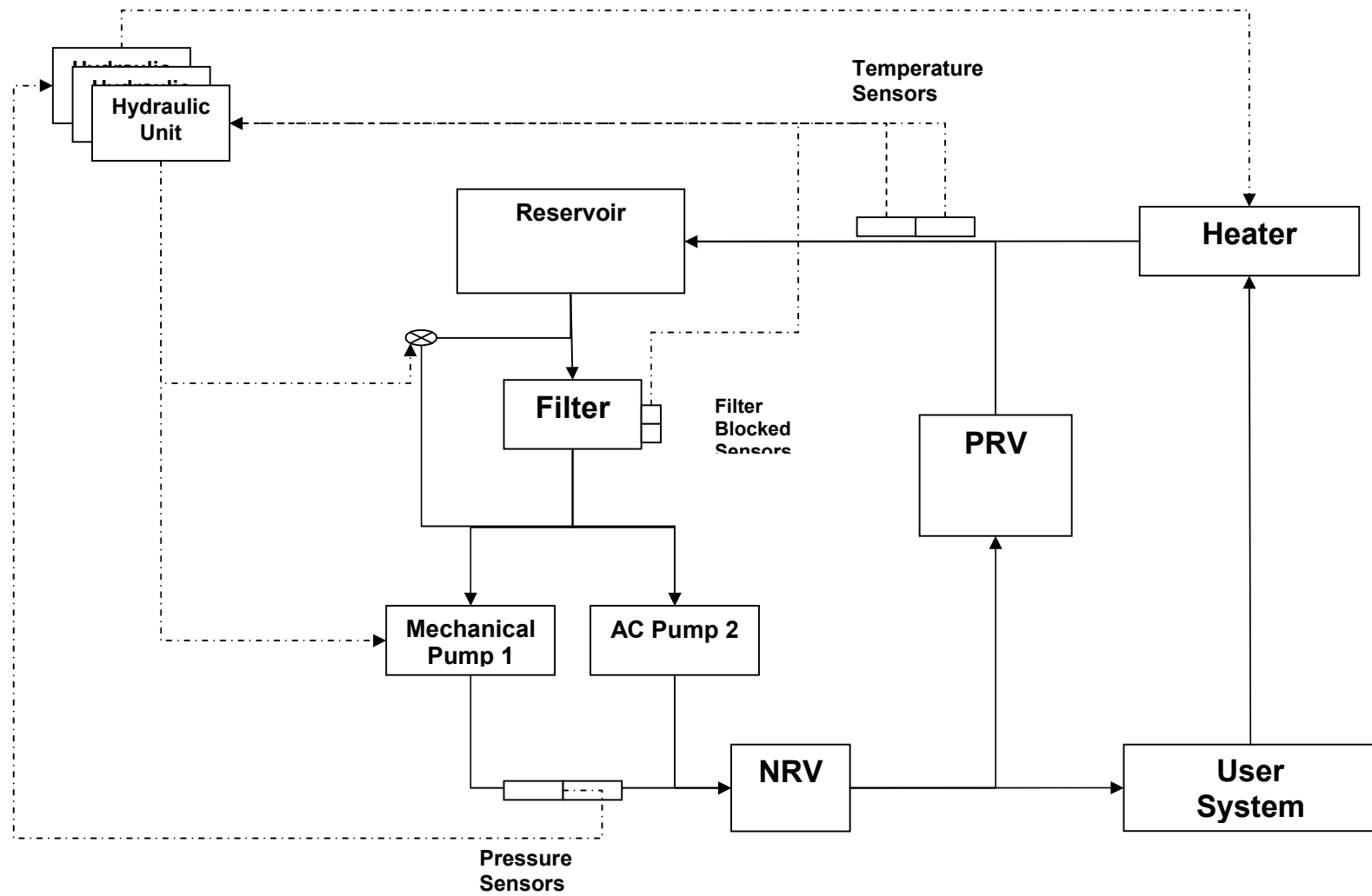


Figure F.3: Hydraulic System

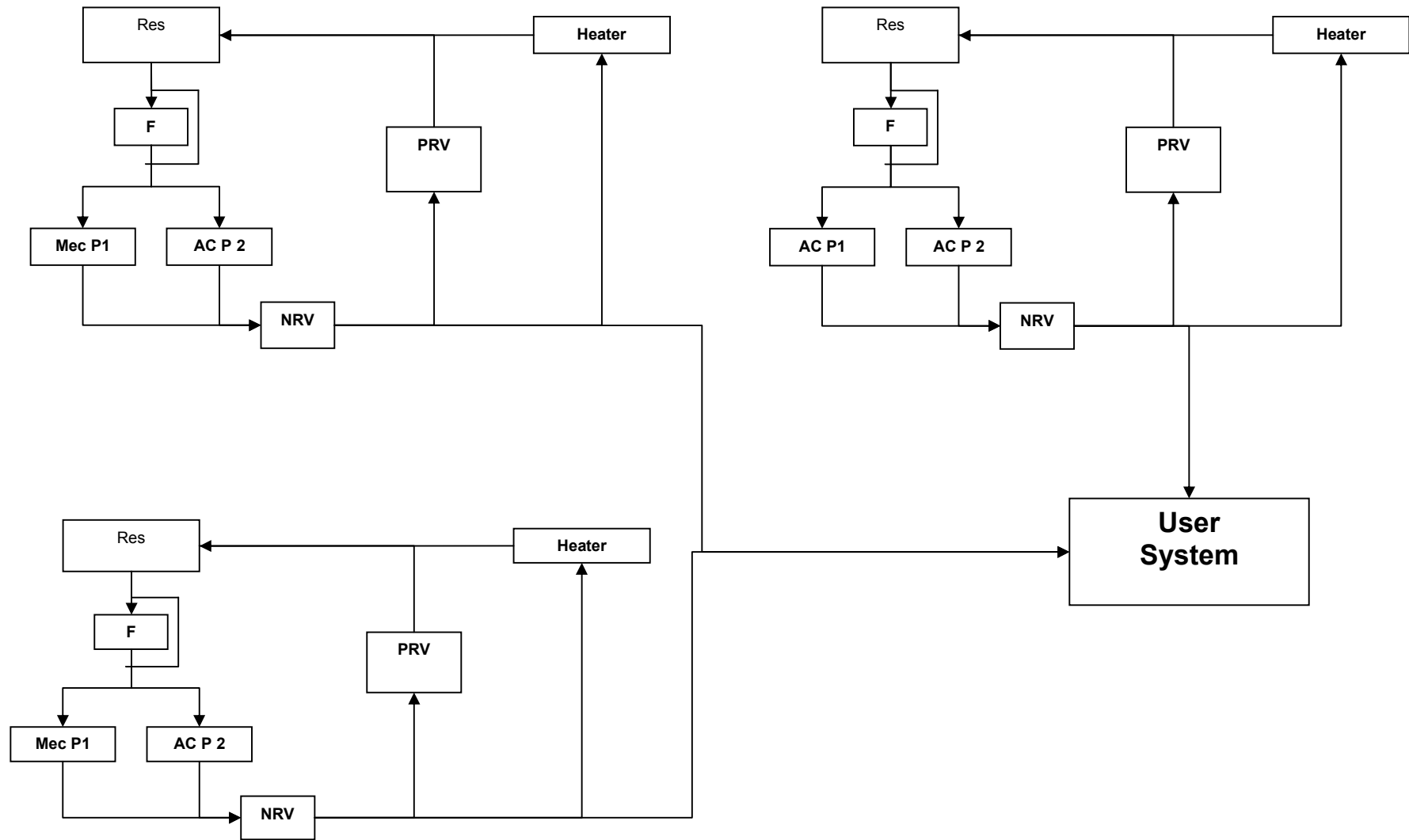


Figure F.4: Hydraulic Systems connection to users

### **F.3.2 Flight Control system**

Flight Control system consists of the flight control surface, connecting linkage and necessary operating mechanisms to control aircraft in flight.

