Efficient Fault Tree Analysis Using Binary Decision Diagrams

by

Karen Ann Reay

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

September 2002

Abstract

The Binary Decision Diagram (BDD) method has emerged as an alternative to conventional techniques for performing both qualitative and quantitative analysis of fault trees. BDDs are already proving to be of considerable use in reliability analysis, providing a more efficient means of analysing a system, without the need for the approximations previously used in the traditional approach of Kinetic Tree Theory. In order to implement this technique, a BDD must be constructed from the fault tree, according to some ordering of the fault tree variables. The selected variable ordering has a crucial effect on the resulting BDD size and the number of calculations required for its construction; a bad choice of ordering can lead to excessive calculations and a BDD many orders of magnitude larger than one obtained using an ordering more suited to the tree. Within this thesis a comparison is made of the effectiveness of several ordering schemes, some of which have not previously been investigated. Techniques are then developed for the efficient construction of BDDs from fault trees. The method of Faunet reduction is applied to a set of fault trees and is shown to significantly reduce the size of the resulting BDDs. The technique is then extended to incorporate an additional stage that results in further improvements in BDD size. A fault tree analysis strategy is proposed that increases the likelihood of obtaining a BDD for any given fault tree. This method implements simplification techniques, which are applied to the fault tree to obtain a set of concise and independent subtrees, equivalent to the original fault tree structure. BDDs are constructed for each subtree and the quantitative analysis is developed for the set of BDDs to obtain the top event parameters and the event criticality functions.

Acknowledgements

I would like to thank my supervisor Professor John Andrews for his guidance, advice and encouragement throughout the course of my Ph.D.

Thanks must also be extended to the members of the Mathematics Department, particularly the administrative staff, who have been so kind and helpful during my time at Loughborough.

Thanks also to my family for their love and support. I'm especially grateful to my Dad, Harry, for his help with proof-reading my thesis and for his continued interest in my work.

I would like to say a huge thank-you to Sally, whose endless enthusiasm, friendship and laughter have helped make the last three years such a fun and memorable time.

Finally, a special thank-you to Alan for his support, understanding, meticulous proof-reading and most of all, for his confidence in me.

Contents

Nomenclature

$A(t)$	Availability function
C	Consequence of an event
C_i	Minimal cut set i
c	Number of output units in the neural network
d	Number of input units in the neural network
E^n	Sum of squares error function for training pattern n
$F(t)$	System unreliability function
$g(a)$	Activation function
$G_i(q(t))$	Criticality function for event i (Birnbaum's measure of importance)
$h(t)$	Conditional failure rate
I_i	Measure of importance for component or cut set i
M	Number of hidden units in the neural network
n	Number of components in a system; all nodes encoding event i in a BDD; training pattern for the neural network
N	Number of neural network training patterns
N_c	Number of minimal cut sets
P	Probability
$P(C_i)$	Probability of existence of minimal cut set i
$P(\theta_i)$	Probability of occurrence of minimal cut set i
$pr_{x_i}(q(t))$	Probability of the path section from the root vertex to the node x_i in the BDD
$po^1_{x_i}(q(t))$	Probability of the path section from the one branch of a node encoding x_i to a terminal one node in the BDD
$po^0_{x_i}(q(t))$	Probability of the path section from the zero branch of a node encoding x_i to a terminal zero node in the BDD
$P[F]$	Probability value of node F in a BDD
$Q_{sys}(t)$	System unavailability function (failure probability)
$q_i(t)$	Component unavailability (failure probability)
R	Risk
t_k	Target response for output unit k
T	Matrix of target responses
$w_{sys}(t)$	System unconditional failure intensity
$w_i(t)$	Component unconditional failure intensity
$W(t_0, t_1)$	Expected number of failures during the interval (t_0, t_1)

$w_{ji}^{(k)}$	Weight from the i^{th} unit in layer k to the j^{th} unit in layer k+1 in the multi-layer perceptron model
$\mathbf{w}$	Weight vector
w_{ji}	Weight from the i^{th} unit in the hidden layer to the j^{th} unit in the output layer of the radial basis function neural network
$\mathbf{W}$	Matrix of weights
x_i	Response of input unit i in the neural network
y_k	Response of output unit k in the neural network
z_j	Response of hidden unit j in the neural network
$Z(\mathbf{q}(t))$	Probability of paths from the root vertex to a terminal one vertex that do not pass through a node encoding x_i
α	Scaling parameter for η
β_i	Binary indictor variable for component states
γ	Scaling parameter for η
δ_i	Errors for output unit j
η	Learning rate parameter
μ	Momentum term
μ_j	Vector determining the centre of basis functions j
$\rho_i(\mathbf{x})$	Binary indicator function for each minimal cut set
σ_j	Width parameter in the Gaussian function for hidden unit j
τ	Time step in iterative algorithms
$\varphi(\mathbf{x})$	Structure function
$\varphi_j(\mathbf{x})$	Basis function j
$\mathbf{\Phi}$	Matrix of basis functions

Chapter 1: Introduction

1.1 Introduction to Reliability and Risk Assessment

The failure of industrial systems, such as those within the nuclear, aeronautical, offshore and transport industries, can have catastrophic consequences. Examples of such incidents include the explosion on the Piper Alpha oil platform in 1988 and the Concorde disaster in Paris in 2000, both of which resulted in multiple fatalities. System safety assessments are now routinely undertaken to increase the reliability of potentially hazardous systems and thus safeguard against undesired incidents in the future.

Reliability and risk assessment techniques have been developed over a number of years, with considerable advancements being made since the Second World War. Both methods are used in system safety analysis in order to calculate the probability and frequency with which a hazardous system failure could occur, and to determine whether the associated 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:

$$R = C \times P \tag{1.1}$$

The risk can therefore be reduced either by reducing the associated consequences of the hazard, or by reducing the probability or frequency of its occurrence.

A quantitative risk assessment of a system involves four basic stages:

1. Identification of potential safety hazards.
2. Estimation of the consequences of each hazard.
3. Estimation of the probability or frequency of each hazard.
4. A comparison of the results against the acceptability criteria.

The consequences of a hazard are usually measured by the expected number of fatalities and indicate the severity of the incident. Consequence modelling is very much industry dependent, as systems and their modes of failure can vary significantly from one industry to another. Reliability assessment techniques, however, which are concerned with calculating the probability or frequency with which system failure can occur, are generic. Methods such as Failure Mode and Effect Analysis (FMEA), Event Tree Analysis, Markov Analysis and Fault Tree Analysis are used extensively in many industries. The most widely used technique for system reliability assessment is Fault Tree Analysis and is discussed later in this chapter.

Having calculated the consequences of each hazard and the probability or frequency with which it can occur, Equation 1.1 is used to determine the associated risk. In order to assess whether a level of risk is acceptable, the HSE (Health and Safety Executive) recommend the use of a three-band approach known as the ALARP principle. This is shown in Figure 1.1.

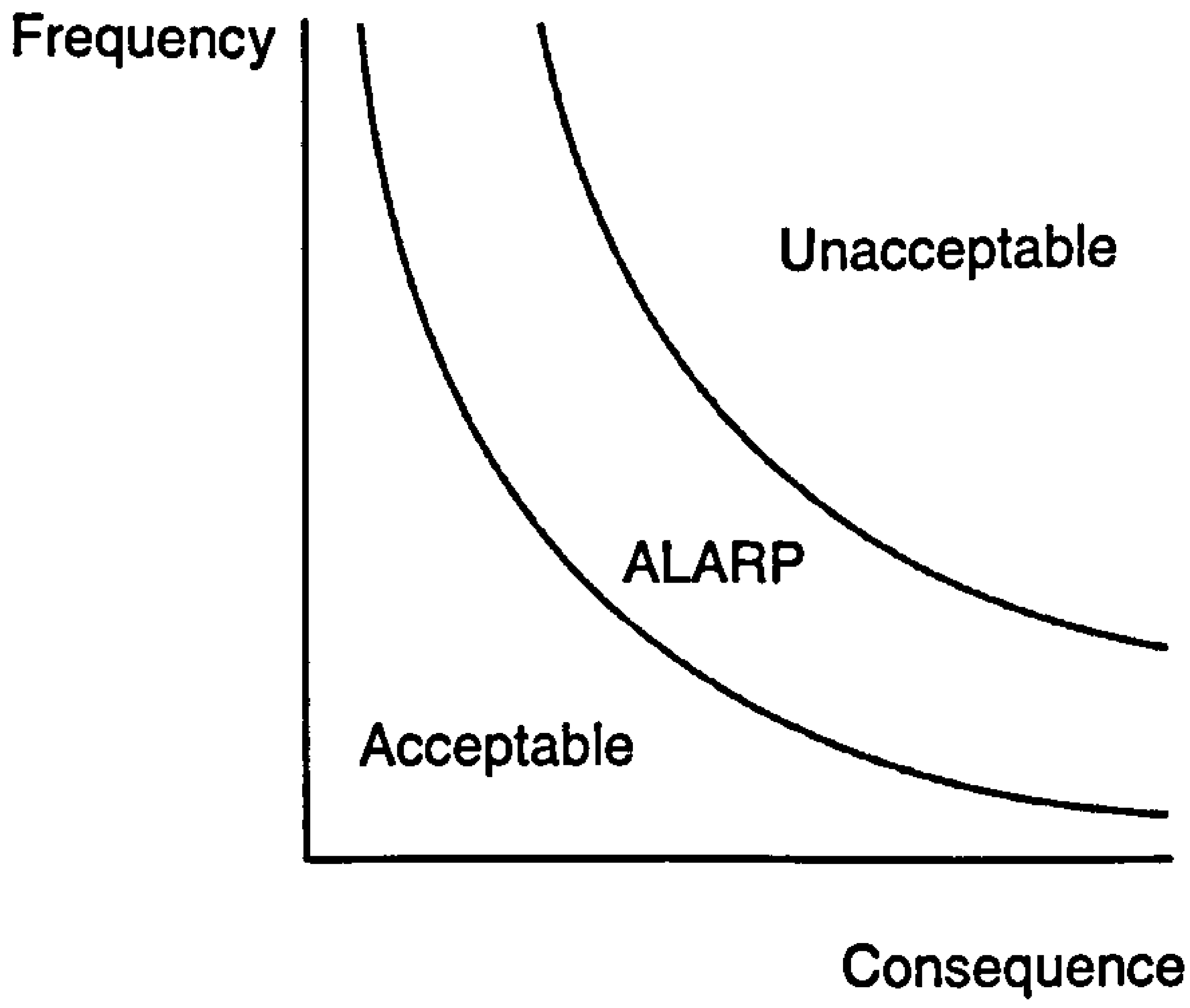


Figure 1.1: The ALARP principle

Risks that fall into the 'acceptable' region are considered low enough to be permissible. Generally, they have a low probability of occurrence and do not have a severe hazard associated with them. Risks that fall into the 'unacceptable' region are not tolerated and either the probability or consequence of the event must be reduced. Between these bands is the 'ALARP' region, where risks must be 'as low as reasonably practicable'. In this case the risks must be shown to be as low as possible, whilst still being economically feasible.

1.2 Quantification Parameters for System Failure

Reliability techniques are employed to assess the reliability performance of a system in terms of the reliability performance of its components. Many quantification measures can be used to describe component and system performance. The common parameters that are used throughout this thesis are defined below[1].

For systems that can be repaired, and so for which failure can be tolerated, a relevant measure of performance is the availability. This is defined as:

> The fraction of the total time that a system (or component) is able to perform the required function.

The complement of availability is unavailability, where:

$$\text{unavailability} = 1 - \text{availability}$$

Unavailability is defined as the probability that a component or system does not work at a given time t, and is denoted by q(t) for a component and $Q_{sys}(t)$ for a system.

Reliability can be defined as:

> The probability that a system or component will operate without failure for a stated period of time under specified conditions.

This measure is relevant for systems where failure cannot be tolerated, and so the successful operation of the system over a stated period of time is an important performance measure. The probability that a system (or component) fails to work continuously over a stated time interval and under specified conditions is known as its unreliability (F(t)) where:

unreliability = 1 - reliability

If a component or system is not repairable and it is working at time t, then it must have worked continuously since t=0. Therefore for non-repairable components and systems the unavailability is equal to the unreliability.

The transition of a component or system to a failed state can be characterised by the conditional failure rate, h(t). This is the rate at which failures occur taking into account the size of the population that has the potential to fail, i.e. those that are still functioning at time t. It is defined as follows:

> The conditional failure rate, h(t), is the probability that a system or component fails in the interval [t, t+dt), given that it has not failed in [0, t).

The unconditional failure intensity of a system or component is defined as:

> The probability of system or component failure in the interval [t, t+dt), given that it was working at t=0.

This measure is denoted by w(t) for a component and $w_{sys}(t)$ for a system. Integrating the unconditional failure intensity with respect to time gives the expected number of failures during the interval (t_0, t_1), denoted by $W(t_0, t_1)$:

$$W(t_0,t_1)=\int_{t_0}^{t_1} w(t)\,dt \qquad 1.2$$

Further component and system quantification measures can be found in reference 1.

A wide range of methods can be used to evaluate the system reliability parameters. One such technique, which is applied extensively in systems safety assessment, is Fault Tree Analysis. This is discussed in the following section.

1.3 Fault Tree Analysis

Fault Tree Analysis was developed by H. A. Watson[2] in the early 1960's, and is a deductive procedure for determining the causes of a particular system failure mode and the probability and frequency with which it could occur. The fault tree diagram provides a visual representation of the combinations of component failures and human errors that could combine to cause system failure. The system failure mode under consideration is referred to as the 'top event' of the fault tree and branches of the tree are constructed below, by taking a 'what causes this' approach. The events are continually redefined in terms of their causes, until each branch ends with a basic event: either a component failure or human error.

Fault Tree Analysis is an example of a 'top-down' technique, as the process starts with the top event and works downwards, building the fault tree beneath. Other methods, such as FMEA, are known as 'bottom-up' techniques, since they start with a set of component failure conditions and identify the possible consequences using a 'what happens if' approach.

The techniques for performing the quantitative analysis of fault trees, known as Kinetic Tree Theory, were not developed until the early 1970's by Vesely[3]. They allow the calculation of various system reliability parameters, such as:

- Probability of top event existence.
- Frequency of top event occurrence.
- Component importance measures.

These are used to determine whether the risk of system failure is sufficiently small and therefore whether or not the system meets the required safety standards.

The disadvantage of the conventional methods of Kinetic Tree Theory is that for large fault trees the analysis can become computationally intensive and can require the use of approximations. This obviously leads to inaccuracies in the calculations. As the techniques are already so well developed, further refinement is unlikely to result in vast improvements. This has led to the development of a new method for analysing fault trees, known as the Binary Decision Diagram technique. This is discussed briefly in the following section.

1.4 Binary Decision Diagrams

The Binary Decision Diagram (BDD) technique for Fault Tree Analysis was developed predominantly by Rauzy[4]. This method does not analyse the fault tree directly, but constructs a BDD, which encodes the fault tree's logic function. Both qualitative and quantitative analyses are then applied to the BDD. The advantage of this technique is that the calculations for the BDD quantification are both exact and efficient; unlike Kinetic Tree Theory, approximations are not required.

However, the structure of the BDD is dependent upon the order in which the fault tree variables (basic events) are considered during the construction process. Many different BDDs can be obtained from one fault tree and their sizes vary considerably, depending on the chosen variable ordering. The wrong choice of ordering scheme can result in a time-consuming construction process and a large BDD, which in turn can lead to increased analysis times. Previous research has failed to identify any ordering scheme that can order the fault tree variables in a manner that produces the smallest possible BDD from every fault tree structure.

1.5 Research Objectives

The aim of this research is to consider techniques for the efficient construction of BDDs from fault trees. Two distinct aspects will be examined. The first of these explores the variable ordering issue and the problem of finding an ordering scheme that produces the smallest BDD for any given fault tree. The second aspect looks at methods of reducing the fault tree size, so that smaller BDDs can be constructed and the choice of variable ordering scheme becomes less critical. The objectives of the project are listed below:

Variable ordering issue:

- Generate and analyse different categories of ordering schemes.
- Examine neural networks as a technique for selecting the most appropriate ordering scheme for a particular fault tree.

Reducing fault tree size:

- Apply modularisation techniques to fault trees.
- Investigate the effect on BDD size of applying reduction techniques to fault trees.
- Extend the BDD quantification methods to consider BDDs that have been constructed from modularised and reduced fault trees.

Chapter 2: Overview of Fault Tree Analysis

2.1 Introduction

Fault Tree Analysis is the most widely used tool in safety and reliability assessment. It is a deductive technique for determining the causes of system failure and the associated reliability parameters. The fault tree itself provides a visual representation of the structure of the system, by expressing a particular system failure mode in terms of component failures and human errors. It produces a complete description of the causes of system failure, which is important during the design stages of a system, as it allows weak areas to be identified and so any problems corrected.

2.2 Construction of the Fault Tree

The initial step in the construction of the fault tree is to identify the system failure mode of concern, known as the top event. A system may have more than one undesirable failure mode and if so, a separate fault tree must be constructed for each. Consequently, several fault trees may be required for the assessment of any given system. Once the top event has been defined, fault tree branches leading to intermediate events are developed underneath, by determining its causes. The intermediate events are then continually redefined in terms of lower resolution events by determining the immediate, necessary and sufficient causes for their occurrence. The process continues until the resolution limit is reached, i.e. all branches end with basic events. These basic events can be component failures or human errors.

The fault tree diagram is composed of gates and events. Events are categorised as either intermediate or basic. Intermediate events, which can be further developed in terms of other events, are represented by rectangles in the tree; basic events cannot be developed any further and are represented by circles. These symbols are shown in Table 2.1. Gates link the events together, depending on their causal relationship. The three fundamental types of gate used in fault trees are the 'AND' gate, 'OR' gate and 'NOT' gate. These gates combine events in the same way as the Boolean operations of 'intersection', 'union' and 'complementation'. Another gate frequently used is the k/n vote gate. This allows the flow of logic through the tree if at least k out of n inputs occur. The vote gate can be expressed in terms of 'AND' and 'OR' logic, but its use reduces the size of the resulting fault tree. The symbols for the gates and their causal relations are shown in Table 2.2.

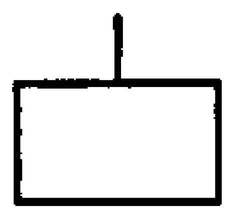
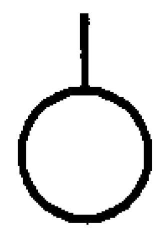
Event symbol	Meaning of symbol
	Intermediate event further developed by a gate
	Basic event

Table 2.1: Event symbols

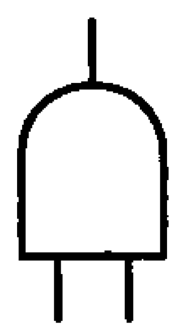
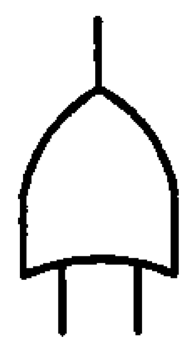
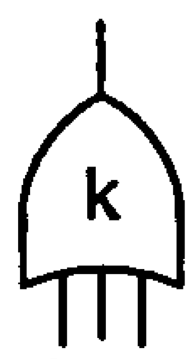
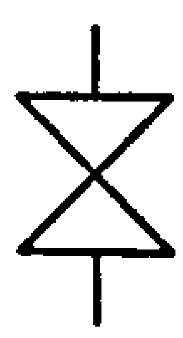
Gate symbol	Gate name	Causal relation
	AND gate	Output event occurs if all input events occur simultaneously
	OR gate	Output event occurs if at least one of the input events occurs
 n inputs	k/n vote gate	Output event occurs if at least k-out-of-n input events occur
	NOT gate	Output event occurs if the input event does not occur.

Table 2.2: Common gate types and corresponding symbols

A system whose failure modes are expressed solely in terms of component failures, is known as a 'coherent' system. A coherent fault tree will contain only 'AND' and 'OR' logic. If the failure modes of a system are expressed in terms of both component failures and successes, it is referred to as a 'non-coherent' system. In addition to the gates used in coherent fault trees, non-coherent fault trees also contain 'NOT' logic. The work within this thesis considers coherent fault trees only.

Once a fault tree has been constructed for a system, two types of analysis are performed: qualitative and quantitative.

- Qualitative analysis involves obtaining the smallest sets of events that combine to cause system failure. In coherent fault trees, these are called 'minimal cut sets'; in non-coherent trees, they are called the 'prime implicants'.
- Quantitative analysis is concerned with calculating the system failure parameters (the top event probability and frequency) and event importance measures.

2.3 Qualitative Analysis

The aim of qualitative analysis is to determine the combinations of basic events that combine to cause system failure. These are termed the cut sets or minimal cut sets of the fault tree and are defined below.

> A cut set is a group of basic events such that if they all occur (i.e. all components fail), the top event also occurs.

However, system failure does not necessarily require the failure of *all* the components in a cut set. Consider for example a cut set that contains three basic events A, B and C. The failure of all three components would guarantee system failure. However, if A and B alone result in system failure, then the state of C is irrelevant and the system will fail regardless of whether C is in a working or failed state. This leads to the definition of a minimal cut set:

> A minimal cut set is the smallest combination of basic events, which if they all occur, cause system failure. If any basic event in the set does not occur (i.e. the component works) then the system will not fail.

Fault trees constructed using different approaches are said to be logically equivalent if their minimal cut sets are identical. The order of a minimal cut set is the number of components within the set. In general, the lowest order minimal cut sets contribute most to system failure, as fewer components failures are needed for the top event to occur. Efforts should therefore be focussed on eliminating lower order minimal cut sets, especially those of order one, which represent single point failures in the system.

If NOT logic is used or implied, the combinations of basic events that cause the top event are called implicants. Minimal sets of implicants are called prime implicants.

To determine the cut sets of a fault tree, the Boolean logic expression for the top event must be transformed to a sum-of-products (s-o-p) form. This can be achieved with the use of a top-down or bottom-up approach, depending on which end of the tree is used to initiate the expansion process. The top-down procedure is described below and illustrated with the use of an example.

The process starts with the top event, which is expanded by continually substituting in the Boolean events appearing lower in the tree, until the expression contains only basic component failures. The product, '.', is used to represent 'AND' gates in the logic equations, and the sum, '+', is used to represent 'OR' gates. Expansion of the resulting equation gives the s-o-p form, from which the cut sets can be determined. If the fault tree contains repeated events then the resulting s-o-p expression may not be minimal and so the minimal cut sets cannot be obtained directly. Redundancies must be removed from the expression using the laws of Boolean algebra, to allow the extraction of the minimal cut sets. The laws are shown in section 2.3.1.

2.3.1 Boolean Laws of Algebra

1. Commutative Laws:

$$A + B = B + A$$
$$A.B = B.A$$

2. Associative Laws:

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

3. Distributive Laws:

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

4. Identities:

$$A + 0 = A \qquad A.0 = 0$$
$$A.1 = A \qquad A + 1 = 1$$

5. Idempotent Laws:

$A + A = A$ (removes repeated cut sets)

$A.A = A$ (removes repeated events within each cut set)

6. Absorption Laws:

$A + A.B = A$ (removes non-minimal cut sets)

$A.(A + B) = A$

7. Complementation:

$$\overline{A} = 1 - A$$
$$A.\overline{A} = 0$$
$$(\overline{\overline{A}}) = A$$

8. De Morgans Laws:

$$\overline{(A + B)} = \overline{A}.\overline{B}$$
$$\overline{(A.B)} = \overline{A} + \overline{B}$$

2.3.2 Example – Obtaining the Minimal Cut Sets

The top-down approach for calculating the minimal cut sets is demonstrated using the example fault tree shown in Figure 2.1.

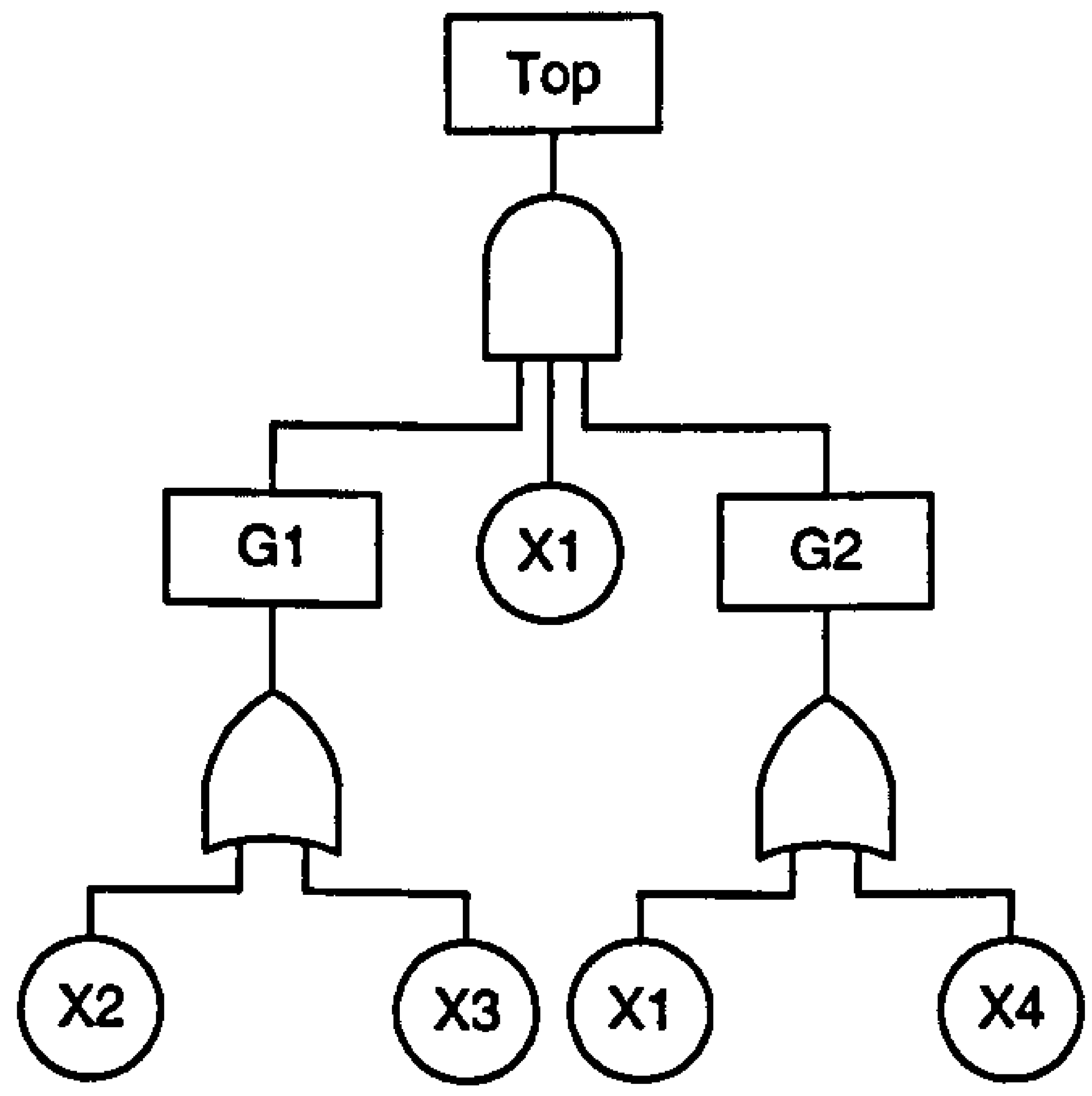


Figure 2.1: Example fault tree

Starting with the top event (Top), it is an 'AND' gate with three inputs, G1, X1 and G2. It can therefore be expressed as a product of these inputs:

$$Top = G1.X1.G2$$

As G1 is an 'OR' gate, made up of two events, X2 and X3, it can be written as:

$$G1 = X2+X3.$$

Substituting this into Top gives:

$$Top = (X2+X3).X1.G2$$

Similarly, G2 can be written as the sum of X1 and X4, so Top becomes:

$$Top = (X2+X3).X1.(X1+X4)$$

The expression now contains only basic events, so is expanded to give:

$$\begin{aligned} Top &= X2.X1.X1 + X1.X3.X1 + X2.X1.X4 + X1.X3.X4 \\ &= X1.X2 + X1.X3 + X1.X4.X2 + X1.X4.X3 \quad (\text{as } X1.X1 = X1) \end{aligned}$$

which gives the cut sets of the fault tree, expressed in s-o-p form. Redundancies can then be removed using the absorption law:

$$Top = X1.X2 + X1.X3$$

This is the minimal disjunctive form of the logic equation, each term of which is a minimal cut set. For this fault tree there are two minimal cut sets, both of order two (i.e. they each contain two basic events). These are {X1, X2} and {X1, X3}.

Obtaining the minimal cut sets for the tree in the example above is relatively straightforward. However, this was for a very small fault tree; a complex system may produce thousands of minimal cut sets. Determining the cut sets of a large system and their conversion to minimal form is a computationally intensive task due to the number of comparisons to be made. Although the algorithms are not complex, the process can be very time-consuming. For this reason, approximations such as culling are often implemented, which cull the cut sets above a certain order (for example above order four) during the calculation process, to reduce the number of computations and the time taken for the analysis. However, this obviously leads to a reduction in the accuracy of the minimal cut sets and so in the resulting quantitative analysis for which they are frequently used.

2.4 Quantitative Analysis

Quantitative analysis of the fault tree allows the calculation of a number of parameters, which are used to assess the system. The top event probability and frequency are used together with the expected number of occurrences of the top event and event importance measures to gain a full understanding of the system.

The methods for fault tree quantification are developed from Kinetic Tree Theory[3], which is a time-dependent methodology for system evaluation. These techniques form the basis of the approach used in the majority of commercial Fault Tree Analysis packages.

2.4.1 Structure Functions

The structure function for the top event of a fault tree shows the system state in relation to its components and is given by:

$$\varphi(\mathbf{x}) = 1 - \prod_{i=1}^{N_C} (1 - \rho_i(\mathbf{x})) \qquad 2.1$$

where $\rho_i(\mathbf{x})$ is the binary indicator function for each minimal cut set C_i, $i = 1..N_c$:

$$\rho_i(\mathbf{x}) = \prod_{j \in C_i} \beta_j \text{ such that } \rho_i = \begin{cases} 1 & \text{if cut set } C_i \text{ exists} \\ 0 & \text{if cut set } C_i \text{ does not exist} \end{cases} \qquad 2.2$$

and for each system component j, β_j is the binary indicator variable such that:

$$\beta_j = \begin{cases} 1 & \text{if component i is failed} \\ 0 & \text{if component i is working} \end{cases}$$

The structure function is also a binary indicator function, taking the following values:

$$\varphi(\mathbf{x}) = \begin{cases} 1 & \text{if the system is failed} \\ 0 & \text{if the system is working} \end{cases}$$

For the fault tree shown in Figure 2.1, which has minimal cut sets C_1 = {X1, X2} and C_2 = {X1, X3}, the structure function is given by:

$$\varphi(\mathbf{x}) = 1\text{-}(1\text{-}X1.X2)(1\text{-}X1.X3) \qquad 2.3$$

The probability of the top event is given simply by the expected value of the structure function,

$$Q_{sys}(t) = E[\varphi(\mathbf{x})] \qquad 2.4$$

If each minimal cut set is independent (i.e. no event appears in more than one cut set), then it is also true that:

$$E[\varphi(\mathbf{x})] = \varphi[E(\mathbf{x})] \qquad 2.5$$

Obtaining the expected value of the structure function for independent minimal cut sets would simply be a matter of substituting the probability of failure of each component into the structure function and calculating the result.

However, the minimal cut sets are not usually independent, and so in this case a full expansion of the structure function and then reduction of the indicator variables (i.e. $X_i = X_i^n$) must be undertaken.

Applying this to the structure function for the example fault tree (Equation 2.3), gives:

$$\begin{aligned} \varphi(\mathbf{x}) &= 1 - (1 - X1.X3 - X1.X2 + X1.X1.X2.X3) && \text{- expansion} \\ &= X1.X3 + X1.X2 - X1.X2.X3 && \text{- reduction} \end{aligned}$$

The probability of the top event is then given by the expected value of the expanded and reduced structure function:

$$Q_{sys}(t) = E(\varphi(\mathbf{x})) = P(X1).P(X3) + P(X1).P(X2) - P(X1).P(X2).P(X3) \qquad 2.6$$

A more efficient method of implementing this uses Shannon's Theorem.

2.4.2 Shannon's Theorem

Shannon's theorem[5] can be expressed as follows.

A Boolean function $f(\mathbf{x})$ where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ can be written as:

$$f(\mathbf{x}) = x_i.f(1_i, \mathbf{x}) + \bar{x}_i.f(0_i, \mathbf{x}) \qquad 2.7$$

where: $\overline{x_i} = 1\text{-}x_i$,

$f(1_i,\mathbf{x}) = f(x_1,\ldots, x_{i-1}, 1, x_{i+1},\ldots, x_n)$ and

$f(0_i,\mathbf{x}) = f(x_1,\ldots, x_{i-1}, 0, x_{i+1},\ldots, x_n)$

$f(1_i,\mathbf{x})$ and $f(0_i,\mathbf{x})$ are known as the residues of $f(\mathbf{x})$ with respect to x_i.

The structure function is pivoted around the most repeated variable using Shannon's expansion. This is continued until no repeated variables exist in the residues.

Shannon's theorem can be applied to the structure function given in Equation 2.3:

$$\varphi(\mathbf{x}) = 1\text{-}(1\text{-}X1.X2)(1\text{-}X1.X3)$$

Pivoting around the repeated variable, X1, gives:

$$\begin{aligned}\varphi(\mathbf{x}) &= X1[1\text{-}(1\text{-}X2)(1\text{-}X3)] + (1\text{-}X1)[0] \\ &= X1[1\text{-}(1\text{-}X2)(1\text{-}X3)]\end{aligned}$$

The probability of the top event is therefore given by:

$$Q_{sys}(t) = E(\varphi(\mathbf{x})) = P(X1).[1\text{-}(1\text{-}P(X2))(1\text{-}P(X3))] \qquad 2.8$$

Expanding this gives exactly the same result as shown in Equation 2.6.

2.4.3 General Method for the Calculation of the Top Event Probability

The general method of calculating the top event probability (i.e. the system unavailability) uses the minimal cut sets obtained from the qualitative analysis. This method can be used whether or not the fault tree contains repeated events.

The top event occurs if at least one minimal cut set exists, therefore for a fault tree that has N_c minimal cut sets, C_i, $Q_{sys}(t)$ is given by:

$$Q_{sys}(t) = P\left(\bigcup_{i=1}^{N_C} C_i\right) \qquad 2.9$$

Expanding gives:

$$Q_{sys}(t) = \sum_{i=1}^{N_C} P(C_i) - \sum_{i=2}^{N_C}\sum_{j=1}^{i-1} P(C_i \cap C_j) + \ldots + (-1)^{N_C-1} P(C_1 \cap C_2 \cap \ldots \cap C_{N_C}) \qquad 2.10$$

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

This expansion is known as the inclusion-exclusion expansion and generates the exact probability of the top event existence.

For example, consider the example fault tree shown in Figure 2.1, which has minimal cut sets $C_1 = \{X1, X2\}$ and $C_2 = \{X1, X3\}$. Equation 2.10 gives the top event probability as:

$$\begin{aligned} Q_{sys}(t) &= P(C_1) + P(C_2) - P(C_1 \cap C_2) \\ &= P(X1.X2) + P(X1.X3) - P(X1.X2.X1.X3) \\ &= P(X1).P(X2) + P(X1).P(X3) - P(X1).P(X2).P(X3) \end{aligned}$$

which is identical to the expression calculated in Equation 2.6.

It can be seen from the above expansion that if the fault tree has a large number of minimal cut sets then calculating this probability will be computationally intensive. For this reason, the calculation is simplified by the use of approximations.

2.4.3.1 Upper and Lower Bounds for System Unavailability

Truncation of the series in Equation 2.10 at an even-numbered term gives a lower bound for the top event probability; truncation at an odd-numbered term gives an upper bound for the top event probability:

$$\underbrace{\sum_{i=1}^{N_C} P(C_i) - \sum_{i=2}^{N_C}\sum_{j=1}^{i-1} P(C_i \cap C_j)}_{\text{Lower bound}} \le \underbrace{Q_{sys}(t)}_{\text{Exact}} \le \underbrace{\sum_{i=1}^{N_C} P(C_i)}_{\text{Upper bound}} \qquad 2.11$$

The upper bound is known as the Rare Event Approximation, P_{RE}(Top), as it is accurate if the component failure events are rare.

$$P_{RE}(\text{Top}) = \sum_{i=1}^{N_C} P(C_i) \qquad 2.12$$

2.4.3.2 Minimal Cut Set Upper Bound

A more accurate approximation for the top event probability is the Minimal Cut Set Upper Bound, P_{MCSUB}(Top). This is derived as follows:

$$\begin{aligned} \text{P(system failure)} &= \text{P(at least one minimal cut set exists)} \\ &= 1 - \text{P(no minimal cut sets exist)} \end{aligned} \qquad 2.13$$

Also,

$$\text{P(no minimal cut sets exist)} \ge \prod_{i=1}^{N_C} \text{P(minimal cut set i does not exist)} \qquad 2.14$$

(Equality exists when the minimal cut sets are independent i.e. when no event occurs in more than one cut set.)

Substituting Equation 2.14 into 2.13 gives:

$$P(\text{system failure}) \leq 1 - \prod_{i=1}^{N_C} P(\text{minimal cut set i does not exist}) \qquad 2.15$$

which gives the Minimal Cut Set Upper Bound:

$$P_{MCSUB}(Top) = 1 - \prod_{i=1}^{N_C}(1 - P(C_i)) \qquad 2.16$$

It can be shown that

$$\underset{\text{Exact}}{Q_{sys}(t)} \leq \underset{\text{Minimal Cut Set Upper Bound}}{1 - \prod_{i=1}^{N_C}(1 - P(C_i))} \leq \underset{\text{Rare Event Approximation}}{\sum_{i=1}^{N_C} P(C_i)} \qquad 2.17$$

2.4.4 Top Event Frequency

The top event frequency is another system parameter that can be calculated – this is useful for systems where unreliability is an important issue.

The system unconditional failure intensity, $w_{sys}(t)$, is defined as the probability that the top event occurs at t per unit time. Therefore, the probability that the top event occurs in the interval [t, t+dt) is given by $w_{sys}(t)dt$.

For the top event to occur in the interval [t, t+dt), no minimal cut sets can exist at time t, and at least one minimal cut set, θ_i, must occur in [t, t+dt). This can be written as:

$$w_{sys}(t)dt = P\left[A \bigcup_{i=1}^{N_C} \theta_i\right] \qquad 2.18$$

where: A is the event that no minimal cut sets exist at time t, $A = \bigcap_{i=1}^{N_C} u_i$.

u_i is the event that the i^{th} minimal cut set does not exist at t.

$\bigcup_{i=1}^{N_C} \theta_i$ is the event that at least one minimal cut set occurs in the interval [t, t+dt).

As $P(A) = 1 - P(\bar{A})$, the right hand side of Equation 2.18 can be written:

$$P\left[A \bigcup_{i=1}^{N_C} \theta_i\right] = P\left[\bigcup_{i=1}^{N_C} \theta_i\right] - P\left[\bar{A} \bigcup_{i=1}^{N_C} \theta_i\right] \qquad 2.19$$

where $\bar{A}$ is the event that at least one minimal cut set exists at t.

Therefore $w_{sys}(t)$ becomes:

$$w_{sys}(t)dt = P\left[\bigcup_{i=1}^{N_c}\theta_i\right] - P\left[\overline{A}\bigcup_{i=1}^{N_c}\theta_i\right] \quad 2.20$$

The first term on the right-hand side gives the contribution from the occurrence of at least one minimal cut set. The second term gives the contribution of the minimal cut set occurrence while other minimal cut sets already exist (i.e. the system is already failed). These terms are denoted by $w_{sys}^{(1)}(t)dt$ and $w_{sys}^{(2)}(t)dt$ respectively to give:

$$w_{sys}(t)dt = w_{sys}^{(1)}(t)dt - w_{sys}^{(2)}(t)dt \quad 2.21$$

The terms on the right of the above equation can be expanded using the inclusion-exclusion principle, but as this is a computationally intensive operation, an approximation is required.

2.4.4.1 Approximation for the System Unconditional Failure Intensity

If component failures are rare, then minimal cut set failures will also be rare events. The term $w_{sys}^{(2)}(t)dt$, which requires minimal cut sets to exist and occur at the same time, would become negligible if component failures are unlikely. Therefore, an upper bound for $w_{sys}(t)dt$ is simply:

$$w_{sys}(t)_{max}dt = w_{sys}^{(1)}(t)dt \quad 2.22$$

As this can be expanded using the inclusion-exclusion principle, the series expansion is truncated after the first term (as for the top event probability) to give the Rare Event Approximation:

$$\begin{aligned} w_{sys}(t)_{max}dt &\leq \sum_{i=1}^{N_c} P(\theta_i) \\ &\leq \sum_{k=1}^{N_c} w_{\theta_k}(t)dt \end{aligned} \quad 2.23$$

where: $P(\theta_i)$ is the probability of the occurrence of minimal cut set i

$w_{\theta_k}(t)$ is the unconditional failure intensity of minimal cut set θ_k.

Note that this is not the same as the Rare Event Approximation for the top event probability. Here $P(\theta_i)$ denotes the probability of **occurrence** of a minimal cut set; for the top event probability, $P(C_i)$ denoted the probability of **existence** of a minimal cut set.

2.4.4.2 Expected Number of System Failures

The expected number of system failures in time t, W(0, t), is given by the integral of the system unconditional failure intensity in the interval t.

$$W(0,t) = \int_0^t w_{sys}(u)du \qquad 2.24$$

For a reliable system, the expected number of system failures is an upper bound for the system unreliability, F(t) (i.e. $F(t) \le W(0,t)$).

2.4.5 Importance Measures

The importance measure of a component or minimal cut set is given by a numerical value and signifies the role that the component or cut set plays in contributing to the top event. This allows the components or cut sets to be ranked in order according to the extent of their contribution to system failure. Importance measures are useful as they can identify weak areas of a system, which is especially important at the design stage.

Importance measures can be categorised as either deterministic or probabilistic. Probabilistic measures can themselves be subdivided into two categories: those dealing with system unavailability assessment and those dealing with the system unreliability assessment.

2.4.5.1 Deterministic Measures

Deterministic importance measures evaluate the importance of a component without considering its probability of failure. One such measure is the structural measure of importance.

Structural Measure of Importance

The structural measure of importance for component i is given by:

$$I_i = \frac{\text{number of critical system states for component i}}{\text{total number of states for the (n - 1) remaining components}} \qquad 2.25$$

A critical system state for component i is a state for which the failure of component i will cause the system to go from a working to a failed state

2.4.5.2 Probabilistic Measures for System Unavailability

Probabilistic measures are generally of more use than deterministic measures in reliability problems as they take into account the components' probability of failure.

Birnbaum's Measure of Importance

This measure[6] is also known as the criticality function and is defined as the probability that the system is in a critical state for component i.

There are two expressions for the criticality function:

- $$G_i(q(t)) = Q_{sys}(1_i, q(t)) - Q_{sys}(0_i, q(t)) \tag{2.26}$$

  where $Q_{sys}(t)$ is the probability that the system fails

  $(1_i, q(t)) = (q_1, \ldots, q_{i-1}, 1, q_{i+1}, \ldots, q_n)$ component i failed

  $(0_i, q(t)) = (q_1, \ldots, q_{i-1}, 0, q_{i+1}, \ldots, q_n)$ component i working

  The above expression gives the probability that the system fails with component i failed, minus the probability of the system failing with component i working, which results in the probability that the system fails only if component i fails.

- $$G_i(q(t)) = \frac{\partial Q_{sys}(t)}{\partial q_i(t)} \tag{2.27}$$

  This is equivalent to Equation 2.26, as:

  $$\frac{\partial Q_{sys}(t)}{\partial q_i} = \frac{Q_{sys}(1_i, q(t)) - Q_{sys}(0_i, q(t))}{1-0} \tag{2.28}$$

This measure of importance forms the basis for many other importance measures.

Criticality Measure of Importance

This calculates the probability that the system is in a critical state for component i and that i has failed. Unlike Birnbaum's measure of importance, it also takes into account the failure probability of component i itself.

$$I_i = \frac{G_i(q(t))q_i(t)}{Q_{sys}(t)} \tag{2.29}$$

Fussell-Vesely Measure of Importance

This measure[7] calculates the probability that component i contributes to system failure and is defined as the probability of the union of the minimal cut sets containing i, given that the system has failed.

$$I_i = \frac{P\left(\bigcup_{k|i\in C_k} C_k\right)}{Q_{sys}(t)} \tag{2.30}$$

This measure gives very similar importance rankings to those obtained using the criticality measure.

Fussell-Vesely Measure of Minimal Cut Set Importance

This measure of importance[7] ranks the minimal cuts sets in the order of their contribution to the top event, rather than considering the individual components. It is defined as the probability of existence of the cut set i, given that the system has failed.

$$I_i = \frac{P(C_i)}{Q_{sys}(t)} \qquad 2.31$$

2.4.5.3 Probabilistic Measures for System Unreliability

These importance measures assess the interval reliability of a system, where the order in which components fail is important. The sequence of failure can be described with the use of **enabling** and **initiating** events. This analysis is of particular use in safety protection systems, where the order in which the protection system fails and some hazardous event occurs is extremely important. For example if the protection system fails first, then the hazardous event occurs, the result will be system failure. However if the hazardous event occurs first, then the protection system will invoke shutdown and a dangerous situation will be avoided. In this case, the protection system failure is an enabling event, which would put the system into a critical state. The hazardous failure is an initiating event, which would result in a dangerous system failure only if the enabling event has already occurred; if the initiating event occurs first, then the safety system would respond as required. The formal definitions of initiating and enabling events are given as:

- Initiating events perturb system variables and place a demand on control/protection systems to respond.
- Enabling events are inactive control/protective systems that permit initiating events to cause the top event.

Barlow-Proschan Measure of Initiator Importance

The Barlow-Proschan measure of initiator importance[8], I_i, is the probability that the system is in a critical state for component i at time t and that the occurrence of initiating event i in the interval [t,t+dt), causes the system to fail.

$$I_i = \frac{\int_0^t G_i(q(u))w_i(u)du}{W(0,t)} \qquad 2.32$$

Modified Measure of Enabler Importance

This importance measure[9] gives the probability that the enabling event i permits the initiating event j to cause system failure in the interval [0,t). The failure of the enabler i is considered only a factor when it is contained in the same minimal cut set as the initiating event j:

$$I_i = \frac{\sum\limits_{\substack{j \\ j \neq i}}^{n} \int\limits_0^t \left[G_{i,j}(q(u)) - G_{M_{i,j}}(q(u))\right] q_i(u) w_j(u) du}{W(0,t)} \qquad 2.33$$

where $G_{i,j}$ is the criticality of components i and j given by:

$$G_{i,j}(q(t)) = \frac{\partial^2 Q_{sys}(t)}{\partial q_i(t) \partial q_j(t)} \qquad 2.34$$

and $G_{M_{i,j}}$ is a correction to the term $G_{i,j}$, which eliminates the separate roles of components i and j. Further discussion of this measure can be found in reference 9.

Measure of Minimal Cut Set Frequency Importance

This measure[10] gives the probability that a minimal cut set of order m causes the system failure in the interval [0,t), given that the system has failed:

$$I(C_i^m) = \frac{\sum\limits_{k=1}^{m} \int\limits_0^t \left[G_{\{C_i^m\}}(q(u)) - G_{M_{\{C_i^m\}}}(q(u))\right] \prod\limits_{\substack{j=1 \\ j \neq k}}^{m} q_j(u) w_k(u) du}{W(0,t)} \qquad 2.35$$

where $G_{\{C_i^m\}}(q(t))$ is the criticality of cut set i, defined as:

$$G_{\{C_i^m\}}(q(t)) = \left. \frac{\partial^2 Q_{sys}(t)}{\partial q_k^m(t)} \right|_{k \in C_i^m} \qquad 2.36$$

and $G_{M_{\{C_i^m\}}}(q(t))$ is a correction to the term $G_{\{C_i^m\}}(q(t))$ that eliminates the separate effects of the components contained in C_i^m

2.5 Fault Tree Modularisation

Modularisation methods can be applied to fault trees in order to reduce their complexity and simplify the resulting analysis. The modularisation procedure identifies independent subtrees within the fault tree, known as modules. A module is defined as a section of the fault tree that is completely independent from the rest of the tree, with no inputs that appear anywhere else in the tree and no outputs to the rest of the tree except from its output event. The advantage of identifying modules is that each one can be analysed independently of the rest of the tree. In effect, the modules can be regarded as individual fault trees and analysed as such.

Several modularisation techniques are available for detecting fault tree modules, but one of particular interest is the linear-time algorithm[11]. The advantage of this algorithm over other techniques is its efficiency, as it requires only two passes through the fault tree to obtain the modules.

2.5.1 Principles of the Linear-Time Algorithm

The modules can be identified after just two depth-first traversals of the fault tree. The first of these performs a step-by-step traversal recording for each gate and event, the step number at the first, second and final visits to that node. To demonstrate this process, refer to the fault tree in Figure 2.2. Starting at the top event and progressing through the tree in a depth-first manner, the gates and events are visited in the order shown in Table 2.3. Event inputs to any gate are considered before the gate inputs. Each gate is visited at least twice: once on the way down the tree and again on the way back up the tree. Once a gate has been visited, it can be visited again, but the depth-first traversal beneath that gate is not repeated. This is shown at step 30 in Table 2.3, where G4 is visited again, but its descendants (any gates and events appearing below that gate in the tree) are not re-visited.

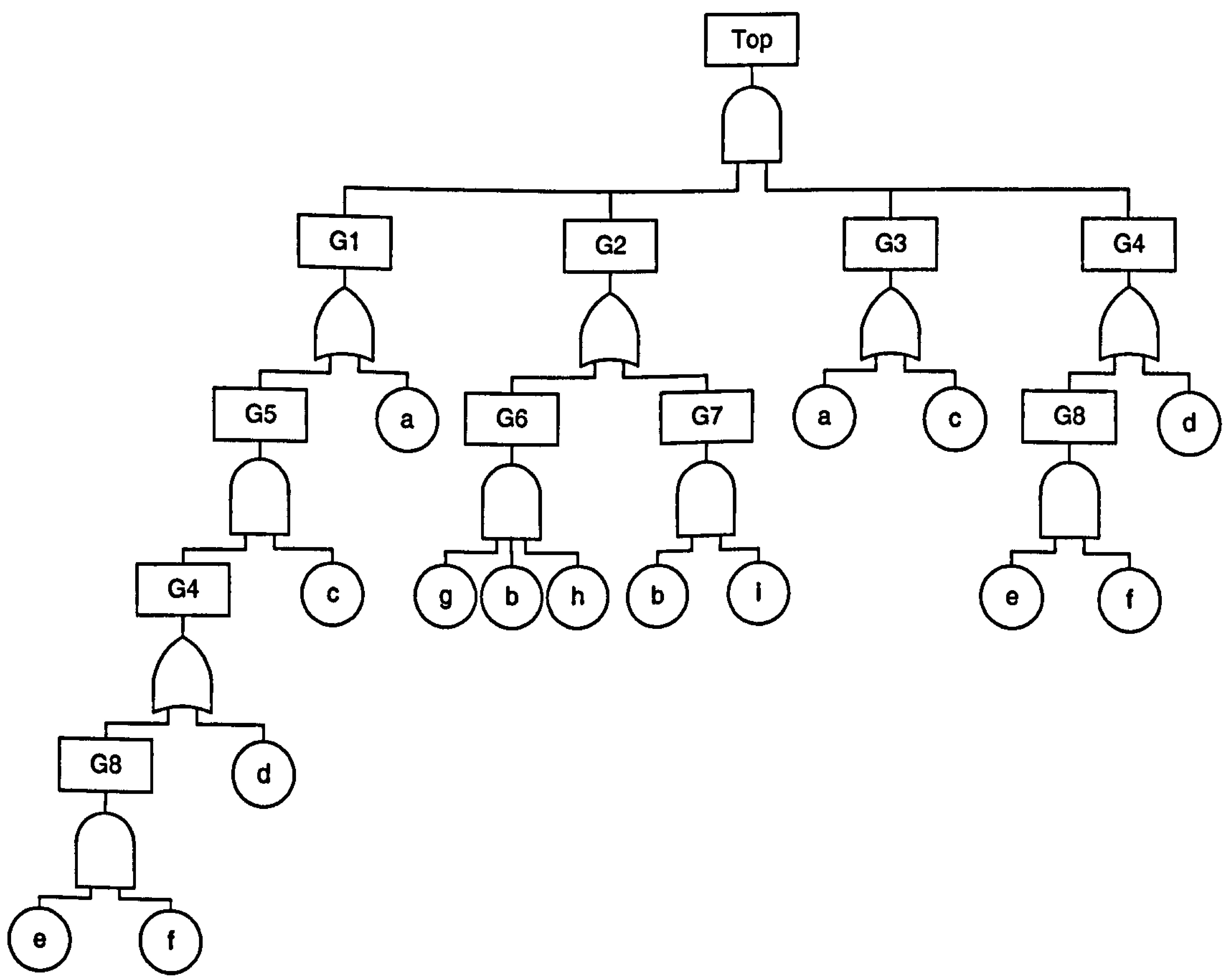


Figure 2.2: Example fault tree to demonstrate the linear-time algorithm

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 2.3: Order in which the gates and events are visited in the depth-first traversal of the fault tree in Figure 2.2

The step numbers of the visits (first, second and final) are recorded during this traversal and the values for the gates are shown in Table 2.4. As G4 is a repeated gate, the step number of the final visit is different to that of the second visit. The equivalent data for the events is shown in Table 2.5. It should be noted that the step number of the second visit to each basic event is always equivalent to the step number of the first visit to that event.

Gate	Top	G1	G2	G3	G4	G5	G6	G7	G8
1st visit	1	2	15	26	6	4	16	21	8
2nd 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 2.4: Data for the gates in the fault tree

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

Table 2.5: Data for the events in the fault tree

The second pass through the tree finds the maximum (max) of the last visits and the minimum (min) of the first visits to the descendants of each gate; these values are also shown in Table 2.4. The principle of the algorithm is that if any descendant of a gate has a first visit step number smaller than the first visit step number of the gate, then it must also occur beneath another gate. Conversely, if any descendant has a last visit step number greater than the second visit step number of the gate, then again it must occur elsewhere in the tree. Therefore, a gate can be identified as heading a module iff:

- 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.

That is, none of the descendents of a gate can appear anywhere else in the tree (unless beneath another occurrence of the same gate). Therefore, the final step of the algorithm simply compares the minimum (min) and maximum (max) values of the descendents visit numbers with the first and second visit step numbers for each gate.

From Table 2.4, it can be seen that gates G1, G5 and G6 cannot be modules, as their descendants have maximum step numbers greater than the second visit step numbers of those gates. Gates G3 and G7 are also not modules, as their descendants have minimum step numbers smaller than the first visit step numbers of the gates.