Figure F.5 shows how a flight control surface works. A Signal from the Avionic of the aircraft is send to the three actuator drive computers, which send a signal to the servos of the particular control surface. The servos let in pressure hydraulic liquid into the actuators which moves the surface. The movement is detected by the two position sensors which make a closed control loop by sending the position information back to the actuator drive computers.

## Flight Control System

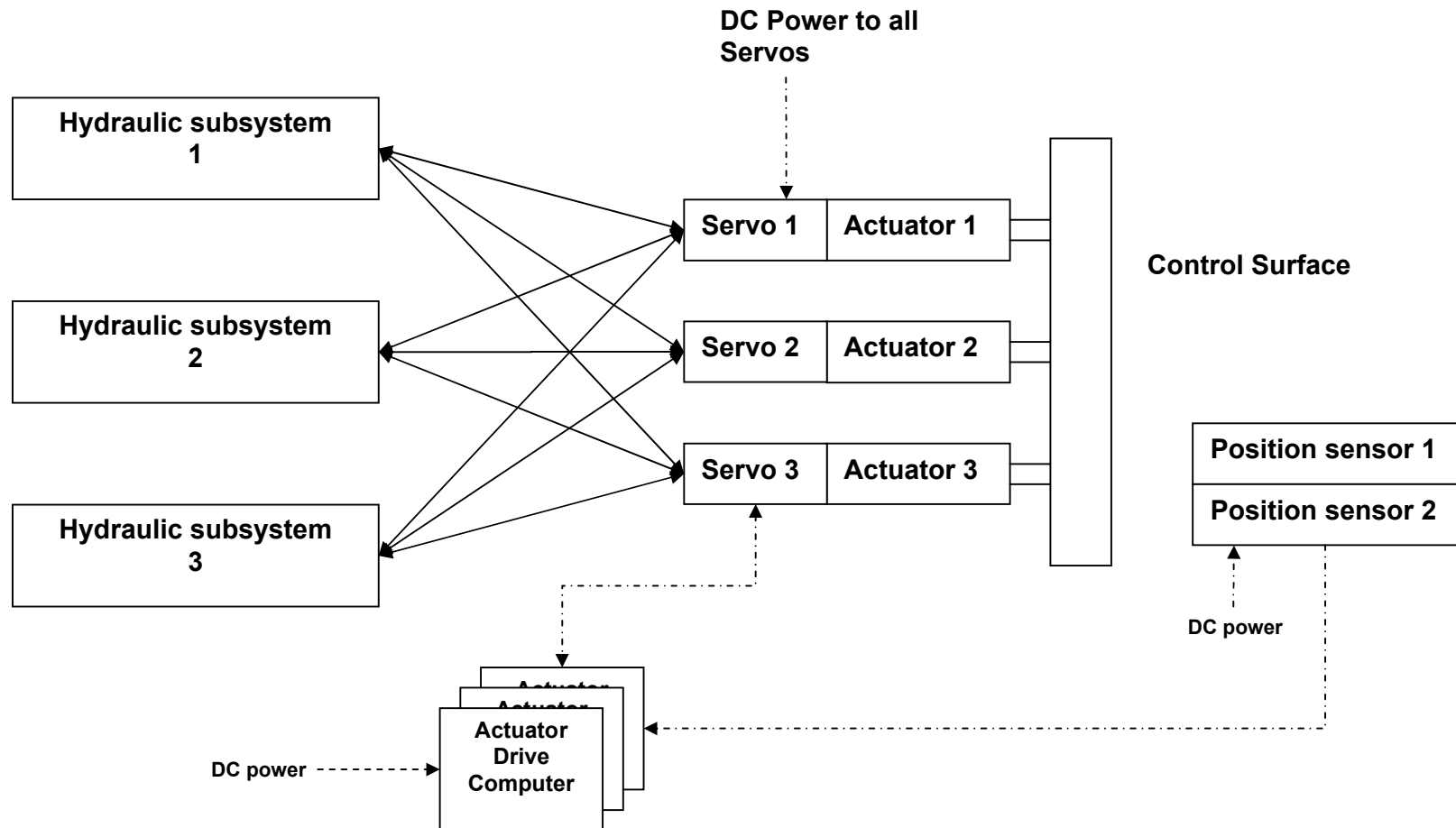


Figure F.5: Flight control system



### **F.3.3 Avionic System**

The Avionic System includes all electronic devices installed in an aircraft. It receives and process information, then output commands by sending signals through the data bus to the particular components. The Avionic system consists of sensors, computers, process and the data bus. The data bus connects all the components together so that they can communicate with each other. Figure F.6 shows the network of communication of the different components. Figure F.7 shows the structure of the avionic which has three of everything being interconnected for redundancy. The heart of the system is the flight management computer FMC. It process received data, control the execution of it and send the instruction to the control unit. The two main inputs for the FMC is Navigation and air data information. The Navigational system consist of Global Positioning system GPS, Internal references System IRS, VHF Omni-Range (VOR), Distance Measuring Equipment DME, Instrument Landing System ILS and Microwave Landing System MLS. ILS and MLS navigated the aircraft for the landing phase. The air data system consists of Air Data Computer and three type of sensors static air pressure, total air pressure and temperature. The FMC will output command to, Flight control Unit FCU to control the position of the control surfaces, Thrust Management Control Unit TMS to control the right thrust needed and The Landing & Braking unit to control the undercarriage of the aircraft, reverse thrust and the braking of the aircraft. The Avionic also monitor different power system such as the Electric, Hydraulic and Fuel system. The computers which perform is task is Utility Management System UMS, FMC, Hydraulic unit, Electric and fuel unit.

## Avionic System

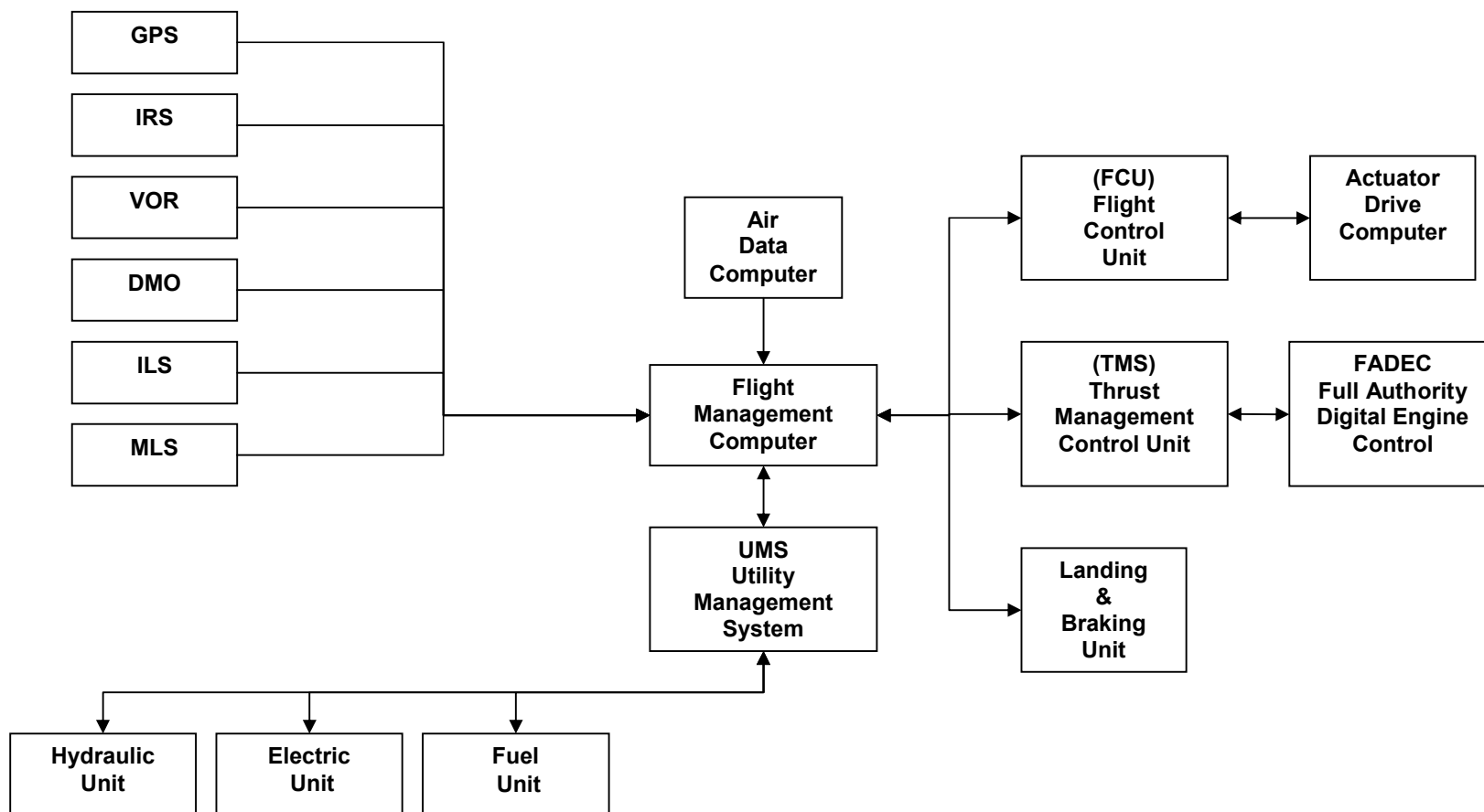


Figure F.6: Avionic functional roots

## Data bus structure

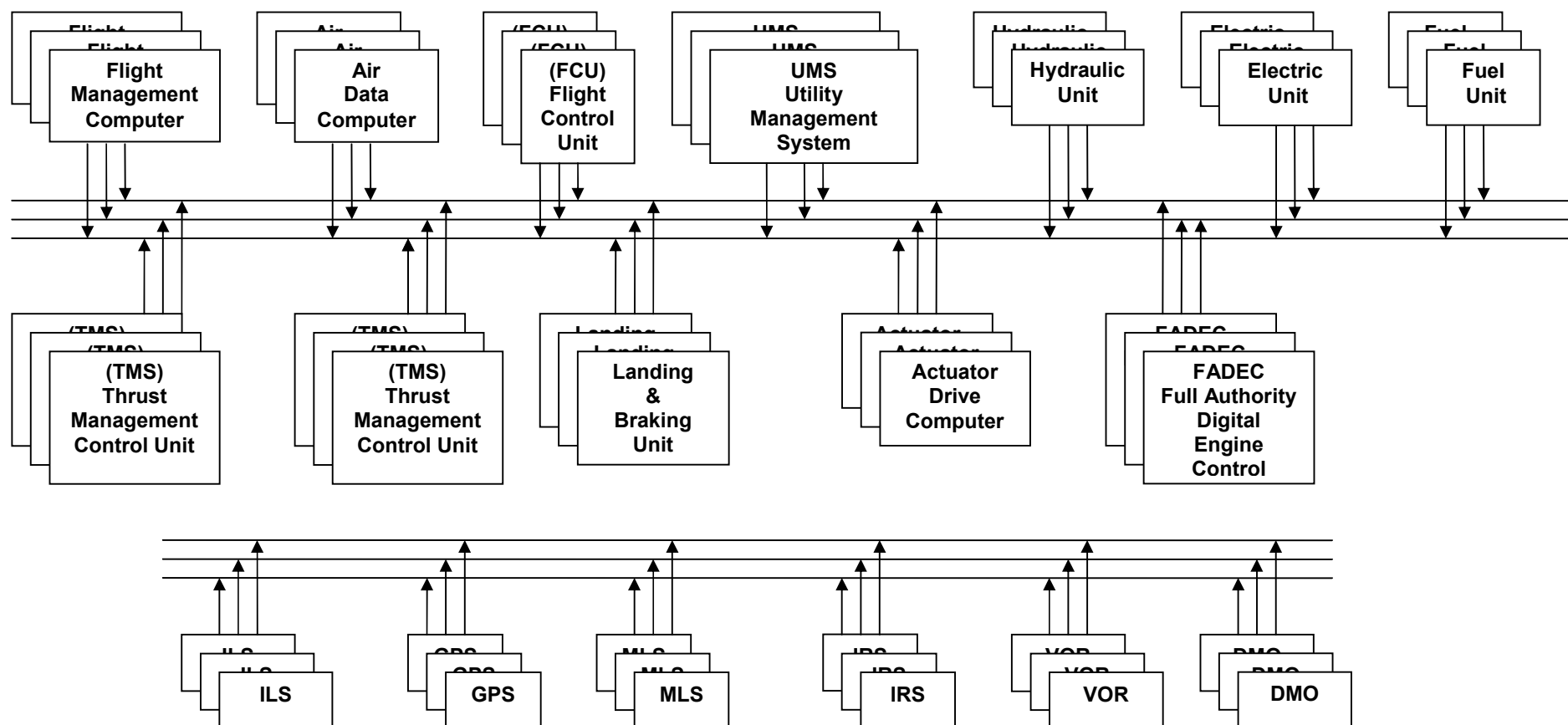


Figure F.7: Avionic System

### **F.3.4 Fuel system**

The Fuel system store, provides and distributes the proper amount of fuel at the correct pressure to the aircraft engines. The Fuel System is shown in figure F.8 and the description on how it works follows. Fuel is stored in two tanks Left and right. Fuel must be at the right pressure and temperature. The right pressure is maintained by monitoring air pressure by air sensors. The measures are sends to the fuel units to process this information. Depending on if the air pressure is too high or too low then the In or out vent valve will open or close, this is a control loop. Similar maintaining the fuel temperature follows the same procedure. Temperature sensors measure the temperature of fuel, sends this information to the fuel unit, then depending on the measurements send a signal to the heater. The Fuel is supply for two engines from two tanks. Focusing on the left hand sized fuel is pumped from the left tank to the left engine by the left AC power pump. The fuel passes a Non returnable valve NRV so that runs the right ways. Pressure sensors are fitted to the pip, then measurement are sends to the fuel units. If the pressure this too low then a signal is send to the crossover valve for it to open, this improves redundancy. If the pressure is dangerous high then the cut off valve is close. The Same action is taken for the right hand sized of the system.

## Fuel system

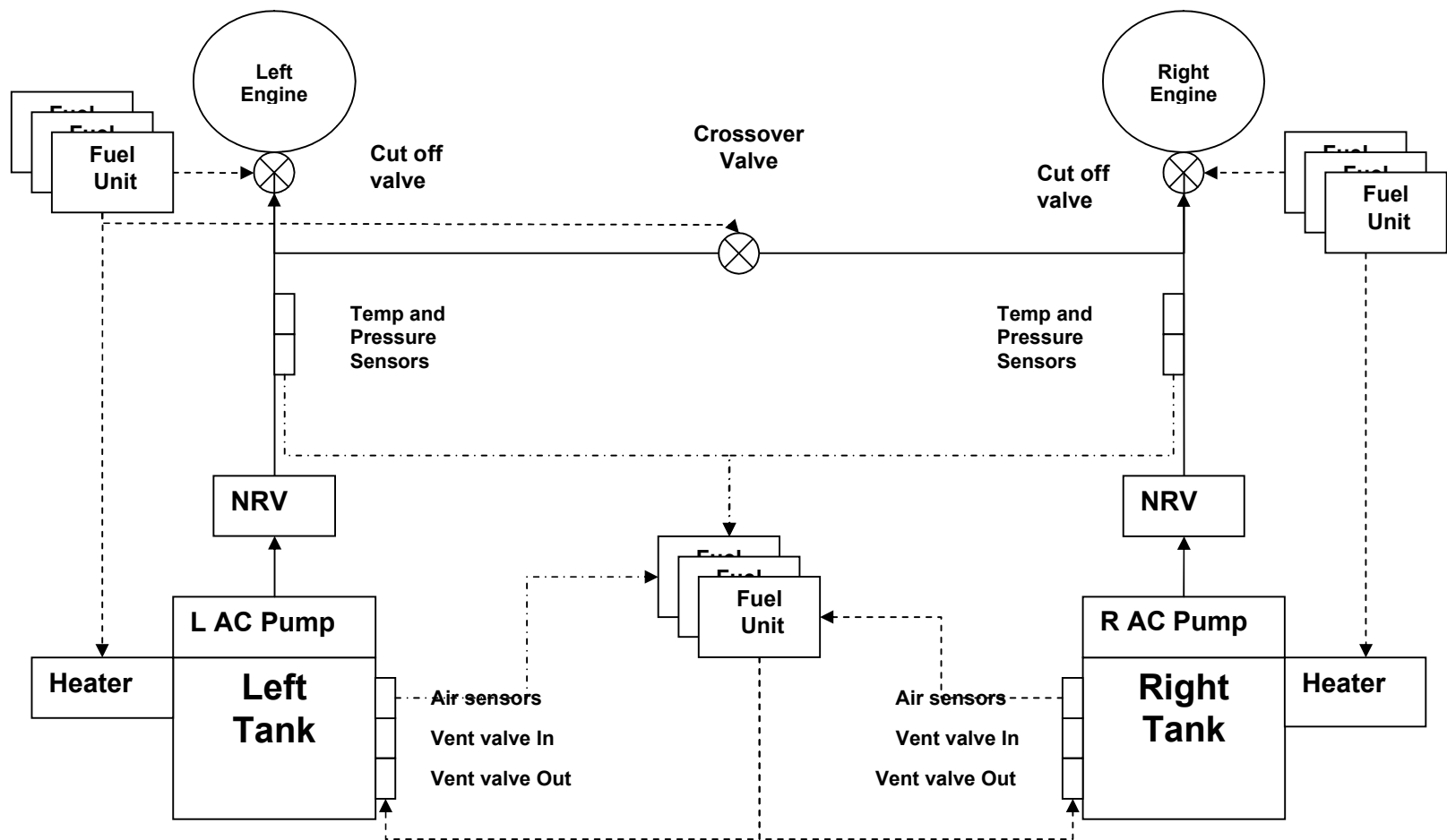


Figure F.8: Fuel System

### F.3.5 Landing gear system

The main function of the Landing gear system is to enable aircraft manoeuvres on ground after (prior) its flight. The two functions of the Landing gear bring the undercarriage up after takeoff and down just before landing. These functions require a sequence of sub functions to happen to order. There are six sub functions which all are control by there own three Actuators and selector valves, they all have to positions. The description of the six sub-functions and there positions are given in the table F.3.