The following gates can therefore be identified as heading modules:

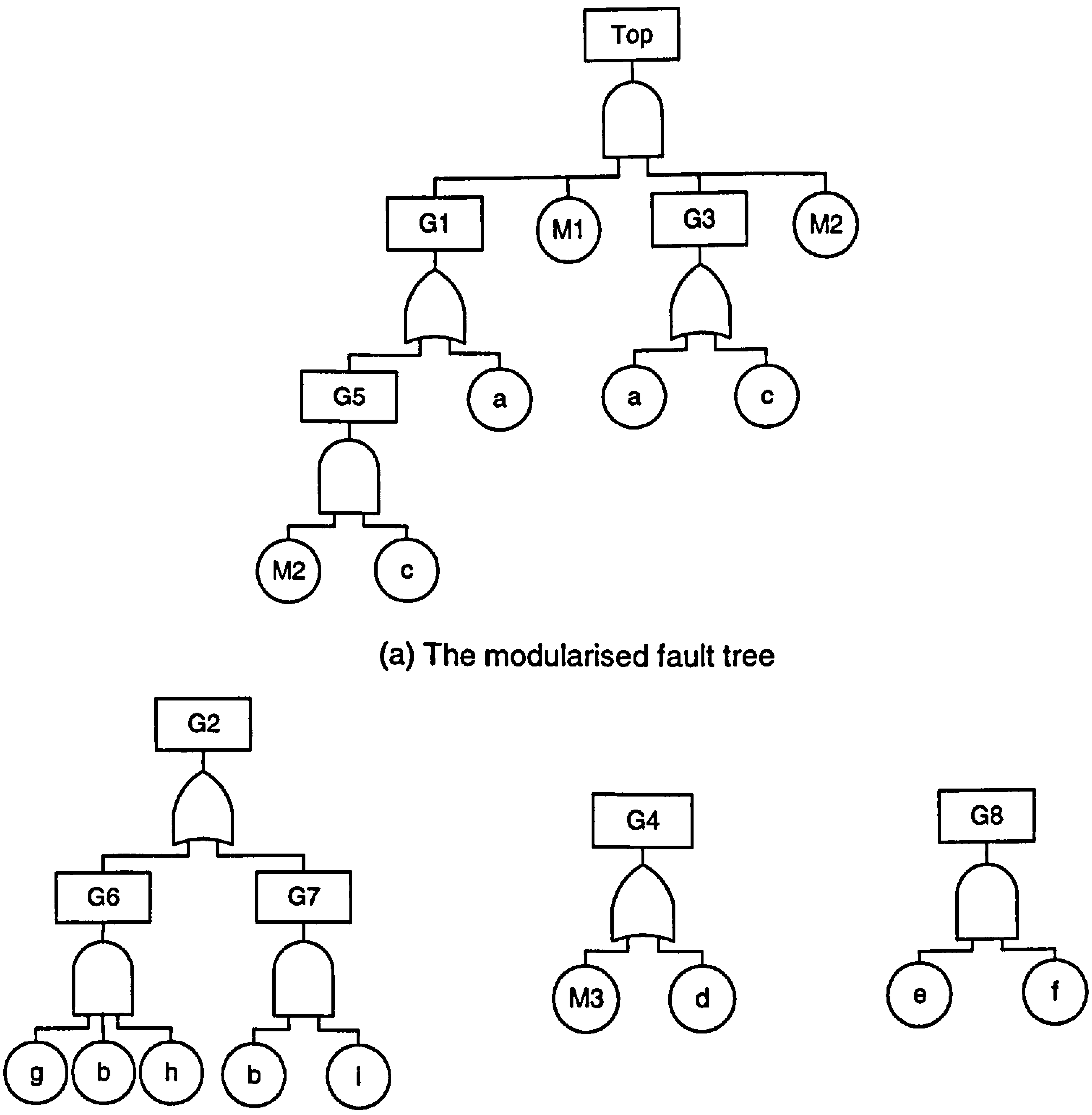
Top, G2, G4, and G8

For completeness, the top event (Top) is included in this list, even though it will always be a module of the fault tree.

Each of the subtrees can be replaced by a single modular event in the fault tree structure and are assigned the following labels:

$G2 \rightarrow M1$, $G4 \rightarrow M2$ and $G8 \rightarrow M3$

Four separate fault trees as shown in Figure 2.3 now replace the single fault tree shown in Figure 2.2.



(a) The modularised fault tree

(b) Module M1 (c) Module M2 (d) Module M3

Figure 2.3: The four modules obtained from the fault tree shown in Figure 2.2

Having identified the modules, each one can be analysed separately and the results substituted into the higher-level fault trees where the modules occur. This process can significantly reduce the number of calculations required in the subsequent analysis. The linear-time algorithm has been programmed as part of the research and a detailed description can be found in Appendix I.

2.6 Summary of Fault Tree Analysis

Fault trees are an extremely good way of representing the failure logic of a system in a visual format. However, if the fault tree is large, then performing analysis upon it (such as finding the minimal cut sets, top event probability, etc) can require extensive calculations and consequently, considerable computing power. Approximations are needed for many parameters, which inevitably leads to a loss of accuracy. Finding more efficient and accurate means of performing these calculations has been the subject of much research, which has led to the introduction of the Binary Decision Diagram technique as an alternative method for this analysis.

Chapter 3: Binary Decision Diagrams

3.1 Introduction

Binary Decision Diagrams (BDDs) were first used by Lee[12] to represent switching circuits. Their use in reliability analysis was developed predominantly by Rauzy[4], who suggested that they might provide an alternative, more efficient technique for performing fault tree analysis.

The BDD method does not analyse the fault tree directly, but converts the tree to a Binary Decision Diagram, which represents the Boolean equation for the top event. This representation of the logic equation is in a form that is much easier to manipulate than a fault tree and so lends itself well to the mathematical analysis. Both qualitative and quantitative analysis can be performed on the BDD, with the advantage that exact solutions can be calculated very efficiently without the need for the approximations necessary in the conventional approach of Kinetic Tree Theory.

3.2 Properties of the BDD

A BDD is a directed acyclic graph, which means that all paths through the BDD are in one direction and that no loops can exist. The BDD is composed of terminal and non-terminal vertices (also called nodes) connected by branches. The non-terminal vertices encode the basic events of the fault tree and the terminal vertices correspond to the final state of the system. These are shown on the BDD in Figure 3.1.

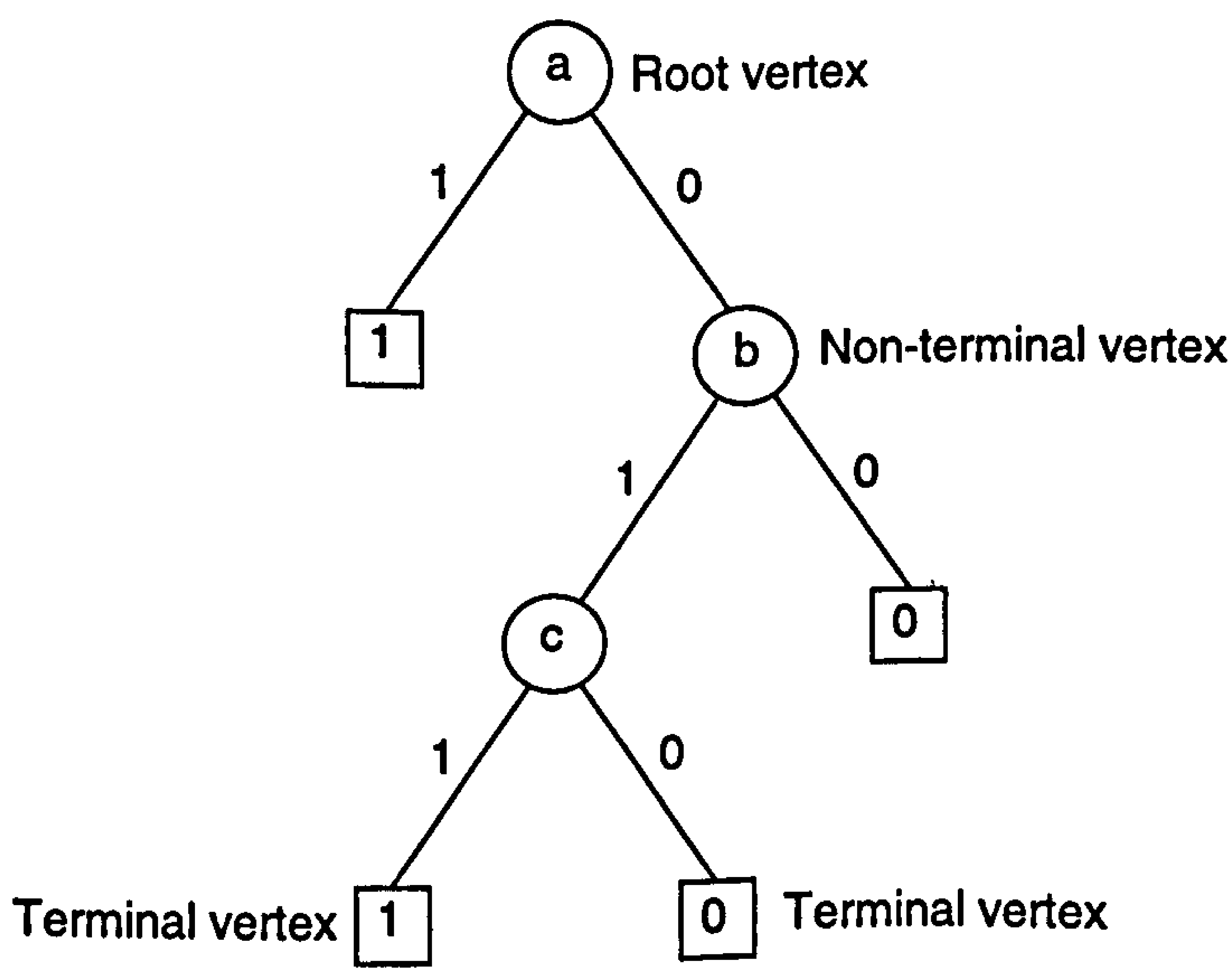


Figure 3.1: Example Binary Decision Diagram

Non-terminal vertices have two outgoing branches. By convention, the left-hand branch is a '1' branch, corresponding to basic event occurrence (i.e. the component fails); the right-hand branch is a '0' branch corresponding to basic event non-occurrence (i.e. the component works). The size of a BDD is usually measured by its number of non-terminal vertices. Terminal vertices have a value of either one or zero, corresponding to top event occurrence (i.e. the system fails) and non-occurrence (i.e. the system works) respectively.

All paths through the diagram start at the root vertex and proceed to a terminal vertex, which marks the end of the path. Each path that terminates in a '1' state gives a cut set of the fault tree, as that particular combination of component failures must result in system failure. Only vertices that lie on the '1' branches of these paths are included in the cut sets. For example, in the BDD shown in Figure 3.1, there are two possible paths that terminate in '1' states. These are:

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

which gives the two corresponding cut sets:

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

In this example, the BDD is in its minimal form and so generates minimal cuts sets. However, this is not always the case, as is discussed later in this chapter.

3.3 Formation of the BDD Using the Structure Function

One method of constructing the BDD uses the structure function of the fault tree. An ordering of the fault tree variables must be chosen, which determines the order in which they are considered in the construction process. The choice of variable ordering also has a significant effect on the size of the resulting BDD, a subject that is discussed in more detail in section 3.6 and Chapter 4. Values of one and zero are then successively substituted into the structure function equation for each node in the BDD, according to the chosen ordering. This process is demonstrated using the fault tree shown in Figure 3.2.

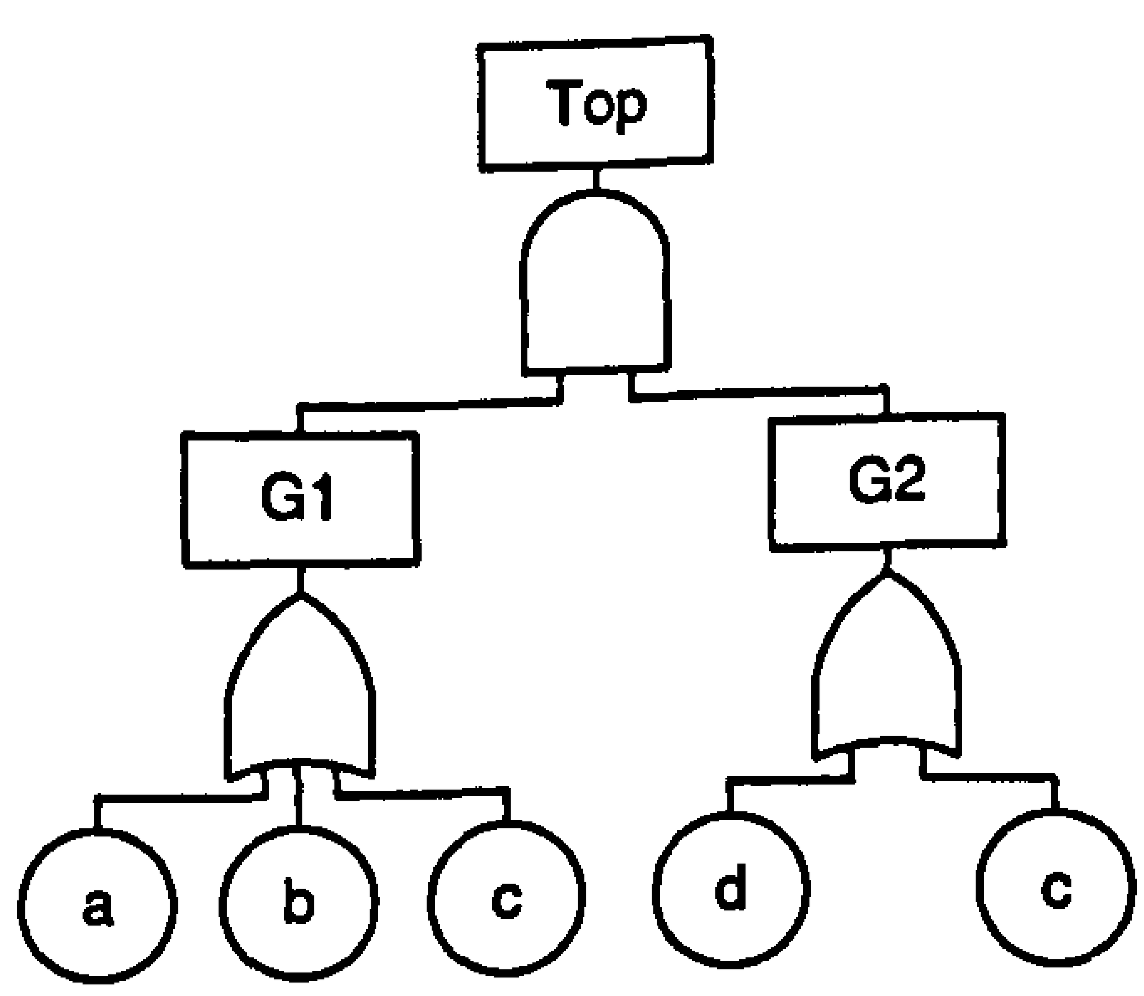


Figure 3.2: Example fault tree

The minimal cuts sets for this tree are:

1. {a, d}
2. {b, d}
3. {c}

which gives the following structure function:

$$\varphi = 1 - (1 - a.d)(1 - b.d)(1 - c)$$

The BDD is constructed according to the variable ordering a<b<c<d, which was chosen by listing the variables as they appear from left to right in the fault tree. The ordering means that event 'a' is considered first, then event 'b' and so on, until the BDD has been fully constructed. The first node, which encodes event 'a', is drawn with its two outgoing branches. The result of the left-hand branch is obtained by substituting the value one into the structure function equation for each occurrence of 'a'; the result of the right-hand branch is found by substituting in the value zero for 'a'. The remaining variables are then considered in the same way, according to the chosen ordering, until the terminal vertices are reached. The resulting BDD with its Boolean equations is shown in Figure 3.3.

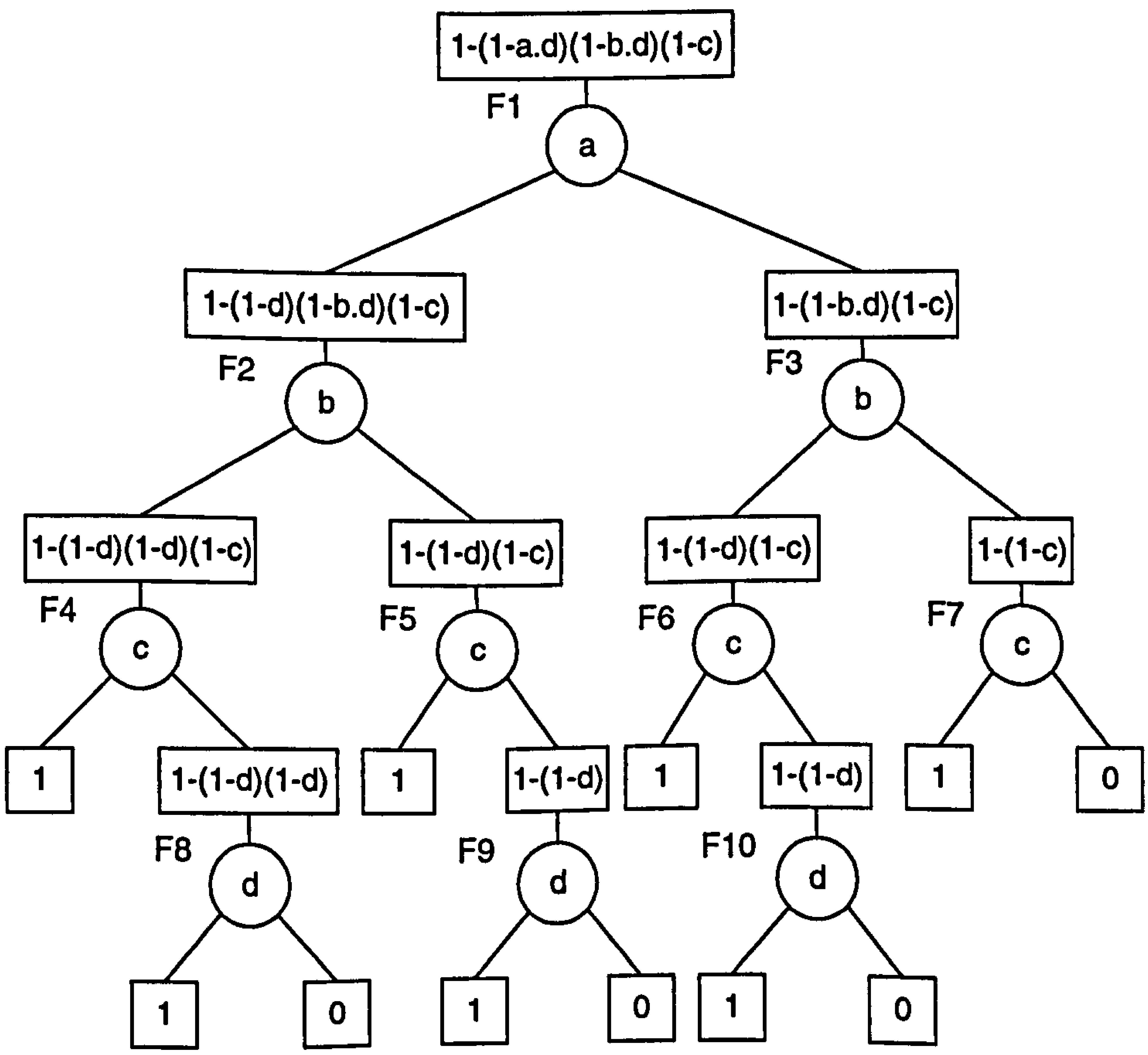


Figure 3.3: Binary Decision Diagram with Boolean equations

The resulting BDD is not, however, in its simplest form. It consists of ten non-terminal nodes, which can be reduced by applying the reduction technique outlined in the following section.

3.3.1 Reduction of the BDD

The following 'collapsing' operations (Friedman and Supowit[13]) can be used to reduce the size of the BDD:

1. If the two sons of a node 'a' are equivalent, then delete node 'a' and direct all of its incoming branches to its left son.
2. If nodes 'a' and 'b' are equivalent, then delete node 'b' and direct all of its incoming branches to 'a'.

The son of a node is the node to which either the one or the zero branch leads.

The above operations can be applied to reduce the BDD in Figure 3.3. Operation 1 is first applied to delete node F2, as both its sons are equivalent. Its incoming branch from node F1 is therefore directed to its left son, node F4. Nodes F5 and F9 are deleted. Then, operation 2 can be applied to the equivalent nodes F4 and F6. Node F6 and its sons are deleted and its incoming branch from node F3 is directed to F4. This is known as 'sub-node sharing' and results in the BDD shown in Figure 3.4.

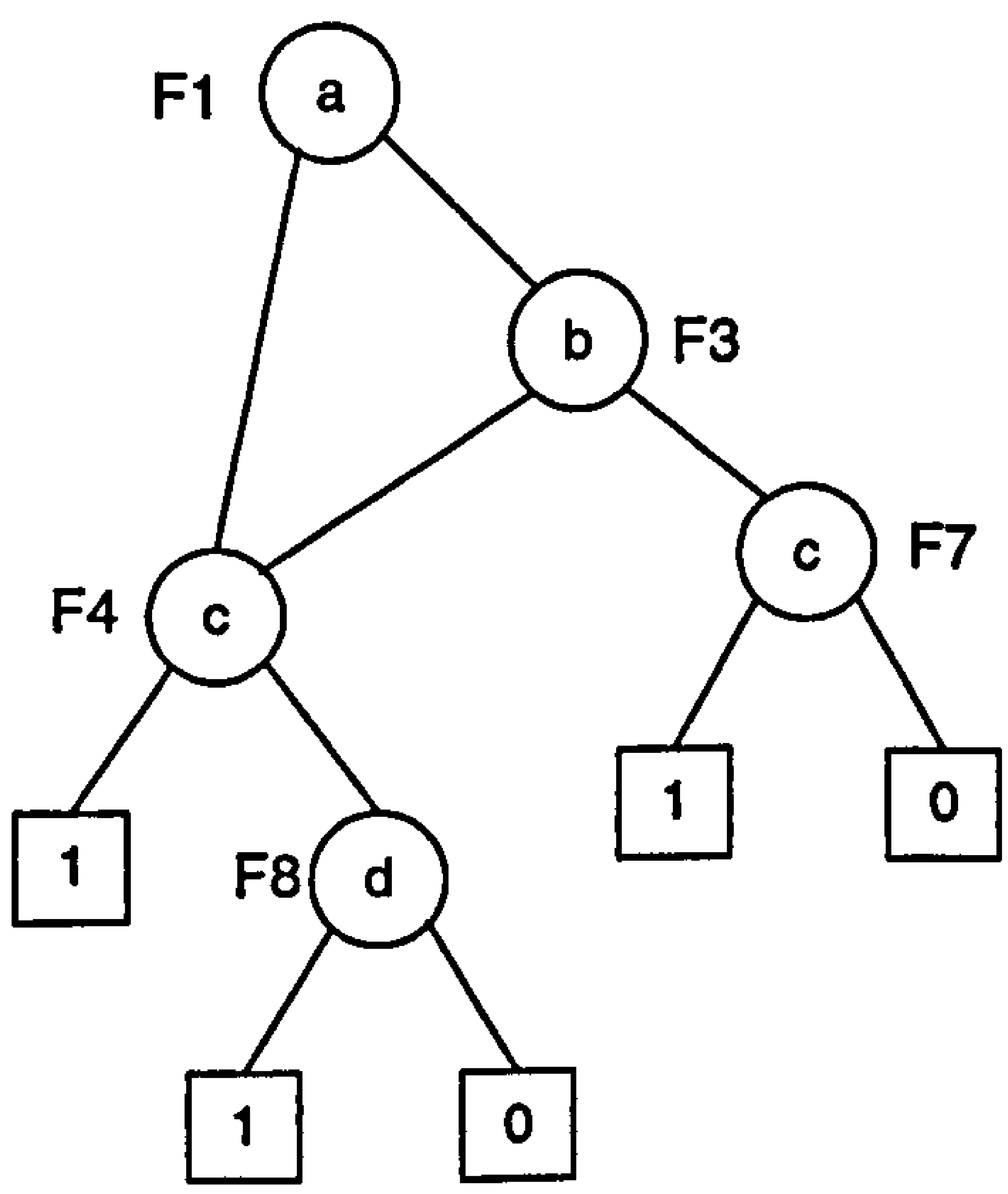


Figure 3.4: The reduced BDD from Figure 3.3

The reduced BDD is significantly smaller than the original, with five non-terminal nodes as opposed to ten. Two of the redundant cut sets that could be found from Figure 3.3 have also been eliminated. However, the reduced BDD is not minimal, as the BDD paths result in a further two non-minimal cut sets. To obtain minimal cut sets from the BDD, it must undergo a minimisation procedure, which is described in section 3.5.

The reduction technique does not alter the logic of the BDD, but it does reduce computer memory requirements.

Although the method of constructing the BDD from the structure function clearly indicates the relationship between the fault tree and the BDD, an obvious disadvantage is that the cut sets must be determined before the BDD can be constructed. As the aim of the BDD method is to perform the analysis more efficiently, an alternative method is implemented.

3.4 Formation of the BDD Using If-Then-Else

This method of constructing the BDD was developed by Rauzy[4] and proceeds by applying an **if-then-else (ite)** technique to each of the gates in the fault tree. The **ite** structure derives from Shannon's formula, which is discussed in detail in Chapter 2. If f(**x**) is the Boolean function for the fault tree top event then by pivoting about any variable X1, Shannon's formula can be written as:

$$f(\mathbf{x}) = X1.f1 + \overline{X1}.f2 \qquad 3.1$$

where f1 and f2 are Boolean functions with X1=1 and X1=0 respectively, and are of one order less than f(**x**).

The corresponding **ite** structure is **ite**(X1, f1, f2), where X1 is the Boolean variable and f1 and f2 are logic functions. This means that **if** X1 fails **then** consider f1, **else** consider f2. Therefore in the BDD structure, f1 lies below the '1' branch of the node encoding X1 and f2 lies below the '0' branch. This is shown in Figure 3.5.

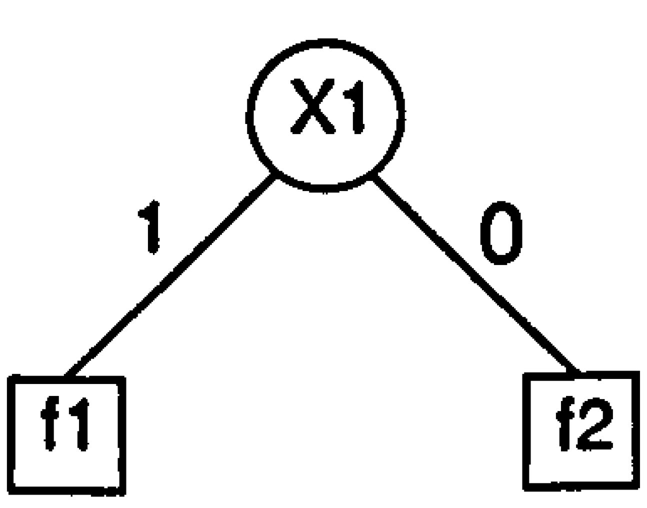


Figure 3.5: BDD showing **ite**(X1, f1, f2)

Once a variable ordering has been established, the following procedure can be implemented to construct the BDD:

- Each basic event Xi is assigned the **ite** structure **ite**(Xi,1,0).

- If X<Y (i.e. X appears before Y in the variable ordering):

  Let J = **ite**(X, F1, F2) and H = **ite**(Y,G1, G2), then

  J<op>H=**ite**(X,F1<op>H, F2<op>H).

- If X=Y:

  J = **ite**(X, F1, F2), H = **ite**(X,G1,G2), then

  J<op>H = **ite**(X,F1<op>G1,F2<op>G2).

where <op> corresponds to a Boolean operation of the gates in the fault tree.

The following identities can also be used to simplify the results:

1<op>H=1, if <op> is an 'OR' gate.
1<op>H=H, if <op> is an 'AND' gate.
0<op>H=H, if <op> is an 'OR' gate.
0<op>H=0, if <op> is an 'AND' gate.

An advantage of the **ite** method for constructing the BDD is that the algorithm automatically makes use of sub-node sharing. This not only reduces the computer memory requirements, as each **ite** structure is only stored once, but it also increases the efficiency, as once an **ite** structure has been calculated, the process does not need to be repeated.

The **ite** method can be demonstrated by constructing a BDD from the fault tree shown in Figure 3.6.

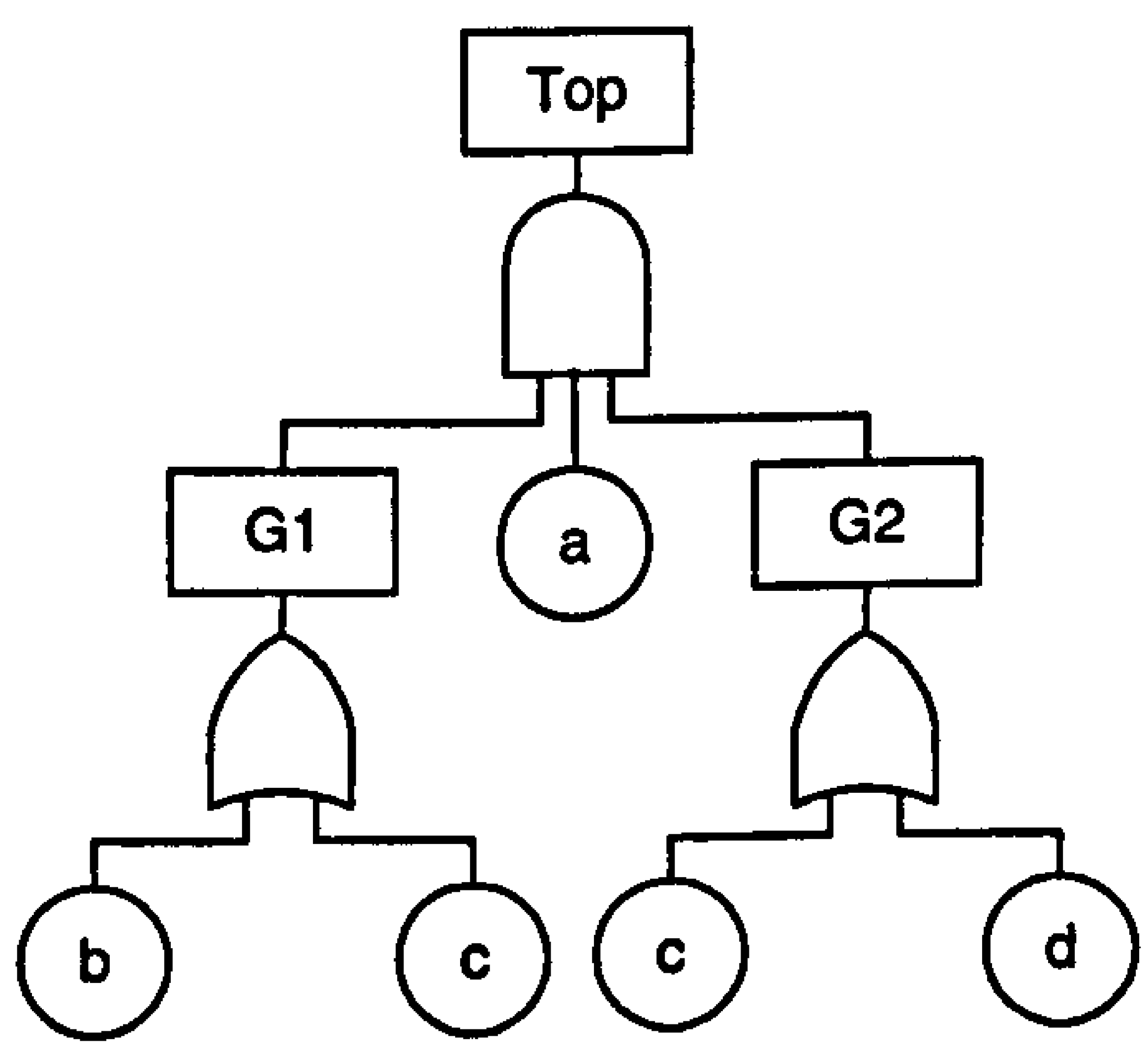


Figure 3.6: Example fault tree for the **ite** method

The ordering $a<b<c<d$ is chosen, which is obtained by a simple top-down, left-right traversal of the fault tree (known as top-down ordering).

G1 is expressed as:

G1 = b + c
= **ite**(b, 1, 0) + **ite**(c, 1, 0)
= **ite**(b, 1, **ite**(c, 1, 0))

G2 is found in a similar way to give:

G2 = **ite**(c, 1, **ite**(d, 1, 0))

The **ite** structure for Top is therefore given by:

Top= a.G1.G2

= **ite**(a, 1, 0) . **ite**(b, 1, **ite**(c, 1, 0)) . **ite**(c, 1, **ite**(d, 1, 0))

= **ite**(a, 1, 0) . **ite**(b, **ite**(c, 1, **ite**(d, 1, 0)), **ite**(c, 1, 0) . **ite**(c, 1, **ite**(d, 1, 0)))

= **ite**(a, 1, 0) . **ite**(b, **ite**(c, 1, **ite**(d, 1, 0)), **ite**(c, 1, 0))

= **ite**(a, **ite**(b, **ite**(c, 1, **ite**(d, 1, 0)), **ite**(c, 1, 0)), 0)

The BDD is constructed by considering the one and zero branches of each variable in turn. In this example, 'a' is the first variable to be considered and is encoded in the root vertex of the BDD. The structure **ite**(b, **ite**(c, 1, **ite**(d, 1, 0)), **ite**(c, 1, 0)) lies below its one branch and the terminal '0' vertex lies below the zero branch. Event 'b' is the next variable to be considered and is encoded in the node beneath the left-hand branch of the root vertex. Its outgoing branches are determined by breaking down the structure **ite**(b, **ite**(c, 1, **ite**(d, 1, 0)), **ite**(c, 1, 0)) into **ite**(c, 1, **ite**(d, 1, 0)) for the one branch and **ite**(c, 1, 0) for the zero branch. This process is continued until all branches end with terminal vertices. The resulting BDD is shown in Figure 3.7.

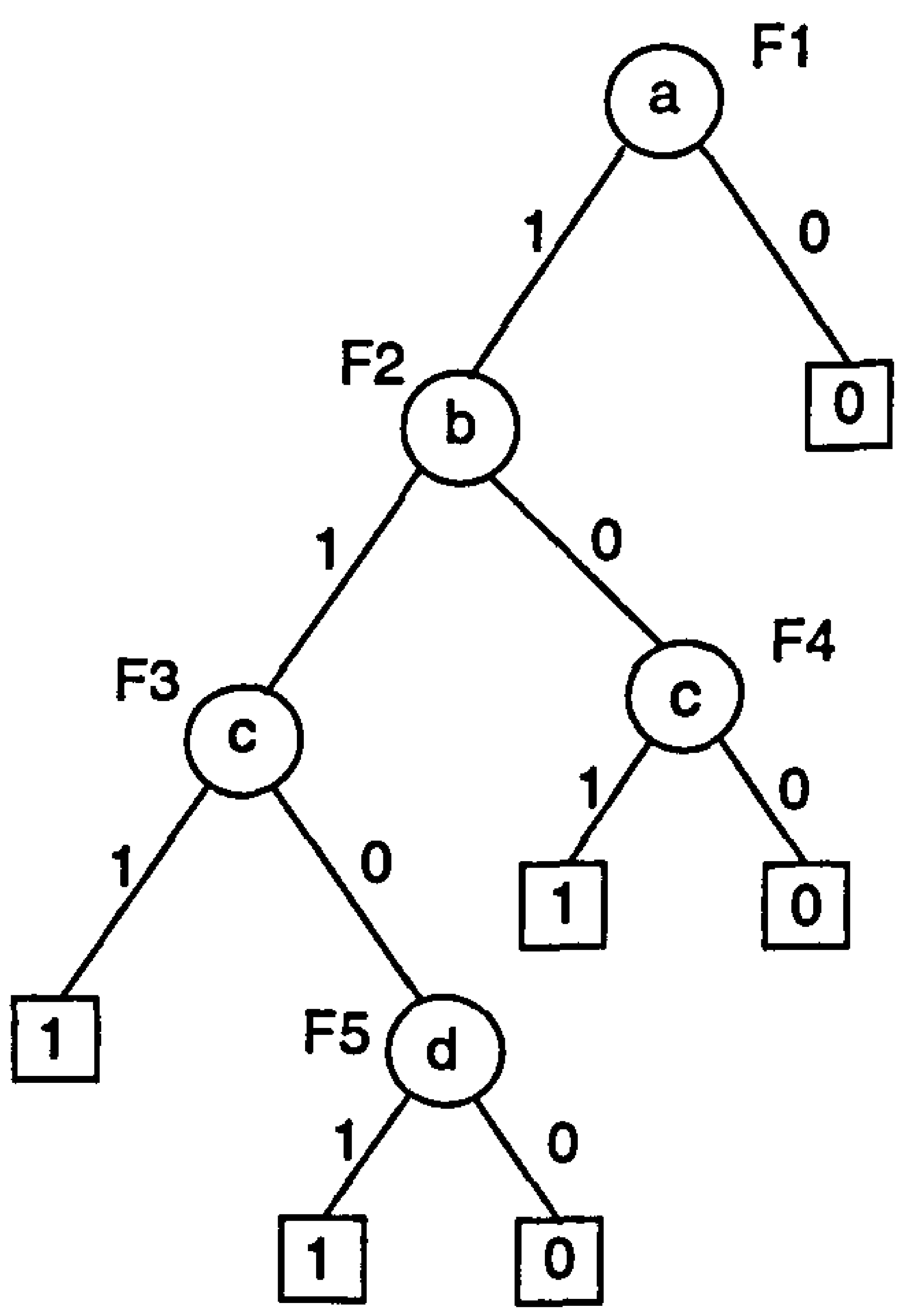


Figure 3.7: BDD obtained from the fault tree in Figure 3.6, with the ordering a<b<c<d

The cut sets, which are obtained from the paths ending with terminal '1' nodes, are:

1. {a, b, c}
2. {a, c}
3. {a, b, d}

The BDD is not minimal and therefore does not generate minimal cut sets. The first cut set is redundant, as it contains the second cut set as a subset. In order to obtain minimal cut sets the BDD has to undergo a minimisation procedure, which is introduced in the following section.

3.5 Minimisation

The cut sets produced from the BDD are only minimal if the BDD is in its minimal form. In order to get a non-minimal BDD into this form, it must undergo a minimising procedure. This process, introduced by Rauzy[4], is applied to the **ite** form of the BDD and creates a new BDD that exactly defines the minimal cuts sets of the fault tree. If there are shared nodes in the BDD, then these must be expanded out prior to minimisation.

Consider a general node in the BDD whose output represents the function F, where

$$F = \textbf{ite}(x, G, H) \tag{3.2}$$

If δ is a minimal solution of G, which is **not** a minimal solution of H, then the intersection of δ and x, $(\{\delta\} \cap x)$, will be a minimal solution of F. The set of all the minimal solutions of F, $sol_{min}(F)$ will also include the minimal solutions of H, so:

$$sol_{min}(F) = \{\sigma\} \tag{3.3}$$

where,

$$\sigma = [\{\delta\} \cap x] \cup [sol_{min}(H)] \tag{3.4}$$

Rauzy has defined the 'without' operator, **without**(G_{min}, H_{min}), which removes all the paths from G_{min} that are included in a path of H_{min}. This ensures that the combined set in Equation 3.4 represents the minimal solutions of F, by removing any minimal solutions of G that are also minimal solutions of H.

This algorithm can be applied to the BDD in Figure 3.7. Each node is considered in turn:

F1 = **ite**(a, F2, 0) - F2 does not contain any paths that are included in the zero branch, as this leads to a terminal vertex.

F2 = **ite**(b, F3, F4) - Event 'c' is included in a path on both the one branch (F3) and the zero branch (F4). Therefore 'c' is removed from the one branch by replacing the terminal '1' vertex with a terminal '0' vertex.

F3 = **ite**(c, 1, F5) - F5 does not contain any paths that are included in the one branch as it leads to a terminal vertex.

F4 = **ite**(c, 1, 0) - Both the one and zero branches are terminal.

F5 = **ite**(d, 1, 0) - Both the one and zero branches are terminal.

The minimised BDD is shown in Figure 3.8.

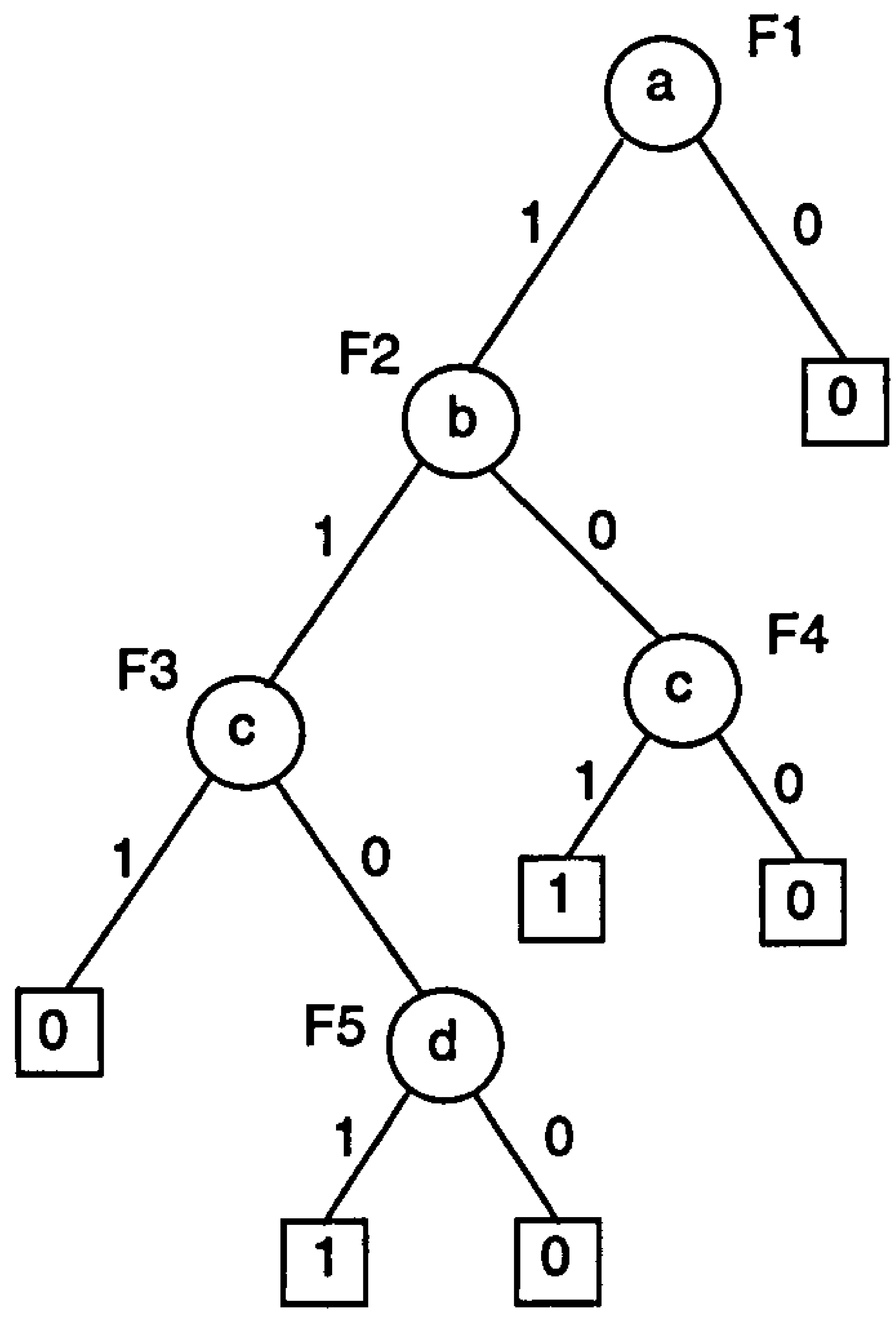


Figure 3.8: The minimised BDD

This produces the following minimal cut sets:

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

Minimising the BDD has therefore removed the redundant cut set {a, b, c}.

It is important to note that as the minimisation procedure changes the structure of the BDD, any quantitative analysis must be performed on the **unminimised** BDD.

3.6 The Influence of Variable Ordering on the BDD

The variables in the fault tree must be ordered before the BDD can be constructed. The chosen ordering not only affects the order in which the variables appear in the BDD, but can also have a crucial effect on the BDD size and the complexity of the calculations required for its construction. For example, consider the fault tree in Figure 3.9.

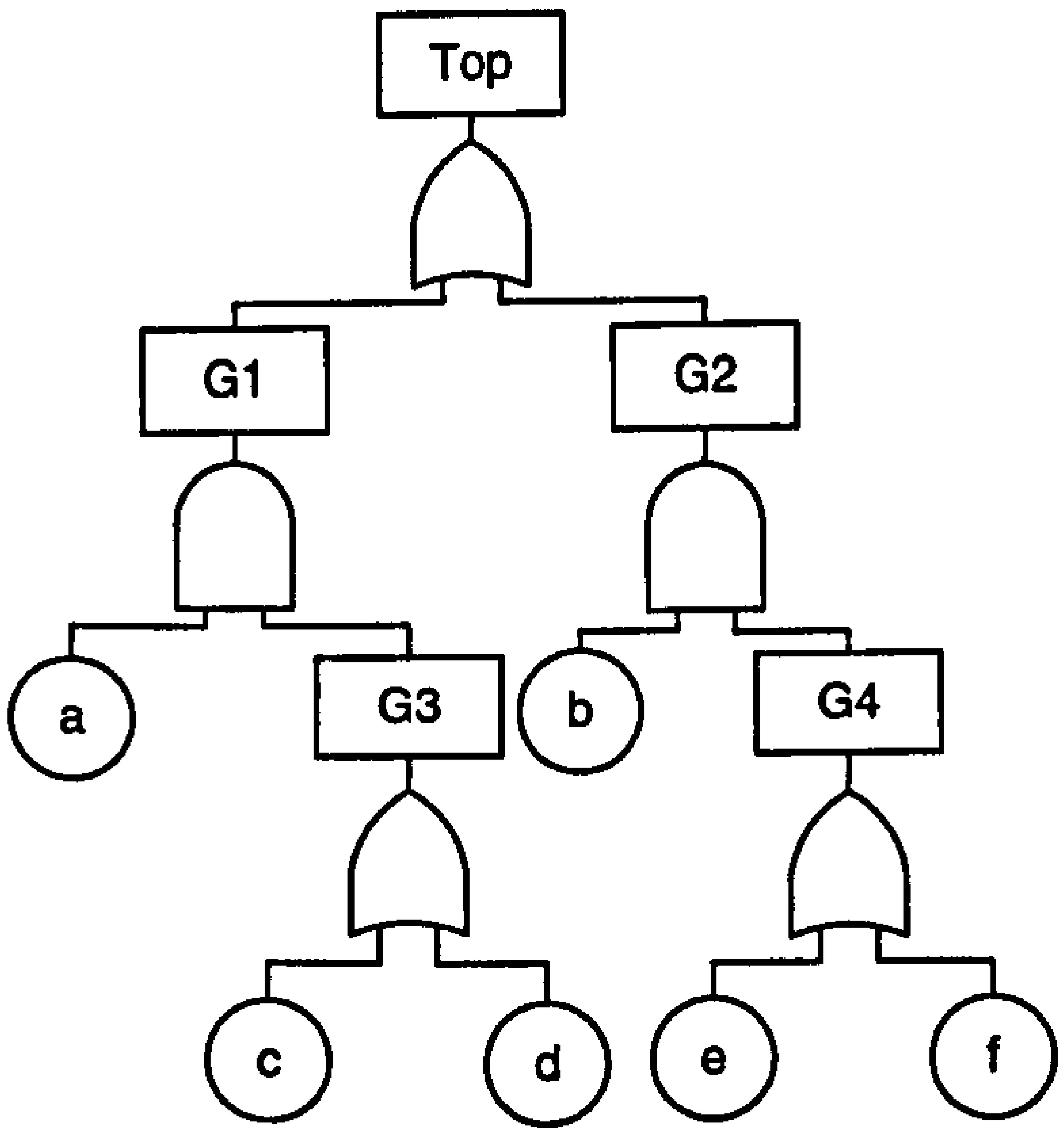


Figure 3.9: Example fault tree for variable ordering

If the 'depth-first' ordering scheme is chosen, which considers the variables in a depth-first, left-right manner, the ordering a<c<d<b<e<f is obtained. The BDD constructed from this ordering is shown in Figure 3.10.

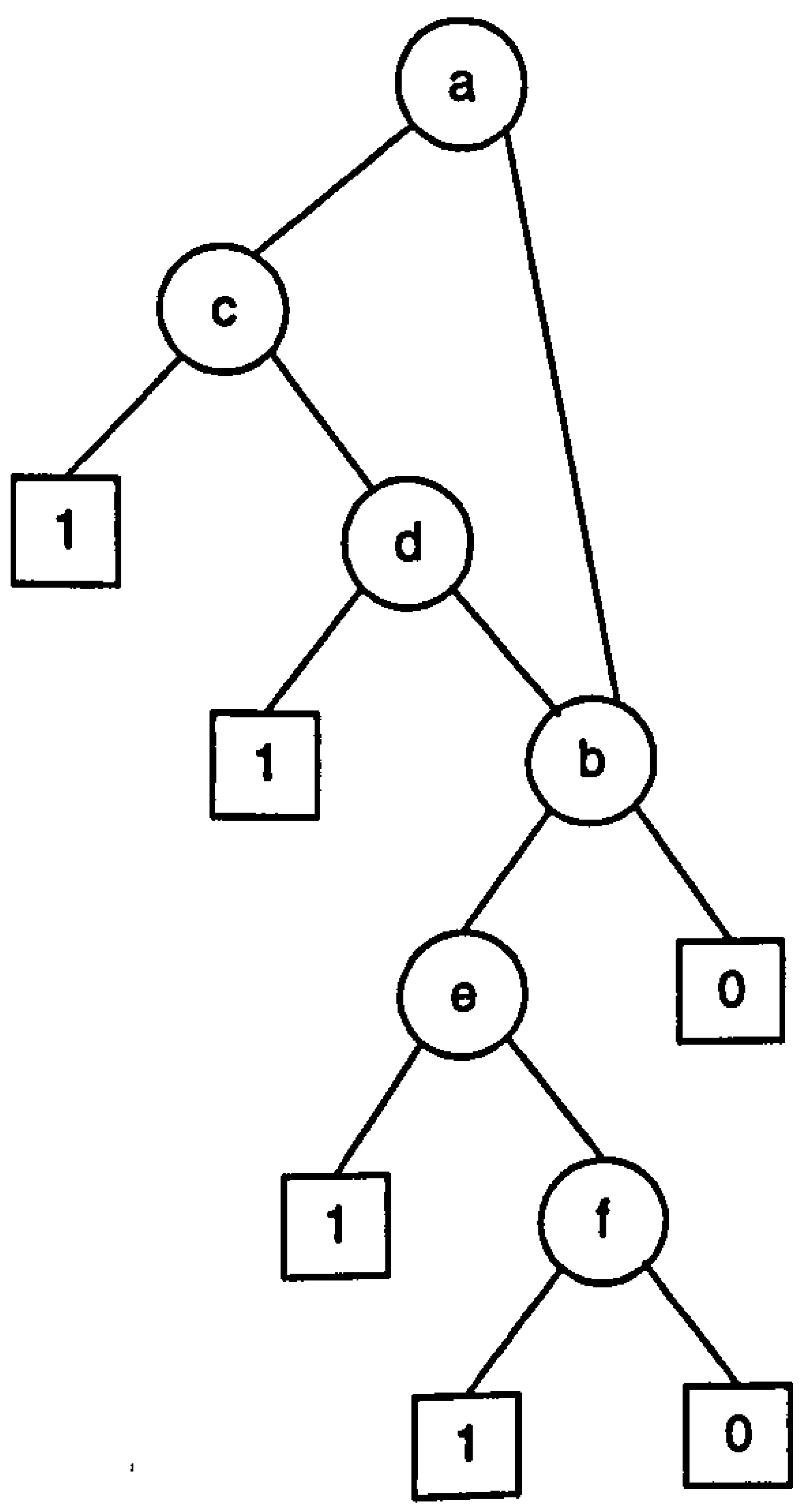


Figure 3.10: BDD obtained from the fault tree in Figure 3.9
using the ordering a<c<d<b<e<f

This ordering gives a simple non-minimal BDD with only six non-terminal nodes. However, if the top-down ordering scheme is used, the following variable ordering is obtained:

a<b<c<d<e<f

which results in the BDD shown in Figure 3.11.

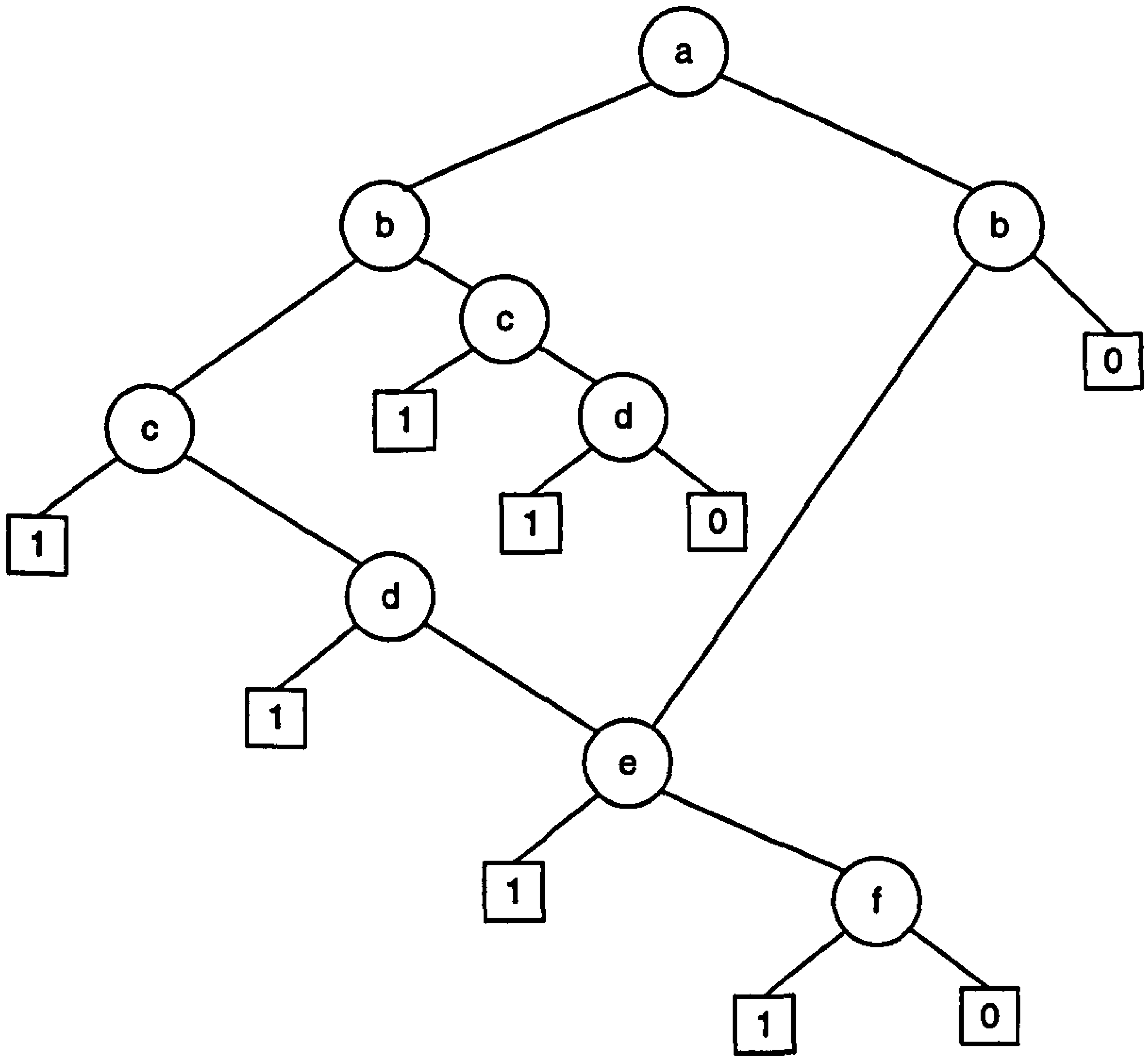


Figure 3.11: BDD for the fault tree in Figure 3.9, using variable ordering a<b<c<d<e<f

The BDD produced from the top-down ordering has nine non-terminal nodes compared with the six non-terminal nodes obtained with the depth-first ordering - an increase of 50%. Although an increase of three nodes is itself not significant, for a fault tree whose BDD contains thousands of nodes, a 50% increase in size could be crucial. For large fault trees, the difference in size produced by a 'good' ordering and a 'bad' ordering can in fact be many orders of magnitude, which can result in the computer storage capabilities being exceeded and the calculations terminated.

Many ordering heuristics have been investigated, but previous research, which is discussed in detail in Chapter 4, has failed to identify any scheme that will always produce a minimal BDD. In fact, no ordering scheme has been found which will produce a BDD, minimal or otherwise, for every fault tree. Although a minimal BDD is advantageous, as the minimal cut sets can be obtained directly so eliminating the need to perform the minimisation procedure, it is not a necessity. The calculations required for the quantitative analysis of BDDs (discussed in Chapter 7) are linear in the size of the BDD, and therefore very efficient. Provided that a BDD can be obtained, the subsequent quantification can be performed. However, it is obviously an advantage to produce as small a BDD as possible to reduce the analysis time, and further research is necessary to ensure that a BDD can be constructed for any given fault tree. The problem of variable ordering is the subject of the literature survey in Chapter 4.

3.7 Modularisation

The BDD construction process can be made more efficient by modularising the fault tree before the conversion procedure takes place. Modularisation identifies independent subtrees (modules) within the fault tree that can be analysed separately from the rest of the tree. A detailed discussion of the modularisation technique is given in Chapter 2.

Modularisation can significantly reduce the complexity of a fault tree, by breaking it down into smaller, more manageable pieces that can be dealt with separately. In terms of the BDD process, the tree can then be analysed in several stages by obtaining smaller BDDs for each subtree. These can then be combined to form a BDD that represents the original fault tree structure. It is possible therefore, that a BDD could be constructed for a tree that could not previously be analysed.

The process can be demonstrated using the fault tree shown in Figure 3.12.

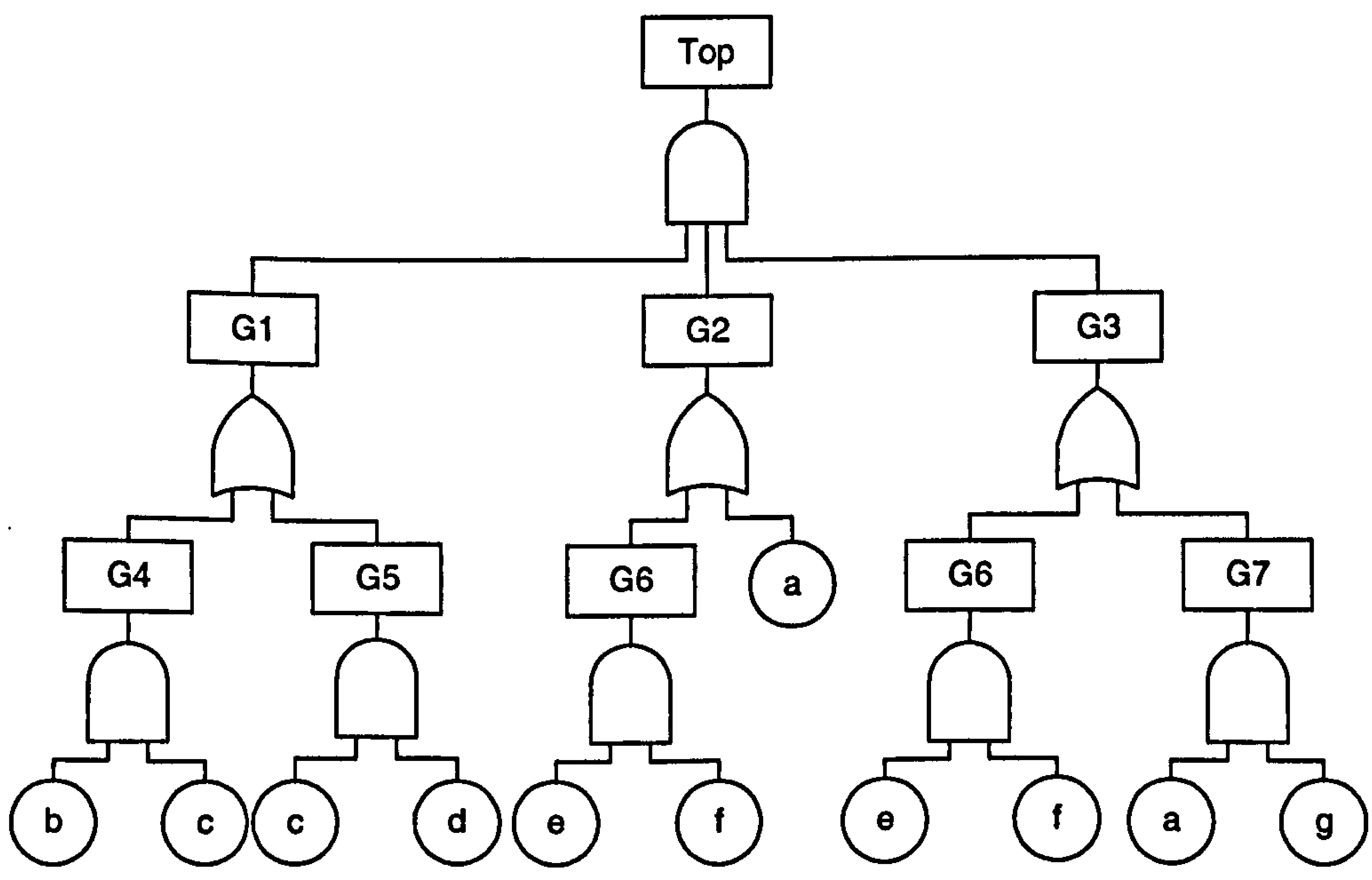
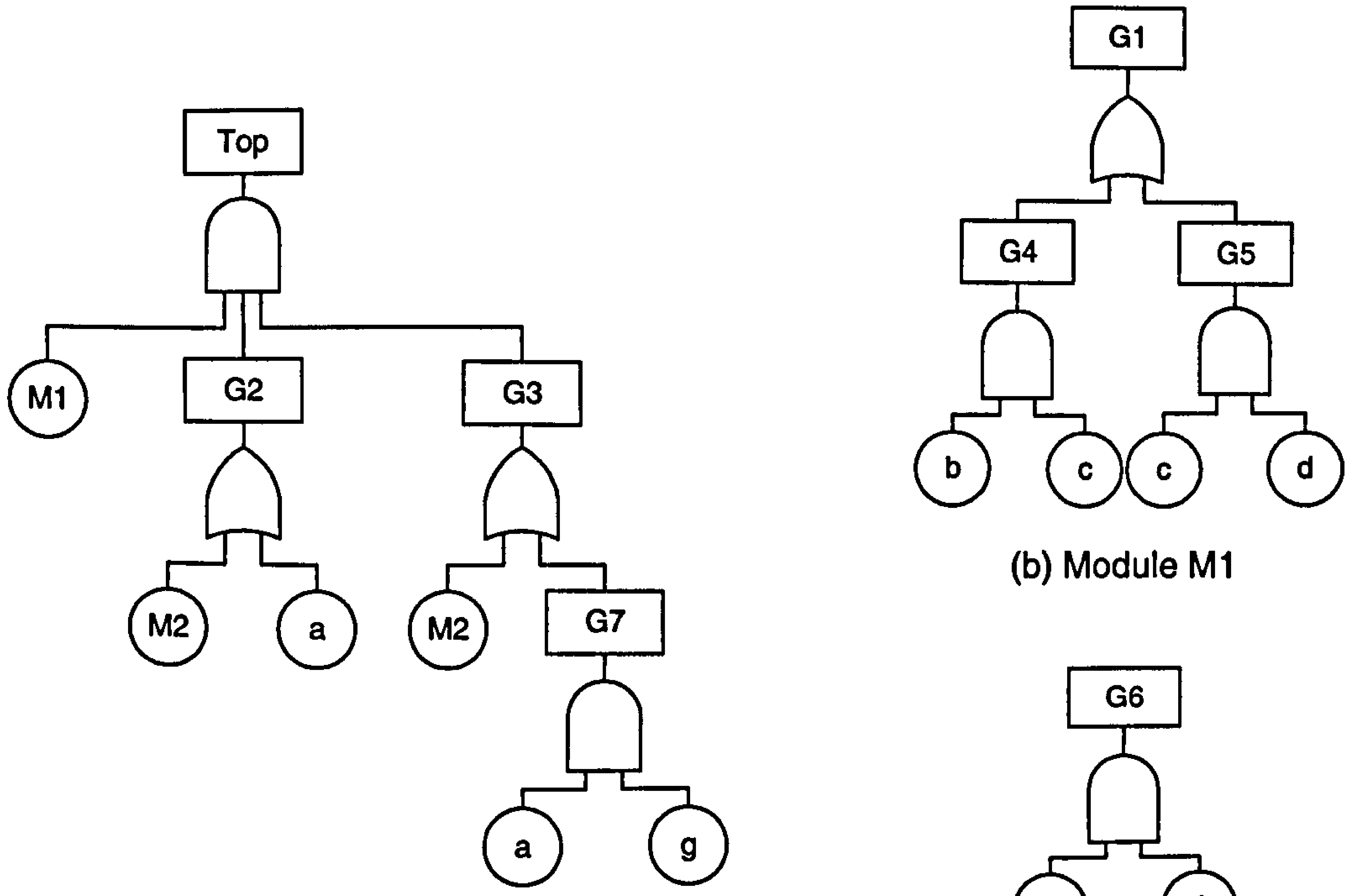


Figure 3.12: A fault tree that can be modularised

The following modules can be identified:

- Gate G1 heads the module M1, as none of its inputs appears as an input elsewhere in the tree.
- Gate G6 heads the module M2. M2 appears twice in the modularised tree, as an input to both G2 and G3.
- Top itself is also a module.

The modularised tree and modules M1 and M2 are shown in Figure 3.13.



(a) The modularised tree

(b) Module M1

(c) Module M2

Figure 3.13: The modularised fault tree and two modules

To form the BDD from the modularised tree, the modules are treated as events and so need to be ordered together with the basic events. Taking the top-down order M1<M2<a<g, the **ite** structure for the top event can be formed:

$$
\begin{aligned}
Top &= M1.G2.G3 \\
&= \mathbf{ite}(M1, 1, 0).\,\mathbf{ite}(M2, 1, \mathbf{ite}(a, 1, 0)).\,\mathbf{ite}(M2, 1, \mathbf{ite}(a, \mathbf{ite}(g, 1, 0), 0)) \\
&= \mathbf{ite}(M1, \mathbf{ite}(M2, 1, \mathbf{ite}(a, \mathbf{ite}(g, 1, 0), 0)), 0)
\end{aligned}
$$

Each module is then analysed independently to form its own BDD. The top-down orderings for the modules are as follows:

M1: b<c<d

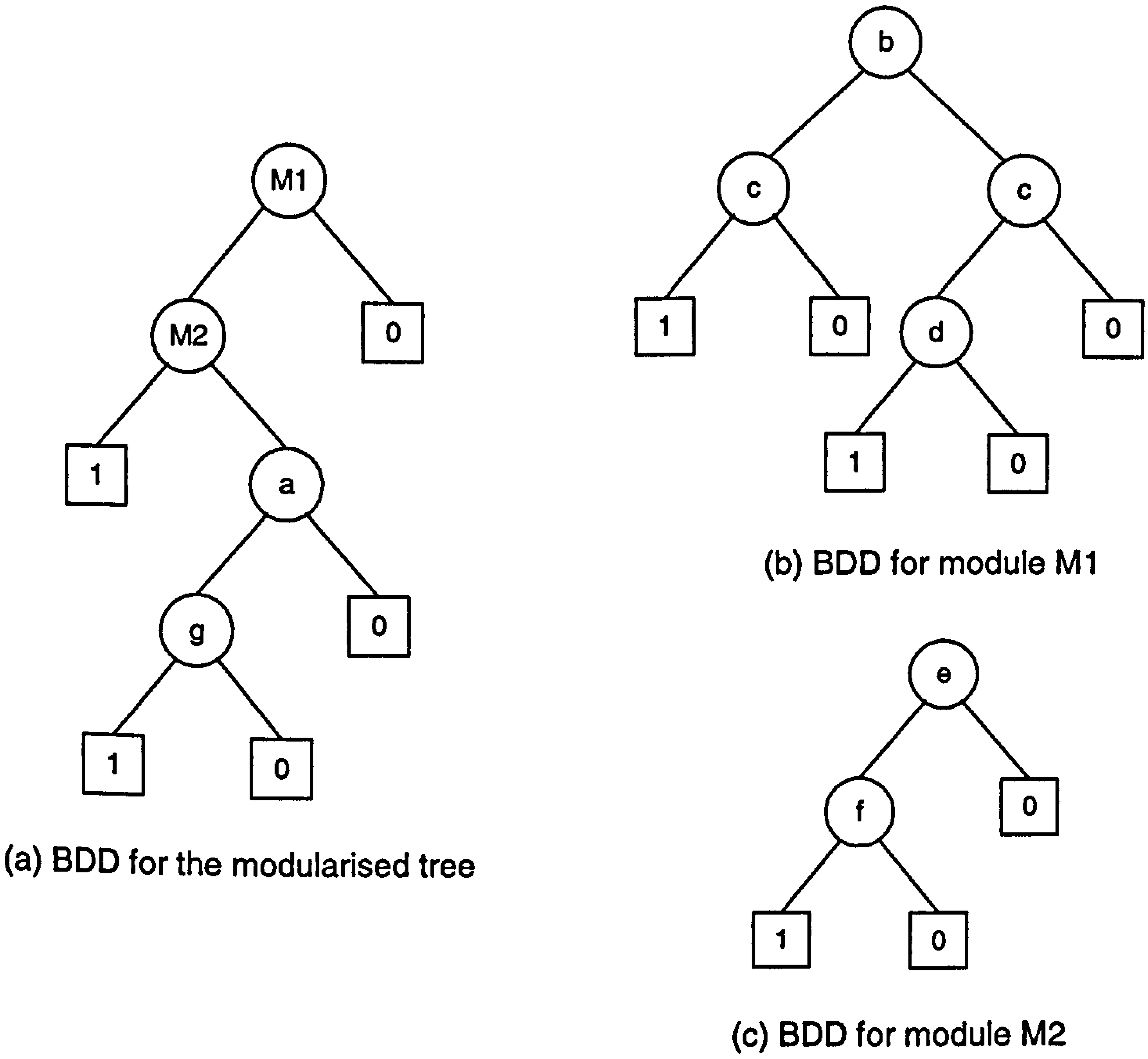
M2: e<f

which result in the **ite** structures given by:

$$
\begin{aligned}
M1 &= G4 + G5 \\
&= \mathbf{ite}(b, \mathbf{ite}(c, 1, 0), 0) + \mathbf{ite}(c, \mathbf{ite}(d, 1, 0), 0) \\
&= \mathbf{ite}(b, \mathbf{ite}(c, 1, 0), \mathbf{ite}(c, \mathbf{ite}(d, 1, 0), 0)
\end{aligned}
$$

$$
\begin{aligned}
M2 &= \mathbf{ite}(e, 1, 0).\mathbf{ite}(f, 1, 0) \\
&= \mathbf{ite}(e, \mathbf{ite}(f, 1, 0), 0)
\end{aligned}
$$

This corresponding set of BDDs is shown in Figure 3.14.



(a) BDD for the modularised tree

(b) BDD for module M1

(c) BDD for module M2

Figure 3.14: The BDDs obtained from the modularised fault tree and two modules

The BDDs for each module are then substituted into Figure 3.14(a) to give one BDD encoding only basic events. This is shown in Figure 3.15.

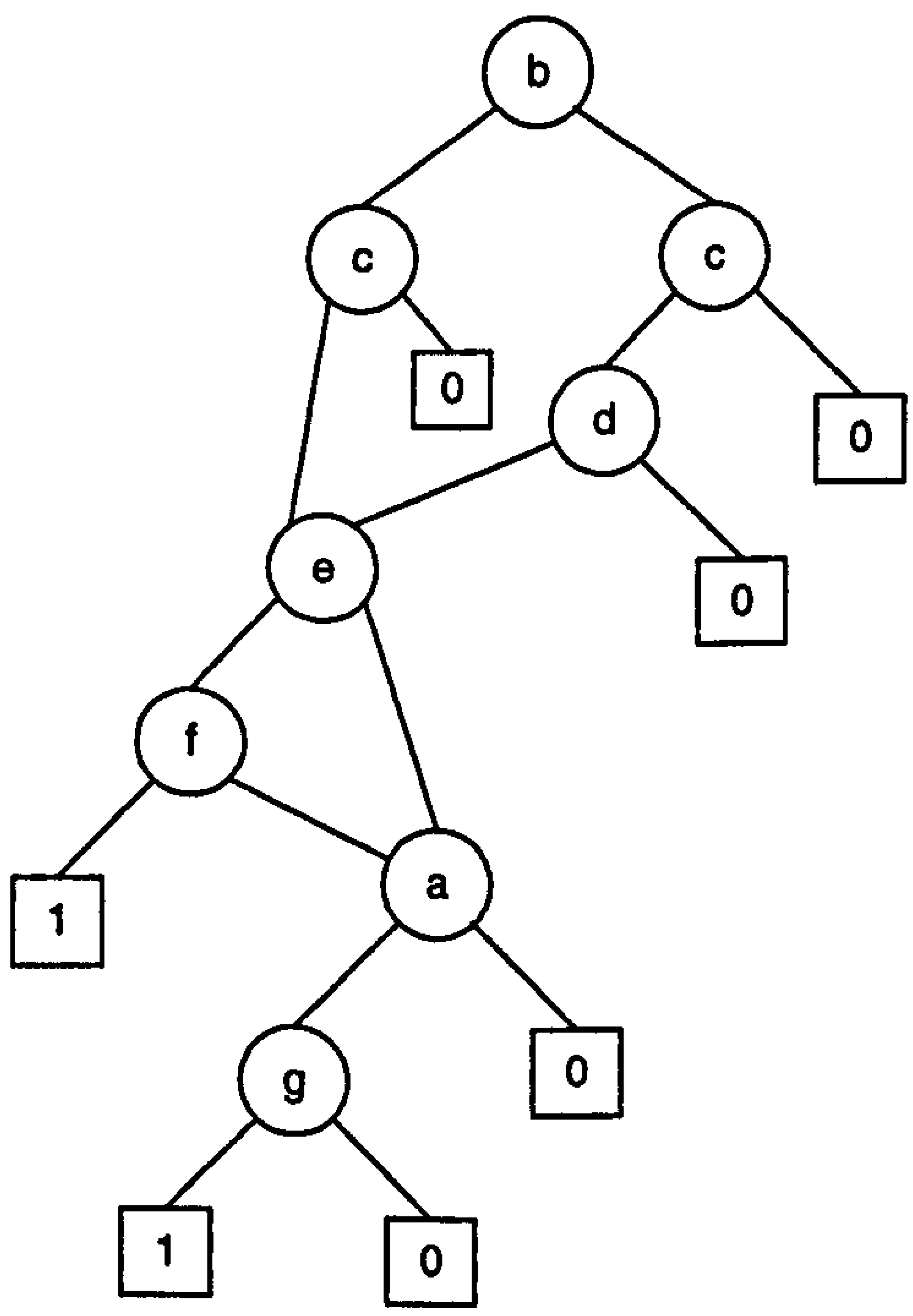


Figure 3.15: The BDD encoding only basic events

Constructing the BDD from the modularised fault tree involves fewer calculations than if the unmodularised fault tree were used. Ordering the variables of the unmodularised tree in a top-down manner (a<b<c<d<e<f<g) results in a BDD with fifteen non-terminal nodes. This is significantly more than for the BDD in Figure 3.15, which has only eight non-terminal nodes.

In the above example, the top-down ordering scheme was used throughout to order the variables of the modularised tree and the two modules. However, the same scheme does not necessarily have to be used for each. The modules can be ordered according to their individual structures, using the scheme that results in the smallest BDD in each case. This could be particularly beneficial in large fault trees, when vast savings could be made in terms of the number of calculations performed and could result in a substantially smaller BDD.

3.8 Summary

The BDD technique is already proving to be of considerable use in reliability analysis. It provides an efficient means of analysing a system, without the need for the approximations previously used in the conventional methods of Kinetic Tree Theory.

The difficulty with this technique is in the conversion of the fault tree to the BDD. The variable ordering can have a crucial effect on the BDD; it can mean the difference between a minimal BDD with few nodes, so providing an efficient analysis, and no BDD at all. There is no ordering scheme capable of generating an efficient BDD structure for all fault trees. Considerable research has been conducted into this area and also into methods of selecting an appropriate scheme from a group of alternatives. A detailed survey of current ordering heuristics and scheme selection techniques is conducted in the following chapter.

Chapter 4: A Survey of Variable Ordering Heuristics

4.1 Introduction to Variable Ordering

The BDD technique introduced in the previous chapter provides an exact and efficient means of analysing fault trees. The difficulty however, lies in the construction of the BDD from the fault tree. An ordering of the fault tree variables must be chosen, which determines the sequence in which the events are considered in the **ite** procedure. The variables are ordered in a systematic manner, according to a particular variable ordering scheme. The choice of ordering scheme can have a crucial effect on the size of the resulting BDD - a 'bad' ordering can result in a BDD many orders of magnitude larger than one obtained from a 'good' ordering.

One of the reasons for the significant variation in BDD size is the rate at which the maximum number of nodes grows as the number of fault tree variables increases. The number of nodes in the BDD cannot be less than the number of variables, n, on which it depends, though this can be less than the number of basic events appearing in the tree if redundancies exist. The maximum number of nodes, however, increases exponentially as 2^n-1, as shown Table 4.1.

Number of variables / minimum number of nodes in BDD, n	5	10	20	50	100
Maximum number of nodes in BDD, 2^n-1	31	1023	~10^6	~10^{15}	~10^{30}

Table 4.1: The rapidly increasing function governing the maximum number of nodes

Another factor that can greatly affect the BDD size is the symmetry of the function. For a symmetrical function, the BDD size does not depend on the variable ordering. However, as the asymmetry increases, so does the variability in the BDD size.

Circuit Analysis is another field in which the BDD technique can be implemented. As with Fault Tree Analysis, an ordering of the system variables must be chosen in order to construct the BDD. The need for a good ordering of the circuit variables was addressed at an early stage by Bryant[14], who noted that the size of the resulting BDD was highly sensitive to the chosen ordering scheme. Consequently there has been much research into finding the optimal variable ordering for circuits and many of the heuristics identified have been considered for fault trees. However, circuits have a different type of logic structure to fault trees and it has been shown by Nikolskaia[15] that traditional variable ordering heuristics for circuits make poor choices for fault trees. Therefore, the survey of heuristics presented in this chapter will not include many of the heuristics and algorithms developed for use in the field of circuits.

Ordering heuristics are categorised as either dynamic or static. Both techniques are discussed below.

- Dynamic ordering: These methods focus mainly on circuits, but can also be applied to fault trees and involve swapping or exchanging variables to produce a smaller BDD[16, 17]. This is achieved either by obtaining the BDD (using any heuristic), then re-ordering the variables to produce a reduced BDD, or by swapping variables during the construction process when the original ordering is not adequate to finish the computation. Although this procedure can significantly reduce the BDD size, it is of limited use in reliability analysis due to the time taken for its implementation. Once a BDD has been constructed, the analysis is a linear function of the number of nodes within the BDD. Therefore, provided a BDD (of any size) can be obtained, it is more efficient simply to perform the required calculations, rather than applying dynamic ordering techniques prior to the analysis. For this reason, dynamic techniques are not considered to be of great use and will not be discussed in this survey.

- Static ordering, whereby a variable order is established prior to the construction of the BDD is the focus of this chapter. Figure 4.1 highlights the techniques that will be covered.

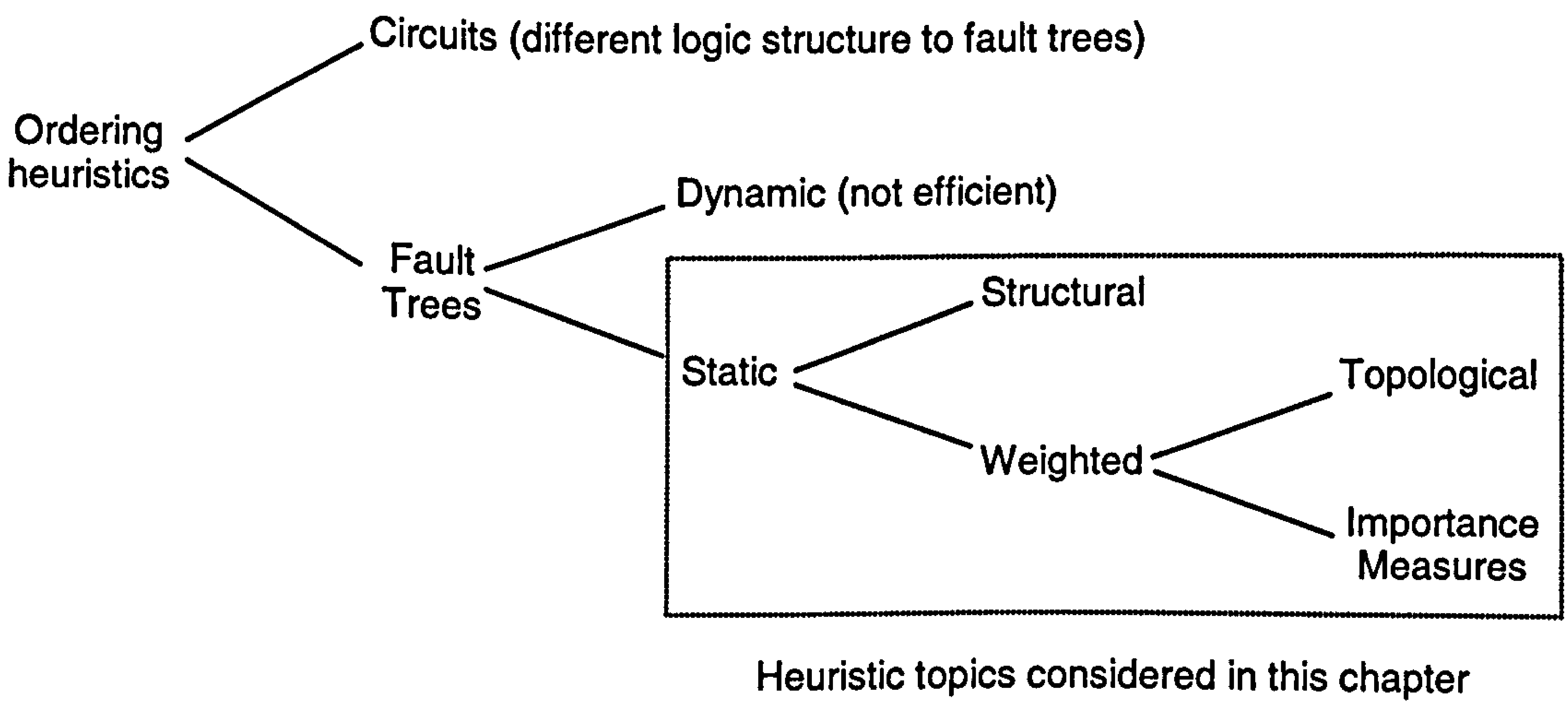


Figure 4.1: Relation of variable ordering heuristics

Many different heuristics have been proposed for selecting a variable ordering and this chapter aims to review and, where possible, compare these schemes. The ordering schemes fall into two categories: structural and weighted. Structural schemes perform a structured traversal of the tree, ordering variables as they are encountered and preserving neighbourhoods. Weighted methods allocate weights to the variables in order to determine the ordering and do not necessarily preserve neighbourhoods. Weighted schemes can be further categorised as being either topological or as based upon importance measures. Structural schemes are the most commonly used ordering techniques and will be discussed first.

4.2 Structural Ordering Schemes

Structural ordering techniques are very widely used and involve ordering the variables via a structured traversal of the fault tree. These schemes tend to preserve the neighbourhoods of the variables, such that those appearing close together in the tree structure are also close in the ordering. The first structural heuristic to be suggested was the depth-first ordering scheme, which Rauzy applied in his initial paper on using the BDD technique for Fault Tree Analysis[4]. However, by far the most common ordering heuristic is the top-down scheme and this is introduced first, as it is referred to in many other techniques.

4.2.1 Top-Down Ordering Scheme

The top-down ordering scheme is the most basic scheme, simply ordering the variables as they are encountered on a top-down, left-right traversal of the fault tree structure. Basic events appearing high in the fault tree will therefore be placed earlier in the ordering than those appearing further down the tree.

For example, the scheme can be applied to the fault tree shown in Figure 4.2. Each level is considered in turn, from the top to the bottom, with the basic events in that level ordered from left to right. Each event is ordered the first time it is encountered; subsequent occurrences are ignored.

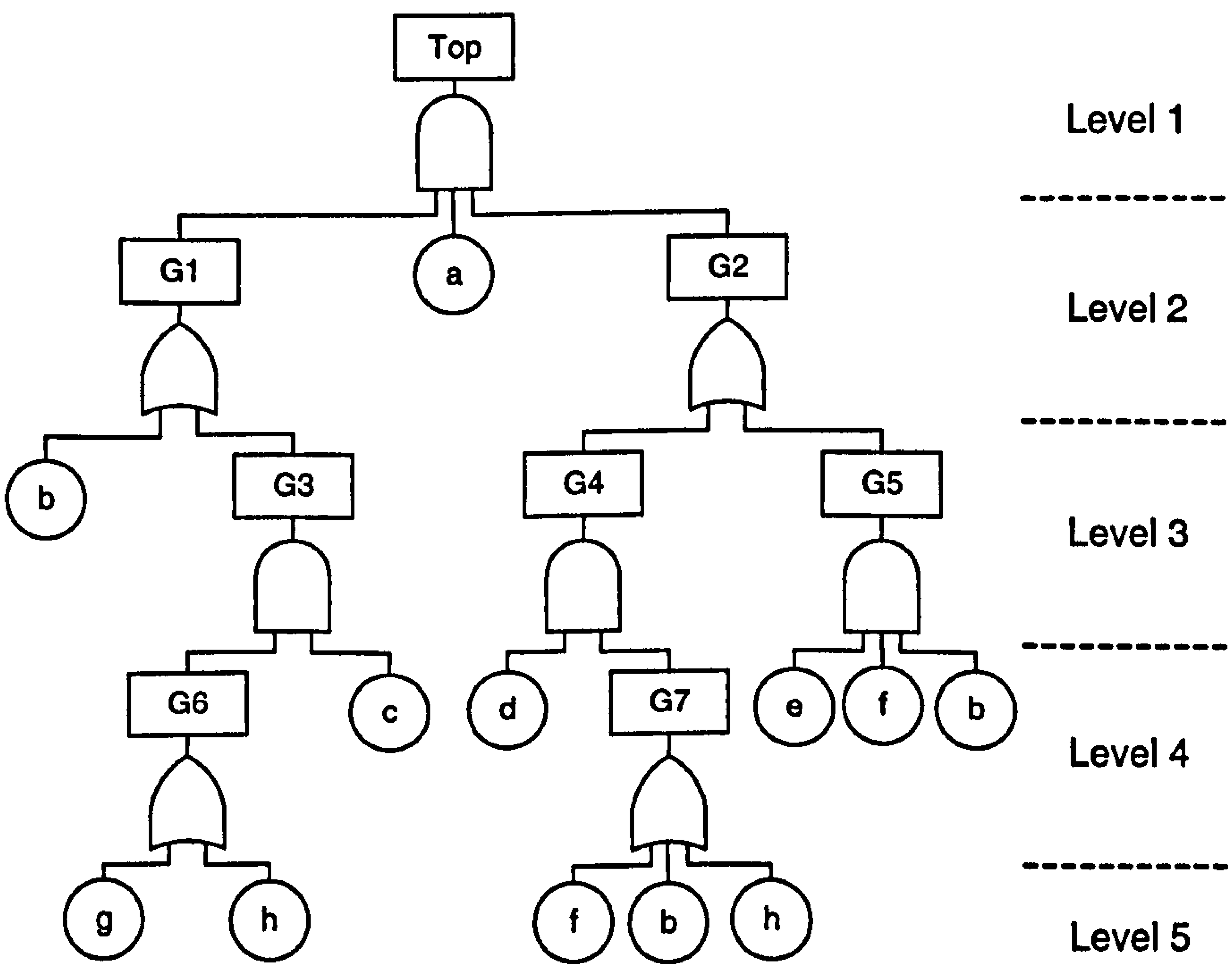


Figure 4.2: Example fault tree used for ordering heuristics

The top-down ordering of basic events is therefore:

$$a<b<c<d<e<f<g<h$$

An obvious feature of this scheme is that it is highly dependent on the way in which the fault tree is written. This is true to some extent in all structural ordering schemes. For example, gates G1 and G2 could be swapped around, as could G4 and G5, or the order in which events are placed as inputs to the gates could be altered, without changing the logic function of the tree. This would affect any structural ordering imposed upon the tree, and the size of the resulting BDD. This is the greatest disadvantage of the structural ordering schemes.

4.2.2 Depth-First Ordering Schemes

In the first paper on the application of BDDs to Fault Tree Analysis, Rauzy suggested a depth-first ordering heuristic[4], which is implemented by 'carrying out a depth-first exploration of the tree and numbering the variables as soon as they appear'. However, an example was not given, so it is unclear exactly how this was implemented.

The following definition of depth-first ordering will be the one used throughout this thesis:

> Depth-first ordering considers the fault tree as being made up of many smaller subtrees, with *each* subtree ordered in a top-down, left-right manner. Starting with the top event, any basic event inputs would be ordered from left to right, and the gate inputs are then considered from left to right. Each of those gates is then considered as the top event and ordered in the same manner, such that lower levels of leftmost subtrees are considered before higher levels of other subtrees.

This ordering scheme can be applied to the fault tree shown in Figure 4.2. The gates are considered in the following order: The traversal starts at the top event, Top, which has two gate inputs. The leftmost gate G1 is considered first. Moving down through the tree, gate G3 is ordered, followed by its input G6. Having completely ordered this subtree, the traversal returns to Top to consider G2. G2 also has two subtrees, headed by G4 and G5. G4 is ordered first, and then its input, G7. Once G4 has been completely ordered, G5 is considered.

At each stage, the basic event inputs are ordered before any gate inputs are considered. This gives the following ordering:

$$a<b<c<g<h<d<f<e$$

Two variations on this depth-first method are:

Alternative 1: The first alternative method[18] proceeds as the technique described above, but does not order the events of a gate before considering any gate inputs.

The events and gates are considered in the order in which they appear in the input list, so any gates that are listed before events will be ordered first.

For example, in Figure 4.2, G1 appears before 'a' as an input to Top. Therefore G1 will be considered, and so its inputs ordered, before 'a' is placed in the ordering. This is also the case for the inputs to G3; G6 appears before 'c' and so is considered first. Thus the ordering would be:

b<g<h<c<a<d<f<e

This method of depth-first ordering is therefore much more dependent on how the fault tree is written than the previous technique, as the way in which the events and gates are placed in relation to one another is now relevant.

Alternative 2: This second alternative method[19] simply considers the subtrees of the top event in turn, ordering each subtree in a top-down, left right manner.

For example consider again the fault tree in Figure 4.2. There are three subtrees of the top event, headed by G1, 'a' and G2, which are considered in this order. Ordering the first subtree, G1 using the top-down method gives the partial ordering b<c<g<h. The second subtree to be considered is simply 'a' itself, so the ordering becomes b<c<g<h<a. The remaining variables are ordered from the final subtree as d<e<f, to give the complete ordering:

b<c<g<h<a<d<e<f

4.2.2.1 Priority Depth-First Ordering

This ordering is an extension to the simple depth-first schemes, incorporating an extra factor thought to have a significant influence on the size of the BDD. Sinnamon and Andrews[20] proposed that the basic events that have more influence on the structure function should be ordered first and that these events frequently lie higher up the fault tree. For this reason, priority should be given to subtrees that only have basic event inputs.

For example, in Figure 4.2, where previously G4 had been ordered before G5, G5 would now be ordered first as it has only basic events as its inputs. This gives the ordering:

a<b<c<g<h<e<f<d

A comparison of the BDD sizes for 51 fault trees using the top-down, depth-first and priority depth-first schemes was conducted by Sinnamon[21]. The results showed that for 21 out of the 51 fault trees, the top-down scheme produced the smallest BDDs. The depth-first method produced the smallest BDDs for 36 of the fault trees. However, the priority depth-first method performed marginally better than this, producing the smallest BDDs for 37 of the 51 fault trees. It is noted that for some fault trees the orderings produced equivalent sized BDDs, so were each considered to have produced the smallest BDD.

Therefore, in this relatively small study of 51 fault trees, the priority depth-first method of ordering was shown to perform better than both the top-down and depth-first schemes.

4.2.2.2 Depth-First, with Number of Leaves

This heuristic, proposed by Rauzy (unpublished) performs a depth-first (first alternative) traversal of the tree, but rather than considering the inputs of a gate from left to right, it chooses the order of the inputs according to their number of leaves. The number of leaves of a gate is the total number of basic events occurring at any level beneath that gate.

The inputs with the smallest number of leaves that have not yet been ordered are considered first. In the case of a tie, the input with the fewest ordered leaves is chosen.

This process can be applied to the fault tree shown in Figure 4.2. The number of leaves beneath each gate is given in Table 4.2.

Gate	G1	G2	G3	G4	G5	G6	G7
Number of leaves	4	7	3	4	3	2	3

Table 4.2: Number of leaves of each gate in Figure 4.2

Starting with the inputs to the top event, 'a' has fewer leaves than G1 or G2 (as it is itself a basic event), so is ordered first. G1 has fewer unordered leaves than G2 (four vs. seven) so is processed first, to give the partial ordering a<b<c<g<h. Events are simply ordered from left to right as they appear in the input list. G2 is considered next and has two inputs, G4 and G5. They have an equal number of unordered leaves (two each), but G5 is processed first, as it has fewer ordered leaves (one vs. two). The partial ordering then becomes a<b<c<g<h<e<f. G4 has the input 'd' which is ordered next, as it has fewest leaves and finally G7 is processed, but contributes nothing further to the ordering as all the basic events have been ordered. The final ordering is:

a<b<c<g<h<e<f<d

This ordering scheme has the advantage that it is less dependent on how the fault tree is written, especially when compared to the first alternative depth-first method, upon which it is based. This is particularly true for gates that have other gates as inputs (either all gate inputs or a mixture of gates and events), as their order is decided by the number of leaves and not by the order in which they appear in the list of inputs. It makes no difference to gates that have only basic events as inputs, as they continue to be ordered from left to right. This heuristic was included in a comparative study[18] with several other ordering heuristics, which is discussed in more detail in section 4.5.

4.2.3 Repeated Events

The top-down ordering scheme was the first to be extended by Sinnamon and Andrews[22] to prioritise repeated basic events (i.e. events that appear more than once in the fault tree) and is called *modified* top-down ordering. It was noted that repeated variables cause the problem of non-minimal cut sets, and so by considering these events first, the size of the resulting BDD structure would be reduced. In this initial study, it was found that by considering repeated events first, 13 out of 15 fault trees resulted in a minimal BDD, whereas using a top-down ordering had previously resulted in redundant BDDs.

The tree is still scanned in a top-down manner. However, variables on the same level that were initially ordered according to their position from left to right, are now ordered according to their number of occurrences within the tree. Those with the greatest number of occurrences are ordered first. If events have an equal number of occurrences, then they are ordered as before.

The variable ordering for the fault tree in Figure 4.2 would therefore be changed slightly due to this new condition. On level three, event 'f' would be ordered before the other events as it has two occurrences and similarly on level four, 'h' is ordered before 'g' as it appears twice in the fault tree. This gives the new ordering:

a<b<f<c<d<e<h<g

Prioritising repeated events was extended to the depth-first and priority depth-first ordering schemes by the same authors[20]. Within these schemes, basic event inputs to each gate were simply ordered in a left-right manner; they are now ordered giving priority to repeated events. Where there is a tie, variables are ordered as before. Ordering the fault tree in Figure 4.2 using these schemes gives the following orderings:

- Modified depth first method: a<b<c<h<g<d<f<e.
- Modified priority depth-first method: a<b<c<h<g<f<e<d.

Sinnamon and Andrews then compared these six ordering heuristics:

- Top-down and modified top-down
- Depth-first and modified depth-first
- Priority depth-first and modified priority depth-first

For six example fault trees, the number of **ite** calculations required to produce the BDD using each different type of ordering scheme (top-down, depth-first, priority depth-first) was found. For the trees with repeated events, the ordering scheme that had been most successful was used to find the number of **ite** calculations using the repeated events option. It was found that there were large differences in the number of computations between the different orderings. However, there was no one scheme that worked best for all the trees. They concluded that it seems unlikely that a general rule-based ordering scheme could be found which would be optimal for all fault trees.

Sinnamon[21] later extended this comparison to consider 51 fault trees. A study using these fault trees to compare the top-down, depth-first and priority depth-first methods had shown that the priority depth-first method had performed the best, producing the smallest fault trees in 37 out of the 51 cases. These results are discussed in section 4.2.2.1. The modified ordering schemes, prioritising repeated events were now included in the comparison. Table 4.3 shows the number of fault trees for which each scheme produced the smallest BDD.

Ordering heuristic	Top-down	Modified top-down	Depth-first	Modified depth-first	Priority depth-first	Modified priority depth-first
Number of trees for which the smallest BDD was produced	17	19	27	35	30	34

Table 4.3: Results for Sinnamon's comparative study of six ordering heuristics

These results show that the modified depth-first method produced the smallest BDD in the most cases (35 out of 51). For nineteen of these trees, the resulting BDDs were minimal. By considering repeated events, the depth-first method has performed better than the previous best method, priority depth first. However, the modified priority depth first method still performs well, producing the smallest BDDs for 34 fault trees.

This investigation was extended in a larger study of 225 fault trees by Bartlett[19], which produced the results shown in Table 4.4.

Ordering heuristic	Top-down	Modified top-down	Depth-first	Modified depth-first	Priority depth-first	Modified priority depth-first
Number of trees for which the smallest BDD was produced	87	169	120	117	36	38

Table 4.4: Results for Bartlett's comparative study of six ordering heuristics

These results show that the modified top-down heuristic significantly outperforms the other schemes for these fault trees. In fact, the priority depth-first and modified priority depth-first methods perform poorly. A reason for the contradicting results could simply be the difference in fault trees used in the studies. It is well documented that heuristics can perform well on some fault trees, but poorly on others, so the fault trees chosen could simply suit one type of heuristic particularly well. However due to the size of Bartlett's study, it is concluded that the modified top-down ordering scheme does seem to produce a better overall performance.

4.2.4 Repeated Gates and Events

A scheme reported by Bouissou et al[18] prioritises both repeated gates and events within the depth-first (first alternative method) scheme. Rather than simply considering the inputs (both gates and events) to a gate from left to right, they are considered according to their number of occurrences in the tree.

This ordering scheme can be applied to the fault tree shown in Figure 4.2. There are no repeated gates in this tree to be considered, but there are repeated events. In G6, 'h' is chosen before 'g', as 'h' appears twice in the tree and 'g' only appears once. There is therefore, a slight change to the ordering obtained with the depth-first (first alternative) method:

$$b<h<g<c<a<d<f<e$$

By prioritising repeated gates and events, the ordering is less sensitive to the way the fault tree is written. Results obtained from a comparative study of heuristics[18] show that this method performs better than the depth-first (first alternative method) heuristic. These results are discussed in more detail in section 4.5.

4.2.5 REBESUL Ordering Scheme

The REBESUL ordering scheme, suggested by Sinnamon[21], incorporates the factors deemed to have the greatest effect in reducing BDD size. Previous results obtained by Sinnamon had shown that a depth-first approach was a good one, and that by employing the priority depth-first ordering scheme, which gives priority to those subtrees that have basic event inputs only, the size of the resulting BDD could be further reduced. Also, the position of repeated events in the ordering has a significant effect on the size of the BDD, so by considering these first, a smaller BDD was likely to be produced. These factors were incorporated into a depth-first approach to give a variable ordering scheme that considers repeated basic events and subtree levels, called REBESUL. The algorithm is based on six steps, which are described overleaf.

1. Create a list of the repeated events in the fault tree; those with the highest number of occurrences are listed first. Repeated events that have an equal number of occurrences are placed in the rows between the next highest and the next lowest.
2. For each repeated event in step 1, create a list of the subtrees (first sons of the top gate) that contain this repeated event in the order of the highest number of different repeated event occurrences within each subtree to the lowest.
   - If two or more subtrees share the same number of repetitions for an event, the subtree with the greatest number of levels takes precedence over how many repetitions there are in a subtree.
3. Create a list of the levels in the subtree at which the repeated event in step 2 occurs.
4. Order the gates (depth-first) starting with the gate that 'contains' the lowest level occurrence (obtained in step 3) of the repeated event, followed by the other gates that 'contain' the next level of occurrence of the repeated event. Note that the term 'contains' does not necessarily mean that the repeated event is a direct input to the gate, it may be an input several levels down. List the repeated events first when ordering the inputs of each gate.
5. If all repeated events have been dealt with in this subtree, order any remaining events to gates in the subtree in a depth-first manner and go to step 6. Otherwise go to step 3 for the next repeated event obtained in step 1.
6. If all subtrees containing repeated events have been dealt with, order any remaining subtrees in a depth-first manner. Otherwise order the next subtree containing repeated events, i.e. go to step 2.

This algorithm can be applied to the fault tree in Figure 4.2 in the following way:

1. 'b' – occurs three times
   'f' – occurs two times
   'h' – occurs two times
2. Subtree three (G2) has the highest number of different repeated events (three), so is ordered first. Subtree one (G1) has two different repeated events.
3. Event 'b' occurs at level two and level three of G2.
4. G5 contains the lowest level of occurrence of 'b' and G4 contains the next level of 'b' ('b' is an input to G7which is an input to G4), therefore take the order of the gates G5, G4, G7, to give the partial basic event ordering:

$$b<f<e<d<h$$

5. All repeated events dealt with, go to step 6.

6. The ordering for subtree one is:

c<g

The ordering for subtree two gives the last basic event 'a'.

All basic events have been dealt with giving the ordering:

b<f<e<d<h<c<g<a

Sinnamon used the REBESUL ordering scheme to calculate the number of BDD nodes for the 51 fault trees that were used to compare the top-down, depth-first and priority depth-first schemes (discussed in section 4.2.2.1). It was found that the REBESUL ordering produced BDDs with the fewest nodes for 41 of the 51 fault trees, compared with the previous best of 37 for the priority depth-first scheme. 19 of these BDDs were minimal. Therefore, the REBESUL ordering scheme proved to be more efficient than the priority depth-first ordering scheme in this case.

4.3 Weighted Ordering Schemes

Weighted ordering techniques allocate weights to the variables, which then determine their position in the ordering. These methods do not necessarily preserve neighbourhoods in the same way as structural ordering schemes, so variables that appear together in the tree structure may not be close in the ordering. Weighted ordering schemes can be divided into two categories: topological schemes, which assign weights according to the positions of the variables in the tree and schemes based on importance measures, which assign weights in a manner that is not dependent on how the tree is written.

4.3.1 Topological Schemes

Two ordering schemes are discussed in this section, which differ by using opposite ends of the fault tree to initiate the weighting process. The ordering produced by each of these schemes is dependent on the way in which the fault tree is written.

Although the use of these schemes for fault trees has been reported[18, 23], few results have been published. The comparative study in section 4.5 however, does include the second of the following schemes.

4.3.1.1 Applying Weights in a Top-Down Manner

Minato et al[24] have applied a weights method to circuits, which can be applied to fault trees in a similar manner. The bases for their method are the following two properties that have been observed in circuits:

1. The inputs that greatly affect the output function should be high in the ordering.
2. The inputs whose connections are topologically close to one another should be close in the ordering.

The corresponding properties applied to fault trees could be:

1. The basic events having the greatest effect on the structure function should be high in the ordering.
2. Basic events topologically close to one another (i.e. events which appear together as inputs to a particular gate) should be close in the ordering.

These two properties form the basis of their approach, termed the 'dynamic weight assignment method'. In terms of fault trees, the method progresses as follows:

- A weight of 1.0 is assigned to the top event and is propagated towards the basic events.
- At each gate, the weight is equally distributed between its inputs.
- Each basic event will then have been assigned a weight. Repeated events have their corresponding weights added together.
- The highest order is given to the basic event with the largest weight.

The ordering could be determined at this point, by ordering the variables according to their weights. However, Minato et al choose the next event by deleting that part of the circuit which can only be reached from the input already chosen (in terms of fault trees, the ordered basic event and any branches leading to it would be deleted) and weights are reassigned from the beginning. By doing this, the largest weight in the last assignment is distributed to the neighbouring events, so that their weights are greatly increased. Therefore in many cases the neighbouring events are given near orders.

To illustrate this method, it will be applied to the fault tree shown in Figure 4.3(a). A weight of 1.0 is assigned to Top and this is propagated through the fault tree to give the distributed weights as shown in Figure 4.3(a). The weights of each basic event are therefore:

$$a = \tfrac{1}{4}$$

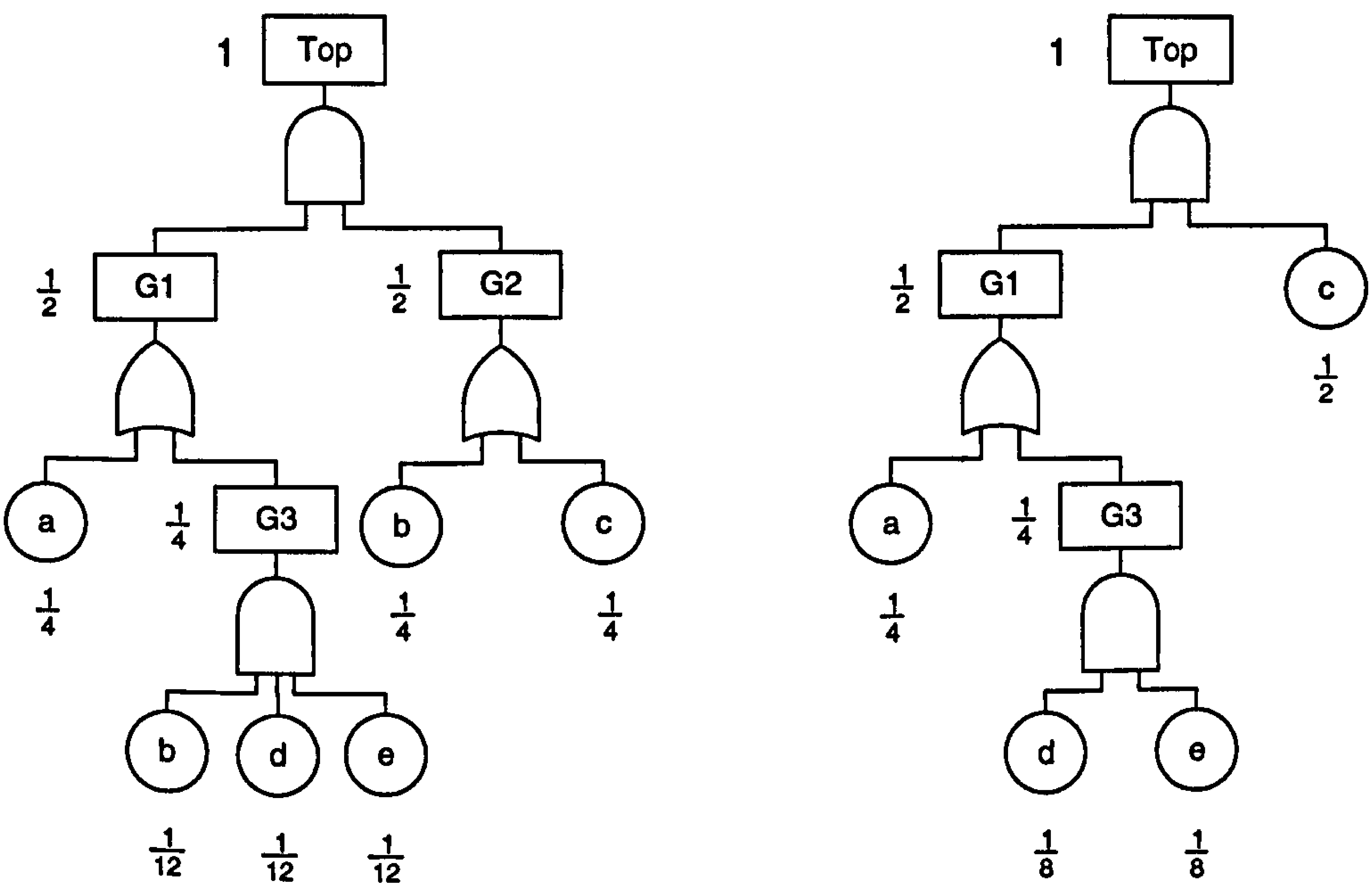
$$b = \tfrac{1}{4} + \tfrac{1}{12} = \tfrac{1}{3}$$

$$c = \tfrac{1}{4}$$

$$d = \tfrac{1}{12}$$

$$e = \tfrac{1}{12}$$

Therefore the first event in the ordering is 'b'. Now, all occurrences of event 'b' are removed from the fault tree, to give the fault tree shown in Figure 4.3(b). It is now possible to see that the next event in the ordering is 'c'. This process is continued until all the events are ordered.



(a) Weights for ordering the first variable

(b) Weights for ordering the second variable

Figure 4.3: Example fault tree for the top-down weighted method

The method does not address the problem of how to order events if they have equal weightings. If this case did arise however, the tie could be broken by selecting the event that occurs the greatest number of times (as it is the repeated events that cause cut set redundancy) or the least number of times (as this would mean that the individual events occur higher in the tree, therefore have more effect on the structure function) in the fault tree. If they had an equal number of occurrences, then a top-down or depth-first approach could be employed.

4.3.1.2 Applying Weights in a Bottom-Up Manner

This ordering scheme proceeds in a similar manner to the previous method, but the technique is initiated from the bottom of the tree, rather than the top. The general method is described below:

- A weight of 1.0 is assigned to each basic event and propagated towards the top event.
- At each gate, the weights of its inputs are added together to give the weight of the gate.

- Once the inputs to the top event have been assigned weights, the tree is explored in a depth-first manner, considering the branches with the largest weight first.
- The events are ordered as they are encountered.

No indication is made as to which branch would be chosen should two or more branches have the same weight. Also, when a gate has two or more basic event inputs, it is unclear as to the order in which the events should be considered.

To demonstrate this ordering, consider the fault tree shown in Figure 4.4. For this example, it is assumed that in the case of two branches having the same weight, the tie is broken by considering the leftmost branch first. Where a gate has two or more basic event inputs, the events shall be ordered in a left-right manner. Each event is given the weight 1. The weights of the gates are therefore calculated to be: G3 = 3, G2 = 2, G1 = 4. The depth-first traversal starts at the top gate and considers G1 first as it has a larger weight than G2. G3 is ordered next and then finally G2. This gives the basic event ordering:

a<b<d<e<c

This ordering seems to give priority to the largest subtrees, whilst preserving neighbourhoods.

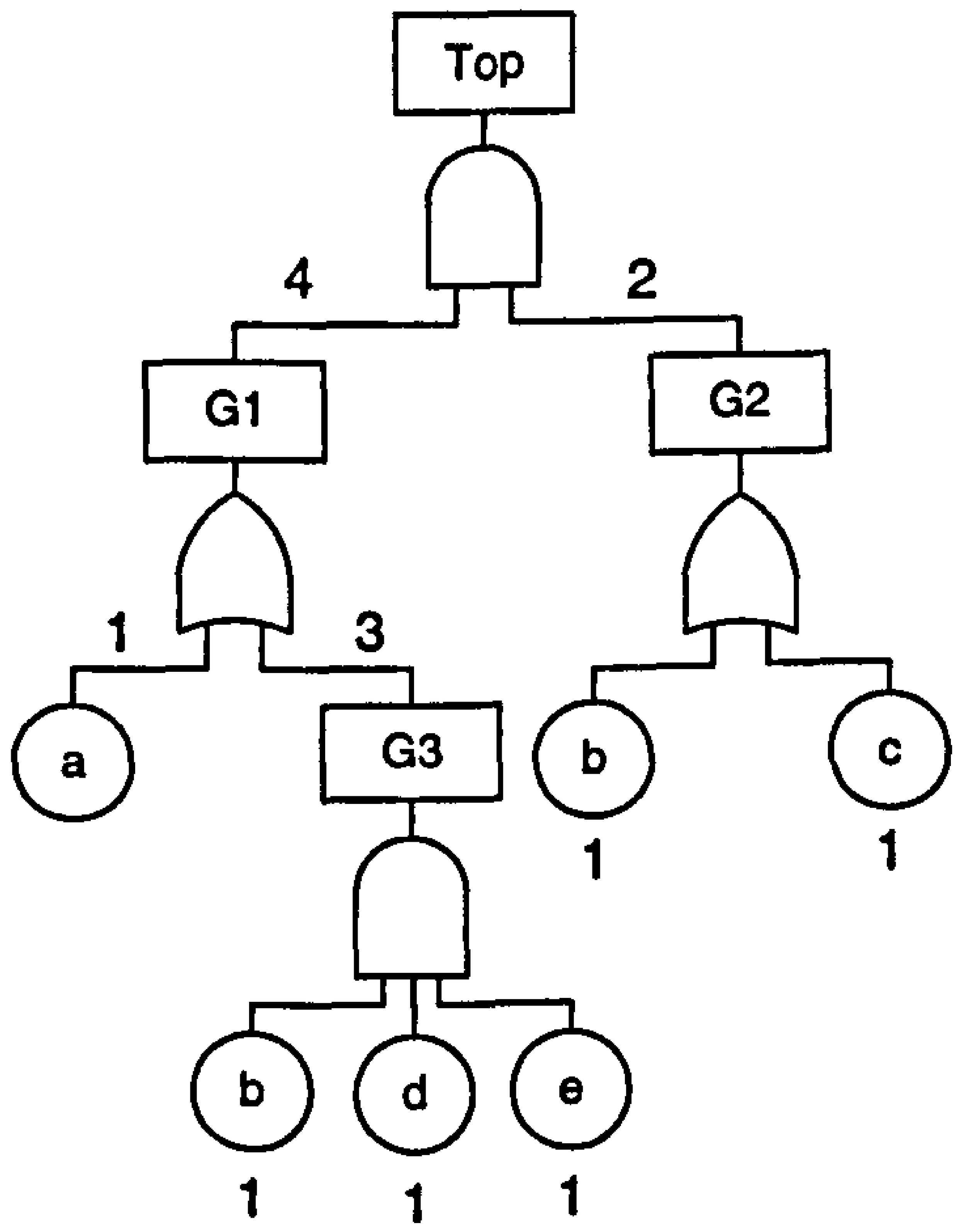


Figure 4.4: Fault tree showing the weights method applied in a bottom-up manner

Several variations on this technique are possible. One alternative is to combine the weights of 'AND' and 'OR' gates differently. For example, adding the weights at an 'OR' gate and multiplying them at an 'AND' gate. This would give 'OR' gates with many inputs precedence over 'AND' gates with many inputs, where it would be fair to assume that the events beneath the 'OR' gate would have more influence over the occurrence of the top event as only one is needed for the logic to flow, compared with the 'AND' gate where every event would need to occur.