Name of component	Description of function	Position 1	Position 2
Lock 1	To lock undercarriage door close.	Lock	Unlock
Undercarriage door Movement	Door to open the entrance where the undercarriage comes out of the aircraft.	Close	Open
Lock 2	To lock undercarriage door open.	Lock	Unlock
Lock 3	To lock undercarriage door up.	Lock	Unlock
Undercarriage Movement	To move the undercarriage up and down.	Up	Down
Lock 4	To lock undercarriage door down.	Lock	Unlock

Table F.3: Description of sub-functions and there positions of Landing gear system

The Landing gear system is shown in figure F.9. The selector valves of the sub function are power by DC power and send signal for command from the Landing and Braking units. Hydraulic Liquid is entered into the selector valves from the Hydraulic system.

The Two main functions of the Landing gear system are performed by activating the sub functions in order by a sequence valve (two are fitted for redundancy), they are also control by signals from the landing and braking units and power by Dc power. The order, sequence and position of the sub functions for the undercarriage to go up and down are has follows:

Undercarriage to go down:

- 1) Lock 1 to unlock
- 2) Undercarriage door to open
- 3) Lock 2 to lock
- 4) Lock 3 to unlock
- 5) Undercarriage to come down
- 6) Lock 4 to lock

Undercarriage to go up:

- 1) Lock 4 to unlock
- 2) Undercarriage go
- 3) Lock 3 to lock
- 4) Lock 2 to unlock
- 5) Undercarriage door to close
- 6) Lock 1 to lock

Two position Sensors are fitted to sense the position of the sub functions. Therefore when one function has competed its tasks the position sensors can send a signal to the landing and braking unit which process this signal and send a new signal to the sequence valve to start the next sub-function in the sequence.