Once the weights have been assigned, the gates could be ordered using an alternative method. For example, each level could be considered in turn, from top to bottom, ordering the gates at each level according to their weight, i.e. largest weight first and ordering the events as they are encountered.

A possible way to decide the order of events that occur together under a gate is to consider repeated events first. This method has proved successful in its application to many other ordering methods and there is no reason why it should not be successfully applied here. Repeated events could also determine which branch is chosen in the case of equal weights; the branch that has the most repeated events below could be the first to be considered.

4.3.2 Importance Measures

Bartlett[19] has performed extensive investigations into the use of importance measures for variable ordering. The aim of this research was to rank the basic events in terms of their significance within the system, in a way that is not dependent on how the fault tree is written.

In order to explore the potential of importance measures for determining a good ordering, Birnbaum's structural importance measure for each component was derived from the fault trees' BDDs. These importance measures were used as an indicator of the importance of each component within the system. An advantage of using importance measures is that they produce the same values regardless of how the fault tree is written. The variables were ranked with those of highest importance appearing earlier in the ordering than those of lower importance. The order for variables with the same value of importance was decided by ordering the one appearing highest in the fault tree structure first.

225 trees were ordered using this measure and the results were compared with the best of six previously identified alternative schemes[25]:

- Top-down.
- Modified top-down.
- Depth-first.
- Modified depth-first.
- Priority depth-first.
- Modified priority depth-first.

It was reported that 76.9% of the 225 trees produced BDDs that had fewer or the same number of nodes as the previous best scheme. Of these six ordering schemes, it was noted that the best results had previously been obtained with the modified top-down scheme (as shown in Table 4.4), with 29.8% of the trees producing BDDs with the fewest nodes compared with the other five orderings. The structural importance measure shows significantly improved results.

The method was then adapted to consider the most repeated event first when the importance measure failed to distinguish between events. If there was still a tie, the events were ordered as before (in a top-down manner). This modified method produced different orderings for 152 of the 225 trees. The percentage with equal or fewer nodes than the previous best of the six structural schemes increased to 77.3%. This is a small improvement on the previous 76.9% obtained without this modification. Bartlett concludes that a different method may be more beneficial in reducing the number of nodes. Either a different approach for ordering those components with equal importance measures could be implemented, or a different importance measure could be used. However, the overall performance is better than that of any other heuristic and shows good potential.

The main drawback of this method is the need to calculate the importance measures from the BDD (obtaining them from the fault tree is very inefficient) and Bartlett addresses this problem by considering the use of approximations that could produce the same ordering. Three possibilities were considered:

i. Look for patterns within the tree that relate to the importance measures, so enabling an ordering to be established by inspection of the tree.
ii. Generate alternative measures, similar to the importance measures, derived by an alternative method.
iii. Apply Birnbaum's structural importance method directly to the tree.

Each of these is now reviewed in more detail:

i. The importance measures for the events of several fault trees were calculated and Bartlett attempted to identify patterns within these trees that related to the measures. The conclusion drawn was that no obvious patterns were identifiable and this option was given no further consideration.

ii. The aim of this approach was to consider a number of alternative weighting methods that are fast and efficient to apply to the fault tree, but which give component rankings similar to those obtained by the calculated importance measures. Three weighting methods were examined:

- **Calculation of Importance by Dividing by the Number of Inputs**

This method is similar to that discussed in section 4.3.1.1, but does not restructure the tree after ordering each event. The top event is given a weight value of one. The weight values of its input events are calculated by dividing the weight of the top event by its number of inputs. For example, if there were three inputs, each is then given a weight value of 1/3. This is continued down through the tree, so that the inputs to any gate are given the weight value of that gate divided by its number of inputs.

No mention is made at this stage of how repeated events are dealt with, but the conclusion drawn is that the orderings obtained do not match those of the calculated structural importance measures.

Bartlett then considers how to approach repeated events: adding together their values disproportionately increases their importance but taking the average of the values would probably underestimate its importance. Therefore a different approach is taken, by scaling the total combination. The weight for the repeated event is calculated by summing the values and multiplying by the square root of the total number of repeated components:

$$w_i = \sqrt{n} \sum_j w_{ij} \qquad 4.1$$

where i is the component, and j each of its occurrences.

Using this scaling mechanism for repeated events, the ordering produced BDDs with fewer or equal nodes than the best of the previous six alternatives in 52.9% of the 225 fault trees.

- **Calculation of Importance by Dividing by the Number of Critical States**

This measure is similar to the one above, except that the criticality of the component is considered when calculating the weights. In the above measure, weight values for the gate inputs are calculated by dividing the gate's weight value by its number of inputs. Here, the weight values of the gate inputs are calculated by dividing the gate value by the number of critical states for the component. Therefore, if there are n inputs to the gate, the criticality of a component is given by $1/2^{n-1}$, compared with the previous measure of 1/n. Repeated events were dealt with by using the weighting method in Equation 4.1.

This ordering produced BDDs with fewer or equal nodes than the best of the previous six alternatives in 50.2% of the 225 fault trees. Neither this nor the method above has produced results close to those obtained using the calculated structural importance measures.

- **Altering the Repeated Events Weighting**

Bartlett considers the problem to be due to the repeated events. Therefore a new weighting method for the repeated events was used, whereby the weight values of the repeated events are added, but the value of its second occurrence is divided by the square root of two and the value of its n^{th} occurrence is divided by the square root of n.

This method produced BDDs with fewer or equal nodes than the best of the previous six alternatives in 62.2% of the 225 fault trees. This is a significant improvement on the previous two methods, however it is still 15.1% lower than the best results obtained

iii. The aim of this third method was to apply the principle of Birnbaum's structural importance measure directly to the tree. This was implemented as follows:

The contribution of each basic event to the occurrence of the top event is calculated according to:

$$I_i = Q(1_i, \underline{½}) - Q(0_i, \underline{½}) \qquad 4.2$$

The selected basic event therefore assumes the failure probabilities of 1 and 0 on two consecutive computations of the top event probability, with the remaining components being given failure probabilities of ½. The event probabilities are worked up through the tree, with the contributions of intermediate events (gates) calculated using Equation 4.3 for 'AND' gates and Equation 4.4 for 'OR' gates.

$$P(\text{gate}) = \prod_{i=1}^{n} q_i \qquad 4.3$$

$$P(\text{gate}) = 1 - \prod_{i=1}^{n} (1 - q_i) \qquad 4.4$$

where n is the number of inputs to the gate.

The result of the second run (with a failure probability of 0) is subtracted from the first run (with failure probability 1) to give the probability value contribution of that basic event to the occurrence of the top event. The basic events are ordered with those giving the largest contribution earlier in the ordering than those with smaller contributions. If events have an equal contribution, then they are ordered according to the top-down ordering scheme.

This ordering technique is demonstrated using the fault tree shown in Figure 4.5.

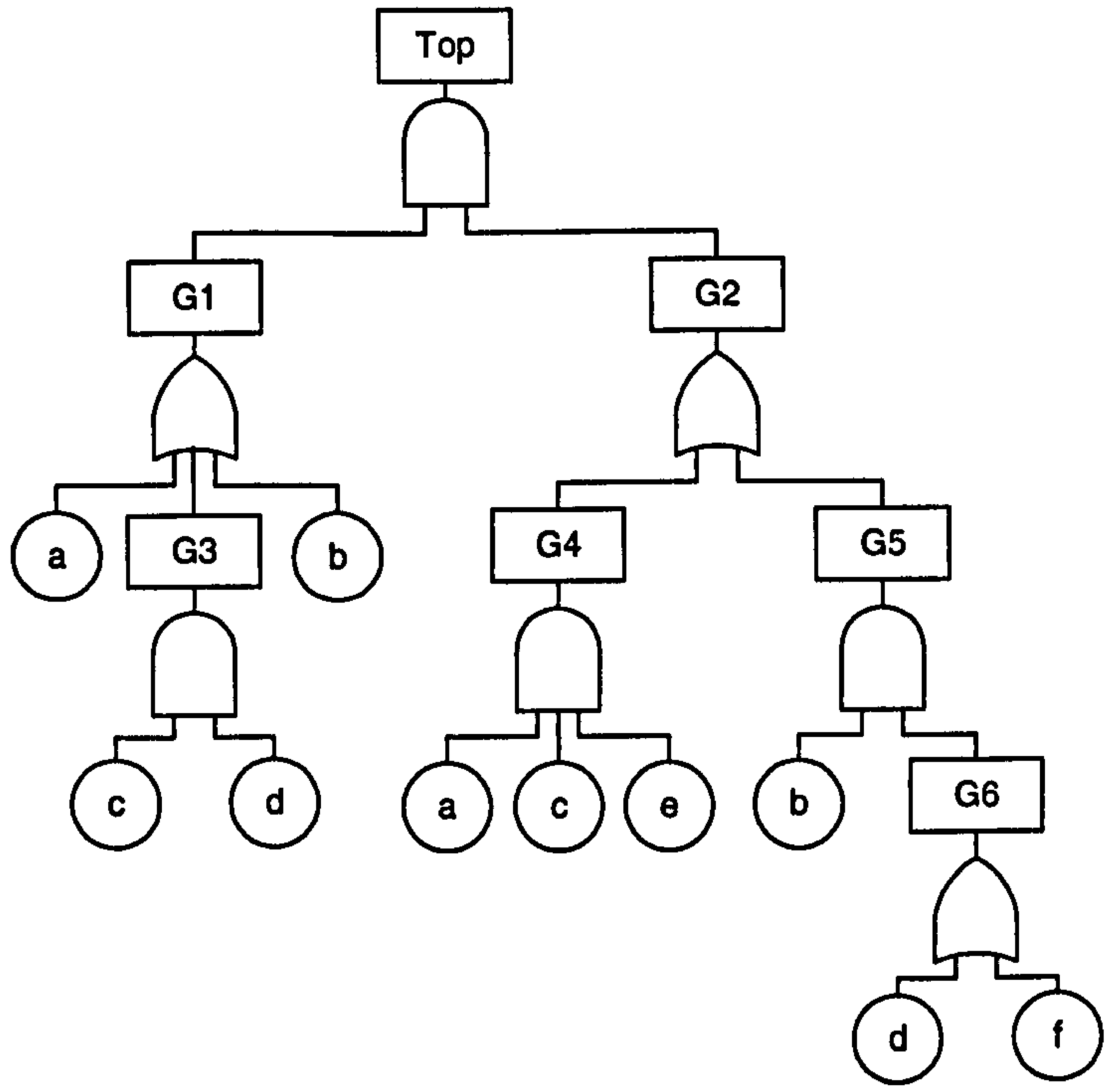


Figure 4.5: Example fault tree

Starting with event 'a', it first assumes a failure probability of 1, with the remaining events assigned probabilities of ½. The probabilities of the gates are then calculated, starting with those containing basic event inputs only and working up through the tree to the top event. The results are shown in Table 4.5.

Gate	Top	G1	G2	G3	G4	G5	G6
P(gate)	$\frac{17}{32}$	1	$\frac{17}{32}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{3}{4}$

Table 4.5: Gate probabilities, with event 'a' assigned a failure probability of 1

The calculations are repeated, with event 'a' assigned a probability of 0. The resulting gate probabilities are shown in Table 4.6.

Gate	Top	G1	G2	G3	G4	G5	G6
P(gate)	$\frac{15}{64}$	$\frac{5}{8}$	$\frac{3}{8}$	$\frac{1}{4}$	0	$\frac{3}{8}$	$\frac{3}{4}$

Table 4.6: Gate probabilities, with event 'a' assigned a failure probability of 0

The contribution of event 'a' to the occurrence of the top event is therefore:

$$I_a = \frac{17}{32} - \frac{15}{64} = \frac{19}{64} \approx 0.297$$

The calculations are repeated for each basic event, giving the contributions shown in Table 4.7.

Event, i	a	b	c	d	e	f
$Q(1_i, \frac{1}{2})$	0.531	0.781	0.465	0.492	0.432	0.457
$Q(0_i, \frac{1}{2})$	0.234	0.078	0.281	0.258	0.305	0.279
I_i	0.297	0.703	0.184	0.234	0.127	0.178

Table 4.7: Contributions of the basic events to the top event occurrence

The events are ordered from the one with the largest contribution to the one with the smallest contribution to give the ordering:

$$b<a<d<c<f<e$$

The difference between this approximation and the exact version of Birnbaum's structural importance measure is that redundant combinations of basic events can occur, as the method of working the probabilities up through the tree means that the intermediate events are not necessarily independent. The values obtained by the approximation method are therefore not exact, but it was thought that they might still offer a good ordering heuristic.

The results obtained in Bartlett's study showed that in 67.1% of cases, the BDD produced had equal or fewer nodes than the BDD produced using the best schemes option from the six structural heuristics.

Bartlett concluded that further improvements could be made by altering the method for dealing with events with the same importance. However, this is the best approximated measure of those considered and although the results are not quite as good as the 77.3% obtained with the structural importance measures calculated from the BDD, it is much more efficient to implement.

The use of importance measures shows great potential as a method of variable ordering. Obviously calculating the importance measures from the BDD is not a viable option, but approximation methods could be used to obtain similar results.

4.4 Optimising the Fault Tree Before Application of Ordering Heuristics

Bouissou[23] suggests that the major problem with conventional ordering heuristics is the lack of theoretically proven properties. He also acknowledges the fact that many heuristics are very sensitive to the way the fault tree is written, which can lead to BDD sizes that differ by many orders of magnitude. In this paper he proposes that the modules of the fault tree should be taken into account when ordering the variables and investigates the effect that this has on the size of the resulting BDD.

Bouissou presents the following theorem:

Let f and g be two functions of disjoint sets of variables and $T_\sigma(f)$ and $T_\sigma(g)$ denote the sizes of the BDDs of those functions respectively, obtained with variable ordering σ. Then

$$T_{\sigma f,\sigma g}(f \cup g) = T_{\sigma g,\sigma f}(f \cup g) = T_{\sigma f}(f) + T_{\sigma g}(g)$$

where σf, σg stands for the ordering obtained by concatenation of σf and σg.

Bouissou states that this equation makes it possible to hope for 'reasonable growth' of the BDD size with an increasing number of variables and suggests the following constraint for any ordering heuristic: the heuristic should group the variables of a module.

A completely modularised tree (i.e. a tree for which each sub-tree is a module) can be represented by a BDD of size n (i.e. the number of variables). For such a tree, the way it is written has no effect on the BDD size.

An optimisation technique is presented, which restructures the fault tree to make the modules appear. The optimiser works in three phases:

- The fault tree is transformed into a sequence of alternating gates. Single input gates and equivalent gates are suppressed.
- The following simplifying rules are repeatedly applied:

  $$(a \cup b_1) \cap (a \cup b_2) \cap \ldots \cap (a \cup b_n) \cap c \rightarrow (a \cup (b_1 \cap b_2 \cap \ldots \cap b_n)) \cap c$$

  $$(a \cap b_1) \cup (a \cap b_2) \cup \ldots \cup (a \cap b_n) \cup c \rightarrow (a \cap (b_1 \cup b_2 \cup \ldots \cup b_n)) \cup c$$

  $$a \cup (a \cap b) \rightarrow a \qquad a \cap (a \cup b) \rightarrow a$$

- Implicit modules are made explicit. For each 'OR' gate in the tree the maximum subset of basic events that always appear together is found.

In fact, many aspects of this optimisation technique are similar to the 'Faunet' reduction approach of Platz and Olsen[26], which is introduced and used in Chapter 6. The first phase is equivalent to the 'contraction' stage, where subsequent gates of the same type are contracted to form a single gate. This gives an alternating sequence of 'AND' and 'OR' gates. Single input gates would automatically be suppressed, as they do not form a true fault tree structure. The first two simplifying rules of the optimisation technique employ the Boolean distributive laws, which form the basis of the extraction step of the Faunet approach. Finally, the third phase of the optimisation technique groups sets of basic events that always occur together in the fault tree. In effect, this is what the factorisation step of the Faunet technique achieves, when it repeatedly takes pairs of events, combining them to form complex events.

The only phase of the optimisation technique that can lead to excessive CPU time is the second one. Therefore, two versions of the optimiser are used: 01 is the simplified version, consisting of only the first and last steps; 02 is the full optimisation.

Once the tree has been optimised, any ordering heuristic can be used, as long as it orders the variables of modules together.

The optimiser was tested on a group of fault trees by finding the number of BDD nodes and the amount of CPU time used for their analysis both before and after optimisation. Each calculation was carried out on 100 randomly generated re-writings of the tree, in order to show the sensitivity of the heuristics to the way the tree is structured. Two heuristics were used, the first of which was the depth-first heuristic, used in the program ARALIA[27]. METAPRIME[28] was also used, which incorporates the following heuristic:

- The level of a gate or variable is defined as:

  $$\text{Level(top)} = 0; \text{level}(f) = \max(\text{level}(g_i))+1$$

  where the g_i are the parents of f.

- The variables are put in order by increasing levels. METAPRIME uses an enhanced version of this heuristic, whereby the variables of a module are ordered together.

It was found that the depth-first heuristic, on average, gives better results than METAPRIME's heuristic.

The optimised version of the trees produced smaller average BDD sizes and used less CPU time than the original fault trees. However, the maximum BDD sizes for the optimised trees increased.

For 20 large trees, which couldn't originally be processed with a reasonable success rate, it was found that all could be processed without failure, for each of the 100 trials, in their optimised form.

The author concludes that restructuring the tree to create as many modules as possible is an efficient pre-processing tool, the cost of which (in terms of CPU time) is negligible when compared with the savings to be made when generating the BDD.

4.5 Results of a Comparative Study of Several Ordering Heuristics

Bouissou et al[18] compared twelve heuristics, presenting the results of the best six (the remaining six were not detailed). The six heuristics used are:

1. Depth-first (first alternative method), as in section 4.2.2.
2. Weights applied bottom-up, as in section 4.3.1.2.
3. Depth-first, considering repeated events and gates first, as in section 4.2.4.
4. Depth-first, with number of leaves, as in section 4.2.2.2.
5. Heuristic 3 applied to heuristic 2.
6. Heuristic 3 applied to heuristic 4.

The authors take into account the fact that heuristics can give significantly different results according to how the fault tree is written. The heuristics were tested on 500 random re-writings of thirteen fault trees, in both their original and optimised forms (using optimisers 01 and 02 as discussed in section 4.4).

The results obtained show that the heuristics fall into two classes. The first class, containing heuristics 2, 4, 5 and 6, tends to give a very low standard deviation on the BDD size, showing that the heuristic is not very sensitive to the re-ordering of branches within the tree. The sizes of the BDDs also tend to be neither excellent, nor bad, usually somewhere between the two. For the optimised trees, the results are usually good or excellent.

The second class, containing heuristics 1 and 3, show a high standard deviation in the BDD size. The heuristics can generate BDDs with fewer nodes than the first class, but can also lead to extremely large BDDs (up to 1500 times larger than the smallest). It seems that in most cases, heuristic 3 gives better results than heuristic 1, in terms of mean, maximum and minimum BDD size.

The results given in the paper suggest that the first class of heuristics are less sensitive to the way the fault tree is written, but only give average results in terms of BDD size. Using the explorative capabilities of the second class of heuristics is more likely to lead to smaller BDDs, but also there is a greater chance of resulting in a large BDD due to their sensitivity to the way the fault tree is written.

4.6 Pattern Recognition Techniques

Pattern recognition techniques can be used to identify patterns within the fault trees and select an appropriate ordering heuristic based on the results. Three different techniques have been explored: the classifier system, the multi-layer perceptron neural network and the radial basis function neural network. The results obtained are reviewed in the following three sections.

4.6.1 The Machine Learning Classifier System Incorporating Genetic Algorithms

The use of classifier systems as a method of selecting the most appropriate ordering scheme for a particular fault tree has been investigated by Bartlett and Andrews[29]. They use a machine learning approach based on genetic algorithms to build a classifier that chooses an ordering scheme according to certain characteristics of the fault tree.

A classifier system is a machine learning system that generates a model of a particular problem by learning the rules that govern the problem through a training process. The rules, which reflect the patterns within the problem, are generated by subjecting the classifier system to large amounts of training data. Once the system adequately models the problem, it can be used for predictive purposes. The system then takes a new input (whose output is unknown) and by applying the rules learnt during training, provides a response. The performance of the algorithm is evaluated from the number of correct responses.

The classifier approach was applied to the ordering problem, where the aim was to learn the rules that govern the relationship between the characteristics of the fault tree and the best ordering scheme option. Once the training had been completed, the system was used to predict ordering schemes for a set of test fault trees, depending on their characteristics and the rules that had been learnt from the training data.

Key features that were thought to describe the fault tree structure were identified, and provide the inputs to the machine learning algorithm in the form of a 19 bit binary string. In total, six characteristics were initially selected:

- Percentage of 'AND' gates in the tree.
- Percentage of different events repeated.
- Percentage of total events repeated.
- Top event gate type.
- Number of outputs from the top event.
- Number of levels in the tree.

The ordering represents the output or response of the classifier in the form of a six bit binary string of 1's and 0's, where a 1 represents the best scheme option and a 0 otherwise. The choices of ordering schemes were based on previous heuristic work by Sinnamon[21]:

- Top-down.
- Modified top-down.
- Depth-first.
- Modified depth-first.
- Priority depth-first.
- Modified priority depth-first.

Each fault tree in the training set was analysed for the best ordering scheme. This, together with the characteristics data was used to produce the training data set, from which the classifier was trained.

The classifier was then used to predict the best ordering schemes for a set of twenty test fault trees. The results were compared with known best schemes for these trees.

The conclusion drawn by the authors was that this model could be trained to predict the best ordering scheme to use on a particular fault tree to produce the most efficient BDD representation. However, they acknowledged that the small group of characteristics used did not adequately represent the fault tree and other characteristics need to be developed. Quantitative results obtained from this investigation are given in Bartlett's doctoral thesis[19] and show that four and five correct predictions out of a possible twenty were obtained.

The work on classifiers was extended by Bartlett and Andrews[25] to use more characteristics to represent the fault tree. Eleven characteristics were considered compared to the previous six. The five additional characteristics are:

- The number of basic events.
- The maximum number of gates in any level.
- Number of gates with gate inputs only.
- Number of gates with event inputs only.
- The highest multiple of a repeated event.

The results are reported to have been more accurate for smaller fault trees. The authors suggest that modifications to the characteristics chosen for larger trees may produce results that are more convincing. Bartlett's doctoral thesis[19] reveals that when using eleven characteristics, the best result was nine out of twenty correct predictions.

4.6.2 Neural Networks: The Multi-Layer Perceptron

Bartlett[19] extended the use of pattern recognition techniques to consider neural networks, which have a more solid theoretical base than the classifier approach. The first neural network model used was the multi-layer perceptron. As with the machine learning approach, the aim is to select the best ordering scheme for a fault tree according to its characteristics.

Neural networks offer a powerful framework for representing non-linear mappings from several input variables to several output variables. The form of the mapping is controlled by a number of adjustable parameters, known as weights, whose values are determined through a training process. In the prediction phase, the weights then determine the path through the network, and so the output response for a given set of inputs.

The multi-layer perceptron consists of a layer of input units, one or more hidden layers of hidden units and a layer of output units. Connections, governed by the weight values, run between every unit in one layer to every unit in the next layer. This is shown in Figure 4.6. The bias units act like adding a constant to an equation.

Numerical values can be applied to the input and output variables, rather than the simple binary representation used in the classifier approach. This has the advantage of being able to give an indication of how good a scheme is in relation to the best.

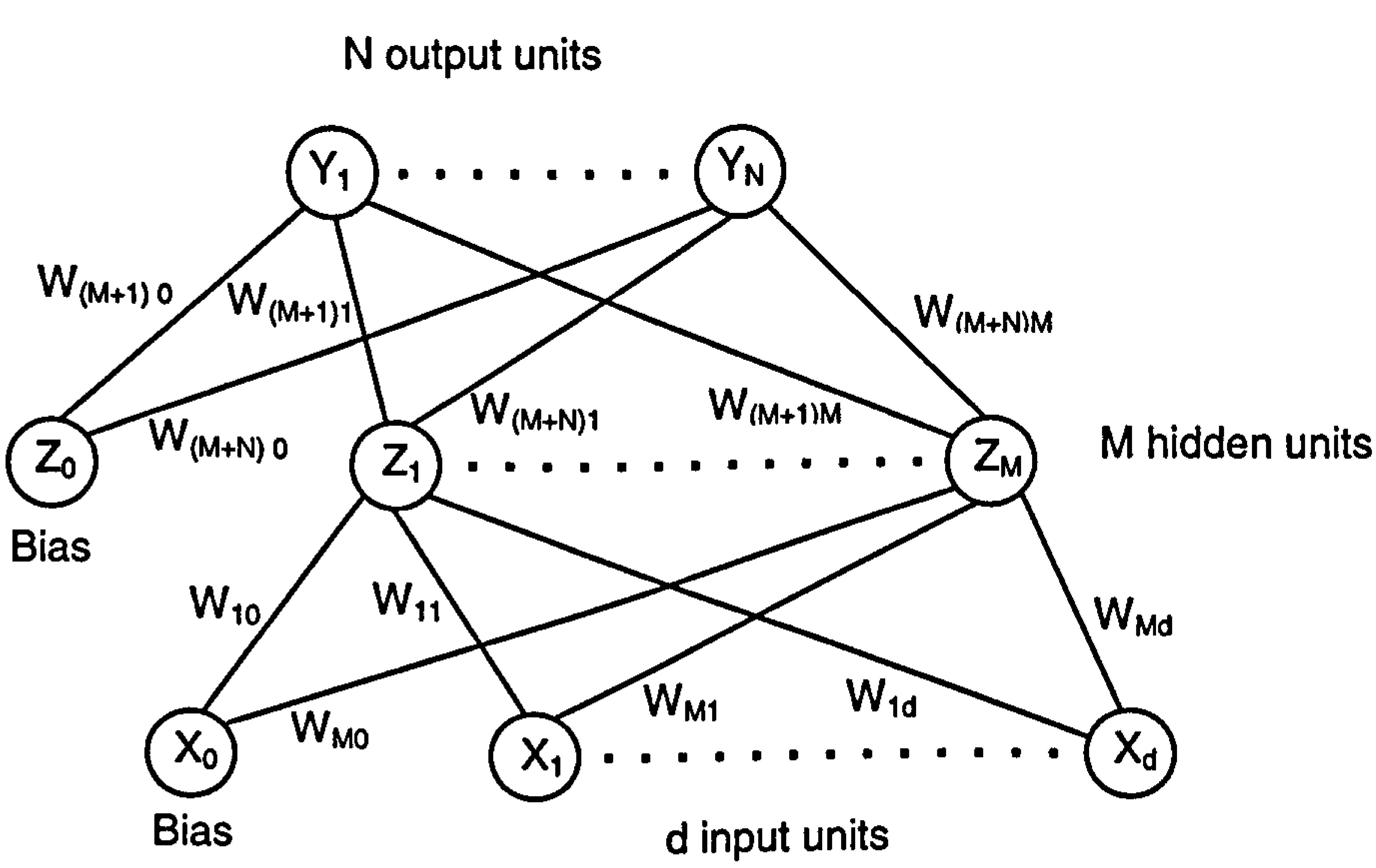


Figure 4.6: Multi-layer perceptron neural network

Bartlett reports that numerous trials were conducted to find the best network architecture for predicting the optimal ordering schemes for a set of twenty test fault trees. The best network was comprised of eleven units in the input layer, each of which represented one of eleven fault tree characteristics:

- Percentage of 'AND' gates in tree.
- Percentage of different events repeated.
- Percentage of total events repeated.
- Top event gate type.
- Number of outputs from top event.
- Number of levels in tree.
- Number of basic events.
- The maximum number of gates in any level.
- Number of gates with gate inputs only.
- Number of gates with event inputs only.
- The highest multiple of a repeated event.

The output layer consisted of six units, one for each of the ordering schemes:

- Top-down.
- Modified top-down.
- Depth-first.
- Modified depth-first.
- Priority depth-first.
- Modified priority depth-first.

With a training set of 198 fault trees, it was found that one hidden layer with five units offered the best results, predicting the correct ordering schemes for 14/20 test trees.

Bartlett suggests that the method is capable of predicting the best ordering scheme for fault trees and that these results could be improved by using a larger training data set. The basis for this hypothesis is that 186 training fault trees were used initially and the best results obtained were 13/20 correct predictions. The addition of extra data into the training set improved these results. Bartlett concludes that the inputs have the most influence on the neural network and so the characteristics used to describe the fault tree structure need to be examined in more detail.

4.6.3 Neural Networks: The Radial Basis Function

The radial basis function is another class of neural network and was also investigated by Bartlett[19] as a method for selecting the best ordering scheme for a particular fault tree.

Diagrammatically, the radial basis function network looks very similar to the multi-layer perceptron, as shown in Figure 4.7. However, the radial basis function network has only one hidden layer, made up of a number of radial basis functions. The connections that run from the input layer to the hidden layer represent the vectors that determine the centres of the basis functions. The connections between the hidden layer and the output layer represent the weights of the network in the same way as with the multi-layer perceptron model.

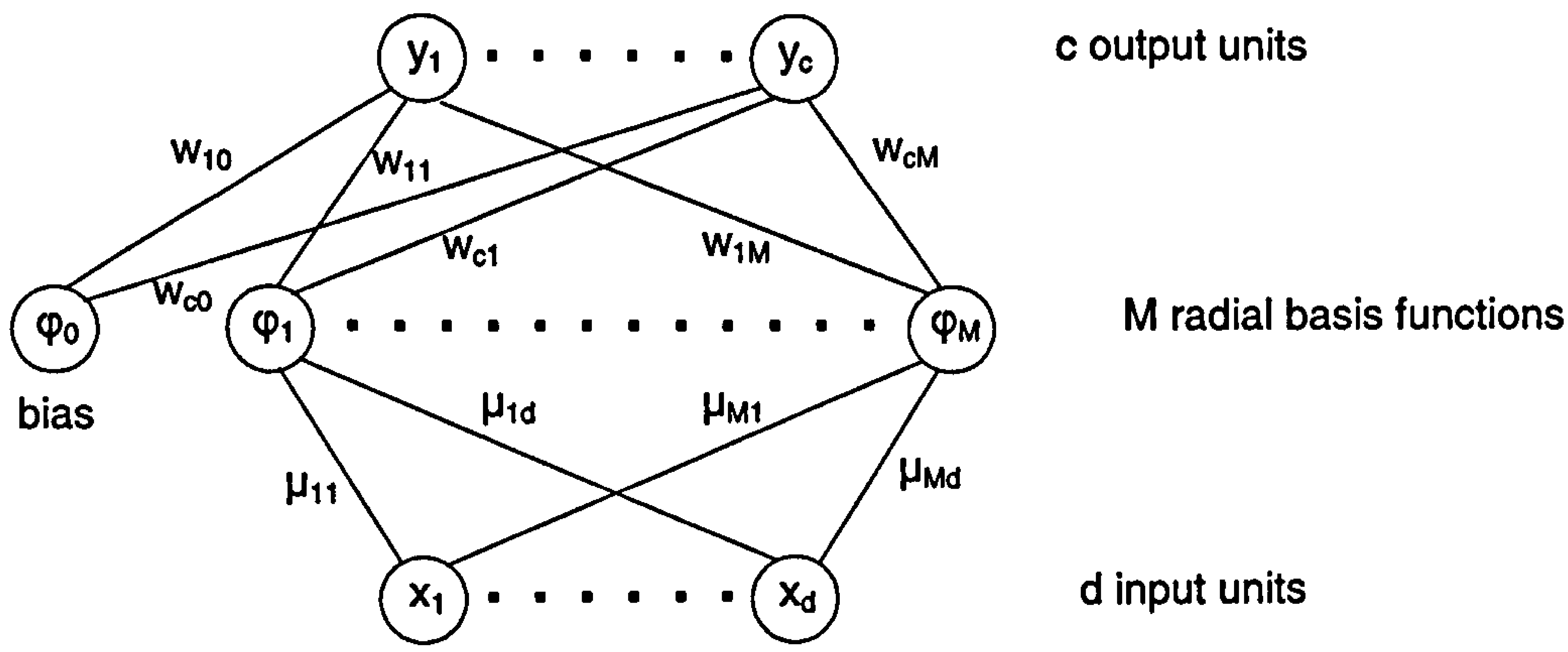


Figure 4.7: Radial basis function neural network

The radial basis function centres and the final layer weights are determined by the training process and are subsequently used in the prediction phase to calculate the output responses from the network for a new set of inputs.

Bartlett reports that numerous trials were carried out to determine the network architecture that predicts the greatest number of correct ordering schemes for twenty test fault trees. As with the multi-layer perceptron, the best networks comprised of eleven units in the input layer and six units in the output layer. Again, numerical values can be applied to the input and output variables, which gives an indication of how good each scheme is in relation to the best. The input units each represented one of the following fault tree characteristics:

- Percentage of 'AND' gates in tree.
- Percentage of total events repeated.
- Percentage of different events repeated.
- Top event gate type.
- Number of levels in tree.
- Number of outputs from top event.
- Number of basic events.
- The maximum number of gates in any level.
- Number of gates with event inputs only.
- Number of gates with gate inputs only.
- The highest multiple of a repeated event.

The output units represent the possible variable ordering schemes:

- Top-down.
- Modified top-down.
- Depth-first.
- Modified depth-first.
- Priority depth-first.
- Modified priority depth-first.

Eight network architectures were identified that were capable of predicting the correct ordering schemes for 14/20 test fault trees. These had between four and nine radial basis function centres. The most efficient of these networks had four centres and in five out of the six incorrect predictions chose the second best ordering scheme.

Bartlett concludes that the radial basis function neural network has the potential to model the variable ordering problem but that improvements could be made by examining the fault tree characteristics in more detail to determine which have the greatest influence on the outcome of the network.

4.7 Summary

There is no ordering heuristic capable of producing a good variable ordering for all fault trees. Many heuristics have been proposed, but most are based on intuition and few conclusions as to the required features of a good heuristic have been drawn.

Much of the research has centred on structural ordering techniques, but results obtained from the weighted scheme based on importance measures appear to be very promising.

Optimising the fault tree before application of the ordering schemes takes relatively little time, but can result in the construction of much smaller BDDs and has been shown to produce BDDs for trees that had not previously been analysed in a reasonable time.

Pattern recognition techniques could offer a good way of selecting an ordering scheme, based on the characteristics of the fault tree. The best results obtained so far, with both the multi-layer perceptron and the radial basis function models, predict the best ordering scheme in 70% of cases. However there were only six structural schemes to choose from, so the method could be extended to include weighted methods as options.

Chapter 5: Comparison of Variable Ordering Schemes

5.1 Introduction

The survey of ordering schemes conducted in the previous chapter has highlighted methods that have not been fully investigated and would benefit from further consideration. Therefore a number of ordering techniques were chosen for a comparative study, in order to assess whether they could provide an alternative means of ordering that would result in a more efficient BDD construction process.

Eight schemes were selected, with modifications made as necessary to incorporate elements from other schemes that had proven advantageous in the BDD construction process. Some of the modifications suggested in the previous chapter were also implemented, including various methods of dealing with 'tied' variables (i.e. variables that remain 'equal' in the ordering after the application of other heuristics). One of the most important features of an ordering scheme is that it is discriminating (i.e. it will always produce the same ordering for a particular fault tree) and it was ensured that each of the ordering techniques fulfilled this criterion. The eight chosen schemes and the reasons for their selection are detailed below.

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

The modified versions of the first three schemes were chosen, as they had performed well in previous work, and gave consistently better results than the non-modified versions. The first two schemes are also very widely used and provide a good benchmark against which to test the other schemes. The fourth scheme (depth-first, with number of leaves) implements an alternative method of choosing the gates within the depth-first scheme, and as the depth-first scheme had proven to be a good choice, this scheme was also considered. The four weighted methods (dynamic and non-dynamic top-down weights, bottom-up weights and event criticality) were chosen as an alternative to the structural ordering schemes. Much of the previous work on ordering heuristics has centred on structural methods and it was felt that the weighted techniques need to be examined in more detail and their performance compared to structural schemes. The final ordering scheme, which applies Birnbaum's measure directly to the tree for an approximate event importance ordering, has produced particularly good results in a previous investigation and was included for this reason.

This chapter describes each of the selected schemes in detail and then discusses their individual performances on a set of example fault trees.

5.2 Descriptions of the Eight Ordering Schemes

As modifications have been made to the schemes introduced in the literature survey, a full description of each scheme and how it is applied to the example fault tree shown in Figure 5.1 is now given.

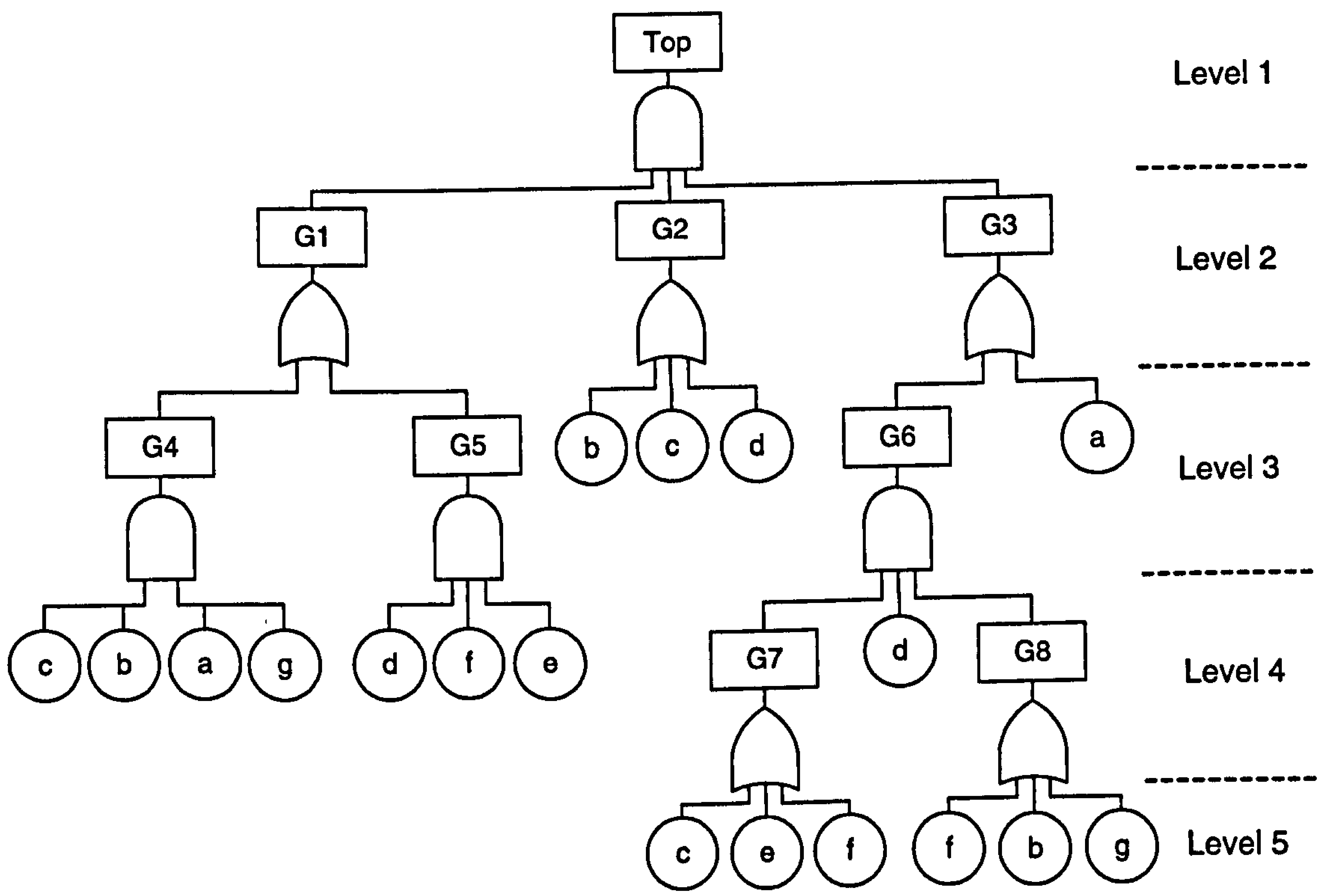


Figure 5.1: Example fault tree used to demonstrate the ordering schemes

5.2.1 Modified Top-Down Ordering

The tree is scanned in a top-down manner. Variables appearing on the same level within the tree are ordered according to their total number of occurrences in the fault tree. Those with higher occurrence are ordered first. If there are two or more variables with an equal number of occurrences, then they are ordered as they appear from left to right on that level. Each event is placed in the ordering the first time it is encountered; subsequent occurrences are ignored.

This scheme can now be applied to the fault tree shown in Figure 5.1. There are no events to consider on the first two levels, so level three is the first level to be examined. Four events appear on this level, which reading from left to right are: 'b', 'c', 'd' and 'a', which need to be ordered according to their number of occurrences elsewhere in the tree. In fact, events 'b', 'c' and 'd' occur an equal number of times (three occurrences) so remain in the left to right order in which they were placed and event 'a' occurs the least number of times (twice) so is ordered after the other events. The partial ordering for this level is therefore:

$$b<c<d<a.$$

Level four is now considered and the events appearing on this level that haven't already been ordered are (listed from left to right): 'g', 'f' and 'e'. Event 'f' appears most often in the tree (three occurrences) so is ordered first, whilst 'g' and 'e' occur an equal number of times, so retain their respective positions. The ordering then becomes:

$$b<c<d<a<f<g<e.$$

There is no need to consider level five, as all the events have been placed in the ordering.

5.2.2 Modified Depth-First Ordering

The modified depth-first ordering scheme considers the gate inputs to any gate in a left-right manner, such that the subtree of the left-most gate is completely explored before considering the remaining gate inputs. Any basic event inputs to a gate are considered before the gate inputs, and are ordered according to their total number of repetitions in the fault tree. The events with the greatest number of occurrences are ordered first, but if there is a tie then they are simply ordered as they appear from left to right in the list of inputs.

The ordering scheme can be applied to the fault tree in Figure 5.1 in the following manner:

The top event, Top, has no event inputs to order, so its three gate inputs, G1, G2 and G3, are considered in turn. The subtree of the leftmost gate, G1, is explored first. Again, this contains no event inputs, but has two gate inputs, G4 and G5, which are processed before returning to consider G2 and G3. G4 appears first in the input list to G1, so is considered next. It has four event inputs: 'c', 'b', 'a' and 'g'. As 'c' and 'b' are the most repeated events (each occurring three times) they appear before 'a' and 'g' in the ordering, which both occur twice. Event 'c' is ordered before 'b' as it appears leftmost in the inputs list and 'a' appears before 'g' for the same reason. This gives the partial ordering:

$$c<b<a<g$$

G4 contains no gates, so the process continues by examining G5, which contains the events 'd', 'f' and 'e'. Events 'd' and 'f' occur the greatest number of times (three appearances) so appear before 'e' in the ordering. Event 'd' is ordered before 'f' as it appears to the left of 'f' in the inputs list. This gives the ordering:

$$c<b<a<g<d<f<e$$

All the events have now been ordered, so it is not necessary to consider gates G2 and G3.

5.2.3 Modified Priority Depth-First Ordering

This ordering scheme is simply an extension of the modified depth-first method, where rather than simply considering the gate inputs from left to right, any gates which themselves have only basic events as inputs, are given preference. Basic events are ordered as in the modified depth-first method, such that the most repeated events are given priority - if there is a tie then they are ordered from left to right as they appear in the list of inputs. Events continue to be considered before any gate inputs.

This ordering technique can be applied to the fault tree in Figure 5.1 in a similar way to the modified depth-first scheme. The top event, Top, has no event inputs to order, so its three gate inputs, G1, G2 and G3, are considered in turn. As G2 has only basic event inputs, it is explored before the other gates, i.e. the inputs to Top are considered in the order G2, G1, G3. G2 contains the events 'b', 'c' and 'd', which, as they already appear with the most repeated events first, retain their respective positions when placed in the ordering:

$$b<c<d$$

The subtree of the next gate, G1, is now explored. This gate contains no event inputs, but has two gate inputs, G4 and G5, which are processed before returning to consider G3. G4 appears leftmost in the input list to G1, so is considered first. G4 contains the unordered events 'a' and 'g'. Both occur twice in the tree, so are placed in the ordering with 'a' first, as it appears first in the input list:

$$b<c<d<a<g$$

G5 adds the final two events to the ordering: 'f' and 'e'. As 'f' is the most repeated event, it is placed in the ordering before 'e', to give the final ordering as:

$$b<c<d<a<g<f<e$$

All the events have now been ordered, so it is not necessary to consider the remaining gates.

5.2.4 Depth-First, with Number of Leaves

This scheme is again an extension to the modified depth-first ordering, with a different method of choosing the order in which gate inputs are explored. They are chosen according to the number of 'leaves' beneath the gate itself. The number of leaves of a gate is the total number of basic events occurring at any level beneath that gate.

The gate inputs with the least number of leaves that haven't been ordered are considered first. In the case of a tie, the gate with the fewest ordered leaves is chosen. If an order still can't be established, then they are simply ordered as they appear from left to right in the input list. A modification has been made to how events are dealt with - they are now ordered in the same way as in the modified depth-first method. So the most repeated events are chosen first but in the case of a tie, they are ordered as they appear from left to right in the list of inputs. Again, they are considered before any gate inputs.

To demonstrate this technique, it is applied to the fault tree in Figure 5.1. The top event, Top, has no event inputs to order, so its three gate inputs, G1, G2 and G3 are considered in turn. The number of leaves, shown in Table 5.1, determines the order in which they are explored.

Gate name	G1	G2	G3
Number of unordered leaves	7	3	8
Number of ordered leaves	0	0	0

Table 5.1: Number of ordered and unordered leaves of fault tree gates G1, G2 and G3

As G2 has the fewest number of unordered leaves, it is considered first, followed by G1, then G3. G2 contains the events 'b', 'c' and 'd', which gives the partial ordering:

b<c<d

The subtree of the next gate, G1, is now explored. This gate contains no event inputs, but has two gate inputs, G4 and G5, which are processed before returning to consider G3. The number of leaves for each gate are shown in Table 5.2.

Gate name	G4	G5
Number of unordered leaves	2	2
Number of ordered leaves	2	1

Table 5.2: Number of ordered and unordered leaves of fault tree gates G4 and G5

Both gates have the same number of unordered leaves, so the number of ordered leaves is considered, of which G5 has fewer. G5 contains the new events 'f' and 'e', and as 'f' has the greatest number of occurrences, appears first in the ordering:

b<c<d<f<e

G4 contains the new events 'a' and 'g', and as both occur twice in the tree are simply placed in the ordering in their respective positions in the tree:

b<c<d<f<e<a<g

This concludes the ordering, so gate G3 is not examined.

5.2.5 Non-Dynamic Top-Down Weighted Ordering

Weights are calculated for each event according to the following steps:

- A weight of 1.0 is assigned to the top event and is propagated through the fault tree towards the basic events.
- At each gate, the weight is equally distributed between its inputs.
- Each basic event will then have been assigned a weight. Repeated events have their corresponding weights added together.
- The highest order is given to the basic event with the largest weight.

The variables are placed in order of decreasing weight. A modification has been made to how events with equal weights are ordered: they are chosen according to their average level of appearance in the tree. The average level is calculated for each variable by summing the levels on which the event occurs and dividing this by the number of occurrences. The variable that appears, on average, highest in the tree is placed earlier in the ordering. If variables still tie for positions then the most repeated event is chosen and if a tie still exists then they are simply ordered as they appeared in the modified top-down ordering.

Figure 5.2 shows the same fault tree as Figure 5.1, but with the weight assignments:

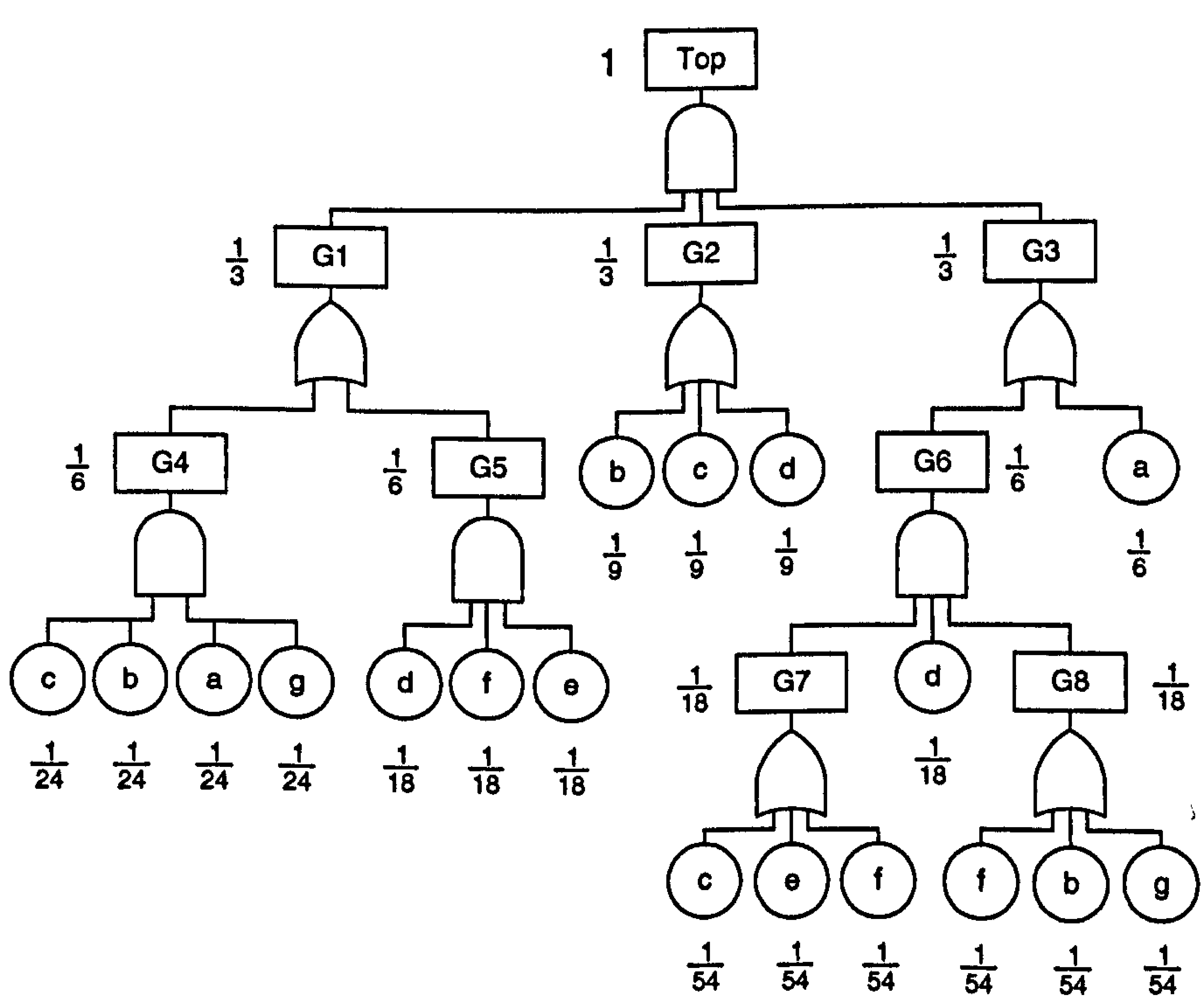


Figure 5.2: Weight assignments for ordering of variables

Weights can be obtained for each variable:

$$a=\frac{1}{6}+\frac{1}{24}=\frac{5}{24}=\frac{45}{216}$$

$$b=\frac{1}{9}+\frac{1}{24}+\frac{1}{54}=\frac{37}{216}$$

$$c=\frac{1}{9}+\frac{1}{24}+\frac{1}{54}=\frac{37}{216}$$

$$d=\frac{1}{9}+\frac{1}{18}+\frac{1}{18}=\frac{2}{9}=\frac{48}{216}$$

$$e=\frac{1}{18}+\frac{1}{54}=\frac{2}{27}=\frac{16}{216}$$

$$f=\frac{1}{18}+\frac{1}{54}+\frac{1}{54}=\frac{5}{54}=\frac{20}{216}$$

$$g=\frac{1}{24}+\frac{1}{54}=\frac{13}{216}$$

The events can now be ordered by decreasing weights. However, events 'b' and 'c' have equal weights, so their average level of occurrence is calculated. This also is found to be equal (both average on level four) and as they both occur the same number of times, they are ordered as in the modified top-down ordering, which was b<c. This gives the non-dynamic top-down weighted ordering:

$$d<a<b<c<f<e<g$$

There are several ways in which events could be ordered should they have equal weights, and some suggestions were made in Chapter 4. It was suggested that the event occurring most could be selected first, as it is the repeated events that cause cut set redundancy. Conversely the event with the lowest number of occurrences could be chosen, as this would mean that the individual events probably occur higher in the tree and therefore have more effect on the structure function. It was decided that calculating the events' average levels of occurrence and choosing the highest would give an improved indicator of which event should be ordered first. So, for example, an event appearing on level two (i.e. as a direct input to the top event) would be chosen before an event that occurs three times on level four. But, an event occurring three times on level four (average level is four) would be ordered before an event appearing once on level three and again on level six (average level is 4.5), even though one occurrence of the second event occurs at a higher level than the first event.

5.2.6 Dynamic Top-Down Weighted Ordering

This ordering progresses in the same way as the non-dynamic version, to calculate the weights for the basic events. However, only the event with the highest weight is placed in the ordering. If two or more events have the same weight, then the event with the highest average level of occurrence is chosen. If they remain indistinguishable, the most repeated event is chosen and if a tie still exists then the event appearing first in the modified top-down ordering is chosen. Once an event has been placed in the ordering, it is removed from the fault tree by deleting all its occurrences. Using the modified fault tree, weights are reassigned from the beginning. This allows another event to be ordered and the process continues until all events have been placed in the ordering. As explained in Chapter 4, applying the dynamic ordering method means that in many cases neighbouring events are given near orders.

Applying this scheme to the example fault tree gives the first set of weights as in the non-dynamic ordering. This means that event 'd' is the first to be placed in the ordering. However, 'd' is now removed from the fault tree to give the modified tree shown in Figure 5.3.