## Landing gear system

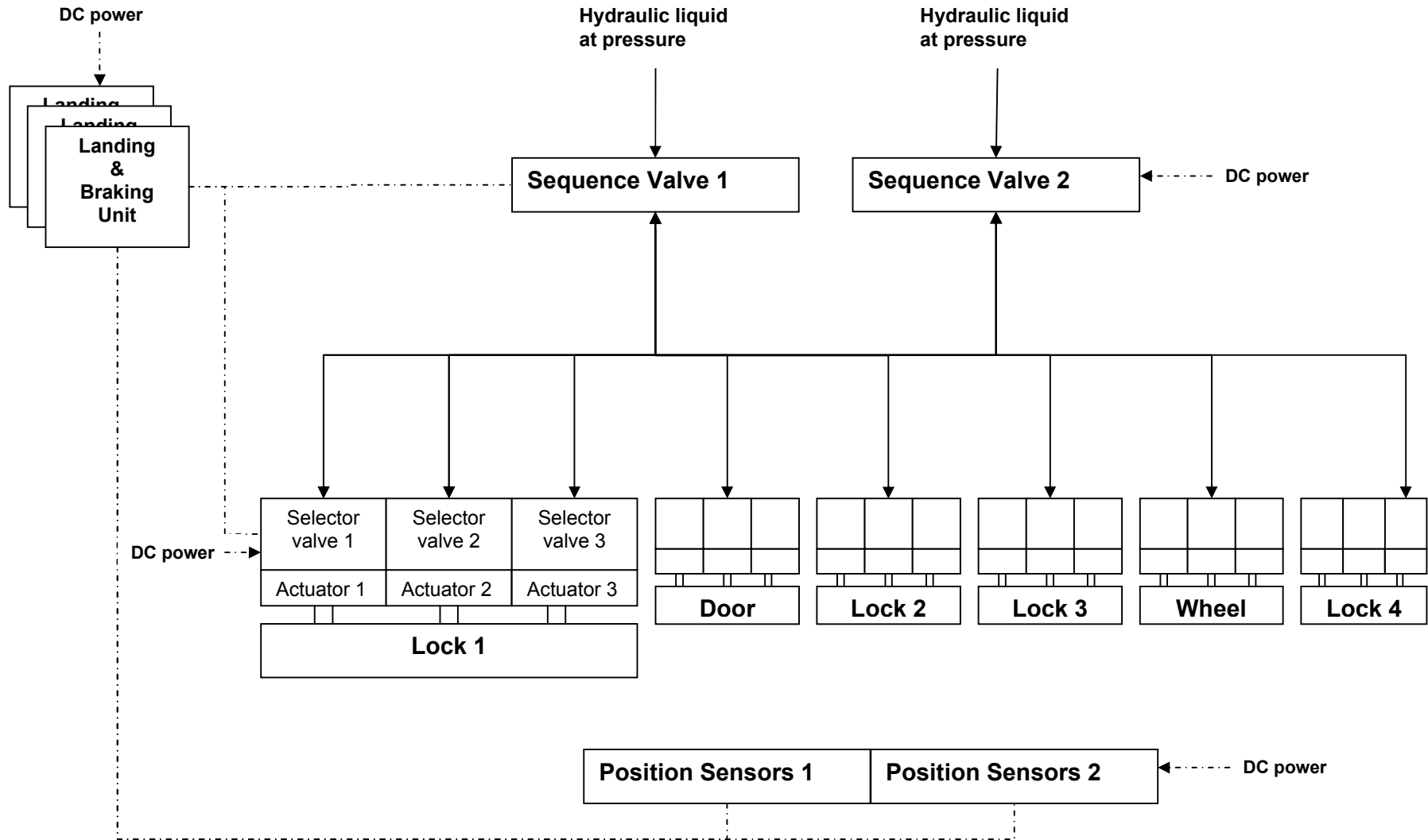


Figure F.9: Landing gear system



### **F.3.6 Braking system**

Braking and antiskid system operates during 'on ground' phases. It includes all those devices that slow or stop the aircraft and prevent aircraft wheels from skidding.

The system provides pressure for the two wheels as shown in figure F.10. Brake panels are push against the wheel. A hydraulic pip is connected to the brake panels therefore hydraulic pressure can be apply to the panels. There are two channel of control valve system for redundancy. They consist of a Antiskid Servo valve and Brake Control Valve. The Antiskid servo valves provide appropriate brake pressure to prevent stoppage of the wheel rotation. The Anti-skid Control unit sends Signal to Antiskid servo valve to control its operating, receives and processes information send by the transducers sensors. The Transducers sensors detect the wheel speed. The Brake Control valve enables hydraulic liquid flow, which produces hydraulic pressure to the system. Brake Control valve are sends signal from the Landing and braking units to activate the system operation. Has can be see from the diagram that a lots of the components of the system are power by DC power.