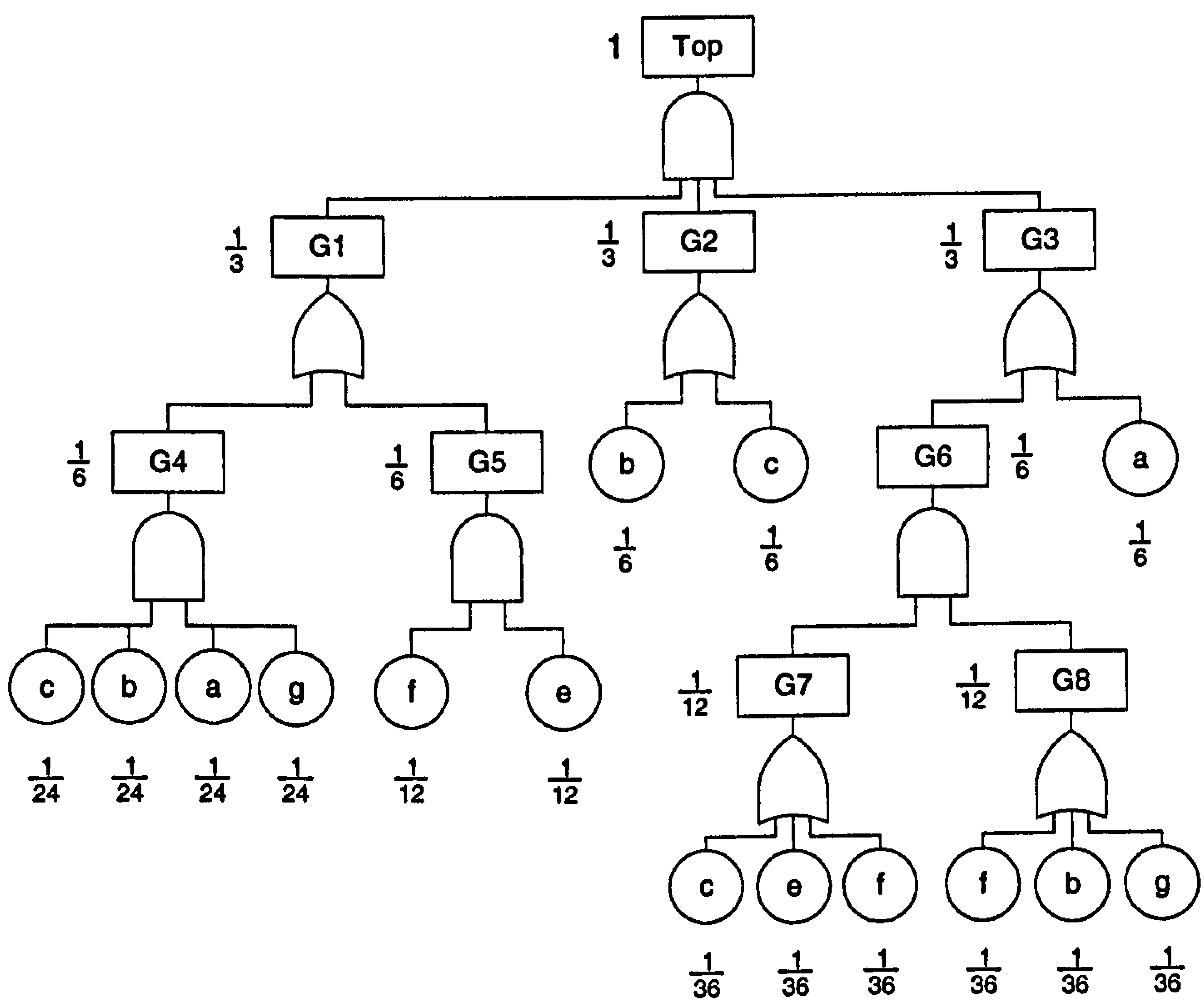


Figure 5.3: Modified fault tree with event 'd' removed

The new weight assignments are:

$$a = \frac{1}{6} + \frac{1}{24} = \frac{5}{24} = \frac{15}{72}$$

$$b = \frac{1}{6} + \frac{1}{24} + \frac{1}{36} = \frac{17}{72}$$

$$c = \frac{1}{6} + \frac{1}{24} + \frac{1}{36} = \frac{17}{72}$$

$$e = \frac{1}{12} + \frac{1}{36} = \frac{1}{9} = \frac{8}{72}$$

$$f = \frac{1}{12} + \frac{1}{36} + \frac{1}{36} = \frac{5}{36} = \frac{10}{72}$$

$$g = \frac{1}{24} + \frac{1}{36} = \frac{5}{72}$$

Events 'b' and 'c' have the largest weight values, and the same average level of occurrence and number of repetitions, so 'b' is chosen as it appears first in the modified top-down ordering.

Continuing in the same manner, event 'b' is removed from the tree and further weights are assigned. This process is repeated until all events have been placed in the ordering. The final dynamic top-down weighted ordering is:

$$d<b<c<a<g<f<e$$

5.2.7 Bottom-Up Weighted Ordering

This technique is initiated from the bottom of the tree, rather than the top and in effect calculates weights for the gates, which are then used to determine the ordering in which they are considered within a depth-first exploration. The way in which this scheme is implemented differs significantly from the method described in Chapter 4. The main features are described below:

- A weight of ½ is assigned to each basic event and propagated towards the top event.
- At each gate, the weights of the inputs are combined as probabilities according to:

$$\text{'AND' gates: } P(\text{gate}) = \prod_{i=1}^{n} q_i \qquad 5.1$$

$$\text{'OR' gates: } P(\text{gate}) = 1 - \prod_{i=1}^{n} (1 - q_i) \qquad 5.2$$

  where n is the number of inputs to the gate.
- Once each of the inputs to the top event has been assigned weights, the tree is explored in a depth-first manner, considering branches with the largest weight first.

Once the weight values of the gates have been established, the method proceeds as in the modified depth-first method, except that the gates are explored according to which has the highest weight rather than simply from left to right. However, if gates do have the same weight then they are considered according to the percentage of repeated events below that gate. This is calculated by adding up the number of repeated events below the gate and dividing by the total number of events below the gate. The gate with the highest number of repeated events is considered first, but if there is a tie, then they are considered from left to right as they appear in the input list. The events of each gate are ordered before the gate inputs are explored and are chosen according to the highest number of occurrences in the fault tree. If events have the same number of occurrences then they are simply chosen from left to right as they appear in the input list.

This scheme can now be applied to the tree in Figure 5.1. Every event is given a weight of ½ and so the weights of the gates can be calculated as in Table 5.3.

Gate name	Gate type	Inputs	Calculated gate weight
G1	OR	G4, G5	23/128
G2	OR	b, c, d	7/8
G3	OR	G6, a	17/32
G4	AND	c, b, a, g	1/16
G5	AND	d, f, e	1/8
G6	AND	G7, G8, d	49/128
G7	OR	c, e, f	7/8
G8	OR	f, b, g	7/8

Table 5.3: Weights of the gates according to the bottom-up weighted method

The top event has three gate inputs: G1, G2 and G3. These are considered in order of highest weight according to Table 5.3, i.e. G2 then G3 then G1. G2 has three event inputs, which gives the partial ordering:

$$b<c<d$$

The subtree of the next gate, G3 is now explored. It contains one event input, 'a', which is added to the partial ordering to give:

$$b<c<d<a$$

Its gate input G6 is then examined. It has one event input, 'd', but as this is already in the ordering, it is not considered. G6 has two gate inputs, G7 and G8, which have equal weights. As they also have the same percentage of repeated events below (both 100%) they are simply considered from left to right, i.e. G7 first. G7 contains two unordered events, 'e' and 'f'. Event 'f' is placed first in the ordering as it has more occurrences in the tree, giving:

$$b<c<d<a<f<e$$

G8 adds the final event 'g' to the ordering to give:

$$b<c<d<a<f<e<g$$

The subtree of the gate G1 is not explored, as all the events have been ordered.

This method differs significantly from the general method detailed in Chapter 4. The general method assigned each event a weight value of one and added the weights up at each gate. However this simply orders the gates according to the number of basic events in its subtree and would not distinguish between 'OR' gates with many inputs and 'AND' gates with many inputs. In this case it would be fair to assume that the events beneath the 'OR' gate would have more influence over the occurrence of the top event as only one is needed for the logic to flow, compared with the 'AND' gate where every event would need to occur. For this reason the events were given weights of ½ and the weights were propagated as probabilities, so keeping the weight values of the gates below one, and giving 'OR' gates higher precedence. If gates have equal probabilities, the order is chosen according to the percentage of repeated events below that gate. This is because repeated events cause the problem of non-minimal cut sets and so by ordering repeated events first the resulting BDD can be smaller. Tied events are dealt with in the same way as in the other depth-first schemes as this has been shown to be a good ordering technique.

5.2.8 Event Criticality

This final ordering scheme is an extension of the one reported in Chapter 4, which applies the principle of Birnbaum's structural importance measure directly to the tree. The contribution of each basic event to the top event is calculated according to:

$$I_i = Q(1_i, \underline{\tfrac{1}{2}}) - Q(0_i, \underline{\tfrac{1}{2}}) \qquad 5.3$$

The selected basic event therefore assumes the failure probabilities of one and zero on two consecutive computations of the top event probability, with the remaining components given failure probabilities of ½. The result of the second run (with failure probability zero) is subtracted from the first run (with failure probability one) to give the contribution of that basic event to the occurrence of the top event

The basic events are ordered such that those with a greater contribution to the occurrence of the top event are ordered before those with smaller contributions. If two events have the same calculated contribution, then the event with the highest average level of occurrence is selected first. If the events are still tied then the most repeated event is selected and if the events are still indistinguishable, then they are simply ordered as they appear in the modified top-down ordering.

This scheme can be applied to the fault tree in Figure 5.1 to give the calculated contributions shown in Table 5.4:

Event	Probability of Top with event failure probability 1	Probability of Top with event failure probability 0	Contribution to the top event
a	0.2051	0.0419	0.1632
b	0.1685	0.0623	0.1062
c	0.1685	0.0623	0.1062
d	0.2621	0.0234	0.2387
e	0.1867	0.0363	0.1504
f	0.1948	0.0350	0.1598
g	0.1474	0.0726	0.0748

Table 5.4: The calculated contributions of each of the basic events to system failure

The events are ranked such that those with larger contributions appear earlier in the ordering than events with smaller contributions. Events 'b' and 'c' have the same contribution, the same average level of occurrence and the same number of occurrences. Therefore, they are simply ordered as they appear in the modified top-down ordering scheme:

d<a<f<e<b<c<g

5.3 Performance of the Schemes on a Set of Fault Trees

A program was written to implement the eight ordering schemes (ordering.c), which were applied to a set of 228 fault trees. Summary details of these trees can be found in Appendix II. BDDs were constructed for each tree, using the variable orderings determined by each of the schemes.

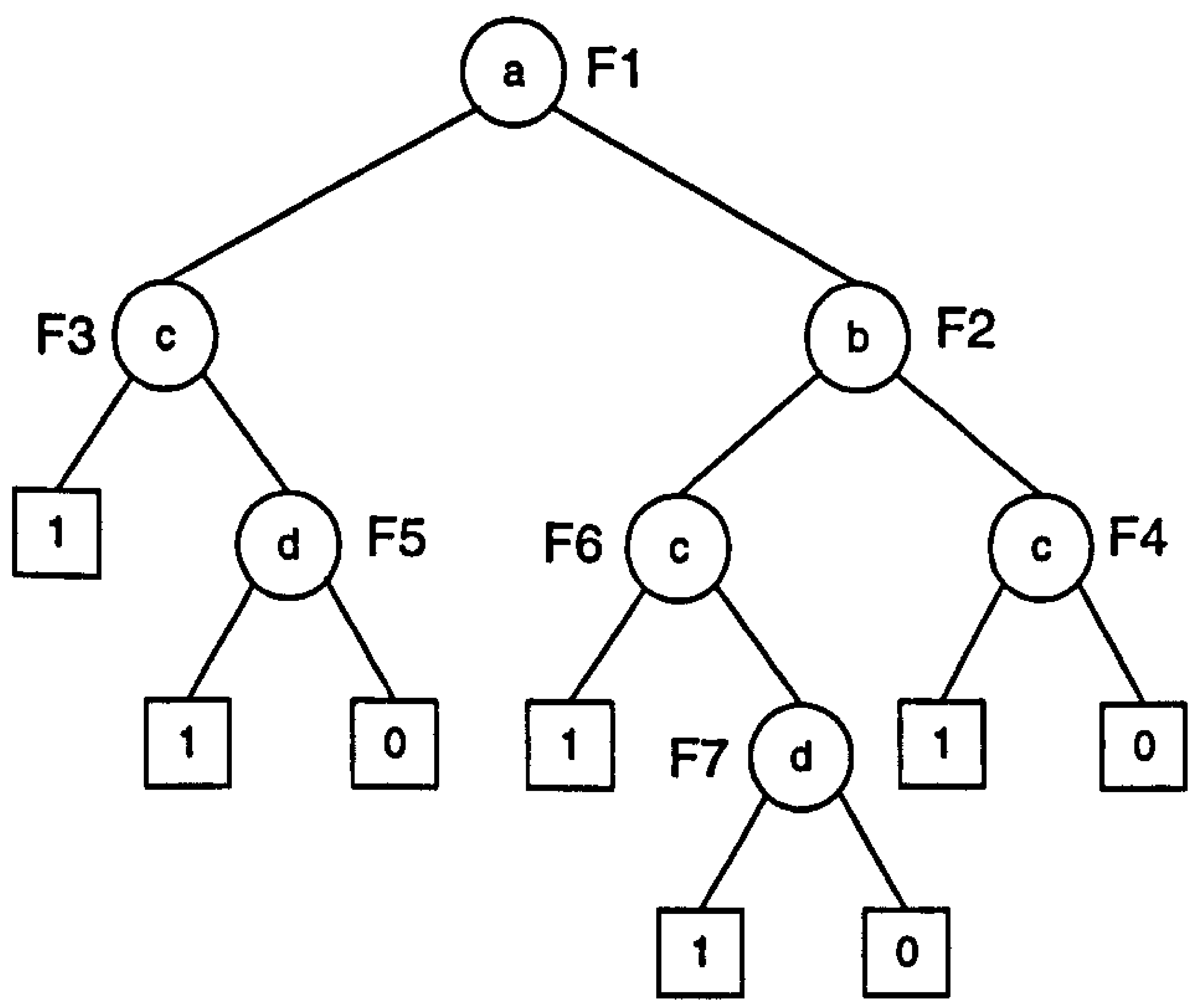
The schemes are ranked in order according to the complexity of the BDD that they produce. The performance of the schemes is then assessed in two ways. Firstly, the number of times that each scheme produces the highest ranking is calculated. This is the usual method of scheme evaluation. The second method considers the average ranking of each scheme across the set of fault trees, so gives an indication of the overall scheme performance. Three different measures of BDD complexity are considered, which are discussed in the following sections.

5.3.1 Measures of BDD Complexity

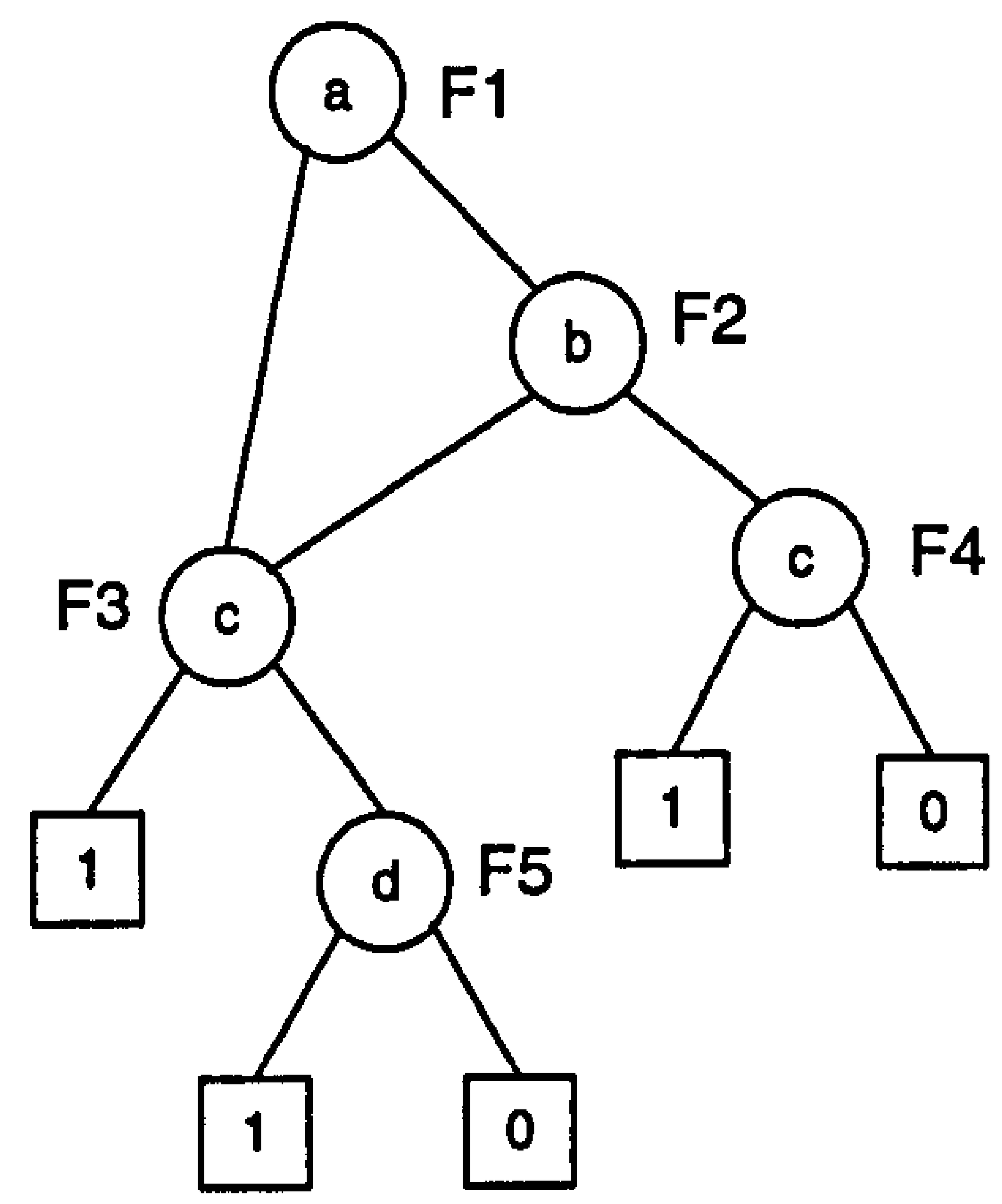
In order to fully compare the ordering schemes, three measures of BDD complexity are used in the investigation. These are the number of non-distinct nodes in the BDD, the number of distinct nodes in the BDD and the number of **ite** calculations required to construct the BDD. Each measure and the reason for employing it, is described in the following sections.

5.3.1.1 Non-Distinct Nodes

The number of non-distinct nodes in the BDD is essentially the size of the BDD when sub-node sharing is not enabled. Therefore if a section of the BDD is repeated, the nodes within this section will be counted as many times as that section appears. For example, nodes F3 and F6 in Figure 5.4(a) are identical, so sub-node sharing can be enabled, as in Figure 5.4(b). The number of non-distinct nodes is therefore measured from 5.4(a), giving a total of seven nodes in this case. This is a particularly useful measure when considering quantitative analysis, as it gives an indication of the number of calculations to be performed.



(a) BDD not enabling sub-node sharing



(b) BDD enabling sub-node sharing

Figure 5.4: Identical BDDs, showing the use of sub-node sharing

Previous results obtained by Sinnamon[21] (comparing schemes 1, 2 and 3) and Bartlett[19] (comparing schemes 1, 2, 3 and 8) have both used this measure as a comparison of BDD size.

5.3.1.2 Distinct Nodes

The number of distinct nodes in the BDD is the total number of non-terminal nodes when sub-node sharing is enabled and is shown in Figure 5.4(b). In this example, the BDD has five distinct nodes. This measure indicates the size of the most compact representation of the BDD and is also the number of nodes that has to be stored in computer memory. This is an important consideration, as it is the very large BDDs that cannot be handled. It is also what is generally referred to as 'the size of the BDD'.

5.3.1.3 Number of If-Then-Else Calculations

This measure represents the number of computations that must be performed and stored during the **ite** procedure (each one is stored with its result so that it is not repeated unnecessarily), and so indicates the size of the arrays that have to be handled. If the computer memory is exceeded and the construction process fails, then the BDD technique cannot be utilised. Therefore reducing this method of BDD complexity is particularly beneficial.

5.3.2 Results: Highest Scheme Rankings

The schemes are ranked in order three times, according to the number of non-distinct BDD nodes, the number of distinct BDD nodes and the number of **ite** calculations required for BDD construction. A count is then made of the number of times that each scheme receives the highest ranking. The numbers of fault trees for which each scheme is 'best', do not add up to the total number of trees, as some have the same result for more than one scheme. The results are not included for the trees that give identical values for each ordering scheme.

The results for each tree, showing the number of non-distinct BDD nodes, distinct BDD nodes and **ite** calculations required to construct the BDD using each ordering, can be found in Appendices III, IV and V.

5.3.2.1 Non-Distinct Nodes

The eight schemes gave identical results for 59 of the 228 fault trees. For the remaining 169 trees, the results are shown in Table 5.5.

Ordering scheme	1	2	3	4	5	6	7	8
Number of trees using non-distinct nodes	18	43	34	37	34	35	46	68

Table 5.5: The number of trees for which each scheme was ranked the highest according to the number of non-distinct BDD nodes

These results clearly show that the event criticality ordering scheme (8), which is a weighted measure, performs significantly better than any other ordering scheme. It produces BDDs with the fewest non-distinct nodes in 68 cases, which is for 22 more trees than the next best scheme, the bottom-up weighted measure (scheme 7).

The modified top-down scheme (1) produced disappointing results, generating BDDs with the fewest non-distinct nodes in only 18 cases. This is substantially worse than for any other scheme – the closest result was obtained by schemes 3 and 5 (the modified priority depth-first and non-dynamic top-down weighted schemes respectively), which were both ranked highest for 34 fault trees.

Although some schemes performed better than others, each has at least one fault tree (and in many cases, several trees) for which it produces a result that cannot be matched by any other scheme. Therefore none can be disregarded at this stage.

5.3.2.2 Distinct Nodes

The eight schemes produced identical results for 64 of the 228 fault trees. The results for the remaining 164 trees are shown in Table 5.6.

Ordering scheme	1	2	3	4	5	6	7	8
Number of trees using distinct nodes	11	64	54	59	25	39	68	39

Table 5.6: The number of trees for which each scheme was ranked the highest according to the number of distinct BDD nodes

These results are significantly different to those obtained for the number of non-distinct BDD nodes. For example in the previous case, the event criticality scheme (8) performed the best. However here it does not perform particularly well at all. One aspect that is mirrored by these results is that again, the modified top-down scheme (1) produces the worst results by a considerable margin.

It is interesting to note that the schemes based on a depth-first approach (i.e. the modified depth-first (2), modified priority depth-first (3), leaves depth-first (4) and the bottom-up weighted measure (7)) perform significantly better than the other schemes. This suggests that in order to draw the BDD in a concise manner, a depth-first approach should be considered. As this was not apparent in the results for the number of non-distinct BDD nodes, it could be that the use of a depth-first method somehow promotes the use of sub-node sharing.

The bottom-up weighted approach (7) performs marginally better than the remaining schemes, so as with the previous section, a weighted technique produces the best results. It could be the combination of a weighted scheme incorporating a depth-first approach that makes this scheme successful.

5.3.2.3 Number of If-Then-Else Calculations

The eight schemes produced identical results for 41 of the 228 fault trees. The results for the remaining 187 trees are shown in Table 5.7.

Ordering scheme	1	2	3	4	5	6	7	8
Number of trees using **ite** calculations	24	52	42	47	33	57	45	58

Table 5.7: The number of trees for which each scheme was ranked the highest according to the number of **ite** calculations

There are similarities between these results and those obtained for the number of non-distinct BDD nodes, in that the modified top-down scheme (1) is ranked highest for the fewest number of trees and the event criticality scheme (8) is ranked highest for the greatest number of trees. However, the results are more evenly spread than for either of the other BDD complexity measures, with a difference of only 34 trees between the best and worst performances, compared with 57 for the number of distinct BDD nodes.

5.3.3 Results: Overall Ranking of the Schemes

This method of evaluating the ordering schemes ranks them in order from the scheme that produces the best results (i.e. the smallest number of nodes or the fewest ite calculations), to the scheme that produces the worst results for each fault tree, where a ranking of one indicates the best performance and a ranking of eight indicates the worst performance. The rankings are then added together over all 228 trees, to show which scheme performs well over all the trees, but does not necessarily perform 'best' each time. This is indicated by the scheme with the lowest added ranking. If a scheme consistently comes second or third, then this could prove a better choice of scheme than one which might perform erratically, producing the highest ranking a number of times, but performing badly on other trees. The results are not included for the trees that give identical values for each ordering scheme.

5.3.3.1 Non-Distinct Nodes

The rankings were added over the 169 trees to give the results shown in Table 5.8.

Ordering scheme	1	2	3	4	5	6	7	8
Added rankings for non-distinct nodes	787	703	708	676	621	638	740	503

Table 5.8: The added rankings for each ordering scheme for 169 fault trees

The event criticality scheme (8) produces the best results, as it did for the number of times it produced the BDD with the fewest number of non-distinct nodes. So in addition to being ranked first for 68 trees, it also performs well over the remaining trees.

Other than the modified top-down approach (1), which again performs badly, most of the remaining schemes produce average results. A significant difference with this measure of scheme performance however, is that the bottom-up weighted measure (7) now produces a result similar to that obtained using the modified top-down method (1), whereas for the number of times it received the highest ranking, it returned the second-best results. This could be because although it is ranked first on 46 occasions, it frequently produced a BDD size that was much larger than that obtained using other schemes.

5.3.3.2 Distinct Nodes

The rankings were added over the 164 trees to give the results shown in Table 5.9.

Ordering scheme	1	2	3	4	5	6	7	8
Added rankings for distinct nodes	887	541	534	570	743	588	625	674

Table 5.9: The added rankings for each ordering scheme for 164 fault trees

These results are similar to those obtained for the number of times each scheme received the highest ranking, with the depth-first measures again performing particularly well. However in this case it is the modified priority depth-first (3) measure rather than the bottom-up weighted measure (7) that produces the best results, with the lowest overall ranking.

As with the results obtained for the non-distinct nodes, the bottom-up weighted measure (7) has not performed as well in the overall rankings as it did when considering the number of times it was ranked highest. It seems that while it produces the best ordering on 68 occasions, it does not maintain a good overall performance on the remaining trees.

If one scheme were to be selected for use when considering the number of distinct nodes, then an appropriate choice would be the modified depth first scheme (2), which performed well in both sets of results. When considering the number of trees for which it produced the smallest BDD, it came a close second place with 64 compared with 68 for the bottom-up weighted measure (7). In this set of results, it had the second lowest ranking (a total of 541 compared with 534 obtained with the modified priority depth-first scheme (3)), which gives it the best overall results and suggests it is a good choice of scheme.

5.3.3.3 Number of If-Then-Else Calculations

The rankings for each scheme were added for the 187 fault trees to give the results shown in Table 5.10.

Ordering scheme	1	2	3	4	5	6	7	8
Added rankings for ite calculations	896	739	727	728	742	606	825	680

Table 5.10: The added rankings for each ordering scheme for 187 fault trees

The results are similar to those obtained for the number of times that each scheme produced a BDD with the fewest **ite** calculations. The dynamic top-down weighted scheme (6) has performed particularly well and although it didn't produce the best results in the previous section, it did come a very close second, using the fewest number of **ite** calculations for 57 trees compared with 58 trees for the event criticality scheme (8). This scheme is therefore very successful at ordering the variables in a manner which minimises the number of **ite** calculations necessary to construct the BDD.

5.4 Conclusions

Previous research has suggested that no scheme will be identified that is capable of producing the smallest possible BDD for any given fault tree, and these results appear to support this theory. It is interesting to see however, that even within a particular fault tree, different schemes work best depending on the measure used to assess the BDD complexity.

For example, Table 5.11 shows the number of distinct BDD nodes, non-distinct BDD nodes and **ite** calculations required to produce the BDDs for the fault tree 'rand155'. Scheme 8 produces the best results for the number of non-distinct nodes, scheme 4 produces the best results for the number of distinct nodes, whilst scheme 6 is best when considering the number of **ite** calculations required to obtain the BDD. Not only does this show that the 'best' choice of scheme can be different for a fault tree according to how BDD complexity is measured, but in this case the schemes that perform well for one measure of BDD complexity do not even produce results that are near to the best for other measures.

Ordering scheme	1	2	3	4	5	6	7	8
Number of non-distinct nodes	1051	1695	1888	1439	894	863	2484	**790**
Number of distinct nodes	226	146	115	**97**	158	106	172	174
Number of **ite** calculations	314	307	297	213	238	**196**	448	250

Table 5.11: The number of non-distinct and distinct nodes for the BDDs obtained by each of the orderings from fault tree 'rand155'

In order to produce the smallest number of non-distinct nodes, the event criticality scheme (8) appears to be the best choice. It produced the smallest BDDs on the greatest number of occasions, but also showed that it performs consistently well by producing the best overall ranking. As there are several ways of distinguishing 'tied' variables within this scheme, it is thought that further work could lead to improved results.

For the smallest number of distinct nodes, the schemes based upon a depth-first approach seemed to provide the best orderings. In particular the bottom-up weighted approach (7), which is a weighted method, produced encouraging results when considering the number of times it was ranked highest and again it is thought that this scheme would benefit from further refinement. As it has already been noted however, the modified depth-first method (2) produced excellent results in both categories and would provide a good choice of scheme when considering distinct nodes.

Two schemes performed particularly well when considering the number of **ite** calculations required to construct the BDD – the dynamic top-down weighted method (6) and the event criticality scheme (8), both of which are weighted measures. It is thought that the dynamic top-down scheme would particularly benefit from further investigation, as it is the first time that results have been obtained using this ordering technique.

The variable orderings produced by each of the schemes are very sensitive to the way in which the fault tree is written. The structure of the tree can vary considerably whilst still satisfying the same logic function and is very rarely written in its most concise form. As well as affecting the variable ordering, this can have a significant effect on the size of the resulting BDD. However, methods can be applied to fault trees to reduce their complexity, with the aim of constructing smaller BDDs. One such technique is considered in the following chapter.

Chapter 6: Fault Tree Reduction

6.1 Introduction

Fault trees are rarely written in their most concise format and this can have a significant effect on the size of the resulting BDDs. Their complexity can be reduced however, by applying fault tree reduction techniques, which optimise the structure of the tree, whilst retaining the underlying logic. This chapter discusses how one such technique, known as the 'Faunet[26]' method, can be used to restructure fault tees to give an equivalent, but simpler, representation of the logic function. The reduced fault trees can then be used within the BDD method, with the aim of producing smaller BDDs than can be obtained using the original (non-reduced) fault trees.

The following sections consider the Faunet reduction technique in detail and discuss the program that was written for its implementation. The performance of the reduction method is then evaluated by comparing the complexity of BDDs constructed from a set of reduced fault trees against those obtained from the original, non-reduced, fault trees.

6.2 The Faunet Reduction Technique

This method of fault tree reduction consists of three stages:

1. **Contraction**

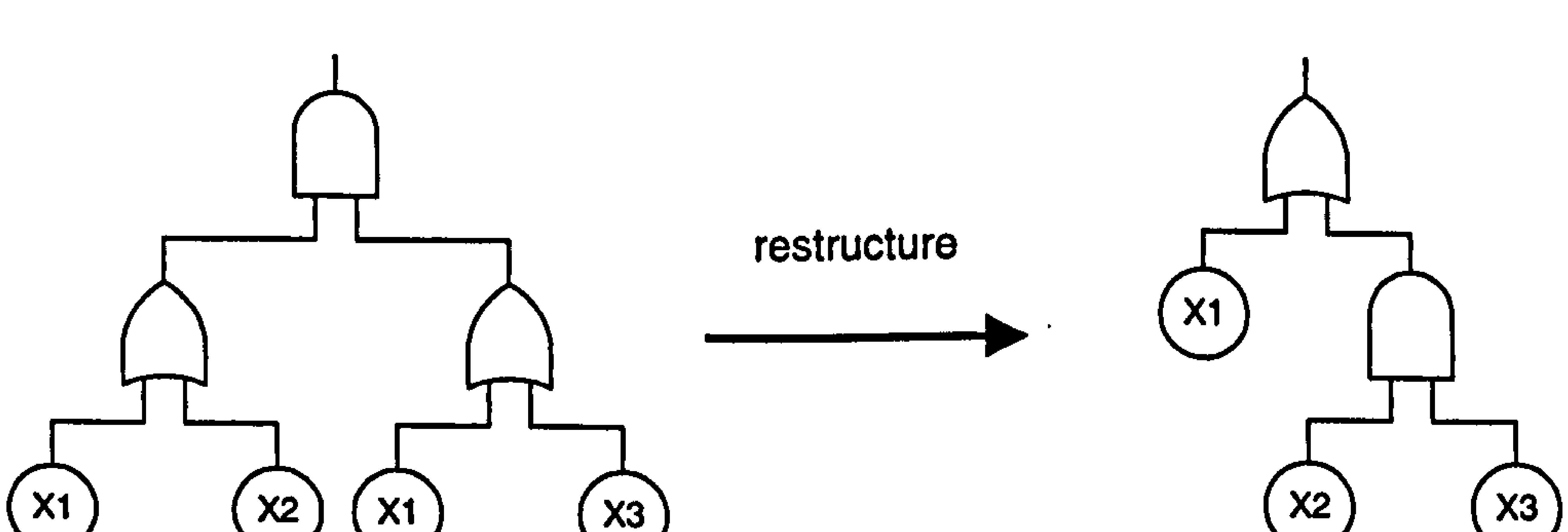
   Subsequent gates of the same type are contracted to form a single gate. This gives an alternating sequence of 'AND' gates and 'OR' gates throughout the tree.

2. **Factorisation**

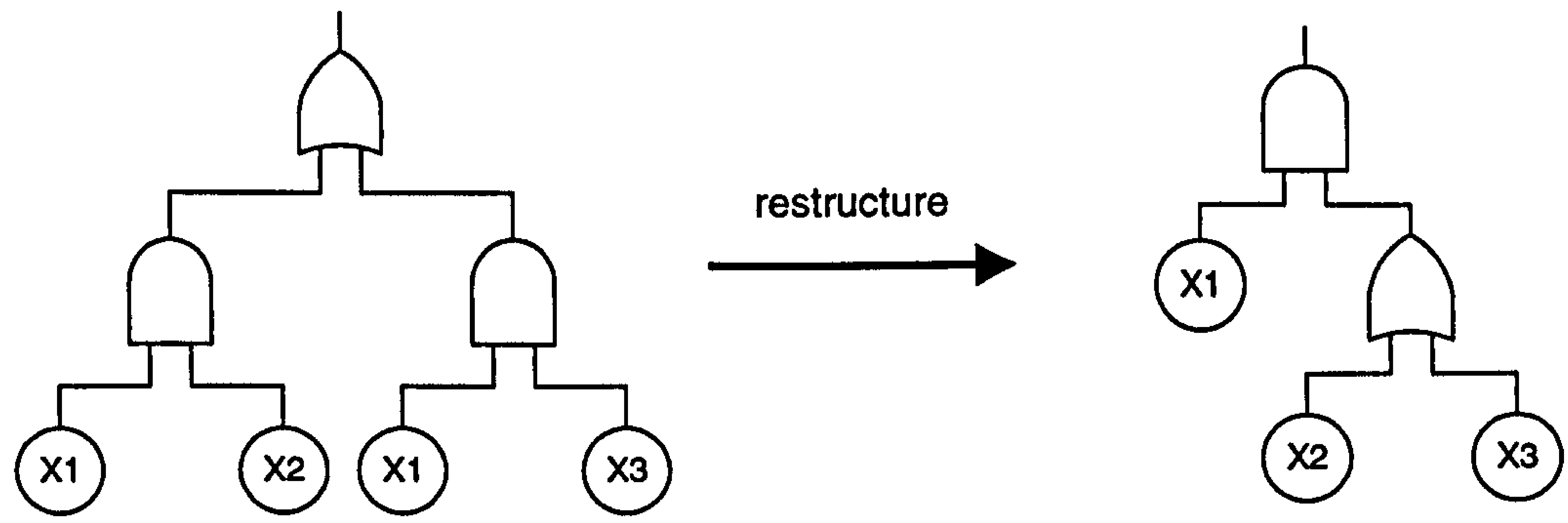
   Pairs of events that always occur together in the *same gate type* are identified. They are combined to form a single complex event.

3. **Extraction**

   The following two structures are identified and replaced:



(a)



(b)

Figure 6.1: The extraction procedure

The above three steps are repeated until no further changes are possible in the fault tree, resulting in a more compact representation of the system.

6.3 Worked Example of the Reduction Technique

In order to demonstrate the reduction process and explain its implementation in the program 'faunet.c', the technique will be applied to the example fault tree shown in Figure 6.2.

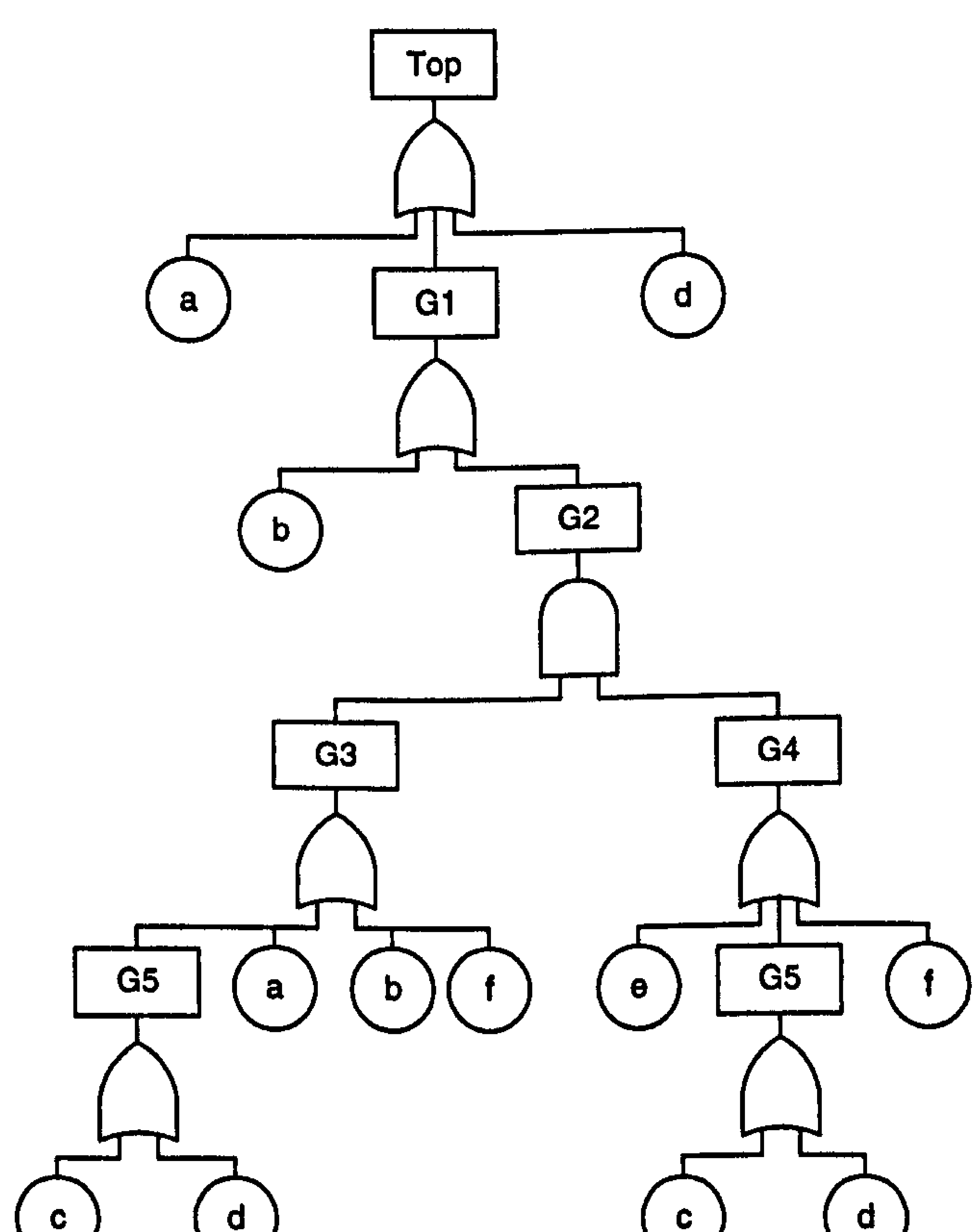


Figure 6.2: Example fault tree

6.3.1 Inputting Fault Tree Data to the Program

The initial step, when given a fault tree such as the one shown in Figure 6.2, is to represent it by a data file. Each gate that appears in the tree is listed in the file, along with its type, the number of inputs (gates and events are numbered separately) and the inputs themselves. A typical file format for the fault tree in Figure 6.2 is shown in Table 6.1.

Gate name	Gate type	Number of gates	Number of events	Inputs		
Top	OR	1	2	G1	a	d
G1	OR	1	1	G2	b	
G2	AND	2	0	G3	G4	
G3	OR	1	3	G5	a	b	f
G4	OR	1	2	G5	e	f
G5	OR	0	2	c	d	

Table 6.1: Fault tree data for Figure 6.2

The data is read into the program with each column of Table 6.1 forming an array. As the data is read into these five arrays, it is also converted to a numerical format and stored in five corresponding arrays. The numerical arrays are used throughout the program, for ease of data manipulation.

- Basic events are numbered from 1 to 999.
- Gates are numbered from 1000 to 1999.
- Complex events are numbered from 2000 upwards.

The numerical arrays for the data in Table 6.1 are shown in Table 6.2.

Gate number	Value of gate 1-OR, 2 - AND	Number of gates	Number of events	Inputs			
1000	1	1	2	1001	1	2	
1001	1	1	1	1002	3		
1002	2	2	0	1003	1004		
1003	1	1	3	1005	1	3	4
1004	1	1	2	1005	5	4	
1005	1	0	2	6	2		

Table 6.2: Numerical fault tree data for Figure 6.2

Other arrays that are created as the data are read in, are the occurrence arrays. These store the number of occurrences in the fault tree data of both gates and basic events. The occurrences of the complex events are also recorded as they are formed. In order for the occurrence arrays to be correct, it is essential that the top event be listed first in the data file. Since the gate representing the top event is the only gate that does not appear as an input to another gate (i.e. it appears in the first but not the fifth column of Table 6.2), it is easily identified. The program scans the data until it identifies the gate with this property, and if this gate doesn't appear on the first line of data, then the data is re-arranged to make this the case.

It is also essential that any gate inputs be listed before events in the inputs list, as it is assumed in the program that the inputs occupying positions 0 to (number of gates - 1) are all gates. This was not the case for some of the data for the test trees and so a piece of code was written that rearranges the data into the required format. The order in which the gates appear in the listing is retained; the first input that is found to be a gate is placed in position 1, the second in position 2 and so on.

6.3.2 The Reduction Process

Once the data has been read in, the reduction process can begin. Figure 6.3 shows the numerical fault tree and Table 6.2 its corresponding data at the start of this process.

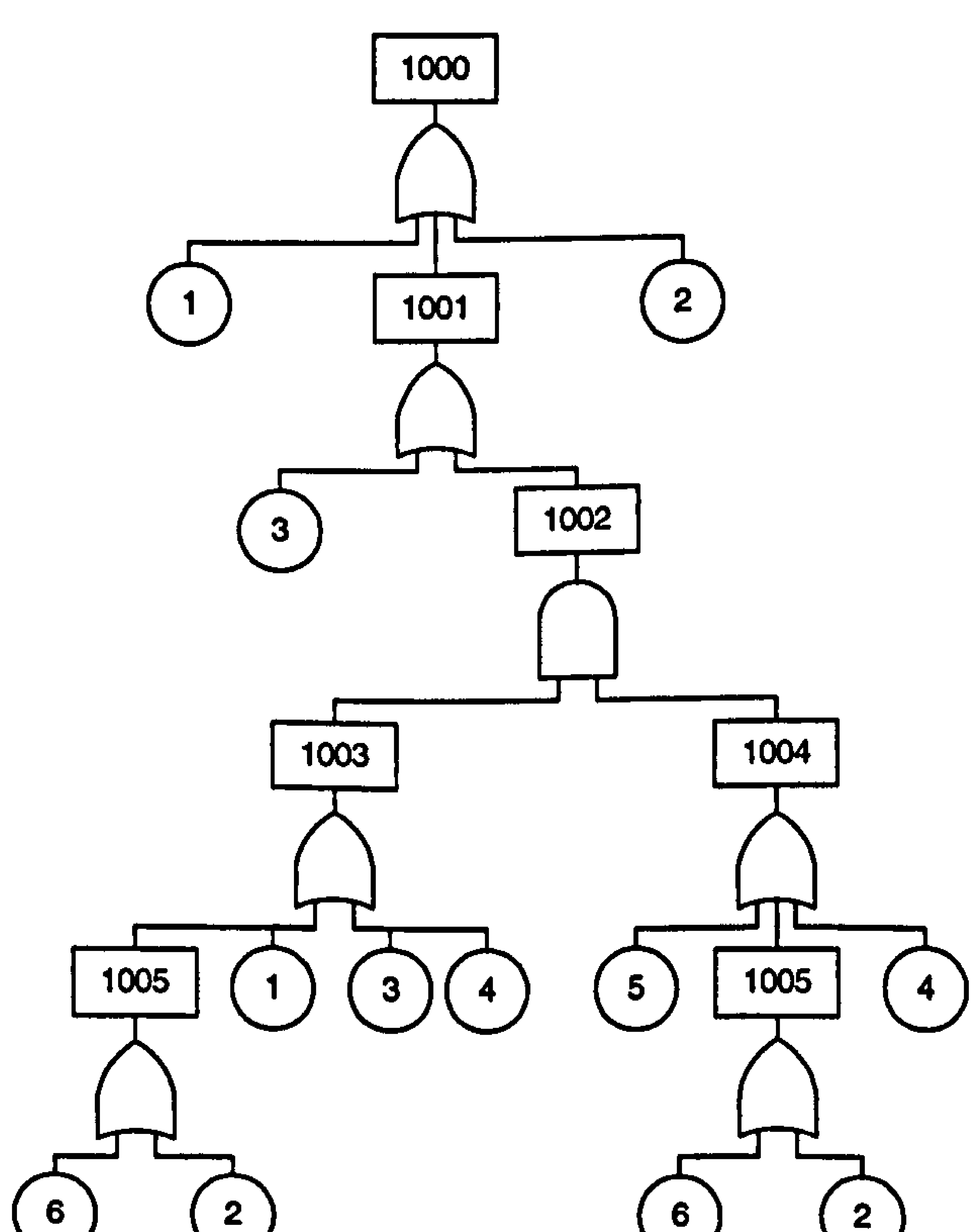


Figure 6.3: The numerical fault tree

Contraction 1

The aim of this first stage is to identify subsequent gates in the tree structure that have the same gate type. In order to do this, the program scans through the inputs to each gate (subsequently referred to as the primary gate) and checks the gate type of each gate input (called the secondary gate). If secondary gate type matches the primary gate type, then the secondary gate can be contracted.

However, before contraction takes place, the number of occurrences of the secondary gate must be checked. If it occurs more than once, then any additional gates to which the secondary gate is an input must also be considered. If any additional gates are of the same type, then contraction can also occur for those cases (resulting in more than one primary gate), however contraction cannot take place for gates that are of a different type.

The process of contraction adds the inputs of the secondary gate to those of the primary gate and deletes the secondary gate from the primary gate's input list. The occurrence arrays containing the number of gates and events are altered accordingly. If there is only one occurrence of the secondary gate, then its line of data can be deleted. However, if it occurs more than once, then its data is only deleted if all the gates to which it is an input are of the same type. If they are not, then the data cannot be deleted as it the gate still occurs as an input elsewhere in the tree and its data is therefore required.

Once contraction has taken place, the inputs to the gates are checked to ensure that each input is listed only once. This is necessary, as the secondary gate could have had an input in common with a primary gate, which would now be listed twice as an input to the primary gate, and would impede the factorisation process.

Application of the contraction stage to the fault tree shown in Figure 6.3:

In this example, gate 1001 appears as an input to gate 1000 and they both have a gate value of 1 (i.e. they are 'OR' gates). Gate 1001 only appears once in the fault tree data, so its inputs are directed to gate 1000 and its line of data deleted. Gate 1005 is also found to be of the same type as gate 1003. However it appears twice in the tree, so its other occurrences must be checked. As it appears as an input to gate 1004, which is also of the same type, it can be contracted in both cases and the line of data can again be deleted. The resulting fault tree and data arrays are shown in Figure 6.4 and Table 6.3.

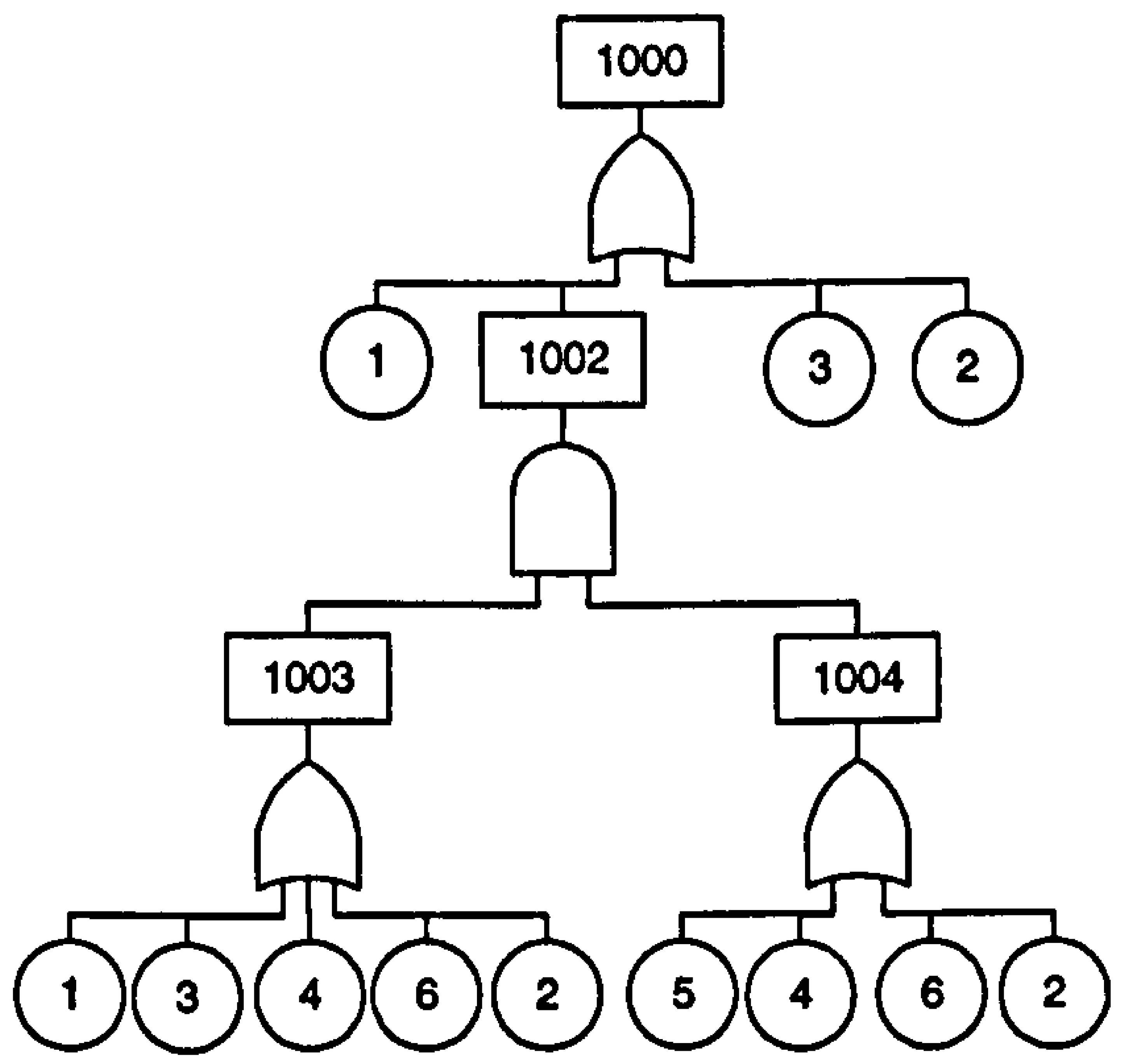


Figure 6.4: The fault tree after contraction 1

Gate number	Gate value	Number of gates	Number of events	Inputs
1000	1	1	3	1002 1 3 2
1002	2	2	0	1003 1004
1003	1	0	5	1 3 4 6 2
1004	1	0	4	5 4 6 2

Table 6.3: Fault tree data after contraction 1

Factorisation 1

The fault tree now has an alternating sequence of 'AND' and 'OR' gates, and can be factorised. The input events to each gate are scanned, looking for pairs that always occur together. This is achieved by systematically examining each possible event pair within the list of inputs. When two events are chosen, the number of occurrences of each is found. If they do not occur the same number of times then the search ends, as this means they cannot always occur together. If they do occur the same number of times, then factorisation can be considered, but each occurrence of the events is checked to ensure that if one event appears as the input to a gate (which must be the same type as the original gate) then the other event is also an input. If each event occurs only once, then they must always occur together, so can immediately be combined.

Once it has been established that they do always occur together and under the same gate type, the events are combined to form a single, complex event. The complex events are numbered from 2000 upwards. The next available number is selected and this is recorded in the complex event array, with the gate type and the two events from which it was formed. The complex event is then substituted into the input array for every occurrence of the first event; occurrences of the second event are deleted. The number of event inputs for the corresponding gates decreases by one.

Application of factorisation to the fault tree shown in Figure 6.4:

Starting with gate 1000, events 1 and 3 are examined. They occur together twice under the same gate type, so can be factorised. Complex event 2000 is created and replaces event 1 in lines 1 and 3 of the input array. Event 3 is deleted. The number of events in both lines of data decreases by one. Events 2000 and 2 are then examined. Event 2000 occurs twice and event 2 occurs three times, therefore they cannot always occur together and are not considered further.

Gate 1002 is now considered, but as it only contains gate inputs, gate 1003 on the next line of data is examined. Events 2000 and 4, then 2000 and 6 are considered, but although they have the same number of occurrences, they do not always occur together. Events 2000 and 2 are again examined, but do not have the same number of occurrences. Events 4 and 6 are considered next, and it is found that they occur together twice under the same gate type. They are therefore combined to form the complex event 2001. Events 2001 and 2 form the next pair, but do not occur the same number of times, so are not considered further. The final gate 1004 is then scanned, but no events can be factorised.

The modified fault tree and data are shown in Figure 6.5 and Table 6.4.