## Braking and Antiskid system

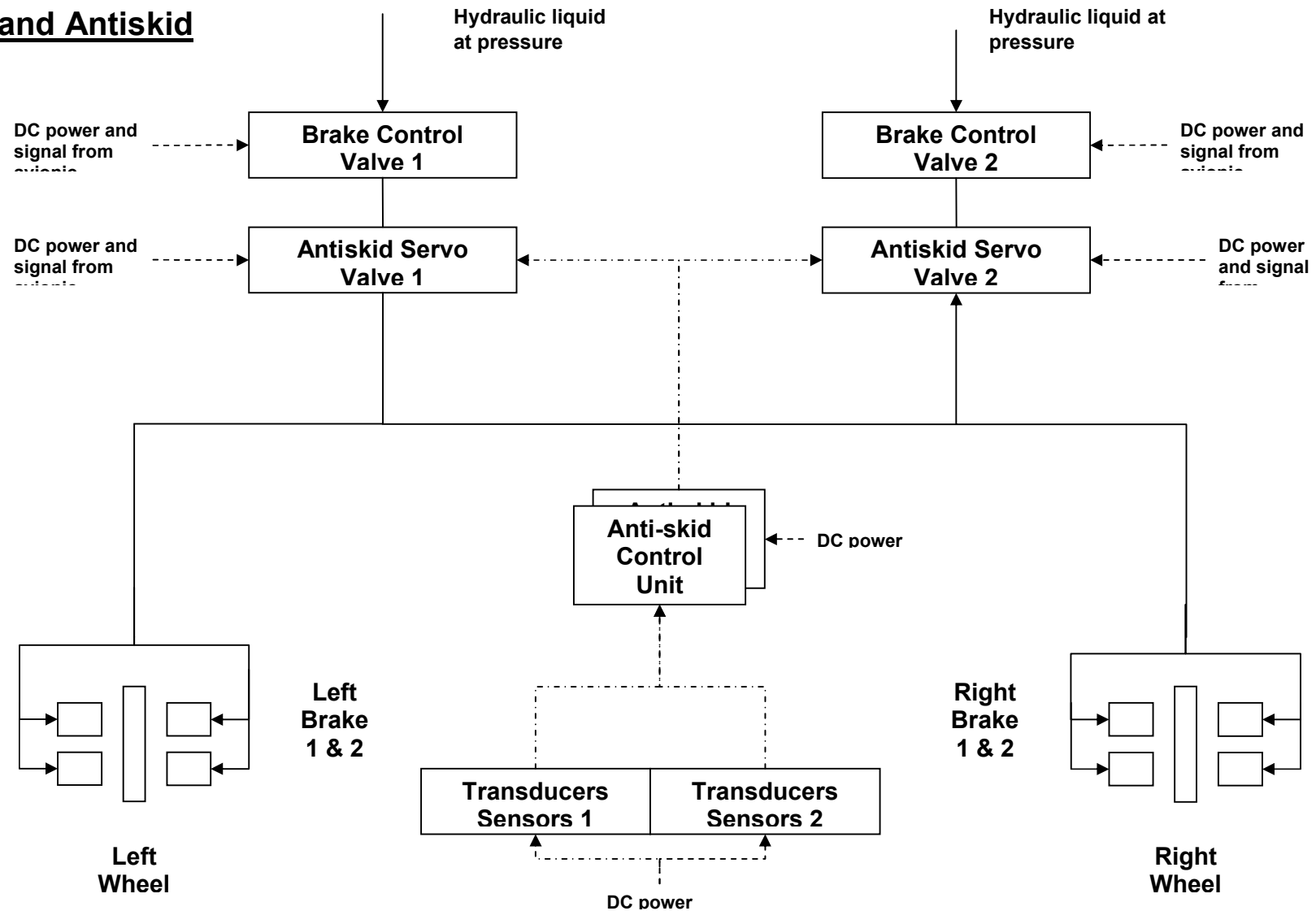


Figure F.10: Braking and Antiskid system

### **F.3.7 Engine Turbo-fan**

The Engine provides thrust for the aircraft and rotational energy, there will be two engines one on each wing. The Engine consists of 8 main components which are shown in figure F.11 and described below:

Fan- The Fan intake air mass into the engine.

Compressor- The Compressor compress air from the fan intake which increase the overall pressure ratio of the mass airflow.

Combustion Chamber- Fuel is added into the chamber with the air and ignited. Therefore results in a high temperature of the mass flow which thermal energy.

Turbine: - The Turbine Conversion the thermal to kinetic energy making the shaft to rotate.

Shaft- The Shaft is rotated by the turbines which therefore rotated power devices on the shaft, such as electrical power generator and gearboxes.

Exhaust- The exhaust releases air flow into the atmosphere in a way which maximum the thrust.

The fuel input system- Fuel is spray into the combustion chamber by the spray nozzle. The quantity of fuel which needed to be lets in to the engine with varies depending on the aircraft modes of operating. Therefore a Engine control valve is control by a process call Full Authority digital engine control FADEC which does all the computations to know what positions the valve should be and sends is information to the valve. In case if the fuel is at too higher pressure then FADEC with send a signal to the High Pressure HP cock to closes to isolated the engine to prevent a fire.

A AC pump is added to the system to get the fuel at the right pressure.

Sensors- FADEC mush have performances feedback from the engines, therefore sensors are fitted to the engine which measures temperature, pressure and speed.

## Engine Turbo-fan

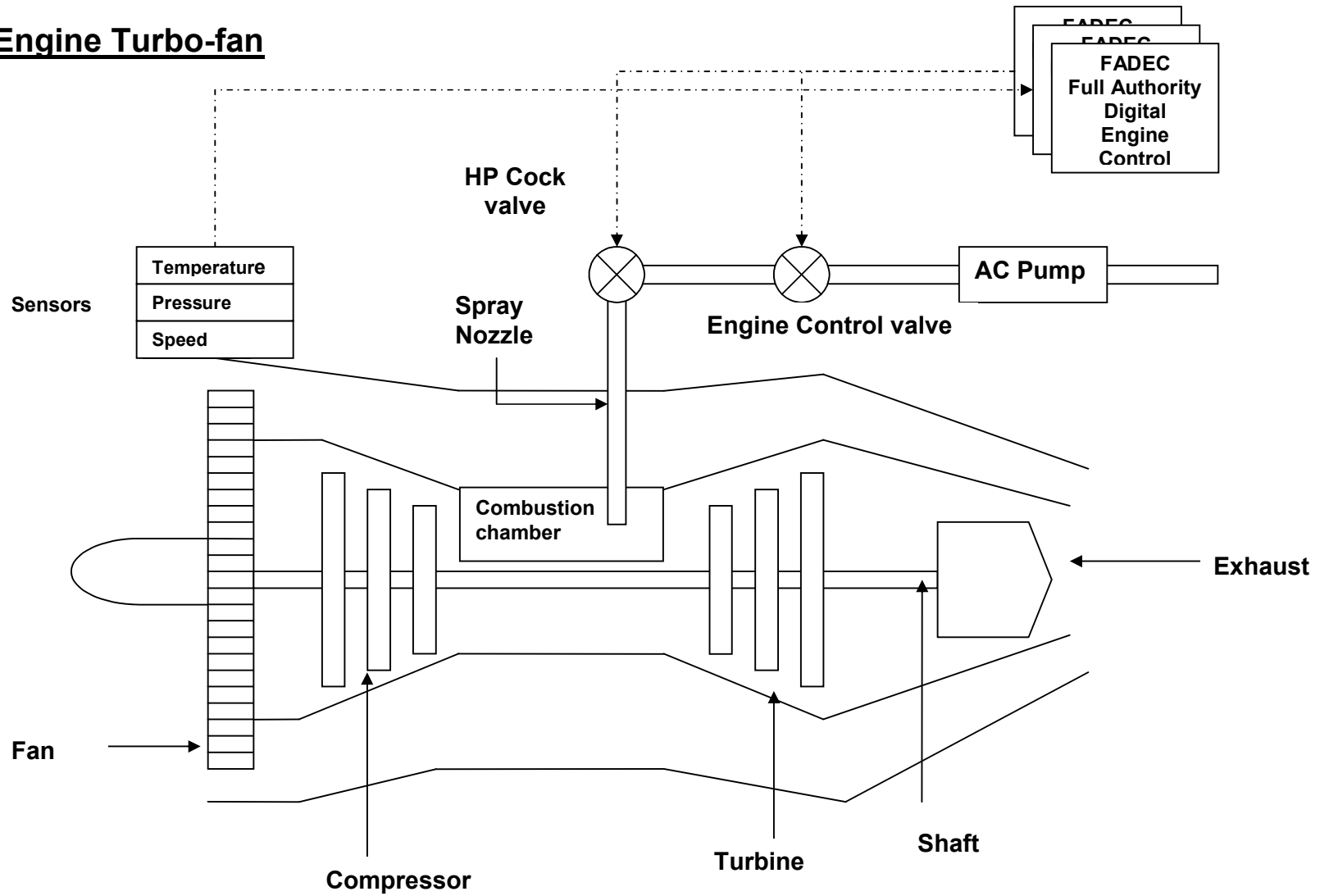


Figure F.11: Engine Turbo-fan

### **F.3.8 Reverse Thrust**

The Reverse thrust provides thrust to the opposite's direction of flight. It is used in the braking phase to help bring the aircraft to a stop.