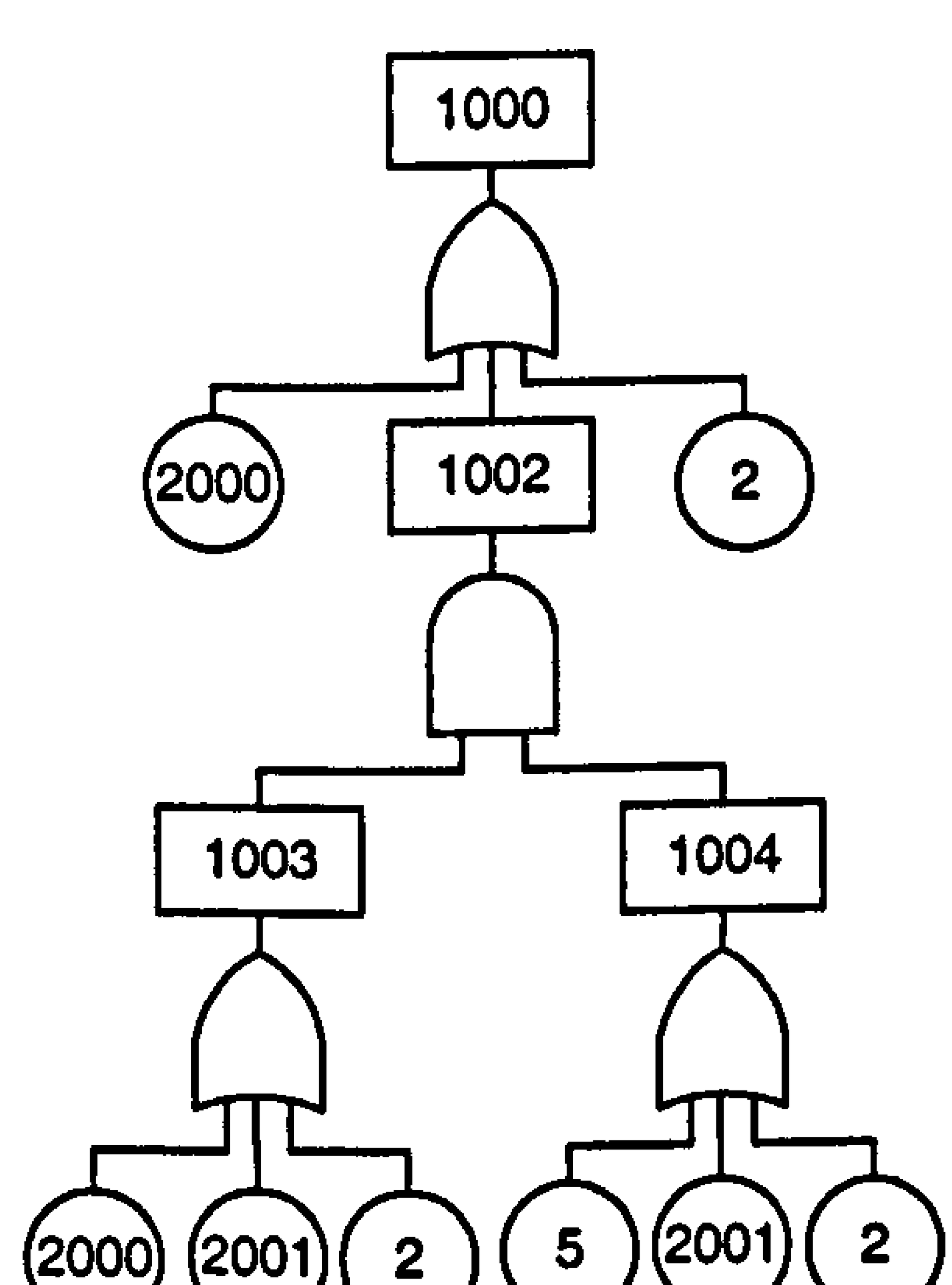


Figure 6.5: The fault tree after factorisation 1

Gate number	Gate value	Number of gates	Number of events	Inputs		
1000	1	1	2	1002	2000	2
1002	2	2	0	1003	1004	
1003	1	0	3	2000	2001	2
1004	1	0	3	5	2001	2

Table 6.4: Fault tree data after factorisation 1

The complex event array has now been started, and is shown in Table 6.5.

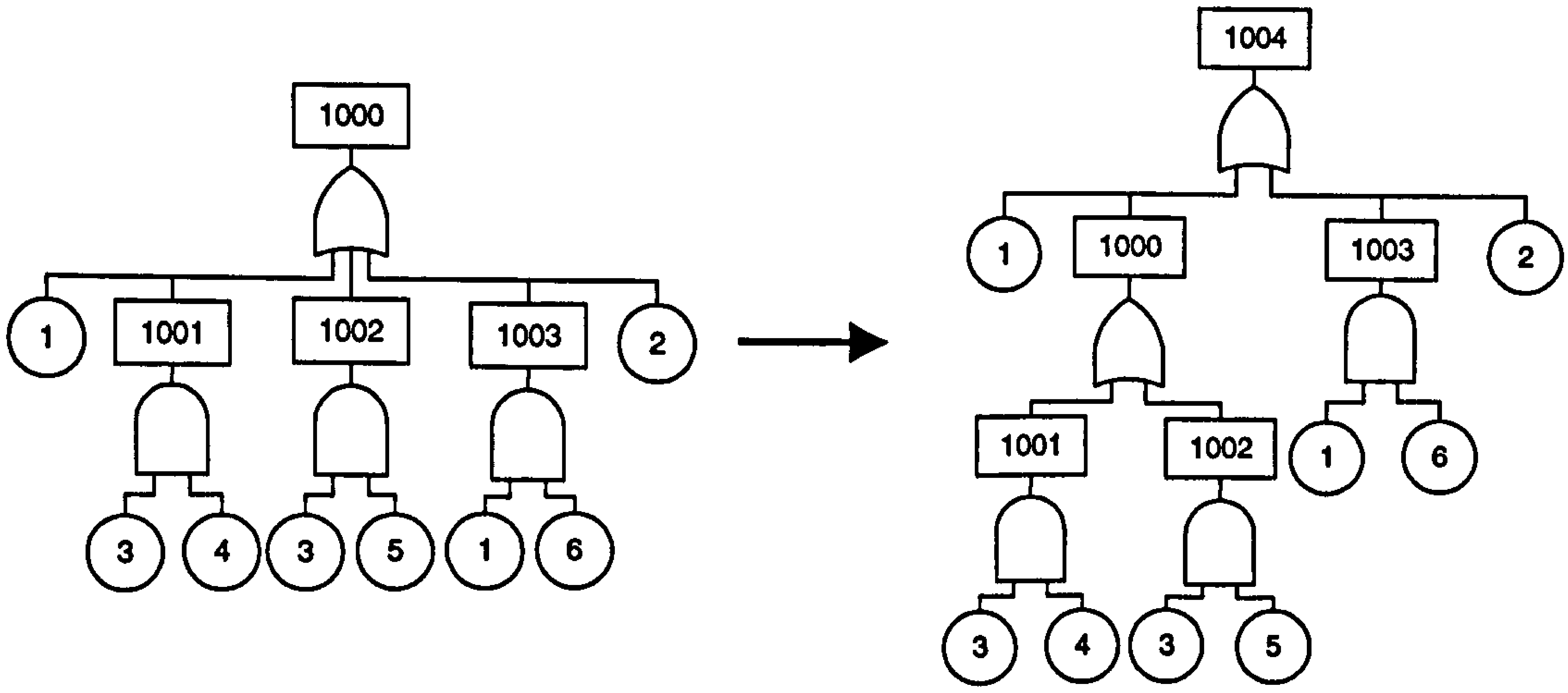
Complex event	Gate value	Event 1	Event 2
2000	1	1	3
2001	1	4	6

Table 6.5: Complex event data after factorisation 1

Extraction 1

The extraction process searches for the structures shown in Figure 6.1. In order to do this, the program scans through each line of data, examining the gate inputs to the primary gate. If the primary gate does not have at least two gate inputs, then the program moves onto the next gate. If it does have two or more gates as inputs (referred to as the secondary gates), then the gates are selected in pairs. Both secondary gates are then checked to see if they are of the same type, but a different type to the primary gate. If so, the inputs to the secondary gates are checked to see if they have a gate or event in common. If they do, then extraction can take place.

Before extraction can occur, however, there may be some necessary adjustments to be made to the data. If the primary gate has more than two inputs, then a new gate must be created which has the same gate type as the primary gate, but which has the primary gate and all its inputs, bar the two secondary gates, as inputs. This restructures the fault tree into the form required for extraction, by using an equivalent representation. An example of this is shown in Figure 6.6.



(a) (b)

Figure 6.6: Equivalent representations of a fault tree

In Figure 6.6(a), the primary gate 1000 has two secondary gates, 1001 and 1002, which have an input in common. In order to get this tree into the required form for extraction, gate 1004 is generated, as shown in Figure 6.6(b). This has gate 1000 as an input, together with events 1 and 2 and gate 1003, which were inputs to gate 1000. Gate 1000 now only has its two secondary gates as inputs. The fault tree data has a new line added for the generated gate, which is listed in the same way as the other gates. The line containing the data for gate 1000 is also adjusted accordingly.

A second adjustment may be required if the secondary gates appear elsewhere in the tree. The secondary gates will be altered (a gate or event extracted as a common input) but any other occurrences of this gate should remain unchanged. This problem is overcome by checking the occurrences of the secondary gates and if either occurs more than once, a new gate must be created. This new gate has exactly the same properties and inputs as the secondary gate, and replaces it in the input list to the primary gate. Therefore, the data for the original secondary gate and its other occurrences in the tree remain unchanged, and the new data can be altered accordingly.

Once the tree is in the correct form, the extraction process can be undertaken. The numbering of the gates is important in order to avoid confusion and is shown in Figure 6.7.

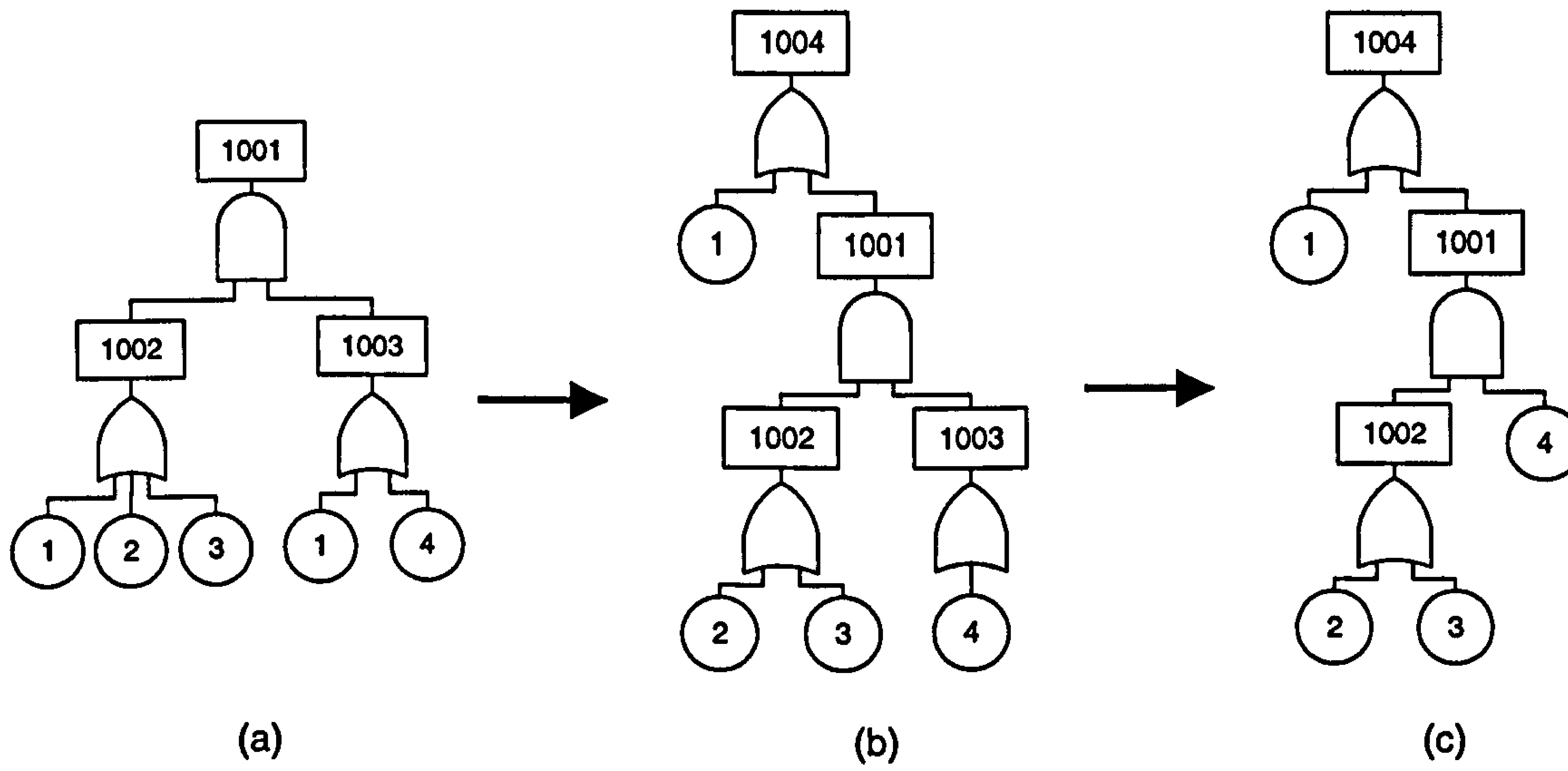


Figure 6.7: Numbering of fault tree gates throughout extraction

In Figure 6.7(b), a new gate is created (1004), which is of the same type as the secondary gates and has the common input, 1, and the primary gate, 1001, as its inputs. The common input 1 is removed from both 1002 and 1003. Figure 6.7(c) shows the next stage, which is the removal of gate 1003 (as it only has one input) with its input directed to gate 1001. This numbering is essential, as the secondary gates may have more than two inputs (as for 1002) and so remain in the tree and the extraction process must take account of this.

Application of the extraction procedure to the fault tree shown in Figure 6.5:

The only gate that has two or more gates inputs is 1002, whose inputs are 1003 and 1004. These secondary gates are both of a different type to the primary gate, and have the event 2001 in common, which can be extracted (Figure 6.8(a)). It is clear from Figure 6.8(a) that another extraction can also be undertaken. Gates 1003 and 1004 also have the event 2 in common, so a second extraction can be undertaken, as shown in Figure 6.8(b).

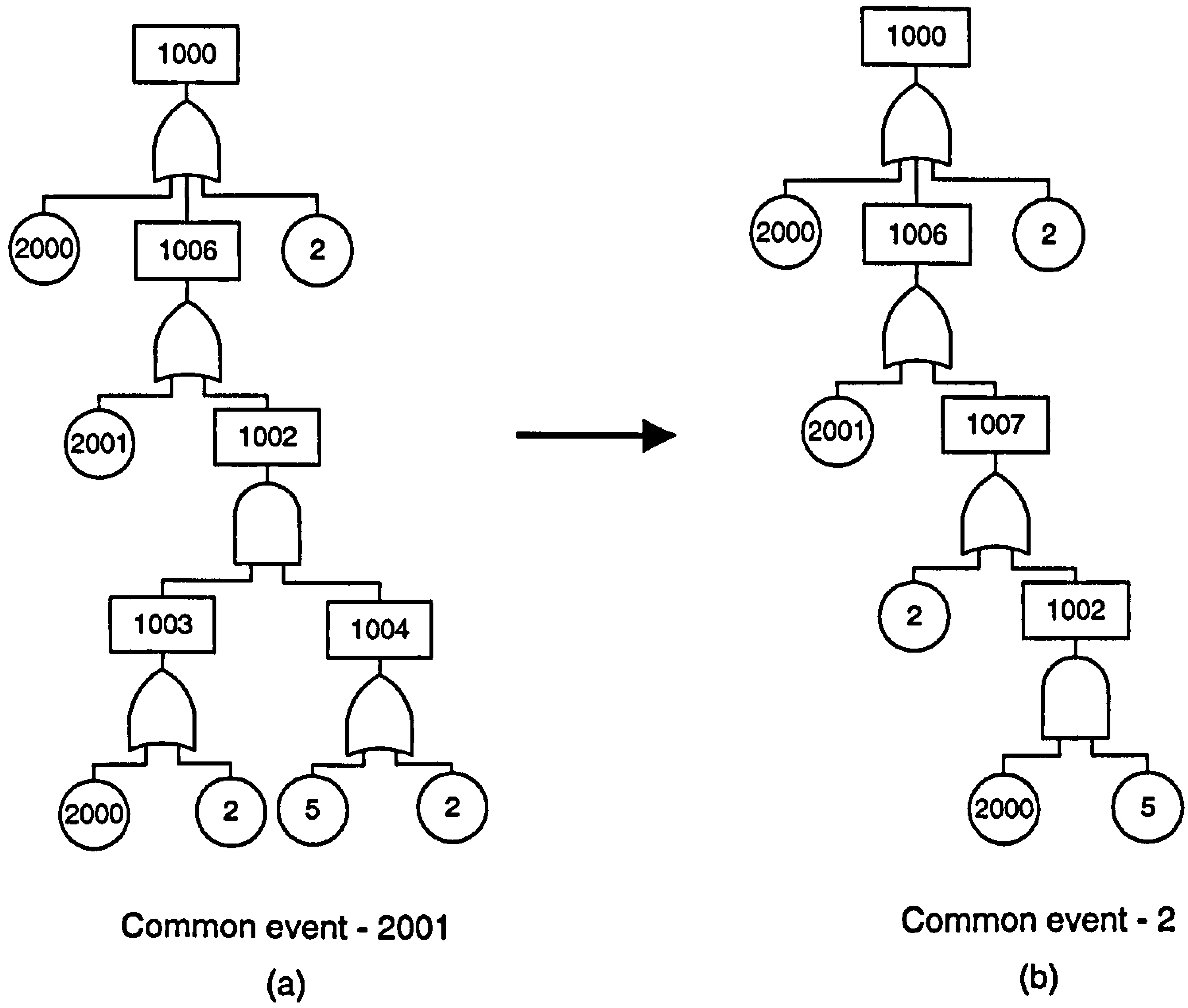


Figure 6.8: Fault tree after extraction 1

The resulting fault tree data no longer lists gates 1003 and 1004, but instead lists the generated gates 1006 and 1007, as shown in Table 6.6.

Gate number	Gate value	Number of gates	Number of events	Inputs
1000	1	1	2	1006 2000 2
1002	2	0	2	2000 5
1006	1	1	1	1007 2001
1007	1	1	1	1002 2

Table 6.6: Fault tree data after extraction 1

Contraction 2

Two further contractions can now take place: gate 1006 can be contracted into gate 1000 and gate 1007 can then also be contracted into gate 1000. This would leave event 2 as occurring twice as an input to gate 1000, but the program checks for this, and deletes one occurrence, updating the occurrence array at the same time. The resulting fault tree and arrays are shown in Figure 6.9 and Table 6.7.

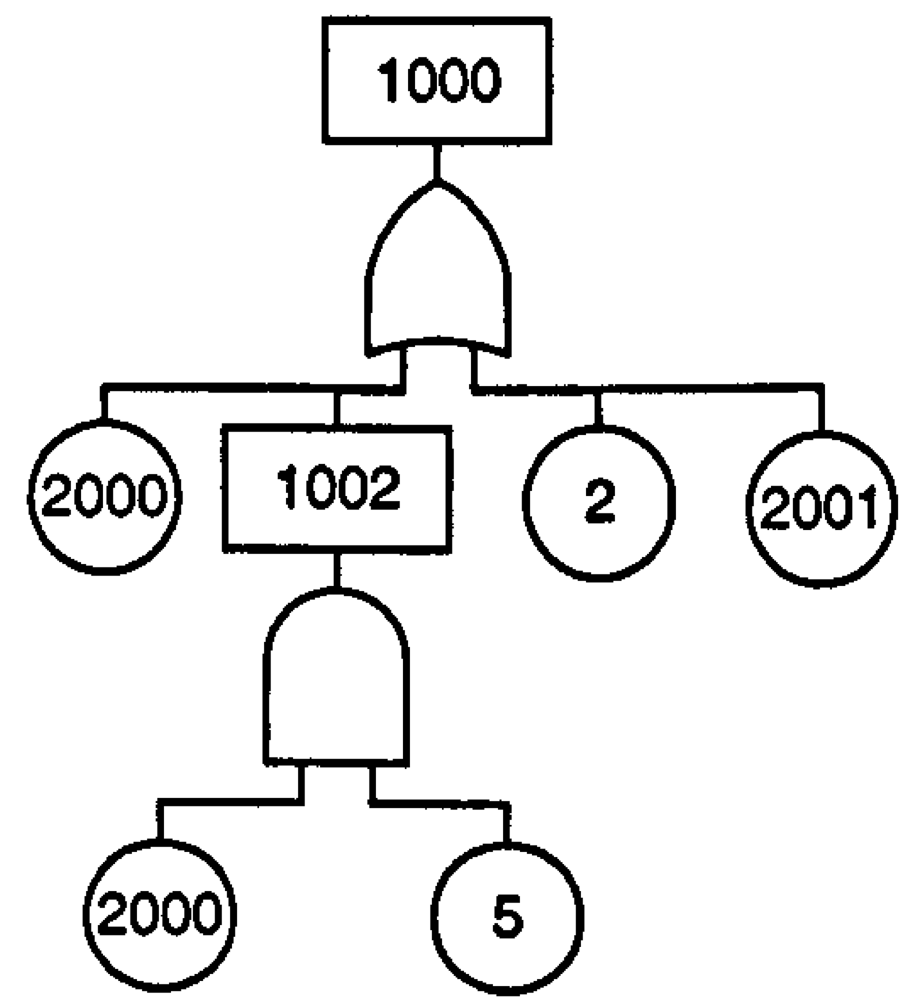


Figure 6.9: Fault tree after contraction 2

Gate number	Gate value	Number of gates	Number of events	Inputs
1000	1	1	3	1002 2000 2 2001
1002	2	0	2	2000 5

Table 6.7: Fault tree data after contraction 2

Factorisation 2

Events 2 and 2001 occur together, so the complex event 2002 is formed, resulting in the fault tree shown in Figure 6.10 and the fault tree data shown in Tables 6.8 and 6.9.

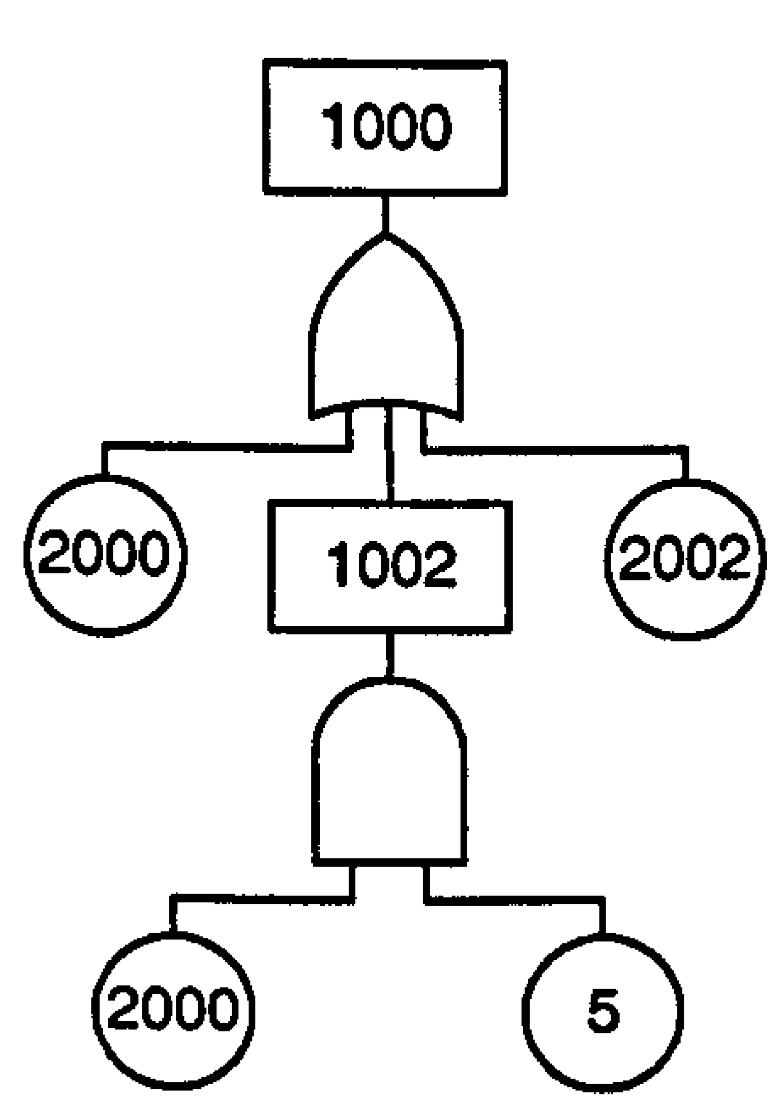


Figure 6.10: Fault tree after factorisation 2

Gate number	Gate value	Number of gates	Number of events	Inputs
1000	1	1	2	1002 2000 2002
1002	2	0	2	2000 5

Table 6.8: Fault tree data after factorisation 2

Complex event	Gate value	Event 1	Event 2
2000	1	1	3
2001	1	4	6
2002	1	2	2001

Table 6.9: Complex event data after factorisation 2

Extraction 2

No extractions can be performed on the fault tree. The program would carry out the three steps again, as there have been changes made, but no further modifications are possible.

6.3.3 The Reduced Fault Tree

The fault tree and complex event data are output to data files in terms of the original gates and event names. Any complex events and generated gate names are output as they were named in the program.

The reduced fault tree and corresponding data files are shown in Figure 6.11 and Tables 6.10 and 6.11 in terms of the original event and gate names.

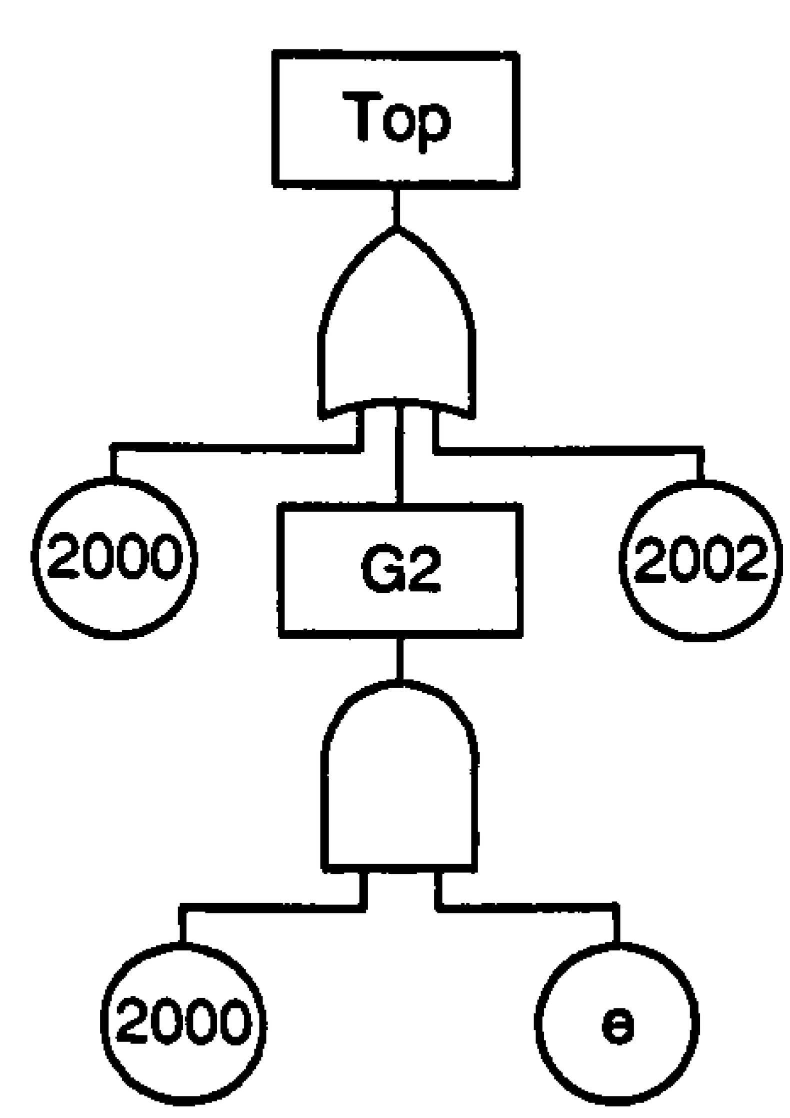


Figure 6.11: The reduced fault tree

Gate name	Gate type	Number of gates	Number of events	Inputs
Top	1	1	2	G2 2000 2002
G2	2	0	2	2000 e

Table 6.10: Reduced fault tree data

Complex event	Gate type	Event 1	Event 2
2000	1	a	b
2001	1	f	c
2002	1	d	2001

Table 6.11: Complex event data

It can be verified that the reduced tree is equivalent to the original tree by examining their minimal cut sets. These will be identical for logically equivalent trees. The original tree has five minimal cut sets of order one:

{a}, {b}, {c}, {d} and {f}

The minimal cut sets for the reduced tree are:

{2000} and {2002}

These can be expanded out in terms of the basic events by taking a 'MOCUS[31]' type of approach. The principle of this method is that 'OR' gates increase the number of cut sets, whilst 'AND' gates increase the number of elements in the cut sets. Therefore using the basic event data in Table 6.11, the minimal cut sets of the reduced tree can be expanded to give:

$$
\begin{aligned}
\text{Top} &= 2000 + 2002 \\
&= a + b + d + 2001 \\
&= a + b + d + f + c
\end{aligned}
$$

which are equivalent to those obtained from the original tree. The technique for obtaining the minimal cut sets of reduced trees in terms of their basic events has been programmed as part of the research (cutsets.c to obtain the minimal cut sets in terms of complex events and complex_cuts.c to expand these out) in order to verify that the trees used to assess the reduction technique have been restructured correctly.

Reduction has simplified the example fault tree considerably. In the original fault tree, there were six gates; in the reduced fault tree, there are two. In the original fault tree there were twelve events, six of them different; in the reduced fault tree there are four events, and only three of them are different. This means that when choosing a variable ordering there are half the number of events to consider, so the number of options for variable ordering is significantly reduced. It is expected that BDDs constructed from reduced fault trees will be substantially smaller than those constructed from non-reduced fault trees and in order to test this hypothesis, the reduction technique was applied to a set of fault trees and their BDD sizes compared. The results are discussed in the following sections.

6.4 Results of the Application of the Reduction Technique

The reduction technique was applied to a set of 228 fault trees. Summary details for the trees are given in Appendix II. BDDs were constructed for each reduced tree using variable orderings determined by the eight ordering schemes analysed in Chapter 5. These are:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

The resulting BDDs were analysed in two ways. Firstly, the success of the reduction technique was evaluated by comparing the complexity of the BDDs constructed from the reduced fault trees against those obtained using the original trees. Then, the performance of the ordering schemes on the reduced trees was analysed by considering the number of times each scheme received the highest ranking and the average ranking of the schemes over all the trees. The results are discussed in the following sections.

6.4.1 Effect of the Reduction Technique on BDD Complexity

The BDDs constructed from the reduced fault trees were compared against those obtained from the original trees for three different measures of BDD complexity: the number of non-distinct BDD nodes, the number of distinct BDD nodes and the number of **ite** calculations required to construct the BDD. The results obtained for each measure of BDD complexity for the set of fault trees are given in Appendices VI, VII and VIII. It was expected that the reduction technique would reduce the trees to a more concise form and that consequently the BDD construction process would be more efficient, requiring fewer **ite** calculations and producing BDDs with fewer non-distinct and distinct nodes.

As there are 228 fault trees with eight ordering schemes used for each, there are a total of 1824 cases to consider. The difference in the number of **ite** calculations and the number of non-distinct and distinct BDD nodes was calculated for each case together with the percentage decrease.

6.4.1.1 Non-Distinct Nodes

Out of a total of 1824 cases, 1751 either showed a decrease or no change (often due to the fact that the BDDs were already minimal for the original fault trees) in the number of non-distinct nodes after reduction. The average decrease over these 1751 cases was 46.72%, which means that on average, the number of non-distinct nodes approximately halved through reduction. This is a substantial decrease in BDD size for a procedure that takes such a short amount of time to apply.

In 73 cases, which account for 4.00% of the sample tested, the number of non-distinct nodes actually increased. This can occur because of the difference in the variable ordering once reduction has taken place. The BDD size is very sensitive to the chosen ordering, so if the fault tree changes sufficiently that the same ordering scheme produces a different variable ordering for the reduced tree, it is possible that this would have an adverse effect and actually increase the number of nodes in the resulting BDD. For example, consider the fault tree 'random1' (summary details for the tree are given in Appendix II) shown in Figure 6.12(a).

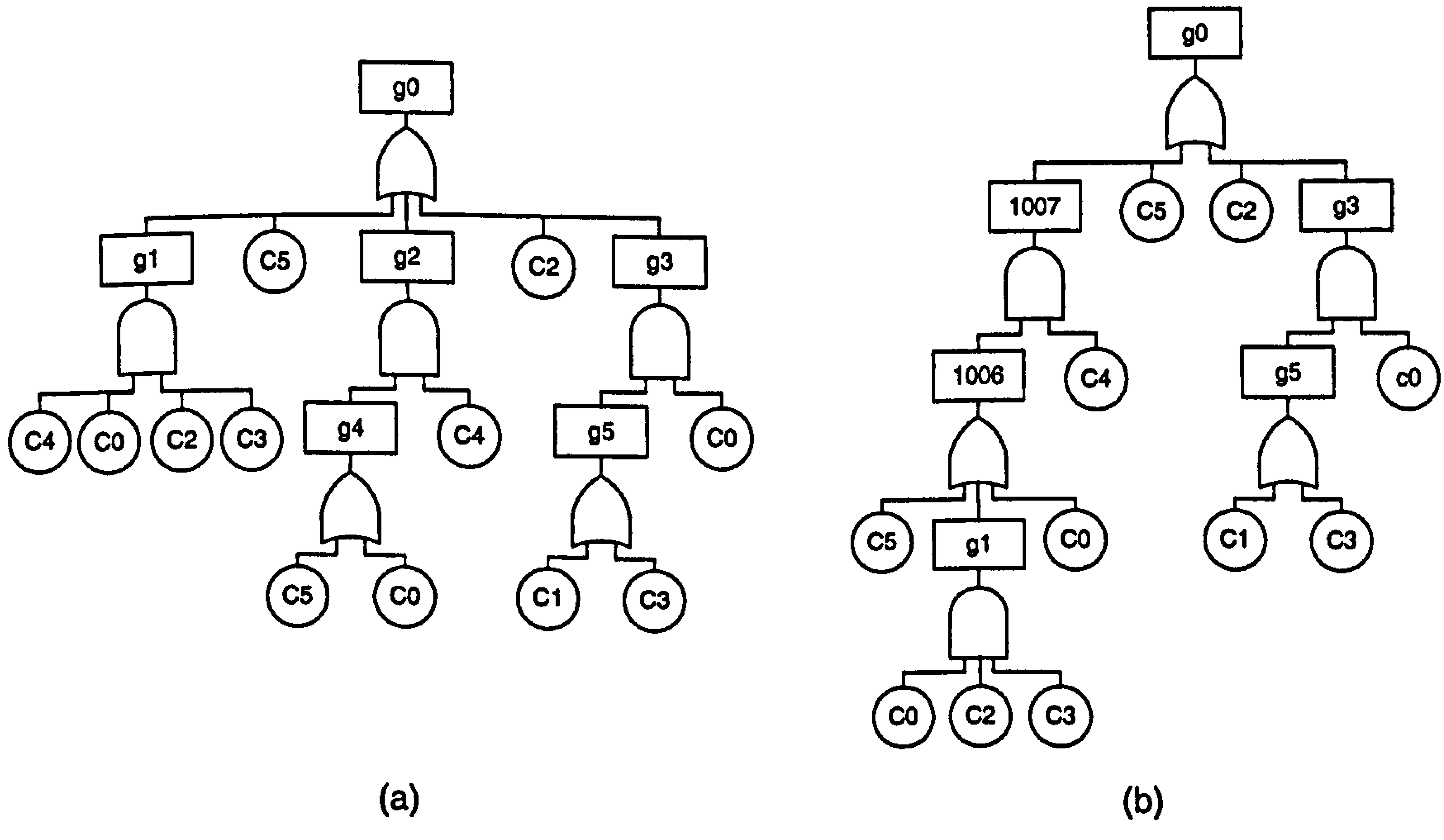


Figure 6.12: The fault tree 'random1' shown before (a) and after (b) reduction

If the variables of the original fault tree are ordered according to the modified depth-first method, the following ordering is obtained:

$$C5<C2<C0<C4<C3<C1$$

The BDD constructed from this ordering is shown in Figure 6.13.

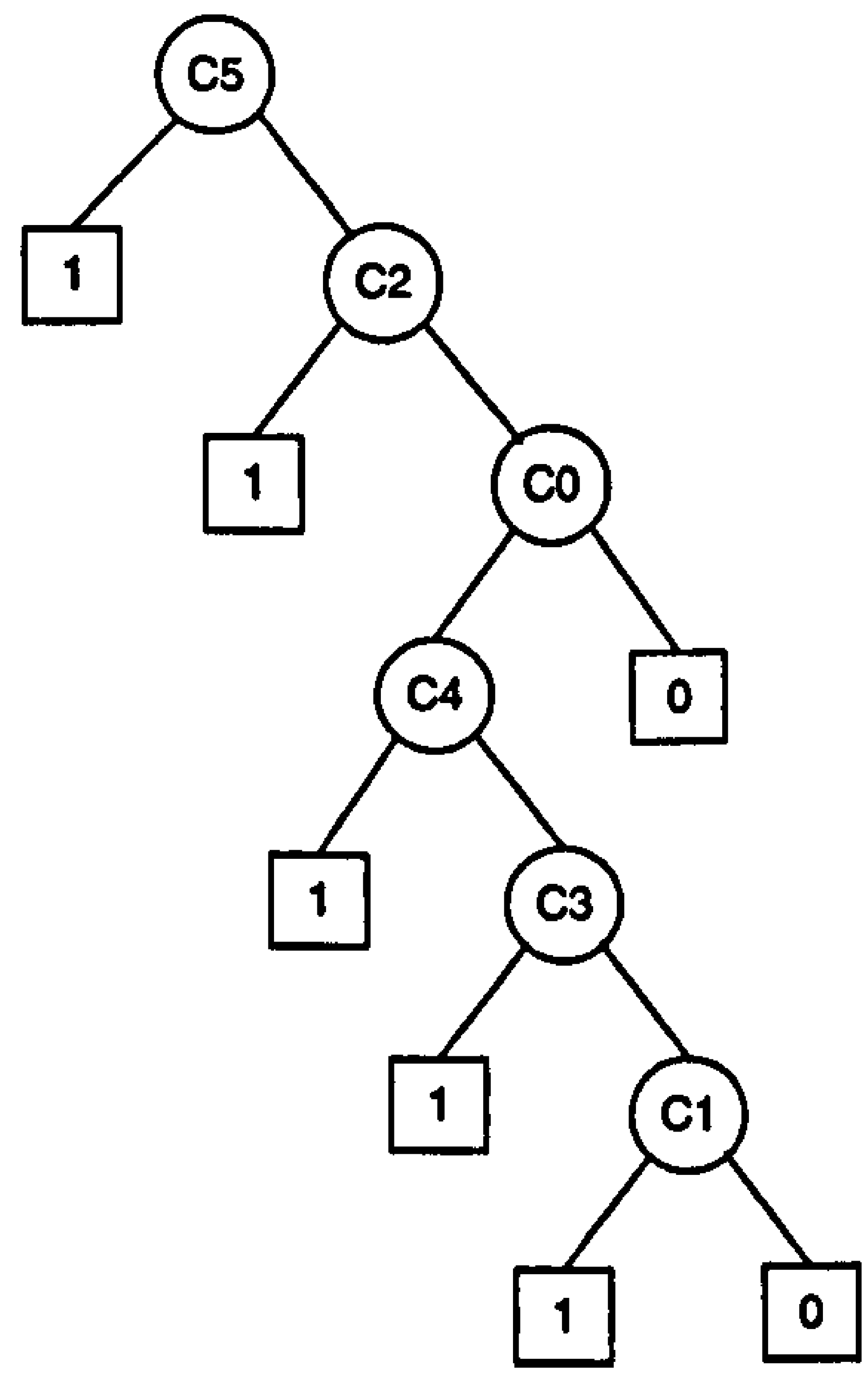


Figure 6.13: The BDD constructed from the fault tree in Figure 6.12(a) using the modified depth-first ordering

The resulting BDD has six non-terminal nodes, which is the number of both distinct and non-distinct nodes. However, by applying the reduction technique to the fault tree, the reduced tree shown in Figure 6.12(b) is obtained. The only stage of the method that can be applied is extraction, where event C4 appears as a common event to both G1 and G2, so can be extracted. The resulting fault tree has the same number of gates as the original (six in total), but has one less basic event (eleven as opposed to twelve). However, when the modified depth-first ordering scheme is applied to the reduced tree, the following ordering is obtained:

C5<C2<C4<C0<C3<C1

Event C4 now appears earlier in the ordering than it did for the original tree. This is because in the original tree it appears under the same gate as C0, so is ordered after C0 as it has fewer occurrences in the tree. In the reduced tree however, it appears as the only event input to the gate above C0 so is ordered first. The resulting BDD is shown in Figure 6.14.

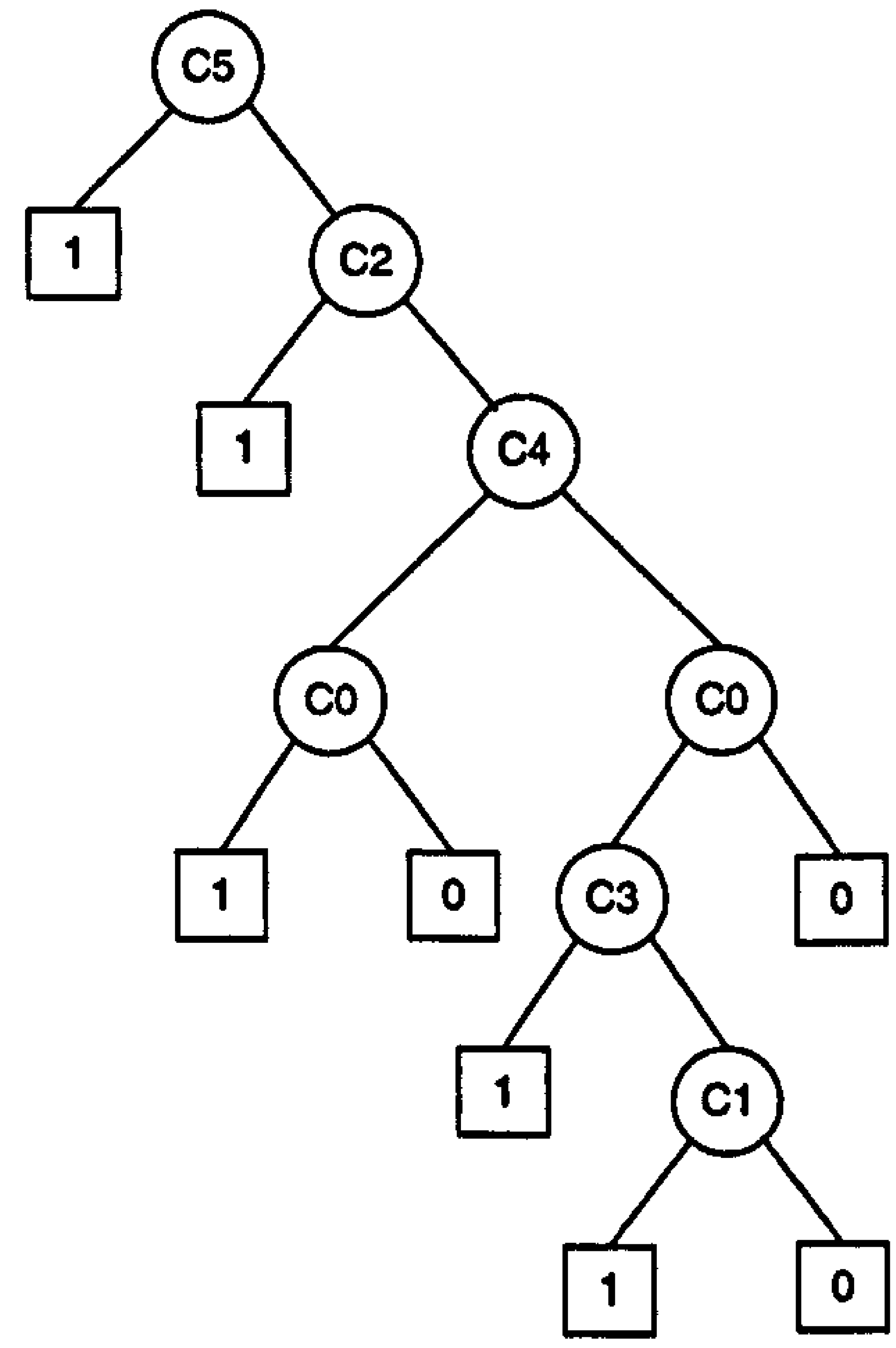


Figure 6.14: The BDD constructed from the reduced fault tree in Figure 6.12(b) using the modified depth-first ordering

This BDD has seven distinct and non-distinct nodes, meaning that reduction has increased the number of nodes in the BDD. However, this is purely down to the variable ordering. If the ordering obtained from the reduced fault tree is used to construct a BDD from the original fault tree, the resulting BDD is exactly the same as that shown in Figure 6.14. Also, if the ordering C5<C2<C0<C4<C3<C1 (i.e. the ordering obtained from the original tree) is used to construct a BDD from the reduced tree, the BDD obtained is the same as the one shown in Figure 6.13, with only six nodes.

The alternate orderings could have been applied to the trees automatically, simply by altering the way in which the original fault tree is written. For example, if g1 and g2 had been swapped in the original fault tree, then C4 would have been ordered first, and the same number of nodes would have been obtained for both the original and reduced tree. Or, the gates (or even the events beneath the gates) could have been written such that C0 was extracted (either by ordering the inputs to g1 such that C0 appears before C4, or by listing g3 before g1 in the inputs to g0), meaning that it would have been ordered before C4 in the ordering, so producing the smaller BDD from the reduced tree. This demonstrates that the ordering scheme chosen for the original trees is not necessarily the best choice of scheme for the reduced trees, as modifications to the fault tree structure can affect the resulting variable ordering. Also, it has been shown that the way in which the original fault tree is written has a direct effect on the structure of resulting reduced tree.

Overall, there are 32 fault trees that showed an increase in the number of non-distinct nodes, for one or more of the ordering schemes. The smallest number of non-distinct nodes obtained over all the orderings from the original tree was compared with the smallest number of non-distinct nodes obtained via the reduced tree. It was found that reduction increased the smallest possible number of nodes in only four trees. This accounts for just 1.75% of cases. So in 28 of these trees, although one or more orderings produced a BDD from the reduced tree with more nodes than the one obtained using the original tree, there was at least one other ordering that resulted in a BDD from the reduced tree of smaller or equivalent size to the smallest produced from the original tree. For example, the tree 'rand147' produced the results shown in Table 6.12.

Ordering scheme	Non-distinct BDD nodes using the original tree	Non-distinct BDD nodes using the reduced tree	Decrease in the number of nodes
1	3017	6692	-3675
2	160475	60575	99900
3	168581	50517	118064
4	6307	54288	-47981
5	2761	2262	499
6	36930	1493	35437
7	11385	6842	4543
8	5460	1655	3805

Figure 6.12: Results for fault tree 'rand147'

This fault tree shows two increases in the number of non-distinct nodes obtained from the reduced tree, in ordering schemes 1 and 4. However, the smallest BDD obtained from the original tree (scheme 5) has 2761 non-distinct nodes. Although ordering five results in a BDD with fewer nodes after reduction, ordering six actually produces an even smaller BDD with only 1493 non-distinct nodes. Therefore the minimum BDD size has been reduced by 45.93%. So, even though an increase occurred using two of the schemes, the fault tree has ultimately benefited from reduction, as a significantly smaller BDD could be constructed.

Each tree was analysed in this way and it was found that the average reduction in the minimum number of non-distinct nodes was 44.86% for the 224 trees that recorded a constant or decreased minimum BDD size. The minimum number of nodes for the four remaining trees was on average 11.37% lower when constructed using the original trees.

6.4.1.2 Distinct Nodes

In 1732 cases out of the total of 1824, the number of distinct BDD nodes decreased or remained the same after the reduction process had been applied to the trees. The average decrease in the number of distinct nodes for these trees was 34.29%. This is not as high as the result obtained for the number of non-distinct nodes, but as there are usually fewer distinct nodes in the BDD than non-distinct nodes, there is less scope for improvement.

As with the results for the non-distinct nodes, a small percentage of the cases actually showed an increase in BDD size. This occurred for 92 cases out of 1824 (5.04%), which is slightly more than when considering non-distinct nodes.

The 92 cases that showed an increase in size account for 45 different fault trees. Of these, reduction had a negative effect on twelve, as the minimum number of nodes that was obtained over all the orderings was smaller before reduction than after reduction, by 12.55%. These twelve trees account for 5.26% of the set of fault trees that were considered. However, reduction had either a positive effect or no effect on the remaining 33 trees, as although one or more orderings resulted in an increase in the number of nodes, another ordering either improved or equalled the smallest number of nodes that was previously attainable. For these trees, together with the 183 trees that showed no increase in the number of distinct BDD nodes after reduction, the average decrease in the minimum number of distinct nodes was 32.47%.

Of the twelve trees whose minimum number of distinct BDD nodes increased through reduction, only two were in the set of four that reduction affected negatively when considering the number of non-distinct nodes. Therefore, for ten of the trees that recorded an increase in the minimum number of distinct BDD nodes after reduction, their minimum number of non-distinct BDD nodes actually decreased or stayed the same (in fact it decreased for nine and remained the same for one). And conversely, there were two trees whose minimum number of non-distinct BDD nodes increased after reduction that did not show an increase in the minimum number of distinct BDD nodes – one showed a decrease, one remained the same. So overall, only two trees (rando33 and rand144) showed an increase in the minimum number of both distinct and non-distinct BDD nodes after reduction.

6.4.1.3 Number of If-Then-Else Calculations

The number of **ite** calculations required to obtain the BDD from the reduced trees either decreased or remained the same when compared to the number needed for the original trees in 1580 out of 1824 cases. On average, the number of calculations was reduced by 40.87%. This is a very promising result, as it shows that not only is the final BDD size significantly affected by the reduction process, but that the number of calculations and the time taken to perform them is also substantially reduced.

In 244 cases, which account for 13.38% of those considered, the number of **ite** calculations actually increased. This is a larger percentage than was obtained for the number of non-distinct and distinct nodes and means that although the final BDD size is smaller, sometimes more calculations are necessary for its construction.

The 244 cases that showed an increase in the number of required **ite** calculations account for 53 different fault trees. Of these, reduction has a negative effect on 27 (11.84% of the sample), as the minimum number of **ite** calculations was smaller by an average of 10.51% when the BDDs were constructed from the original trees. Only one of these trees (rand144) also showed an increase in the minimum number of distinct and non-distinct BDD nodes after reduction. The remaining 26 trees actually benefit from or are not affected by the reduction process, as although one or more orderings resulted in an increased number of calculations, the previous minimum was either improved or equalled by other schemes. For these trees, together with the 175 trees that showed no increase in the number of **ite** calculations after the reduction process has been applied, the average decrease in the minimum number of calculations was 40.39%.

6.4.1.4 Summary of Results

The BDDs constructed from the reduced fault trees were compared against those constructed using the original fault trees for three different measures of BDD complexity. The reduction technique has been shown to perform well according to each, with average decreases of 46.72% over 96.00% of the 1824 cases for the number of non-distinct BDD nodes, 34.29% over 94.96% of cases for the number of distinct BDD nodes and 40.87% over 86.62% of cases for the number of **ite** calculations required to construct the BDD.

The smallest attainable values of BDD complexity (i.e. the minimum obtained over all eight ordering schemes) were also compared for each of the original and reduced trees. Average decreases were recorded of 44.86% over 224 trees for the number of non-distinct nodes, 32.47% over 216 trees for the number of distinct nodes and 40.39% over 201 trees for the number of **ite** calculations required to obtain the BDD.

Only one tree (rand144) recorded an increase in each measure of BDD complexity. Nine other trees (benjiam, rstree5, worrell, random4, random7, rando27, rando45, rando72 and rand152) show no improvement in any of the measures, but reduction has a positive effect on the remaining 218 trees, which each produce BDDs with at least one improved complexity measure.

6.4.2 Performance of the Ordering Schemes on the Reduced Fault Trees

The performance of the ordering schemes on the reduced trees is assessed in two ways. The first method considers the number of times that each scheme produces the best results and the second method examines the average ranking of each scheme over all the trees. The results are discussed in the following sections.

6.4.2.1 Results: Highest Scheme Rankings

The schemes are ranked in order according to the complexity of the BDDs that they produce and a count is then made of the number of times that each scheme receives the highest ranking. Three measures of BDD complexity are considered: the number of non-distinct BDD nodes, the number of distinct BDD nodes and the number of **ite** calculations required for BDD construction. Results are not included for the trees that give identical values for each ordering scheme.

6.4.2.1.1 Non-Distinct Nodes

The eight ordering schemes gave identical results for 90 of the 228 trees. This is significantly more than the number of identical results obtained using the non-reduced trees. In fact, the number has increased by more than 50% from 59 trees. This is due to two factors. Firstly, the reduced trees generally have fewer variables, meaning there are fewer variations in the orderings produced by the schemes and consequently identical BDDs are constructed. The second reason is that smaller fault trees produce BDDs that are not so variable is size and so different orderings are more likely to produce BDDs with the same complexity, but that are not necessarily identical. Whatever the reason for producing identical results, it is obviously advantageous if it reduces the importance of choosing just one 'correct' scheme.

The results for the remaining 138 trees are shown in Table 6.13.

Ordering scheme	1	2	3	4	5	6	7	8
Number of trees using non-distinct nodes	13	24	23	24	30	32	28	63

Table 6.13: The number of reduced trees for which each scheme was ranked the highest according to the number of non-distinct BDD nodes

The event criticality scheme (8) performs significantly better than the other schemes, producing BDDs with the fewest non-distinct nodes for 63 fault trees. This echoes the result obtained for the non-reduced trees.

The four weighted methods (schemes 5-8) have all performed well, producing better results than the structural schemes. This was not seen in the results for the non-reduced trees and could be due to the way in which the reduced trees are now structured

6.4.2.1.2 Distinct Nodes

The ordering schemes produced identical results for 90 trees. Again, this is significantly more than the number of identical results obtained for the non-reduced trees, which was 64. Table 6.14 shows the results obtained for the remaining 138 trees.

Ordering scheme	1	2	3	4	5	6	7	8
Number of trees using distinct nodes	12	40	35	39	14	32	37	35

Table 6.14: The number of reduced trees for which each scheme was ranked the highest according to the number of distinct BDD nodes

The schemes based upon a depth-first approach (2, 3, 4 and 7) perform well, as they did for the non-reduced trees. However, the results are far closer when using the reduced trees, with a difference of only 28 between the best and worst performers (modified depth-first (scheme 2) and modified top-down (scheme 1) respectively) compared with a difference of 57 when using the non-reduced trees. This again suggests that the choice of ordering scheme is of less importance when considering the reduced trees. There are still 'good' and 'bad' schemes for each tree, but overall the difference is not as marked as it was with the non-reduced trees.

6.4.2.1.3 Number of If-Then-Else Calculations

The eight schemes produced identical results for 64 of the 228 fault trees. Again, this is a significant increase on the 41 identical results obtained using the non-reduced trees. The results for the remaining 164 trees are shown in Table 6.15.

Ordering scheme	1	2	3	4	5	6	7	8
Number of trees using ite calculations	20	44	28	19	20	43	33	55

Table 6.15: The number of reduced trees for which each scheme was ranked the highest according to the number of **ite** calculations

The event criticality ordering scheme (8) outperforms the other schemes, producing BDDs from the fewest **ite** calculations in 55 cases. The results are similar to those obtained for the non-reduced trees, with schemes 2, 6 and 8 producing the best results, though the dynamic top-down weighted measure (scheme 6) has not performed as well as it did previously.

6.4.2.2 Results: Overall Ranking of the Schemes

The schemes are ranked in order from the one that produces the best results (i.e. the smallest number of nodes or the fewest **ite** calculations) to the one that produces the worst results for each fault tree, where a ranking of one indicates the best performance and a ranking of eight indicates the worst performance. The rankings are then added together over all 228 trees, to give an indication of the overall behaviour of the schemes. The best performance is indicated by the scheme with the lowest added ranking The results are not included for the trees that give identical values for each ordering scheme.

6.4.2.2.1 Non-Distinct Nodes

The rankings were added over the 138 trees to give the results shown in Table 6.16.

Ordering scheme	1	2	3	4	5	6	7	8
Added rankings for non-distinct nodes	627	640	583	595	460	508	659	371

Table 6.16: The added rankings for each ordering scheme for 138 reduced fault trees

The event criticality scheme (8) performs better than the other seven ordering schemes, which means that in addition to producing BDDs with the fewest number of non-distinct nodes for the most trees, it also produces consistently good results for the remaining trees. This is a result that was also seen for the non-reduced trees. In general, the weighted measures (with the exception of scheme 7, which is discussed below) perform better than the structural ordering schemes, both in the number of times they produce the smallest BDD and in these results for the overall scheme rankings.

The bottom-up weighted measure (7) performs badly in this scheme assessment method, though the results for the number of times it received the highest ranking placed the scheme in fourth position. This suggests that although it produces the smallest BDDs for a considerable number of fault trees, it does not perform well over the remaining trees. This conclusion was also drawn for the non-reduced trees.

6.4.2.2.2 Distinct Nodes

The rankings were added over the 138 trees to give the results shown in Table 6.17.

Ordering scheme	1	2	3	4	5	6	7	8
Added rankings for distinct nodes	717	503	491	505	583	468	578	528

Table 6.17: The added rankings for each ordering scheme for 138 reduced fault trees

The dynamic top-down weighted scheme (6) produced the best results, which is the first time that the depth-first schemes have been outperformed when considering the number of distinct BDD nodes (for both reduced and non-reduced trees).

It was noted in the results for the number of times that each scheme produced the best ranking that they were much closer for the reduced trees than for the non-reduced trees. This is also the case here with a difference of only 249 between the best and worst performing schemes, compared with 353 for the non-reduced trees. This again suggests that the choice of scheme becomes less critical when considering the number of distinct BDD nodes for reduced trees.

6.4.2.2.3 Number of If-Then-Else Calculations

The rankings for each scheme were added for the 164 fault trees to give the results shown in Table 6.18.

Ordering scheme	1	2	3	4	5	6	7	8
Added rankings for **ite** calculations	785	614	647	739	627	531	746	542

Table 6.18: The added rankings for each ordering scheme for 164 reduced fault trees

The dynamic top-down weighted measure (6) and the event criticality scheme (8) both perform well, as they did for the number of times each produced BDDs using the smallest number of **ite** calculations. The event criticality scheme (8) would probably be a marginally better choice of scheme as it produced BDDs with the fewest **ite** calculations for 12 more trees than the dynamic top-down weighted measure (6). However, the dynamic top-down weighted measure has shown great potential and proved the better choice of scheme for the non-reduced trees (when considering the number of **ite** calculations) and would benefit from further investigation.

6.4.2.3 Summary of Results

The event criticality ordering scheme (8) performs well when considering the number of non-distinct BDD nodes, producing the smallest BDDs most often and the best overall ranking. It is also a good choice of scheme when considering the number of **ite** calculations required to obtain the BDD. The dynamic top-down ordering (6) also produced good results for the number of **ite** calculations and was the best choice of scheme when considering the overall rankings for the number of distinct BDD nodes. The modified depth-first scheme (2) produced BDDs with the fewest distinct nodes for the most trees and is the only category in which the event criticality (8) and dynamic top-down orderings (6) were outperformed. In fact, the four schemes based on the depth-first approach provided the best results in this category, as they did when considering the non-reduced fault trees.

The ordering schemes produced identical results for a significant number of the reduced trees. For the number of non-distinct and distinct BDD nodes, identical results were obtained for 90 trees whilst for the number of **ite** calculations, the total number of trees with identical results was 64. These figures are substantially higher than for the non-reduced trees and suggest that it is less critical to choose just one 'correct' scheme. This was also shown in the results for the number of distinct BDD nodes, where there is less difference between 'good' and 'bad' schemes.

6.5 Conclusions

The Faunet reduction technique has been shown to be an effective pre-processing tool for fault trees. BDDs constructed from a set of 228 reduced trees were compared against those obtained from non-reduced trees for three different measures of BDD complexity: the number of non-distinct BDD nodes, the number of distinct BDD nodes and the number of **ite** calculations required to obtain the BDD. The results showed a significant decrease in each measure of BDD complexity for a large percentage of the trees tested. The performance of eight ordering schemes on the reduced trees was also assessed according to these measures and the results obtained suggest that the choice of ordering scheme becomes less critical when dealing with reduced trees. The use of the Faunet reduction technique is therefore recommended for application to fault trees before constructing BDDs.

Chapter 7: Quantitative Analysis of Binary Decision Diagrams Incorporating Modules and Complex Events

7.1 Introduction

The quantitative analysis of BDDs is an exact and efficient procedure, which determines many properties of the system under consideration. To date, the methods have only been applied to BDDs consisting entirely of basic events. However, the techniques of reduction and modularisation have been investigated as methods of optimising fault trees and so can result in BDDs encoding both complex and modular events. The current methods therefore need to be extended to consider these additional factors.

In this chapter, the current procedures for performing the basic elements of quantitative analysis, such as calculating the system unavailability, the unconditional failure intensity and the criticality functions of the basic events, are explained. The methods are then extended to incorporate both complex events and modules into the analysis, so that BDDs obtained from reduced and modularised fault trees can be quantified.

7.2 System Unavailability

The **ite** structure encoded in the BDD is derived from Shannon's theorem, which can be used to express the structure function for the top event as:

$$f(\mathbf{x}) = x_i.f_1(x_1, x_2,\ldots, x_{i-1}, 1, x_{i+1},\ldots, x_n) \qquad 7.1$$
$$+ \overline{x_i}.f_2(x_1, x_2,\ldots, x_{i-1}, 0, x_{i+1},\ldots, x_n)$$

where: x_i is the pivoting variable

f_1 and f_2 are Boolean functions with $x_i = 1$ and $x_i = 0$ respectively.

If $f(\mathbf{x})$ represents the root vertex of the BDD, encoding the event x_i, then the equations for the next level in the BDD will be f_1 for the one branch and f_2 for the zero branch. The probability of the top event (i.e. system unavailability) can be found by taking the expectation of each term of Equation 7.1, to give:

$$E[f(\mathbf{x})] = q_i(t).E[f_1] + (1-q_i(t)).E[f_2] \qquad 7.2$$

where $q_i(t) = E[x_i]$, the probability that event i occurs.

Therefore the system unavailability can be calculated by summing the probabilities of the disjoint (mutually exclusive) paths through the unminimised BDD. The disjoint paths can be found by tracing all paths from the root vertex to terminal one vertices. Each disjoint path represents a combination of working and failed components that leads to system failure and therefore events lying on both one and zero branches are included in the probability calculation.

In order to implement the calculation procedure, Equation 7.2 can, in effect, be applied to each node in the BDD to get its 'probability value'. This only has a physical representation for the root vertex, as it is equivalent to the top event probability; for any other node in the BDD it is simply used as a means of calculation and has no physical significance. For any BDD node, F = **ite**(x_i, J, K), the probability value is given by:

$$P[F] = q_i(t).P[J] + (1-q_i(t)).P[K] \tag{7.3}$$

where P[J] is the probability value of the node on the one branch of F

P[K] is the probability value of the node on the zero branch of F

Equation 7.3 is applied to the BDD in a bottom-up manner. Nodes that have terminal vertices on both their one and zero branches are considered first, as terminal one and zero vertices simply have probability values of one and zero respectively. The values are then worked up through the BDD until the top event probability can be evaluated.

7.3 System Unconditional Failure Intensity

The system unconditional failure intensity, $w_{sys}(t)$, which is defined as the probability that the top event occurs at t per unit time, is given by:

$$w_{sys}(t) = \sum_i G_i(\mathbf{q}(t)).w_i(t) \tag{7.4}$$

where $G_i(\mathbf{q}(t))$ is the criticality function for each component

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

The criticality function is defined as the probability that the system is in a critical state with respect to component i and that the failure of component i would cause the system to go from a working to a failed state. Therefore:

$$G_i(\mathbf{q}(t)) = Q(1_i, \mathbf{q}(t)) - Q(0_i, \mathbf{q}(t)) \tag{7.5}$$

where $Q(1_i,\mathbf{q}(t))$ is the probability of system failure with $q_i(t) = 1$ and $Q(0_i,\mathbf{q}(t))$ is the probability of system failure with $q_i(t) = 0$.

An efficient method of calculating the criticality function from the BDD[32] considers the probabilities of the path sections in the BDD up to and after the relevant nodes. For example, consider the variable x_i, which occurs at two intermediate nodes in the BDD, as shown in Figure 7.1.

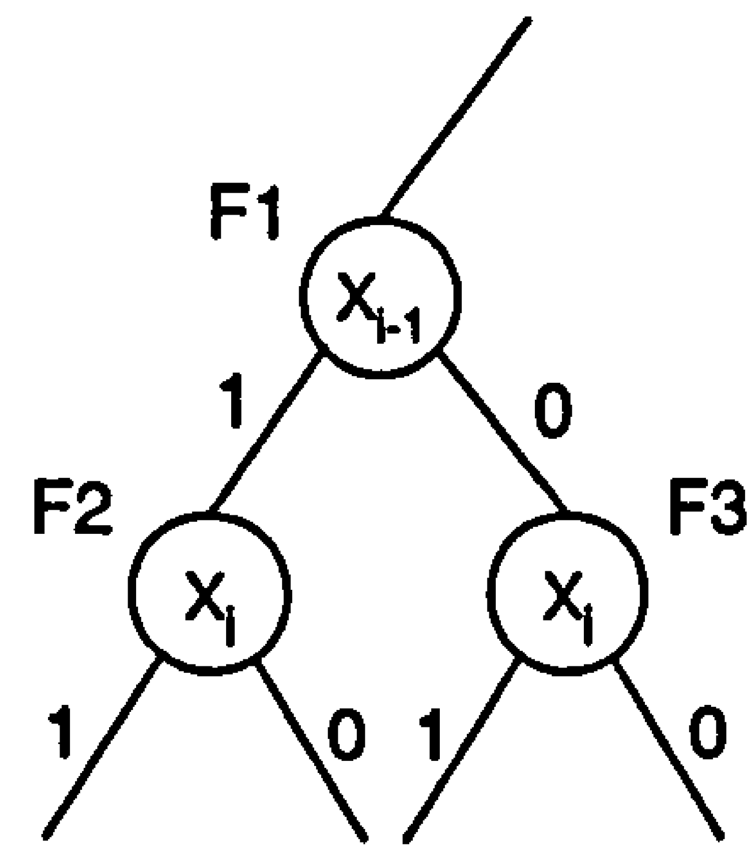


Figure 7.1: BDD section showing the locations of variable x_i

$Q(1_i,q(t))$ and $Q(0_i,q(t))$ can be defined for this variable as:

$$Q(1_i,q(t)) = \sum_n (pr_{x_i}(q(t)).po^1_{x_i}(q(t))) + Z(q(t)) \tag{7.6}$$

$$Q(0_i,q(t)) = \sum_n (pr_{x_i}(q(t)).po^0_{x_i}(q(t))) + Z(q(t)) \tag{7.7}$$

where: $pr_{x_i}(q(t))$ - the probability of the path section from the root vertex to the node x_i (set to one for the root vertex).

$po^1_{x_i}(q(t))$ - the probability of the path section from the '1' branch of a node encoding x_i to a terminal '1' node (or the probability value of the node beneath the '1' branch of x_i).

$po^0_{x_i}(q(t))$ - the probability of the path section from the '0' branch of a node encoding x_i to a terminal '1' node (or the probability value of the node beneath the '0' branch of x_i).

$Z(q(t))$ - the probability of paths from the root vertex to the terminal '1' node that do not go through a node encoding x_i.

n - all nodes encoding variable x_i in the BDD.

By substituting Equations 7.6 and 7.7 into Equation 7.5, the criticality function for each event can be expressed as:

$$G_i(q(t)) = \sum_n pr_{x_i}(q(t))[po^1_{x_i}(q(t)) - po^0_{x_i}(q(t))] \tag{7.8}$$

As this summation is over all the nodes encoding a particular event, the algorithm must calculate $pr_{x_i}(q)$, $po^1_{x_i}(q)$ and $po^0_{x_i}(q)$ for each node and record the values separately. For this reason, $pr[F]$, $po^1[F]$ and $po^0[F]$ are referred to as the corresponding values calculated for the nodes, which are then used in the evaluation of Equation 7.8 according to the encoded variable. Only when they have been found for each occurrence of the event in the BDD can the criticality function for that event be calculated.

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

The calculation of the system unavailability can be performed simultaneously, as po^1[F] is equivalent to the probability value of the node beneath the one branch of F, and po^0[F] is equivalent to the probability value of the node beneath its zero branch. Therefore at each stage of the calculation, both the path probabilities and the terms of Equation 7.3 are evaluated. The algorithm that encodes this calculation procedure is shown in Figure 7.2.

```
prob_value(F)
{
  F = ite(xi, J, K)
  Consider '1' branch:
    if (J = 1) then po1[F] = 1
    else
    {
      if (visited[J] = 1) then temp = pr[J]
                           else temp = 0
      pr[J] = qi.pr[F]
      po1[F] = prob_value(J)
      pr[J] = pr[J] + temp
    }
  Consider '0' branch:
    if (K = 0) then po0[F] = 0
    else
    {
      if (visited[K] = 1) then temp = pr[K]
                           else temp = 0
      pr[K] = (1-qi).pr[F]
      po0[F] = prob_value(K)
      pr[K] = pr[K] + temp
    }
  Calculate the probability value of the node:
    if (visited[F] = 0) then
    {
      probability[F] = qi.po1[F] + (1-qi).po0[F]
      visited[F] = 1
    }
    return(probability[F])
}
```

Figure 7.2: Algorithm to calculate the system unavailability and node path probabilities

The algorithm returns the probability value of the node under consideration, so the original calling function will receive the top event probability. The variable 'visited', which is used throughout the algorithm, is used to determine whether or not a node has previously been considered in the calculations. Due to sub-node sharing, a node may be reached by more than one path and its value of pr[F] needs to include the probabilities of all the possible path sections from the root vertex to that node. Therefore if a node has previously been visited and assigned a value of pr[F], this is held in a temporary variable, whilst the new value from the current path is used to calculate the increase in the values of pr[F] for the nodes beneath. For example, consider the section of a BDD shown in Figure 7.3:

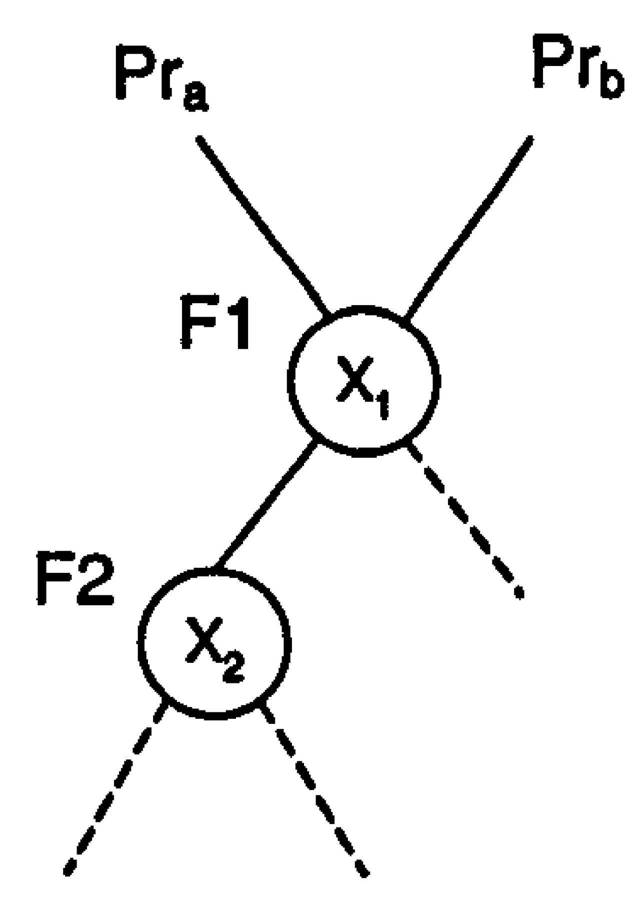


Figure 7.3: A section of a BDD, reachable by two different paths

If the probability of the path sections from the root vertex to node F1 is Pr_a by one route and Pr_b by a second route, then the total value of pr[F1] is $Pr_a + Pr_b$. However, the depth-first pass through the BDD would assign these values in two separate calculations. On the first visit to node F1, pr[F1] is evaluated as Pr_a. This is subsequently used to calculate the value pr[F2] (resulting in $Pr_a.q_1$) and values of pr[F] for any other nodes beneath F1 in the BDD. When node F1 is visited for the second time, the probability of the paths sections by the second route, Pr_b, must be used to calculate the increase in the values of pr[F] for the nodes beneath. The initial values are kept in temporary variables, and on returning through the BDD the values from the two separate passes are added together to give the correct value of pr[F] for each of the nodes. For example, for node F2 the second pass assigns a value of $Pr_b q_1$ to pr[F2]. Adding the two values together results in a total of $q_1.(Pr_a + Pr_b)$, which is equivalent to what would have been calculated had a single pass been made through the BDD using the total value for pr[F1] of $(Pr_a + Pr_b)$. However, although some nodes will be encountered more than once, it is still more efficient to carry out this depth-first calculation, rather than continually searching through the **ite** structure to find whether or not the nodes can be reached by alternative paths and then performing the calculations once the final values of pr[F] have been established.

The values of $po^1[F]$ and $po^0[F]$ are stored for each node and the algorithm shown in Figure 7.4 is used to calculate both the criticality functions for each of the basic events and the unconditional failure intensity of the system.

```
calc_criticality
{
  set G_i(q) = 0 for each event.
  for (each node F = ite(x_i, J, K) in the BDD)
  {
    G_i(q) = G_i(q) + pr[F].(po^1[F] - po^0[F])
  }
  w_sys = 0.0
  for (each event, x_i in the system)
  {
    w_sys = w_sys + G_i(q).w_i
  }
}
```

Figure 7.4: Algorithm to calculate the event criticality functions and the system unconditional failure intensity

The calculation procedure is demonstrated in the following section, by means of a worked example.

7.4 Worked Example

To demonstrate the calculation of the system unavailability and unconditional failure intensity, consider the BDD shown in Figure 7.5.

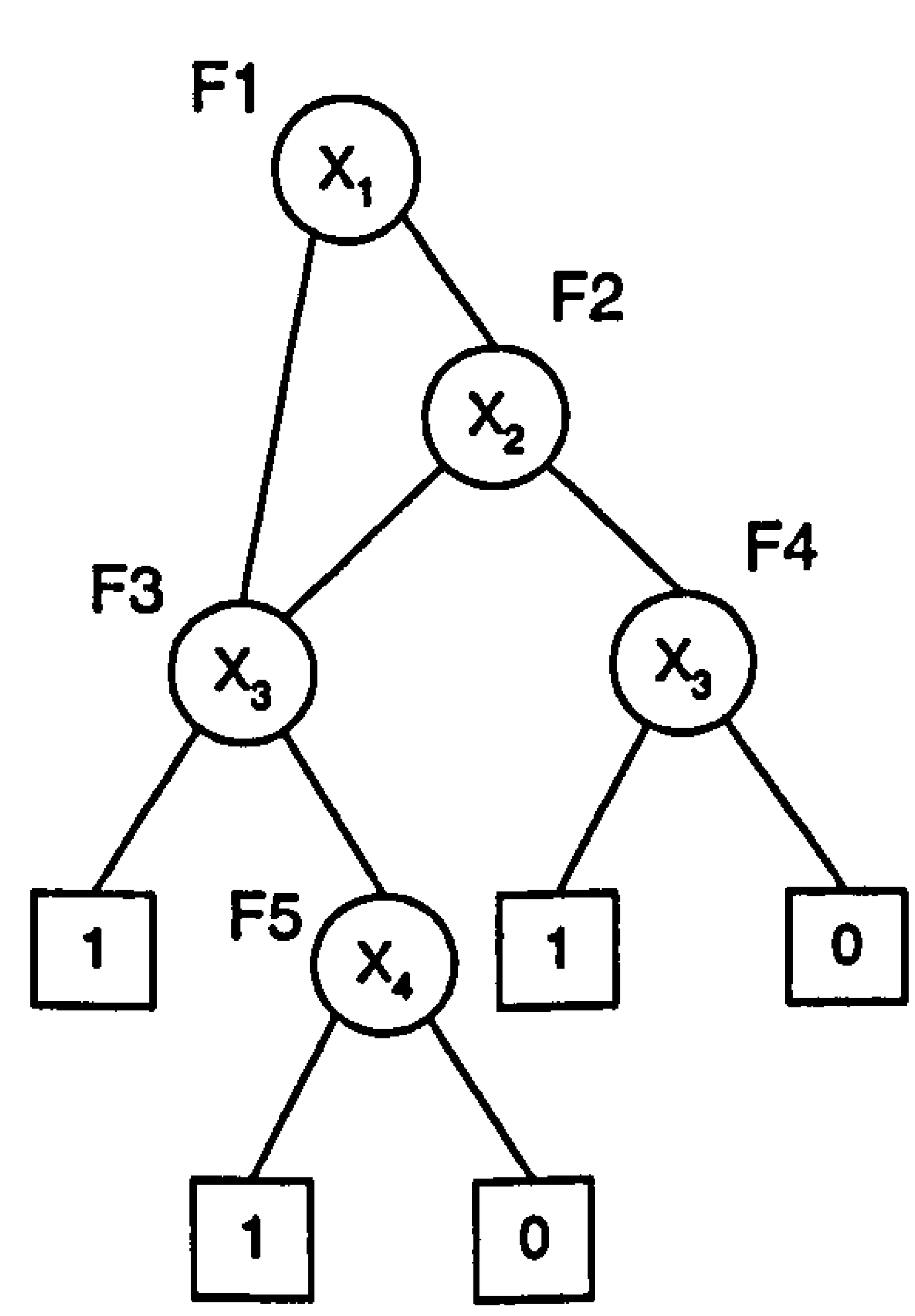


Figure 7.5: Example Binary Decision Diagram

There are three paths through the BDD that end with a node that has terminal vertices on both branches. These paths must be considered in turn.

1. The one branch of node F1:

$$F1 - F3 - F5$$

2. The zero branch of node F1, which splits into two sub-branches at node F2:

F1 - F2 - with sub-branches (a) - F3 - F5

(b) - F4

One Branch: F1 - F3 - F5

Starting at the root vertex F1, the value 1.0 is assigned to pr[F1]. Node F3 is reached by descending the one branch of F1 and the current value of pr[F3] can then be calculated (this will not be the total value, as the node can be reached by another path):

$$\begin{aligned} pr[F3] &= pr[F1].q_1 \\ &= q_1 \end{aligned}$$

The current value of pr[F5] can also be evaluated:

$$\begin{aligned} pr[F5] &= pr[F3].(1-q_3) \quad \text{[F5 lies on the zero branch of F3]} \\ &= q_1(1-q_3) \end{aligned}$$

Having reached the foot of this BDD branch, the procedure continues by working back up through the BDD, calculating $po^1[F]$, $po^0[F]$ and probability values for the nodes. As the probabilities of the paths beneath branches leading directly to one and zero terminal vertices are one and zero respectively, the probability value of node F5 is simply q_4.

Node F3 has a terminal one vertex on its one branch (which therefore has a probability value of one) and the probability of the paths beneath its zero branch is equal to the probability value of the node beneath, i.e. node F5. Therefore the probability value of F3 is calculated as:

$$\begin{aligned} P[F3] &= q_3.po^1[F3] + (1-q_3).po^0[F3] \\ &= q_3 + (1-q_3).P[F5] \\ &= q_3 + (1-q_3)q_4 \end{aligned}$$

This value therefore becomes the probability of the paths beneath the one branch of node F1 and concludes the calculations on this branch.

Zero Branch: F1 - F2 -

The probability of the paths from the root vertex to node F2 is given by:

$$\begin{aligned} pr[F2] &= pr[F1].(1-q_1) \\ &= 1-q_1 \end{aligned}$$

There are two possible paths through the BDD from node F2, but as the one branch of any node is always explored before the zero branch, this is considered first.

Sub-Branch (a): - F3 - F5

Moving down the BDD to node F3, it is noted that it has already been visited, so current values of pr[F] of both F3 and F5 are temporarily stored whilst new ones are utilised. The additional probabilities arising from this path are now calculated:

$$\begin{aligned} pr[F3] &= pr[F2].q_2 \\ &= (1-q_1)q_2 \end{aligned}$$

and,

$$\begin{aligned} pr[F5] &= pr[F3].(1-q_3) \\ &= q_2(1-q_1)(1-q_3) \end{aligned}$$

These are then added to the previous values to give totals of:

$$pr[F3] = q_1 + (1-q_1)q_2$$

and,

$$pr[F5] = (q_1 + (1-q_1)q_2).(1-q_3)$$

The path probabilities $po^1[F]$ and $po^0[F]$ and the probability values for these nodes are not re-evaluated, as they do not change. The probability of the paths below the one branch of F2 is therefore assigned the probability value of node F3 that has already been calculated. The second sub-branch of node F2 is considered before returning to node F1.

Sub-Branch (b): - F4

Descending the zero branch of node F2 allows the calculation of the probability of the paths from the root vertex to node F4:

$$\begin{aligned} pr[F4] &= pr[F2].(1-q_2) \\ &= (1-q_1).(1-q_2) \end{aligned}$$

The probability value of node F4 is simply q_3. This value is assigned to the probability of the paths beneath the zero branch of node F2 and the probability value of F2 can then be calculated as:

$$\begin{aligned} P[F2] &= q_2.\, po^1[F2] + (1-q_2).po^0[F2] \\ &= q_2(q_3 + (1-q_3)q_4) + (1-q_2)q_3 \\ &= q_3 + q_2q_4(1-q_3) \end{aligned}$$

Finally, the probability value of the root vertex is calculated, which gives the top event probability:

$$Q_{sys} = P[F1] = q_1.po^1[F1] + (1-q_1).po^0[F1]$$
$$= q_1(q_3 + (1-q_3)q_4) + (1-q_1).(q_3 + q_2q_4(1-q_3))$$
$$= q_3 + q_4(1-q_3).[q_1 + q_2(1-q_1)]$$

The calculation results are summarised in columns 3 to 6 of Table 7.1.

Node	Variable	Probability value (P)	Pr	Po^1	Po^0	Criticality
F1	x_1	$q_3 + q_4(1-q_3).[q_1+ q_2(1-q_1)]$	1	$q_3 + q_4(1-q_3)$	$q_3 + q_2q_4(1-q_3)$	$q_4(1-q_3)(1-q_2)$
F2	x_2	$q_3 + q_2q_4(1-q_3)$	$1-q_1$	$q_3 + q_4(1-q_3)$	q_3	$q_4(1-q_1)(1-q_3)$
F3	x_3	$q_3 + q_4(1-q_3)$	$q_1 + q_2(1-q_1)$	1	q_4	$(1-q_4).[q_1+q_2(1-q_1)]$
F4	x_3	q_3	$(1-q_1)(1-q_2)$	1	0	$(1-q_1)(1-q_2)$
F5	x_4	q_4	$(1-q_3).[q_1+q_2(1-q_1)]$	1	0	$(1-q_3).[q_1+q_2(1-q_1)]$

Table 7.1: Quantitative results for the BDD in Figure 7.5.

The final column of Table 7.1 shows the criticality values that are calculated according to the algorithm in Figure 7.4. This gives the correct criticality functions for variables x_1, x_2 and x_4 as they each appear only once in the BDD. However, as variable x_3 is encoded in both nodes F3 and F4, their criticality values must be added to give the total criticality function for x_3:

$$G_3 = (1-q_4).[q_1+q_2(1-q_1)] + (1-q_1)(1-q_2)$$
$$= 1 - q_4[q_1 + q_2(1-q_1)]$$

The final stage of the analysis is to calculate the system unconditional failure intensity, which is given by:

$$w_{sys}(t) = G_1w_1 + G_2w_2 + G_3w_3 + G_4w_4$$
$$= w_1q_4(1-q_3)(1-q_2) + w_2q_4(1-q_1)(1-q_3) + w_3(1-q_4[q_1 + q_2(1-q_1)]) + w_4(1-q_3).[q_1+q_2(1-q_1)]$$

The analysis so far has considered BDDs containing only basic events. In the following sections this is extended to incorporate both complex events and modules.

7.5 Incorporating Complex Events and Modules Into the Analysis

The following sections describe the extension of the current quantification methods to consider BDDs encoding complex events and/or modular events. The aim of the analysis is to obtain not only the system unavailability and unconditional failure intensity, but to be able to extract the criticality functions for the basic events that contribute to the complex events and modules. This is essential, as although reduction and modularisation may be used to help construct the BDDs, it must be possible to analyse the system in terms of its original components.

7.5.1 Syntax

When modules are identified and extracted from a fault tree, the result is a set of subtrees, which together describe the original system. Each of these trees is converted to a BDD and the analysis is performed on the resulting set of BDDs. The BDD that represents the top event, and from which the top event probability can be calculated, is referred to as the 'primary' BDD. The remaining BDDs encode the structure of the subtrees and are labelled according to the 'modular event' that replaces the subtree in the higher-level fault tree structure.

7.5.2 Overview of the Calculation Procedure

The calculation process starts at the root vertex of the primary BDD and proceeds down through the branches, calculating the probabilities of the paths from the root vertex to each of the nodes. The unavailability of each encoded event is required as it enables the calculation of pr[F] for the nodes beneath. Therefore, the probabilities of both the complex and modular events are necessary for the analysis.

Values of $po^1[F]$ and $po^0[F]$ are calculated for the nodes on the way up through the primary BDD. If a node is encountered that encodes either a complex or modular event, then the complex event or module must be further analysed to assign appropriate values of pr[F], $po^1[F]$ and $po^0[F]$ to its component nodes. This allows the calculation of the criticality functions of the basic events with the complex events and modules.

The criticality functions of basic events encoded within the primary BDD are calculated according to Equation 7.8 at the end of the analysis, once the path probabilities of the nodes have been evaluated. The criticality functions of *all* the basic events are then used together with their unconditional failure intensities to calculate the system unconditional failure intensity.

It is also possible to calculate $w_{sys}(t)$ by considering only the events encoded in the primary BDD. This would require both the criticality functions of any encoded modular and complex events and their unconditional failure intensities. Although these are relatively simple to calculate[33], they are values that have no further use in the analysis. Instead, the criticality functions of all basic events are calculated, which allows the analysis of the contributions to system failure through component or basic event importance measures.

The techniques for calculating the complex and modular event probabilities and the criticality functions of their constituent basic events are described in the following sections.

7.5.3 Unavailability of Complex and Modular Events

The probabilities of the complex events are used during the depth-first pass of the BDD to calculate the values pr[F], po^1[F] and po^0[F] for other nodes in the BDD.

The probabilities of the complex events are calculated as they are formed, which ensures the process is as efficient as possible. Determining their probabilities is a straightforward procedure, as they are only a combination of two component events. The calculation depends on whether the events were combined under an 'AND' gate or an 'OR' gate, so for a complex event X_c that has constituent events X_1 and X_2, the unavailability is given by:

$$\text{'AND' Gate: } q_c = q_1 q_2 \tag{7.9}$$

$$\text{'OR' Gate: } q_c = q_1 + q_2 - q_1 q_2 \tag{7.10}$$

The probabilities of the modular events are not calculated before the quantitative analysis takes place, but are determined as and when required during the analysis (once a value has been calculated it is stored for later use). The calculation of the unavailability of a modular event is effectively that of finding the probability of the 'top event' of the module. A depth-first algorithm (similar to the one shown in Figure 7.2) is used, which sums the probabilities of the disjoint paths through the module's BDD. If another modular event, x_i, is encoded within the module, the algorithm identifies its root vertex, M[x_i], and proceeds to call itself to calculate the required probability. Thus, the unavailability of modules encoding only basic and complex events will necessarily be evaluated first. The algorithm is shown in Figure 7.6.

```
module_prob(F)
{
  F = ite(xi, J, K)

  Consider '1' branch:

    if (J = 1) then po1[F] = 1
    else po1[F] = module_prob(J)

  Consider '0' branch:

    if (K = 0) then po0[F] = 0
   else po0[F] = module_prob(K)

  Calculate and return probability value of node:

    if (xi is a modular event whose probability is
    unknown) then qi = module_prob(M[xi])

    probability[F] = qi.po1[F] + (1-qi).po0[F]
 return(probability[F])
}
```

Figure 7.6: Algorithm for calculating the probability of a module

The calculation procedures for evaluating the probabilities of the complex and modular events are therefore relatively straightforward. At this stage they could be used alone to determine the system unavailability by performing the depth-first calculations (as in the algorithm for analysing single BDD structures in Figure 7.2) on the primary BDD only[34]. The calculation of the basic events' criticality functions does however require further analysis. This is discussed in the following sections.

7.5.4 Criticality of Basic Events Within Complex Events

Once the path probabilities have been calculated for a node encoding a complex event, that complex event must be further analysed by assigning appropriate values of $pr_{x_i}(q)$, $po^1_{x_i}(q)$ and $po^0_{x_i}(q)$ to its component events. These are required so that the criticality functions of the basic events can be evaluated. Consider a node encoding the complex event X_c, as shown in Figure 7.7.

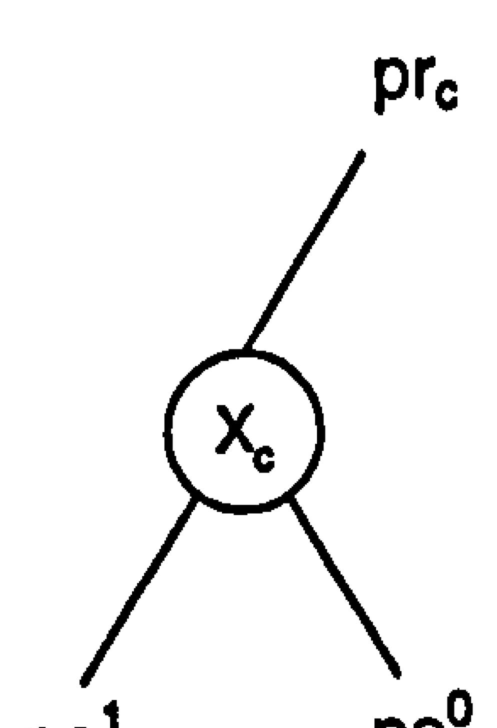


Figure 7.7: A complex event node within a BDD

The two events that combine to form this complex event are joined either by an 'AND' gate or an 'OR' gate, which gives the possible ite structures and corresponding BDDs as shown in Figure 7.8.

'AND': $X_c = X_1.X_2$

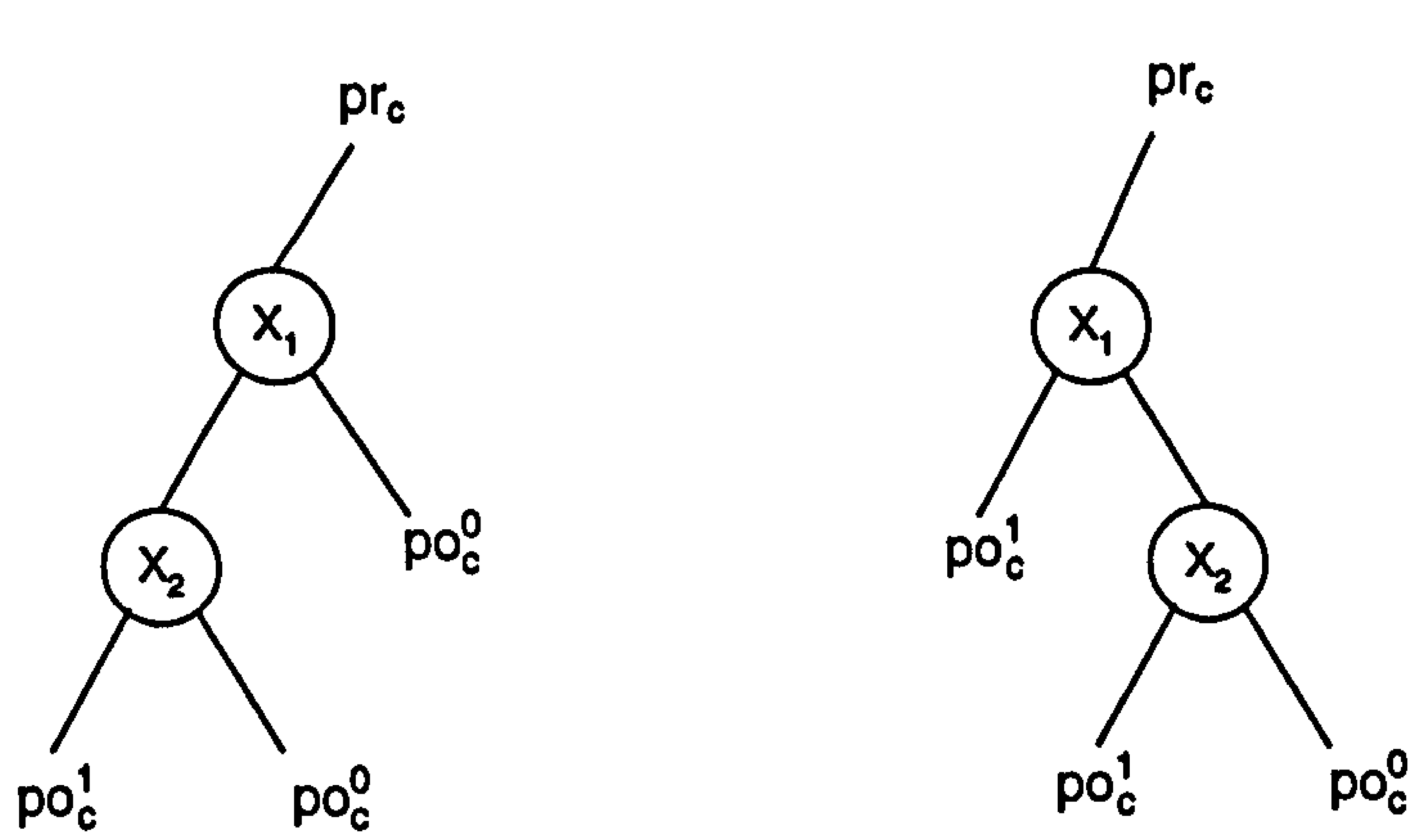
$X_c = \text{ite}(X_1, \text{ite}(X_2, 1, 0), 0)$

'OR': $X_c = X_1 + X_2$

$X_c = \text{ite}(X_1, 1, \text{ite}(X_2, 1, 0))$

Figure 7.8: The possible BDD structures of a complex event

The complex event node effectively replaces one of these structures in the original BDD (either the primary BDD or the BDD of a module). In order to evaluate the path probabilities of the nodes encoding these component events, the terminal one vertices are simply replaced with the probability of the paths below the one branch of the complex event node and the terminal zero vertices are replaced with the probability of the paths below the zero branch of the complex event node. The probability of the paths preceding the root vertex does not have the usual value of one, but takes the value of pr[F] of the complex event node (pr_c). This is shown in Figure 7.9.



(a) $X_c = X_1 . X_2$

(b) $X_c = X_1 + X_2$

Figure 7.9: The complex event structure

Using Figure 7.9, the values of $pr_{x_i}(q)$, $po^1_{x_i}(q)$ and $po^0_{x_i}(q)$ can be calculated for the variables X_1 and X_2. The resulting expressions are shown in Equations 7.11 - 7.22.

'AND' gate:

X_1:

$$pr_1 = pr_c \tag{7.11}$$

$$po_1^1 = q_2.po_c^1 + (1-q_2).po_c^0 \tag{7.12}$$

$$po_1^0 = po_c^0 \tag{7.13}$$

X_2:

$$pr_2 = pr_c.q_1 \tag{7.14}$$

$$po_2^1 = po_c^1 \tag{7.15}$$

$$po_2^0 = po_c^0 \tag{7.16}$$

'OR' gate:

X_1:

$$pr_1 = pr_c \tag{7.17}$$

$$po_1^1 = po_c^1 \tag{7.18}$$

$$po_1^0 = q_2.po_c^1 + (1-q_2).po_c^0 \tag{7.19}$$

X_2:

$$pr_2 = pr_c.(1-q_1) \tag{7.20}$$

$$po_2^1 = po_c^1 \tag{7.21}$$

$$po_2^0 = po_c^0 \tag{7.22}$$

As the events X_1 and X_2 may be either basic events or other complex events, this process is repeated until values have been calculated for all contributing basic events. The criticality functions of the basic events are then calculated according to Equation 7.8. The algorithm implementing this method is shown in Figure 7.10.

```
complex_calc(xc)
{
  xc = x1 <op> x2
  Calculate probabilities:

    pr[x1] = pr[xc]
    po^1[x2] = po^1[xc]
    po^0[x2] = po^0[xc]

    if (<op> = 'AND')
    {
      po^1[x1] = q2.po^1[xc] + (1-q2).po^0[xc]
      po^0[x1] = po^0[xc]
      pr[x2] = pr[xc].q1
    }
    if (<op> = 'OR')
    {
      po^1[x1] = po^1[xc]
      po^0[x1] = q2.po^1[xc] + (1-q2).po^0[xc]
      pr[x2] = pr[xc].(1-q1)
    }

  If contributing events are basic, then calculate criticality,
  otherwise call function again:

    if (x1 is a basic event) then G1 = G1 + pr[x1].(po^1[x1] - po^0[x1])
    else complex_calc(x1)

    if (x2 is a basic event) then G2 = G2 + pr[x2].(po^1[x2] - po^0[x2])
    else complex_calc(x2)
}
```

Figure 7.10: Algorithm for the calculation of the criticality functions of basic events within complex events

7.5.4.1 Repeated Complex Events

Any complex event can appear more than once in the BDD, resulting in new values of $pr_{x_i}(q)$, $po_{x_i}^1(q)$ and $po_{x_i}^0(q)$ being calculated for its component events on each occasion. The criticality function for each of the contributing basic events must therefore be calculated in stages, using the newly assigned values each time. Once this additional criticality value has been calculated for each of the contributing basic events, it is added to the current value so that it is calculated as the analysis proceeds, rather than as a separate procedure at the end of the analysis as is the case for the basic events in the primary BDD.

7.5.5 Criticality of Basic Events Within Modules

Modular events are dealt with in a similar way to complex events. Once the path probabilities of the modular event node are known, the module is further analysed to determine the path probabilities of its component nodes. These probabilities must be assigned as they would have been, had the module not been replaced by the single modular event. In order to do this, the values of $po^1[F]$ and $po^0[F]$ of the modular event node replace any terminal one and zero vertices within the module, and the probability of the paths preceding the root vertex of the module is assigned the value of pr[F] of the modular event node. This is shown in Figure 7.11.

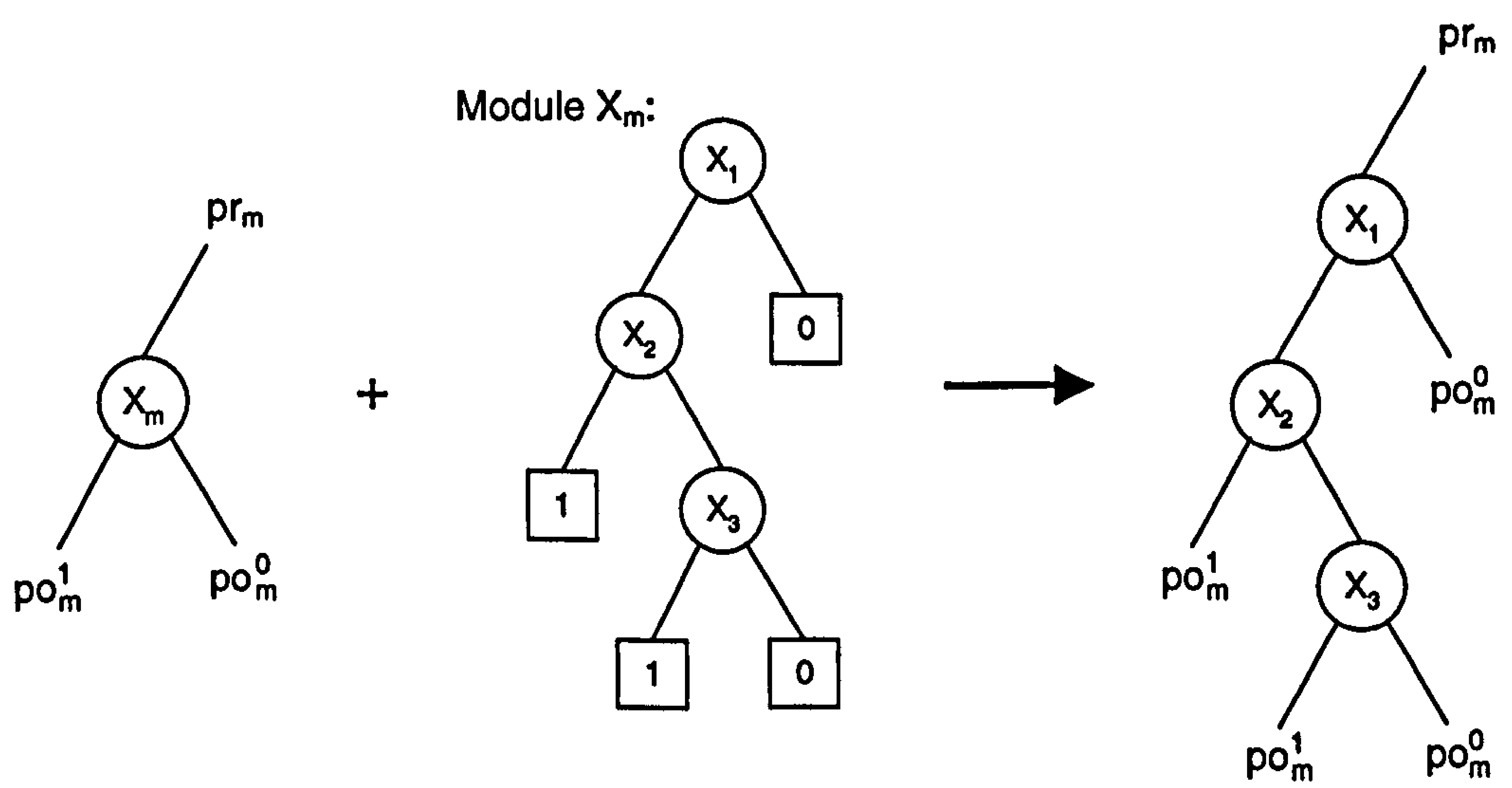


Figure 7.11: Replacing a modular event with the entire module structure

Unlike complex events, the structure of modules is not fixed. They can contain any number of events (basic, complex, or indeed other modular events), connected by any number of gates. Therefore, the path probabilities are assigned to the nodes by means of a depth-first process, which is capable of dealing with any BDD structure. The method is very similar to that used for analysing a single BDD, the algorithm for which is shown in Figure 7.2. The difference is that whenever a terminal node is encountered, the probability of the paths below either the one or the zero branch of the modular event node is used, rather than the terminal vertex probability values of one and zero. Obviously, pr[F] of the root vertex will also be set to equal the probability of the paths preceding the modular event node.

As with complex events, the calculations required to obtain the path probabilities for the nodes within the module must be repeated for each occurrence of the modular event in the BDD. These values are used to calculate the additional contributions to the criticality functions of the basic events that arise due to the further occurrences of the modular event.

7.6 The Algorithm for Incorporating Complex Events and Modules Into the Analysis

The analysis of the primary BDD is conducted in a similar manner to the analysis of single BDD structures, except for the processes instigated when a modular or complex event is encountered. As the probabilities of complex events are calculated as they are formed, they are treated as basic events when descending the BDD. However, once the path probabilities have been evaluated for a complex event node, the algorithm 'complex_calc' (Figure 7.10) is used to calculate the criticality functions of its constituent basic events.

If a modular event is encountered when descending the BDD, the algorithm 'module_prob' (as shown in Figure 7.6) is called to calculate the probability of the modular event if it has not already been evaluated. When ascending the BDD, a depth-first algorithm is used to calculate the criticality functions of the basic events that contribute to the module.

As the process for determining the path probabilities of the nodes within a module is so similar to the procedure used for dealing with the primary BDD, a separate algorithm is not needed. The existing method is simply extended to include both options. The resulting algorithm is shown in Figure 7.12 and deals with the primary BDD or any of its modules, depending upon how the parameters are set. It requires three initial variables, which are set each time the function is called: F, subtree and m_node. These are described below:

F: The node currently being considered.

subtree: The variable that determines whether the node belongs to a module or the primary BDD - set to '1' if it occurs in the BDD of a module, '0' otherwise.

m_node: If node F belongs to a module, m_node is the modular event that has replaced that module structure in the higher-level BDD.

Further variables that are used within the algorithm are:

visited[F]: Determines whether or not node F has previously been considered in the calculations - set to '1' if it has been considered, '0' otherwise.

$M(x_i)$: The root node of the module replaced by the modular event x_i.

If 'subtree' is set to zero, the algorithm performs calculations on the primary BDD, resulting in the calculation of the top event probability and values of pr[F], po^1[F] and po^0[F] for each of its nodes.

If 'subtree' is set to one, the calculations will determine pr[F], po^1[F] and po^0[F] for nodes in the module and upon exiting the module, the algorithm evaluates the criticality functions for each of its basic events.

When the algorithm is initialised, the node to be considered is set as the root vertex of the primary BDD and the variable 'subtree' is set to zero. The algorithm then performs all the necessary calculations and returns the top event probability.

```
calc_prob(F, subtree, m_node)
{
  F = ite(x_i, J, K)

    if (x_i is a modular event whose probability is
    unknown), then q_i = module_prob(M[x_i])

  Consider '1' branch:
    if (J = 1) then
      if (subtree = 0) then po^1[F] = 1
      else  po^1[F] = po^1[m_node]
    else
    {
      if (visited[J] = 1) then temp = pr[J]
                          else temp = 0
      pr[J] = q_i.pr[F]
      po^1[F] = calc_prob(J, subtree, m_node)
      pr[J] = pr[J] + temp
    }

  Consider '0' branch:

    if (K = 0) then
      if (subtree = 0) then po^0[F] = 0
      else po^0[F] = po^0[m_node]
    else
    {
      if (visited[K] = 1) then temp = pr[K]
                          else temp = 0
      pr[K] = (1-q_i).pr[F]
      po^0[F] = calc_prob(K, subtree, m_node)
      pr[K] = pr[K] + temp
    }

  Calculate the probability value of the node:

    if (visited[F] = 0) then
    {
      probability[F] = q_i.po^1[F] + (1-q_i).po^0[F]
      visited[F] = 1
    }

  If x_i is a complex or modular event, calculate the
  additional criticality of its component basic events:

    if(x_i is a modular event)
    {
      Calculate pr and po values for events within the
      module; the root node of the module is M[x_i].

        set pr[M[x_i]] = pr[F]
        set subtree = 1
        calc_prob(M[x_i], subtree,  F)

        for(all basic event nodes in the module)
        {
          G[event] = G[event] +pr[node].(po^1[node] -
                          po^0[node])
        }
    }
    else if (x_i is a complex event)
    {
      complex_calc[x_i]
    }
   return(probability[F])
}
```

Figure 7.12: The algorithm for the quantitative analysis of BDDs encoding modular and complex events

7.7 Worked Example of the Calculation Procedure

The method of dealing with BDDs encoding complex and modular events is demonstrated with the following example. Consider the BDDs shown in Figure 7.13, where (a) shows the 'primary' BDD for the fault tree containing the top event, and (b) and (c) show the BDDs for modules M1 and M2 contained within the primary BDD. Note that each node in the set of

BDDs is labelled uniquely, for ease of identification. The data for the complex events, which shows their constituent events and the gate type under which they were combined, is given in Table 7.2.

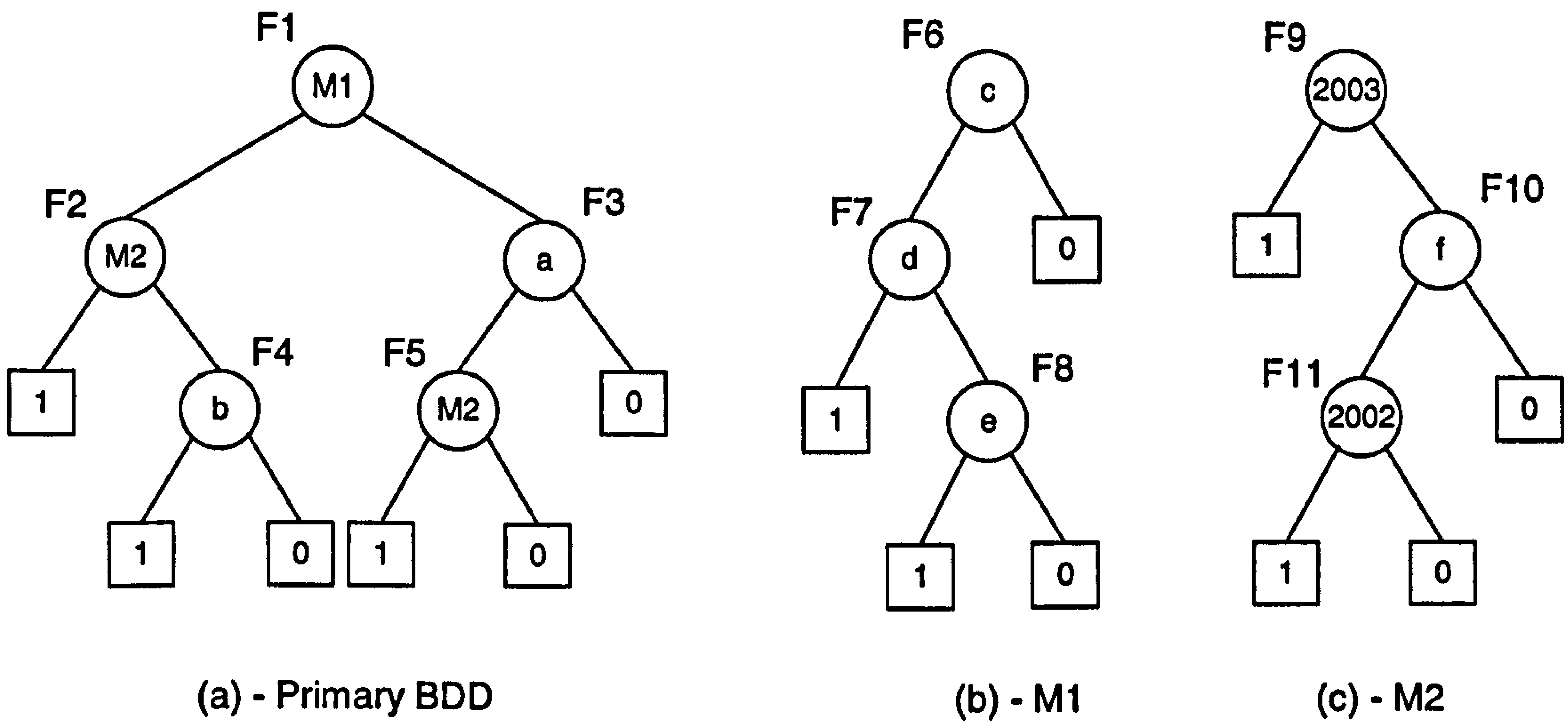


Figure 7.13: Example BDD set

Complex event	Gate value	Event 1	Event 2
2000	OR	g	k
2001	OR	2000	j
2002	AND	i	h
2003	AND	l	2001

Table 7.2: Complex event data

The basic event data (unavailability, q_i, and unconditional failure intensity, w_i, of each event) are shown in Table 7.3. The probabilities of the complex events are calculated as they are formed, according to Equations 7.9 and 7.10 and are also shown in Table 7.3. The unconditional failure intensities of the complex events are not required for the analysis.

Event	a	b	c	d	e	f	g	h
q_i	0.008	0.005	0.008	0.006	0.007	0.010	0.003	0.002
w_i (hr^{-1})	3.92×10^{-6}	2.88×10^{-6}	1.94×10^{-5}	9.90×10^{-7}	4.67×10^{-5}	7.23×10^{-6}	1.10×10^{-5}	8.30×10^{-7}

Event	i	j	k	l	2000	2001	2002	2003
q_i	0.004	0.009	0.005	0.015	7.985×10^{-3}	1.691×10^{-2}	8.000×10^{-6}	2.537×10^{-4}
w_i (hr^{-1})	1.65×10^{-5}	4.20×10^{-5}	5.58×10^{-7}	2.15×10^{-6}	-	-	-	-

Table 7.3: Event data for the BDDs shown in Figure 7.13

There are two paths through the primary BDD that end with a node that has terminal vertices on both branches; the first starts with node F1 and includes the nodes on its one branch (F2 and F4), the second path also begins at node F1, but comprises of the nodes on the zero branch (F3 and F5). The analysis is therefore considered in four stages - descending the one branch of F1, ascending the one branch, descending the zero branch of F1 and finally ascending the zero branch.

Descending the One Branch of F1

The process begins at the root vertex F1 with the value 1.0 assigned to pr[F1]. No further calculations can be undertaken until the unavailability of the encoded modular event, M1, is known. Therefore the procedure for calculating the probability of a module is implemented.

Unavailability of M1:

Evaluating the unavailability of the module M1 simply requires the summation of the probabilities of the disjoint paths through its BDD (Figure 7.13(b)). The algorithm shown in Figure 7.6 performs this procedure efficiently.

The disjoint paths through the BDD are:

1. c.d
2. $c.\bar{d}.e$

Therefore the unavailability of the module is given by:

$$q_{M1} = q_c.q_d + q_c.(1 - q_d).q_e$$
$$= 1.037\text{x}10^{-4}$$

Having calculated the module's probability, the calculations in the primary BDD can continue.