The system consists of the two clamshell Doors in to engine and the engine which was described above. The system is shown in figure F.12.

The Clamshell Doors are supply focus from three actuators which three selector valves are connected to them. The Selector valve lets in hydraulic liquid at pressure from the hydraulic system and is power by the Dc power.

The selector valves are activated by a signal send by the Landing & Braking unit.

## Reverse Thrust

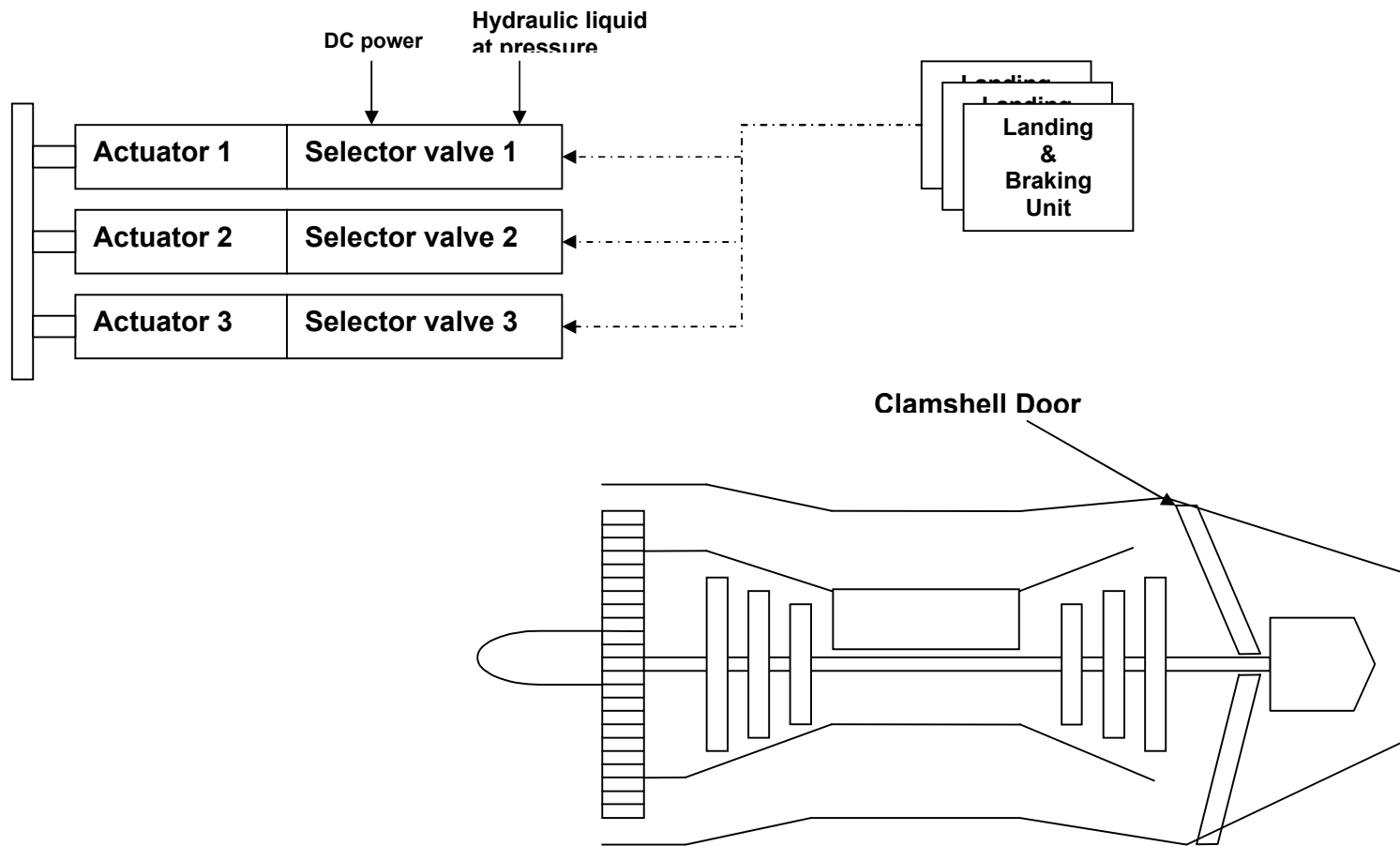


Figure F.12: Reverse Thrust

### **F.3.9 Electrical system**

Electrical system generate, regulate and supply the UAV Aircraft with DC and AC power, this is shown in figure F.13. The primary AC system has two main generators driven by the accessory gearbox of the respective engine. The system has a third generator called the Auxiliary Power Unit APU. There are also two back up generators. There are two main power channels left and right. The left generator supply power to the left AC main bus and Xtr bus and similar the right generator distributed to the right buses. The system is monitor by the control protection unit CPU. The CPU controls the Generator control breakers GCB for the three generators. For example if the generator is producing too much power then the GCB is close to isolate the generator. Also if the generator is not producing a enough power then a contactor can be open to being another sources of power on line, for example the Bus Tie Breaker BTB and Bus tie contactors can be open to distributed power to the other sized of the system. The system distributed power to the left and right AC main bus and to ac Xtr bus. The AC main buses distributed power the AC load of the aircraft. The Xtr bus distributed power for the second part of the system which supply DC power. AC power is converted to DC Power by a Transformer Rectifier unit TRU To the Dc bus of each channel. It the case of a emergence DC power can be generated by the Ram Air Turbine RAT. The essential DC power loads such as FCS servos are supply power from three channels of FCDC buses. The left bus is connected to the central FCDC bus. The FCDC buses are supply power from the DC buses, PMRs and the Main Battery which the power run though a hot battery bus. It the case of a overload of power from any of the FCDCs there a circuit breaker for each FCDCs. There is a standby power for the AC Power load which is connect to the left AC Xtr bus and Inverter, converted DC to AC power, the inverter is supply Dc power from the Battery bus.

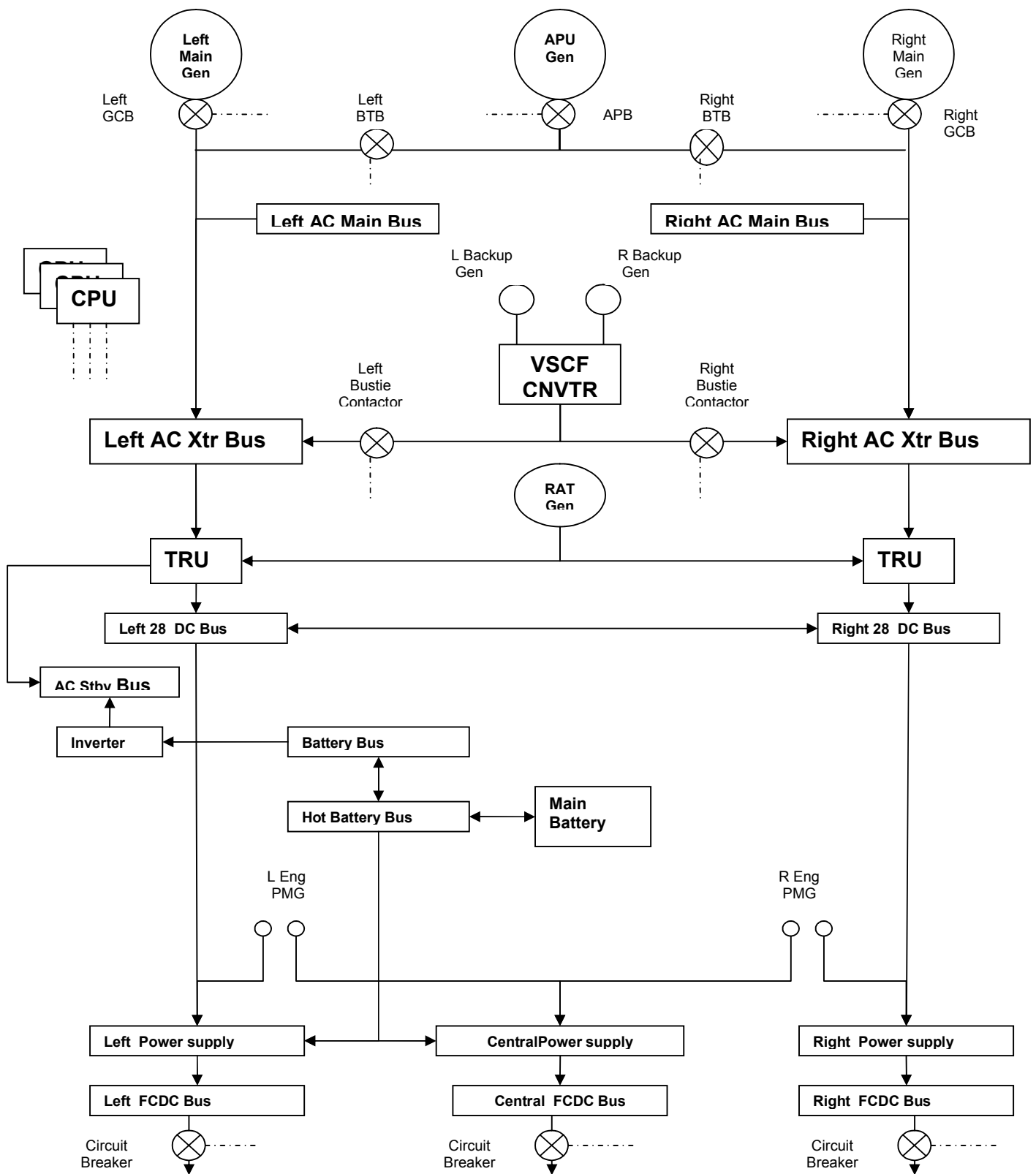


Figure F.13: Electrical system



#### F.4 Method of building fault trees

The fault trees have been made by the structure shown in figure F.14. The top level of the tree considers the external, internal and external protected by internal system. The majority of the tree will be the internal effects since that is where the complex aircraft is. The second level of the tree is the functions require, this information is given from table F.1 and F.2. Each function has different failure modes, for example does not work, operating when not meant to. These failure modes will dependent on the function. The functions are in term of the subsystems. Each function failure mode has been traces backwards though the aircraft system considering the unit, power supply and the signal information given to the unit. This has be broken down into the component and component failure modes.

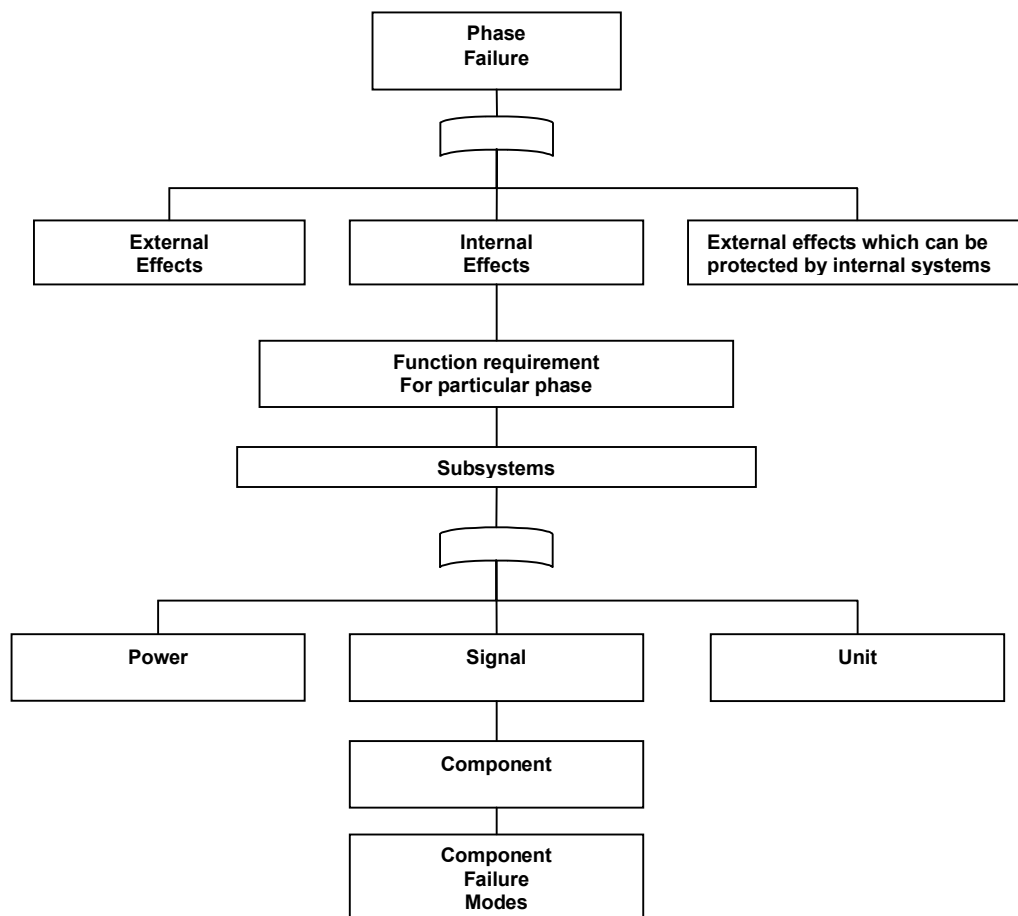


Figure F.14: General fault tree structure