Descending the one branch of node F1 in the primary BDD leads to node F2. The probability of the paths from the root vertex to this node is calculated as follows:

$$pr[F2] = q_{M1}.pr[F1]$$
$$= 1.037\text{x}10^{-4}$$

Node F2 also encodes a modular event whose probability is unknown. This must be calculated before moving down the BDD branches.

Unavailability of M2:

The unavailability of module M2 is calculated in the same manner as M1, so is considered independently of the primary BDD. Although it contains complex events, it is treated in exactly the same way at this stage, as their probabilities have already been calculated. The disjoint paths through the BDD are:

1. 2003
2. $\overline{2003}$.f.2002

Therefore the unavailability of the module is given by:

$$q_{M1} = q_{2003} + (1 - q_{2003})q_f.q_{2002}$$
$$= 2.538\text{x}10^{-4}$$

The calculations in the primary BDD can now continue.

Having calculated the unavailability of the modular event M2, the nodes on the branches of node F2 can now be examined. As the one branch is a terminal one vertex it needs no further consideration, except to set po^1[F2] equal to 1.0. The node F4 lies on the zero branch, so is considered next.

The probability of the paths from the root vertex to node F4 is calculated as follows:

$$pr[F4] = pr[F2].(1-q_{M2})$$
$$= 1.036\text{x}10^{-4}$$

Both the one and zero branches of F4 lead to terminal nodes, therefore po^1[F4] and po^0[F4] are set to 1.0 and 0.0 respectively and the process of moving back up through the BDD starts.

Ascending the One Branch of F1

The probability values of the nodes are calculated on the way back up through the BDD branches. Also, any nodes encoding complex or modular events are explored so that the criticality functions of their constituent basic events can be calculated.

The node currently being considered is F4, whose probability value is simply the unavailability of the node variable 'b', which is equal to 0.005. This value also becomes the probability of the paths below the zero branch of node F2, po^0[F2].

As node F2 encodes the modular event M2, and the path probabilities for this node have all been calculated, the module must be explored and probability values assigned to its nodes.

Assigning Values to the Nodes Within Module M2

The nodes within the module's BDD are assigned probabilities as they would have been, had the module not been replaced in the higher-level BDD structure by the single modular event. Therefore pr[F9] is given the value of pr[F2] (i.e. 1.037×10^{-4}), as detailed earlier. The probabilities of the paths below branches that lead to terminal one and zero vertices are assigned the values po^1[F2] (i.e. 1.0) and po^0[F2] (i.e. 0.005) respectively.

The calculations are summarised in Table 7.4.

Node	Event	Pr	Po^1	Po^0	Probability value
F9	2003	1.037×10^{-4}	1.0	5.000×10^{-3}	$q_{2003} + po^0[F9].(1-q_{2003})$ $= 5.253\times10^{-3}$
F10	f	$pr[F9].(1-q_{2003})$ $= 1.036\times10^{-4}$	5.008×10^{-3}	0.005	$po^1[F10].q_f + po^0[F10].(1-q_f)$ $= 5.000\times10^{-3}$
F11	2002	$pr[F10].q_f$ $=1.036\times10^{-6}$	1.0	0.005	$q_{2002} + po^0[F11].(1-q_{2002})$ $= 5.008\times10^{-3}$

Table 7.4: Assigning values to the nodes of module M2

The criticality functions of the basic events within this module are also evaluated. For event 'f', the values of node F10 are used, giving:

$$G_f = 1.036\times10^{-4}.(5.008\times10^{-3} - 0.005)$$
$$= 8.250\times10^{-10}$$

In order to calculate the criticality functions of the basic events that form the complex events 2002 and 2003 (which has further complex events 2000 and 2001 as components), Equations 7.11-7.22 are used to evaluate $pr_{x_i}(q)$, $po^1_{x_i}(q)$ and $po^0_{x_i}(q)$ for each basic event. The results of applying these equations, together with the calculated criticality values are shown in Table 7.5.

Complex event	Gate type	Component event	pr	po^1	po^0	Criticality
			of the component event			
2002	AND	$X_1 = i$	$pr_{2002} = 1.036x10^{-6}$	$q_h.po^1_{2002} + (1-q_h). po^0_{2002} = 6.990x10^{-3}$	$po^0_{2002} = 0.005$	$2.062x10^{-9}$
		$X_2 = h$	$pr_{2002}.q_i = 4.146x10^{-9}$	$po^1_{2002} = 1.0$	$po^0_{2002} = 0.005$	$4.125x10^{-9}$
2003	AND	$X_1 = l$	$pr_{2003} = 1.037x10^{-4}$	$q_{2001}.po^1_{2003} + (1-q_{2001}). po^0_{2003} = 2.183x10^{-2}$	$po^0_{2003} = 5.000x10^{-3}$	$1.745x10^{-6}$
		$X_2 = 2001$	$pr_{2003}.q_l = 1.555x10^{-6}$	$po^1_{2003} = 1.0$	$po^0_{2003} = 5.000x10^{-3}$	-
2001	OR	$X_1 = 2000$	$pr_{2001} = 1.555x10^{-6}$	$po^1_{2001} = 1.0$	$q_j.po^1_{2001} + (1-q_j).po^0_{2001} = 1.396x10^{-2}$	-
		$X_2 = j$	$pr_{2001}.(1-q_{2000}) = 1.543x10^{-8}$	$po^1_{2001} = 1.0$	$po^0_{2001} = 5.000x10^{-3}$	$1.535x10^{-6}$
2000	OR	$X_1 = g$	$pr_{2000} = 1.555x10^{-6}$	$po^1_{2000} = 1.0$	$q_k.po^1_{2000} + (1-q_k).po^0_{2000} = 1.889x10^{-2}$	$1.526x10^{-6}$
		$X_2 = k$	$pr_{2000}.(1-q_g) = 1.550x10^{-6}$	$po^1_{2000} = 1.0$	$po^0_{2000} = 1.396x10^{-2}$	$1.526x10^{-6}$

Table 7.5: Calculation of the criticality of basic events within complex events 2002 and 2003

Having calculated the current criticality values of the basic events within module M2, the calculation process continues in the primary BDD.

The probability value of node F2 is now calculated. This also gives the probability below the one branch of the root vertex, F1:

$$po^1[F1] = P[F2] = q_{M2}.po^1[F2] + (1-q_{M2}).po^0[F2]$$
$$= 5.253x10^{-3}$$

This concludes the second stage of the analysis - the current calculated values are shown in Table 7.6.

	Node	Event	One branch	Zero branch	Pr	Po^1	Po^0	Probability value
	F1	M1	F2	F3	1.0	5.253×10^{-3}		
	F2	M2	1	F4	$pr[F1].q_{M1} = 1.037\times10^{-4}$	1.0	0.005	$po^1.q_{M2} + po^0.(1-q_{M2}) = 5.253\times10^{-3}$
	F3	a	F5	0				
	F4	b	1	0	$pr[F2].(1-q_{M2}) = 1.036\times10^{-4}$	1.0	0.0	$q_b = 0.005$
	F5	M2	1	0				
M2:	F9	2003	1	F10	1.037×10^{-4}	1.0	5.000×10^{-3}	5.253×10^{-3}
	F10	f	F11	0	1.036×10^{-4}	5.008×10^{-3}	0.005	5.000×10^{-3}
	F11	2002	1	0	1.036×10^{-6}	1.0	0.005	5.008×10^{-3}

Table 7.6: Current calculated values for the primary BDD and module M2

Descending the Zero Branch of F1

As the probabilities of both modular events have been determined, the calculations required for descending this branch of the BDD are straightforward. They simply involve calculating the probability of the path sections from the root vertex to nodes F3 and F5.

$$pr[F3] = pr[F1].(1-q_{M1})$$
$$= 9.999\times10^{-1}$$

and,

$$pr[F5] = pr[F3].q_a$$
$$= 7.999\times10^{-3}$$

The probabilities below the one and zero branches of node F5 can be set to 1.0 and 0.0 respectively as they lead to terminal one and zero vertices. The final stage of the analysis now begins.

Ascending the Zero Branch of F1

As node F5 encodes the second occurrence of the modular event M2, additional criticality values must be calculated for the basic events within the module.

Assigning Values to the Events Within Module M2 - Second Occurrence

The probability preceding node F9 is set to the value of pr[F5] (7.999×10^{-3}) and the probabilities of the paths below branches that lead to terminal one and zero vertices are assigned the values po^1[F5] and po^0[F5] respectively (simply 1.0 and 0.0).

The calculations are repeated with these new values for all nodes within M2, overwriting the previous results. The summarised calculations are shown in Table 7.7.

Node	Event	Pr	Po^1	Po^0	Probability value
F9	2003	7.999×10^{-3}	1.0	8.000×10^{-8}	$q_{2003} + po^0(1-q_{2003})$ $= 2.538\times10^{-4}$
F10	f	$pr[F9].(1-q_{2003})$ $= 7.997\times10^{-3}$	8.000×10^{-6}	0.0	$po^1.q_f + po^0.(1-q_f)$ $= 8.000\times10^{-8}$
F11	2002	$pr[f10].q_f$ $= 7.997\times10^{-5}$	1.0	0.0	$q_{2002} = 8.000\times10^{-6}$

Table 7.7: Assigning values to the nodes of module M2

Additional criticality values of the basic events within the module are now evaluated. For event 'f', the values of node F10 are used, giving:

$$G_f = 7.997\times10^{-3}.(8.000\times10^{-6} - 0.0)$$
$$= 6.398\times10^{-8}$$

As for the previous occurrence of M2, Equations 7.11 - 7.22 are used to obtain values of $pr_{x_i}(q)$, $po^1_{x_i}(q)$ and $po^0_{x_i}(q)$ for the basic events. The results of the calculations are shown in Table 7.8.

Complex event	Gate type	Component event	pr	po^1	po^0	Criticality
			of the component event			
2002	AND	$X_1 = i$	7.997×10^{-5}	2.000×10^{-3}	0.0	1.599×10^{-7}
		$X_2 = h$	3.199×10^{-7}	1.0	0.0	3.199×10^{-7}
2003	AND	$X_1 = l$	7.999×10^{-3}	1.691×10^{-2}	8.000×10^{-8}	1.353×10^{-4}
		$X_2 = 2001$	1.200×10^{-4}	1.0	8.000×10^{-8}	-
2001	OR	$X_1 = 2000$	1.200×10^{-4}	1.0	9.000×10^{-3}	-
		$X_2 = j$	1.190×10^{-4}	1.0	8.000×10^{-8}	1.190×10^{-4}
2000	OR	$X_1 = g$	1.200×10^{-4}	1.0	1.396×10^{-2}	1.183×10^{-4}
		$X_2 = k$	1.196×10^{-4}	1.0	9.000×10^{-3}	1.186×10^{-4}

Table 7.8: Calculation of the criticality of basic events within complex events 2002 and 2003

The new values of the events' criticality functions are added to the values calculated previously, to give their total criticality functions.

The probability value of node F5 is given by the unavailability of the encoded modular event, M2, which was previously calculated to be 2.538×10^{-4}. This also determines the value of $po^1[F3]$. The probability value of F3 can therefore be computed, which in turn gives the probability of the paths below the zero branch of the root vertex:

$$po^0[F1] = P[F3] = q_a.po^1[F3] + (1-q_a).po^0[F3]$$
$$= 2.030 \times 10^{-6}$$

As node F1 encodes a modular event, its component basic events are considered before its probability value (and so the probability of the top event) is calculated.

Assigning Values to the Events Within Module M1

The probability preceding the root vertex, F6 is assigned the value of pr[F1] (1.0) and the probabilities of paths below branches that lead to terminal one and zero vertices are assigned the values $po^1[F1]$ (5.253×10^{-3}) and $po^0[F1]$ (2.030×10^{-6}) respectively.

The calculations to determine the remaining path probabilities and criticality functions for the basic events are straightforward, as all the nodes encode basic events. The calculations are summarised in Table 7.9.

Node	Event	Pr	Po^1	Po^0	Probability value	Criticality
F6	c	1.0	7.007×10^{-5}	2.030×10^{-6}	$po^1.q_c + po^0.(1-q_c) = 2.575 \times 10^{-6}$	6.804×10^{-5}
F7	d	$pr[F6].q_c = 8.000 \times 10^{-3}$	5.253×10^{-3}	3.878×10^{-5}	$po^1.q_d + po^0.(1-q_d) = 7.007 \times 10^{-5}$	4.171×10^{-5}
F8	e	$pr[f7].(1-q_d) = 7.952 \times 10^{-3}$	5.253×10^{-3}	2.030×10^{-6}	$po^1.q_e + po^0.(1-q_e) = 3.878 \times 10^{-5}$	4.175×10^{-5}

Table 7.9: Assigning values to the nodes of module M1

Once the criticality functions have been evaluated, the final calculations in the primary BDD can be performed.

The top event probability, which is given by the probability value of the root vertex F1, is calculated as follows:

$$Q_{sys} = P[F1] = q_{M1}.po^1[F1] + (1-q_{M1}).po^0[F1]$$
$$= 2.575 \times 10^{-6}$$

All the calculations are summarised in Table 7.10. There are two sets of values for the module M2, as it has two occurrences in the primary BDD.

	Node	Event	One branch	Zero branch	Pr	Po^1	Po^0	Probability value
	F1	M1	F2	F3	1.0	5.253×10^{-3}	2.030×10^{-6}	$po^1.q_{M1}$ + $po^0.(1-q_{M1})$ $= 2.575\times10^{-6}$
	F2	M2	1	F4	$pr[F1].q_{M1}$ $= 1.037\times10^{-4}$	1.0	0.005	$po^1.q_{M2}$ + $po^0.(1-q_{M2})$ $= 5.253\times10^{-3}$
	F3	a	F5	0	$pr[F1].(1-q_{M1})$ $= 9.999\times10^{-1}$	2.538×10^{-4}	0.0	$po^1.q_a$ + $po^0.(1-q_a)$ $= 2.030\times10^{-6}$
	F4	b	1	0	$pr[F2].(1-q_{M2})$ $= 1.036\times10^{-4}$	1.0	0.0	$q_b = 0.005$
	F5	M2	1	0	$pr[F3].q_a =$ 7.999×10^{-3}	1.0	0.0	2.538×10^{-4}
M1:	F6	c	F7	0	1.0	7.007×10^{-5}	2.030×10^{-6}	2.575×10^{-6}
	F7	d	1	F8	8.000×10^{-3}	5.253×10^{-3}	3.878×10^{-5}	7.007×10^{-5}
	F8	e	1	0	7.952×10^{-3}	5.253×10^{-3}	2.030×10^{-6}	3.878×10^{-5}
M2:	F9	2003	1	F10	1.037×10^{-4}	1.0	5.000×10^{-3}	5.253×10^{-3}
					7.999×10^{-3}	1.0	8.000×10^{-8}	2.538×10^{-4}
	F10	f	F11	0	1.036×10^{-4}	5.008×10^{-3}	0.005	5.000×10^{-3}
					7.997×10^{-3}	8.000×10^{-6}	0.0	8.000×10^{-8}
	F11	2002	1	0	1.036×10^{-6}	1.0	0.005	5.008×10^{-3}
					7.997×10^{-5}	1.0	0.0	8.000×10^{-6}

Table 7.10: Final calculated probabilities for the primary BDD and its modules

The criticality functions of the basic events within the primary BDD are now calculated according to Equation 7.8:

$$G_a = 0.9999.(2.538\times10^{-4} - 0.0)$$
$$= 2.538\times10^{-4}$$

$$G_b = 1.036\times10^{-4}.(1.0 - 0.0)$$
$$= 1.036\times10^{-4}$$

Table 7.11 shows the criticality functions for all the basic events.

Event	a	b	c	d	e	f
Criticality	2.538×10^{-4}	1.036×10^{-4}	6.804×10^{-5}	4.171×10^{-5}	4.175×10^{-5}	6.480×10^{-8}

Event	g	h	i	j	k	l
Criticality	1.198×10^{-4}	3.240×10^{-7}	1.620×10^{-7}	1.206×10^{-4}	1.201×10^{-4}	1.370×10^{-4}

Table 7.11: The criticality functions of the basic events

The system unconditional failure intensity can be found from Equation 7.4:

$$w_{sys}(t) = \sum_{i} G_i(q(t)).w_i(t)$$

$$= 1.135\text{x}10^{-8}\ \text{hr}^{-1}$$

This concludes the quantitative analysis of the BDD. If required, the methods could be developed to obtain further basic event importance measures, such as those detailed in Chapter 2. The criticality functions are needed for many of these and are a major element required to evaluate the criticality measure of component importance.

7.8 Conclusions

In this chapter the quantitative analysis has been developed for BDDs that encode modular and/or complex events. It has been shown how the analysis proceeds to enable the calculation of the top event probability and the system unconditional failure intensity. In addition, a technique for extracting the criticality functions of the basic events, which are constituents of both complex events and modules, has been developed. This enables the system to be assessed in terms of its original components and allows analysis of the contributions to system failure through basic event importance measures.

Chapter 8: A Fault Tree Analysis Strategy Using Binary Decision Diagrams

8.1 Introduction

The BDD technique for Fault Tree Analysis provides a more accurate and efficient means of system assessment than the conventional approach of Kinetic Tree Theory. However, there is currently no method of selecting an appropriate ordering scheme that can be used to guarantee the successful construction of a BDD for all fault trees. As such, emphasis in the research has turned to applying alternative techniques that increase the likelihood of obtaining a BDD for any given fault tree structure, by ensuring that the associated calculations are as efficient as possible. This chapter introduces an analysis strategy for fault trees, which aims to implement this requirement by providing a structured framework for the BDD construction process, so that the BDD method can be used successfully for any given system.

The initial stage of the analysis strategy applies two pre-processing techniques to the fault tree: reduction and modularisation. The reduction technique optimises the fault tree structure, whilst modularisation identifies modules that can be analysed independently of the rest of the tree. This results in a set of concisely written subtrees, which are logically equivalent to the original fault tree structure. BDDs are constructed for each, using a variable ordering determined by one of eight ordering schemes. Quantitative analysis is then performed simultaneously on the resulting set of BDDs to obtain the top event probability, the system unconditional failure intensity and the criticality functions of the basic events.

The stages of the analysis strategy are detailed in the following sections and demonstrated throughout with the use of an example fault tree. The program written to implement the technique is also discussed and results are given at the end of the chapter for its application to a set of fault trees.

8.2 Pre-Processing of the Fault Tree

The aim of applying the pre-processing techniques is to obtain the smallest possible fault trees, so that the process of constructing the BDDs becomes as simple and efficient as possible. Two simplification procedures are used. The first of these is Faunet reduction, a technique that restructures the tree to a more concise format. This is followed by linear-time modularisation, which identifies modules existing within the tree that can be analysed separately. The result is a set of simple, independent fault tree structures that together describe the original system.

8.2.1 Faunet Reduction

Faunet reduction is a technique that is used to reduce the complexity of fault trees, so eliminating any 'noise' from the system, without altering the underlying logic. Its effectiveness was demonstrated in Chapter 6, where its application to a large set of fault trees significantly reduced the complexity of the resulting BDDs.

The fault tree shown in Figure 8.1 is used to demonstrate the analysis strategy.

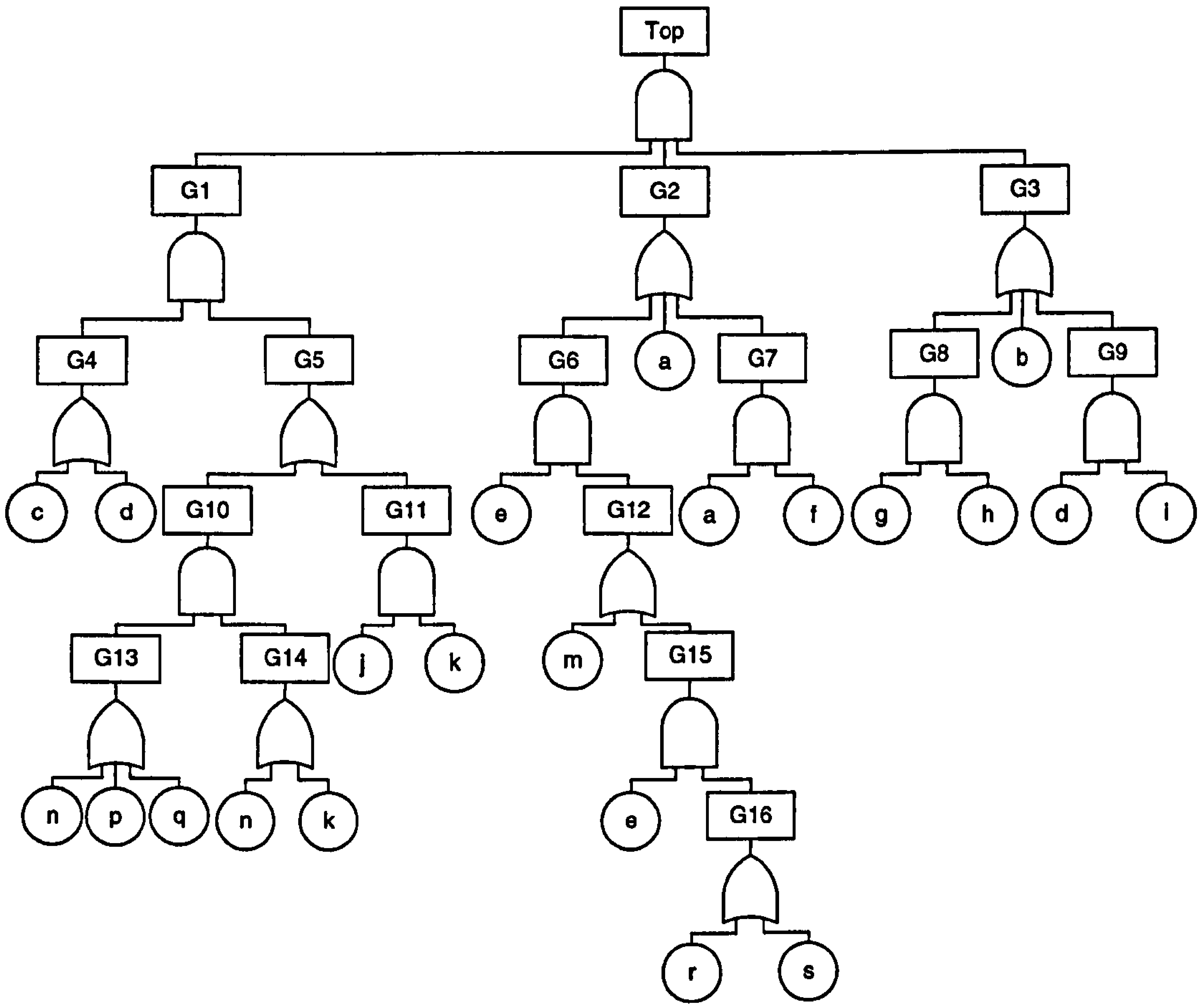


Figure 8.1: Example fault tree

The fault tree is represented by a data file throughout the program and it is this that is manipulated, rather than the actual fault tree structure. As details of the data manipulation for the Faunet reduction technique were discussed in depth in Chapter 6, only the effect of applying the technique will be considered here and the data file for the fault tree will be introduced in the following section.

The basic event data for the fault tree is shown in Table 8.1 and is read into the program at the same time as the data file containing the fault tree structure.

Event	a	b	c	d	e	f
q_i	0.003	0.0045	0.008	0.01	0.0035	0.0025
w_i	1.94×10^{-4}	9.90×10^{-7}	2.15×10^{-6}	1.37×10^{-5}	3.92×10^{-6}	8.50×10^{-7}

Event	g	h	i	j	k	m
q_i	0.015	0.012	0.009	0.004	0.007	0.015
w_i	2.44×10^{-6}	6.40×10^{-7}	2.27×10^{-6}	3.92×10^{-6}	6.22×10^{-5}	8.76×10^{-6}

Event	n	p	q	r	s
q_i	0.005	0.008	0.0065	0.012	0.006
w_i	4.86×10^{-6}	1.12×10^{-4}	9.90×10^{-7}	3.53×10^{-5}	7.86×10^{-6}

Table 8.1: Basic event data for the fault tree in Figure 8.1

Upon application of the Faunet reduction technique to the tree in Figure 8.1, a significantly smaller fault tree is obtained, as shown in Figure 8.2. The corresponding fault tree data is shown in Table 8.2. The fault tree data lists each gate that appears in the tree, together with its type, the number of inputs (gates and events are numbered separately) and the inputs themselves. This forms a complete description of the fault tree structure.

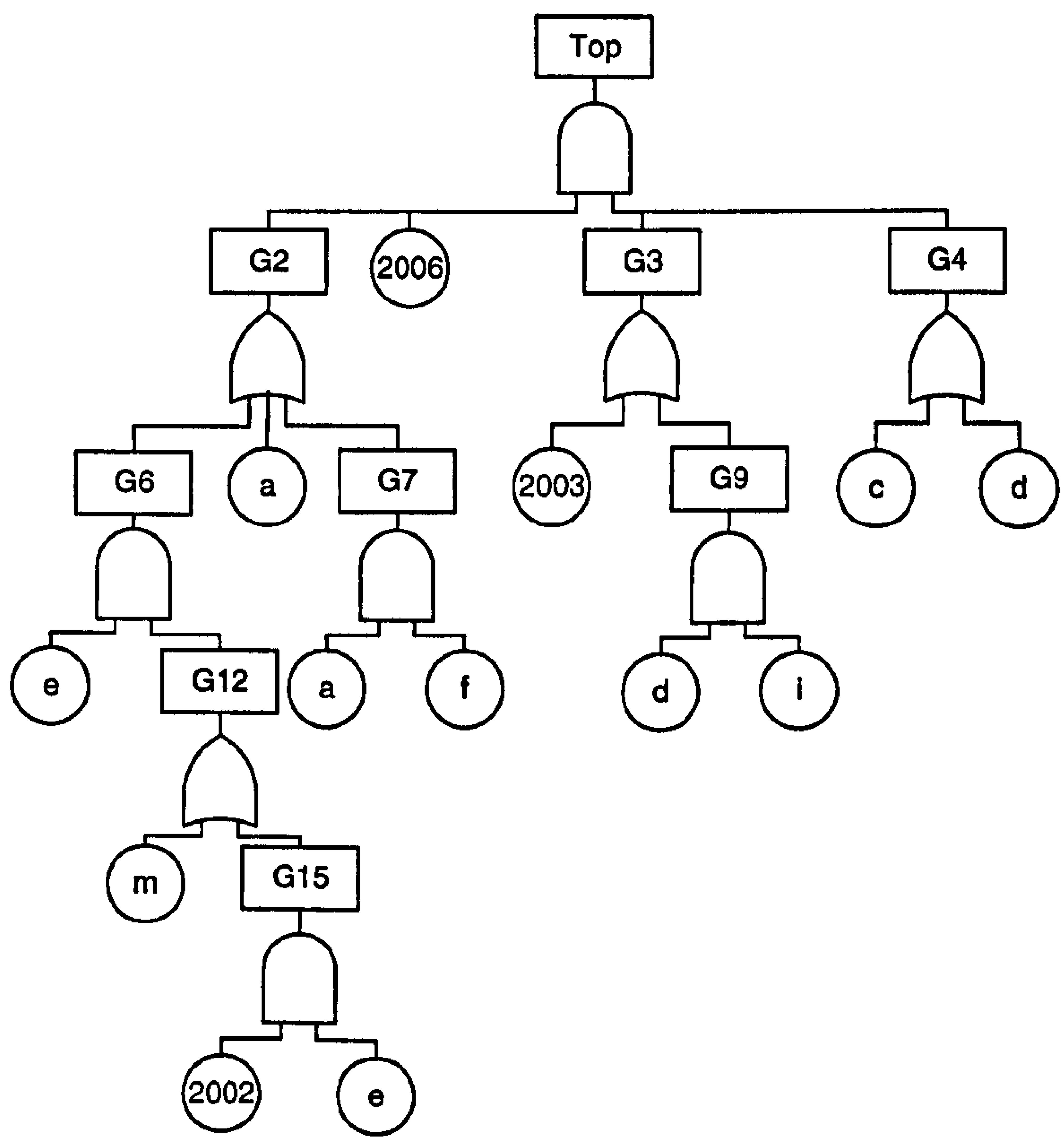


Figure 8.2: The resulting fault tree after the application of Faunet reduction

Gate name	Gate type	Number of gates	Number of events	Inputs
Top	AND	3	1	G2 G3 G4 2006
G2	OR	2	1	G6 G7 a
G3	OR	1	1	G9 2003
G4	OR	0	2	c d
G6	AND	1	1	G12 e
G7	AND	0	2	a f
G9	AND	0	2	d i
G12	OR	1	1	G15 m
G15	AND	0	2	2002 e

Table 8.2: The fault tree data for the tree shown in Figure 8.2

The complex event data are shown in Table 8.3. The probabilities of the complex events, which are required for the quantification process and are calculated as the complex events are formed, are also shown in Table 8.3.

Complex event	Gate type	Event 1	Event 2	Unavailability
2000	AND	g	h	1.800×10^{-4}
2001	OR	p	q	1.445×10^{-2}
2002	OR	r	s	1.793×10^{-2}
2003	OR	2000	b	4.679×10^{-3}
2004	OR	j	2001	1.839×10^{-2}
2005	AND	2004	k	1.287×10^{-4}
2006	OR	2005	n	5.128×10^{-3}

Table 8.3: The complex event data after Faunet reduction

Having reduced the fault tree to a more concise form, the second simplification technique of modularisation is now considered.

8.2.2 Modularisation

The linear-time algorithm, introduced in Chapter 2, is an efficient method of modularisation, which is capable of identifying the fault tree modules after only two depth-first traversals of the tree. The advantage of identifying such modules is that each one can be analysed independently of the rest of the tree, and the results substituted into the higher-level fault trees where the modules occur.

The modularisation technique is applied to the tree in Figure 8.2, identifying the gates that head modules as:

Top, G2 and G6

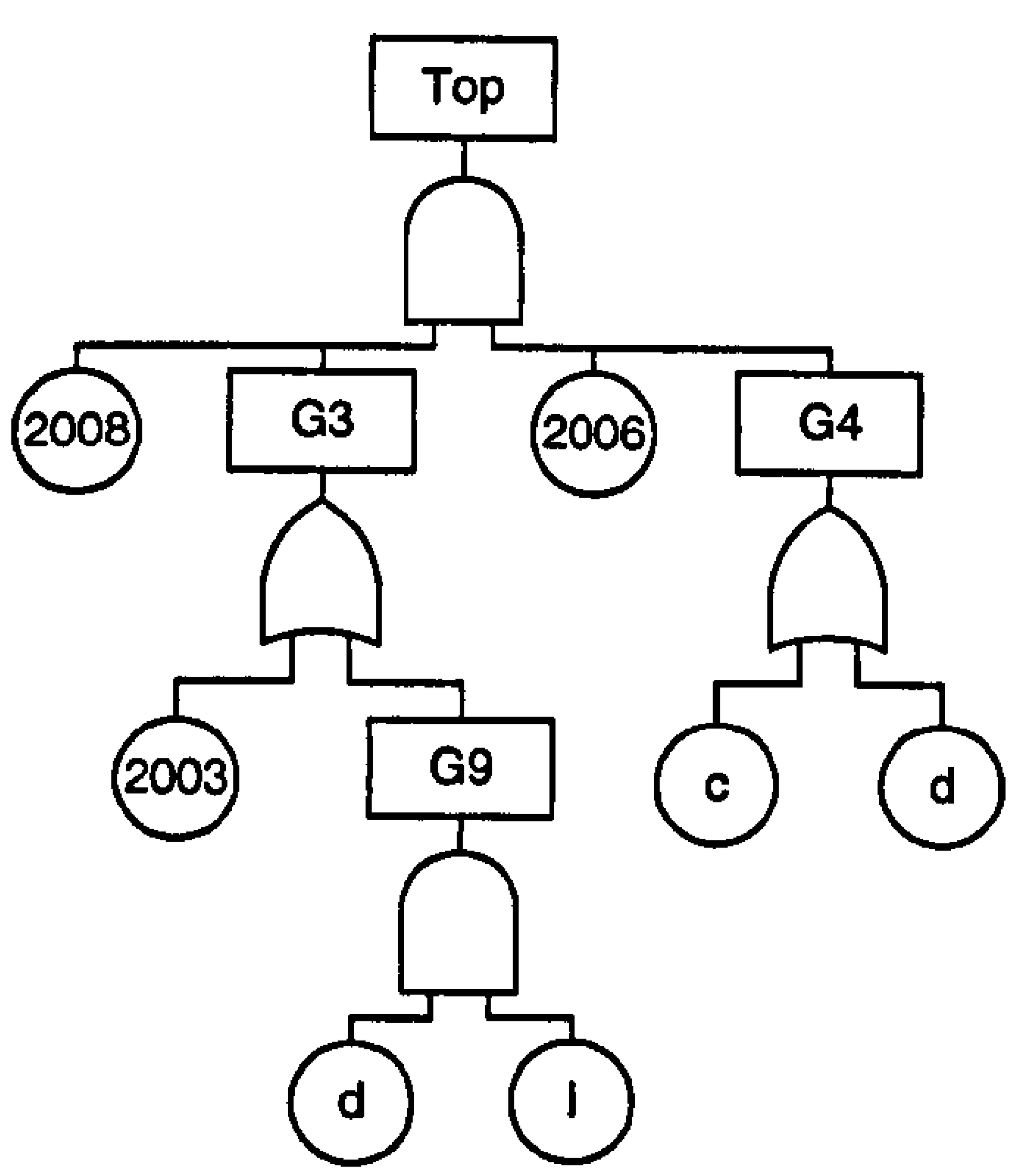
The occurrences of these subtrees are replaced in the fault tree structures by single modular events, which are named in the same way as complex events (i.e. they take on the next available value above 2000):

Top - 2007, G2 - 2008, G6 - 2009

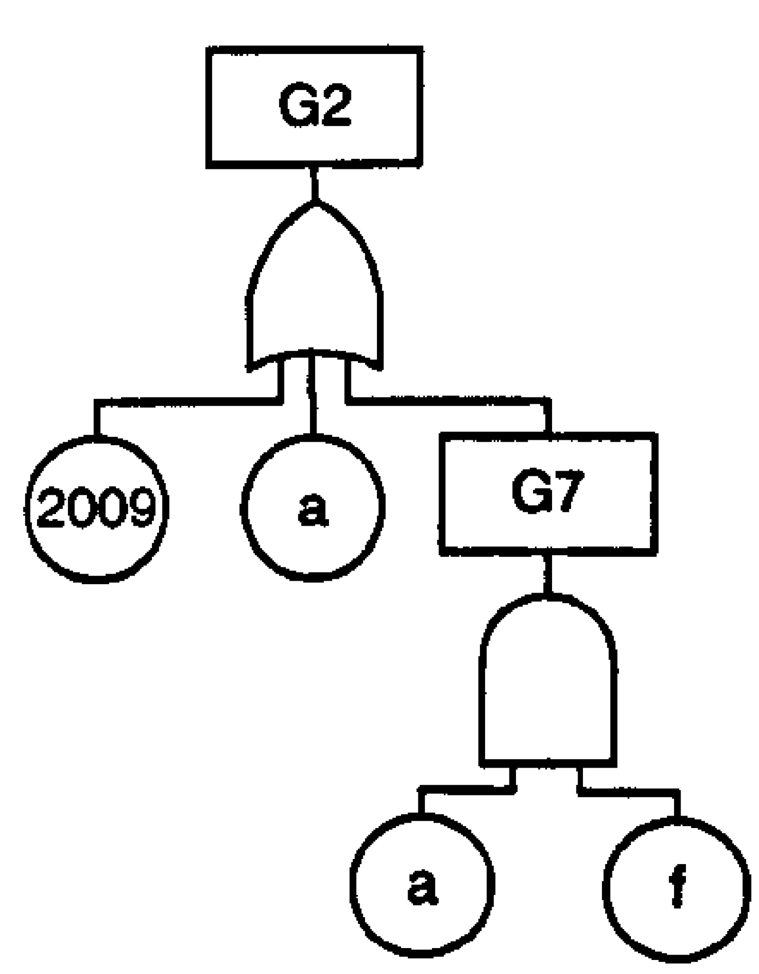
In the program, this is achieved by replacing each occurrence of these gates in the list of inputs to other gates by the appropriate modular event. This is shown in Table 8.4, where the fault tree data now essentially incorporates three separate fault trees. The corresponding module structures are shown in Figure 8.3.

Gate name	Module name	Gate type	Number of gates	Number of events	Inputs			
Top	2007	AND	2	2	G3	G4	2008	2006
G2	2008	OR	1	2	G7	2009	a	
G3	-	OR	1	1	G9	2003		
G4	-	OR	0	2	c	d		
G6	2009	AND	1	1	G12	e		
G7	-	AND	0	2	a	f		
G9	-	AND	0	2	d	i		
G12	-	OR	1	1	G15	m		
G15	-	AND	0	2	2002	e		

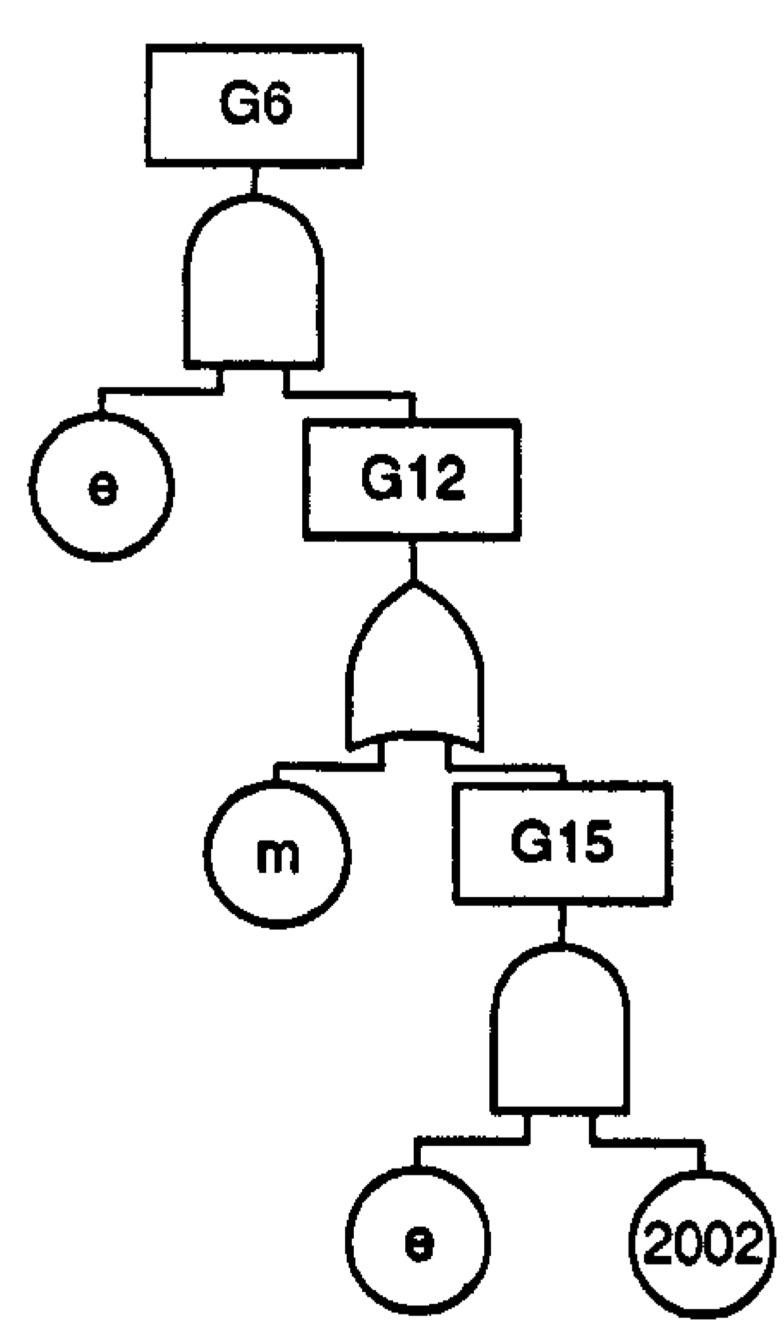
Table 8.4: The fault tree data after modularisation



(a) Module 2007



(b) Module 2008



(c) Module 2009

Figure 8.3: The three modules obtained from the fault tree shown in Figure 8.2

Having reduced the fault tree to a more concise form and identified all the independent subtrees, the pre-processing stage is complete and the next step is to obtain the associated BDDs.

8.3 Construction of the BDDs

A BDD is constructed for each of the modules, using a variable ordering determined by one of eight ordering schemes, which are detailed in Chapter 5:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

The choice of ordering scheme for each module should be less critical than for the original tree, due to the pre-processing techniques applied. At this stage the schemes are selected randomly. An alternative option, however, would be to incorporate a method of scheme selection based on the characteristics of the individual modules. This would ensure that the most appropriate scheme was chosen on each occasion. One such approach is the neural network technique, which is a pattern recognition method that has previously been considered as a mechanism for selecting ordering schemes for BDD construction[19, 30]. The following chapter investigates this as an option for inclusion within this analysis strategy.

The fault tree data for each module must be extracted from the collective data, so that it can be considered independently. Taking each module in turn, its variables are ordered using the chosen ordering scheme and a BDD constructed. The BDD data is stored in an **ite** array, and is added to as the BDDs are constructed for the remaining modules. This technique is now applied to the example modules shown in Figure 8.3.

The extraction of the data for any module starts on the line on which the gate heading that module is located. Therefore, for module 2007, which is headed by the gate Top, the process starts on the first line of the fault tree data, which is then copied into the module data array. Every gate that is referenced in the inputs to Top is included in the module data (G3 and G4). Each gate that appears as an input to either G3 or G4 is also listed, and so on until every gate that exists within the module is included in the module data. The self-contained data for module 2007 is shown in Table 8.5.

Gate name	Module name	Gate type	Number of gates	Number of events	Inputs
Top	2007	AND	2	2	G3 G4 2008 2006
G3	-	OR	1	1	G9 2003
G4	-	OR	0	2	c d
G9	-	AND	0	2	d i

Table 8.5: Data for module 2007

This data forms a completely independent subtree, for which a variable ordering must now be determined. A suitable scheme would be the modified priority depth-first scheme, which results in the following ordering:

2008<2006<d<c<2003<i

The BDD obtained from this ordering is shown in Figure 8.4. It is known as the 'primary' BDD, as it represents the top event of the original fault tree and can be used to calculate the system unavailability.

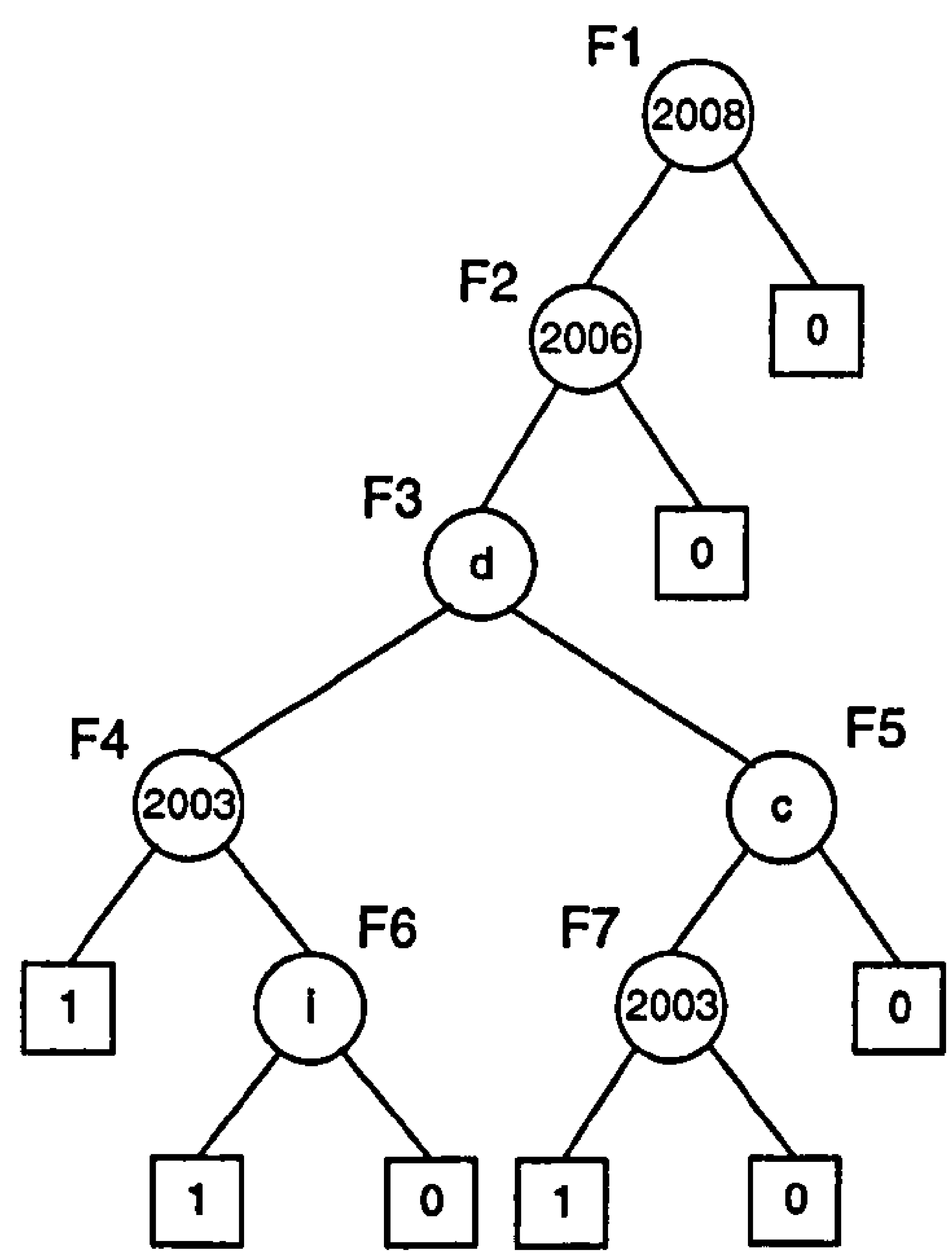


Figure 8.4: The primary BDD (module 2007) obtained from the ordering 2008<2006<d<c<2003<i

The program stores the BDD data in an **ite** array. Each node is identified by its unique label (F1, F2, and so on), as shown in Figure 8.4. The node labels are stored together with the encoded event and the names of the nodes that appear on the one and zero branches. The **ite** data contains all the information necessary to describe the BDD, as shown in Table 8.6.

Node	Event	One branch	Zero branch
F1	2008	F2	0
F2	2006	F3	0
F3	d	F4	F5
F4	2003	1	F6
F5	c	F7	0
F6	i	1	0
F7	2003	1	0

Table 8.6: The **ite** array, currently containing the data for module 2007

Each module is considered in the same way, and its **ite** data is stored in the same array as the first module. Obviously, this means that the nodes in the BDDs must be labelled differently - therefore, the number of the first node of the next BDD follows on from the number of the final node in the previous BDD.

The extracted data for the modules 2008 and 2009 are shown in Tables 8.7 and 8.8.

Gate name	Module name	Gate type	Number of gates	Number of events	Inputs
G2	2008	OR	1	2	G7 2009 a
G7	-	AND	0	2	a f

Table 8.7: Data for module 2008

Gate name	Module name	Gate type	Number of gates	Number of events	Inputs
G6	2009	AND	1	1	G12 e
G12	-	OR	1	1	G15 m
G15	-	AND	0	2	2002 e

Table 8.8: Data for module 2009

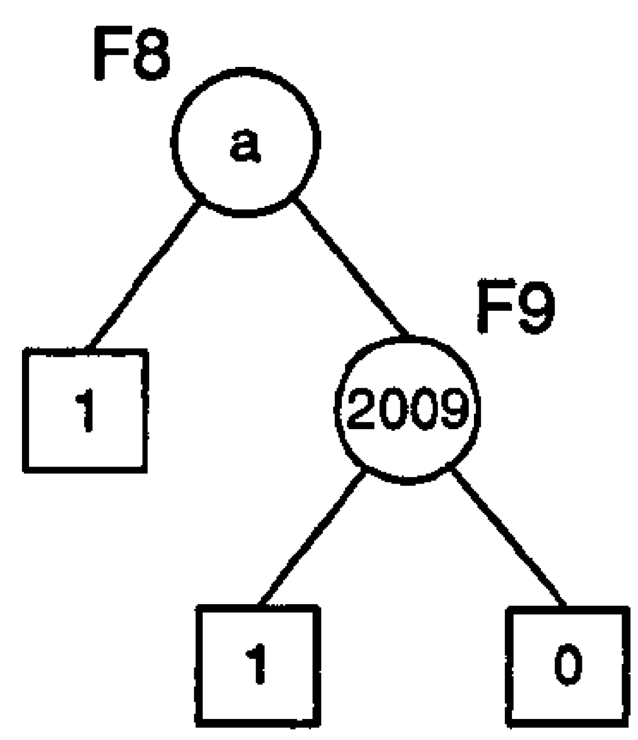
Again, variable ordering schemes must be chosen to construct the BDDs. The event criticality ordering scheme is used for module 2008, giving the event ordering:

$$a<2009<f$$

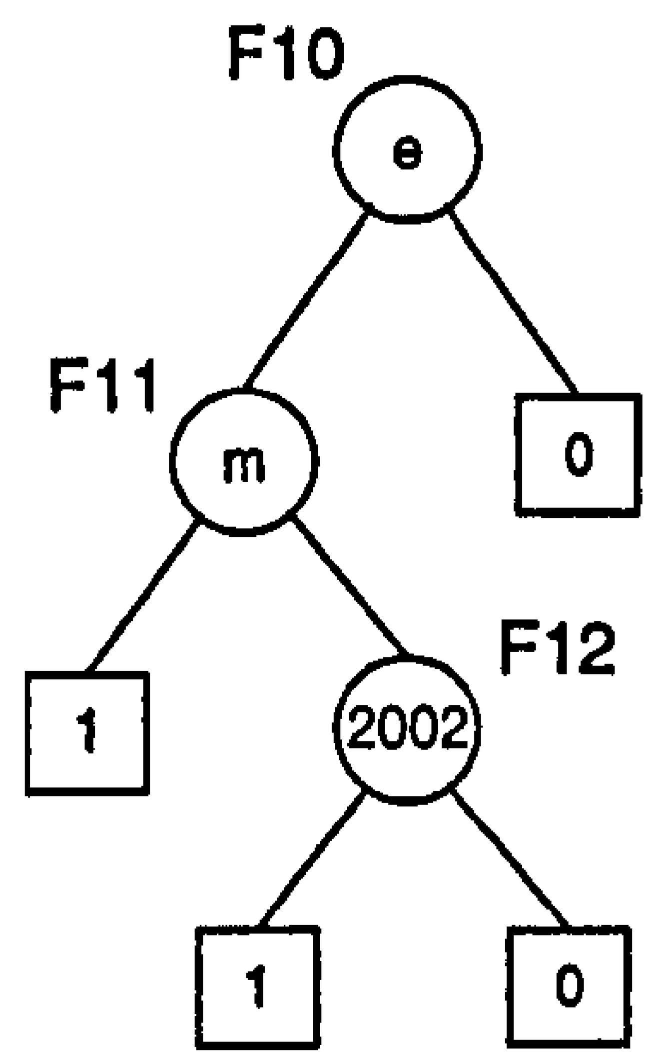
The modified top-down scheme is used for module 2009 giving:

$$e<m<2002$$

The resulting BDDs, which also illustrate the node labelling, are shown in Figure 8.5.



(a) BDD for module 2008



(b) BDD for module 2009

Figure 8.5: The BDDs for modules 2008 and 2009, demonstrating the node labelling

The BDD data for these modules are added to the **ite** array, shown in Table 8.9. This now completely represents the original fault tree structure.

Node	Event	One branch	Zero branch
F1	2008	F2	0
F2	2006	F3	0
F3	d	F4	F5
F4	2003	1	F6
F5	c	F7	0
F6	i	1	0
F7	2003	1	0
F8	a	1	F9
F9	2009	1	0
F10	e	F11	0
F11	m	1	F12
F12	2002	1	0

Table 8.9: The **ite** array, containing the data for all the modules

Once the set of BDDs has been constructed, the quantitative analysis can begin.

8.4 Quantitative Analysis

The basic event data (i.e. unavailability, q_i, and unconditional failure intensity, w_i,) are input to the program with the fault tree data and are shown in Table 8.1. The probabilities of the complex events are calculated as they are formed and are shown in the final column of Table 8.3.

The unavailabilities of the modules are also required and are evaluated by calculating the probability of the modules' 'top event'. This procedure is described in further detail in Chapter 7. The probabilities of modules 2008 and 2009 in this example are:

$$q_{2008} = 3.11\text{x}10^{-3}$$

$$q_{2009} = 1.14\text{x}10^{-4}$$

The quantitative analysis described in Chapter 7 is performed simultaneously on the three BDDs, the results of which are summarised in Table 8.11.

The top event probability is given by the probability of the root vertex of the primary BDD:

$$Q_{sys}(t) = 2.77\ \text{x}10^{-9}$$

The criticality functions of the basic events are required, so that the system can be analysed in terms of its basic components. These are shown in Table 8.10, and are also calculated according to the methods described in Chapter 7.

Event	a	b	c	d	e	f
Criticality	8.89x10^{-7}	2.85x10^{-7}	7.40x10^{-8}	2.17x10^{-7}	8.71x10^{-11}	0.0
Event	g	h	i	j	k	m
Criticality	3.40x10^{-9}	4.25x10^{-9}	1.59x10^{-7}	3.71x10^{-9}	9.88x10^{-9}	9.17x10^{-12}
Event	n	p	q	r	s	
Criticality	5.40x10^{-7}	3.72x10^{-9}	3.72x10^{-9}	1.39x10^{-13}	1.38x10^{-13}	

Table 8.10: Criticality functions for the basic events

The system unconditional failure intensity is calculated using the criticality functions and unconditional failure intensities of the basic events:

$$w_{sys}(t) = \sum_i G_i(q(t)).w_i(t)$$
$$= 1.80\text{x}10^{-10}$$

This concludes the analysis of the example fault tree.

	Node	Event	One branch	Zero branch	Pr	Po^1	Po^0	Probability value
2007:	F1	2008	F2	0	1.0	8.89×10^{-7}	0.0	$q_{2008}.po^1 + (1-q_{2008}).po^0 = 2.77\times10^{-9}$
	F2	2006	F3	0	$pr[F1].q_{2008} = 3.11\times10^{-3}$	1.73×10^{-4}	0.0	$q_{2006}.po^1 + (1-q_{2006}).po^0 = 8.89\times10^{-7}$
	F3	d	F4	F5	$pr[F2].q_{2006} = 1.60\times10^{-5}$	1.36×10^{-2}	3.74×10^{-5}	$q_d.po^1 + (1-q_d).po^0 = 1.73\times10^{-4}$
	F4	2003	1	F6	$pr[F3].q_d = 1.60\times10^{-7}$	1.0	9.00×10^{-3}	$q_{2003}.po^1 + (1-q_{2003}).po^0 = 1.36\times10^{-2}$
	F5	c	F7	0	$pr[F3].(1-q_d) = 1.58\times10^{-5}$	4.68×10^{-3}	0.0	$q_c.po^1 + (1-q_c).po^0 = 3.74\times10^{-5}$
	F6	i	1	0	$pr[F4].(1-q_{2003}) = 1.59\times10^{-7}$	1.0	0.0	$q_i = 9.00\times10^{-3}$
	F7	2003	1	0	$pr[F5].q_c = 1.26\times10^{-7}$	1.0	0.0	$q_{2003}.po^1 + (1-q_{2003}).po^0 = 4.68\times10^{-3}$
2008:	F8	a	1	F9	$pr[F1] = 1.0$	$po^1[F1] = 8.89\times10^{-7}$	1.02×10^{-10}	-
	F9	2009	1	0	$pr[F8].q_a = 3.00\times10^{-3}$	$po^1[F1] = 8.89\times10^{-7}$	$po^0[F1] = 0.0$	$q_{2009}.po^1 + (1-q_{2009}).po^0 = 1.02\times10^{-10}$
2009:	F10	e	F11	0	$pr[F9] = 3.00\times10^{-3}$	2.90×10^{-8}	$po^0[F9] = 0.0$	-
	F11	m	1	F12	$pr[F10].q_e = 1.05\times10^{-5}$	$po^1[F9] = 8.89\times10^{-7}$	1.59×10^{-8}	$q_m.po^1 + (1-q_m).po^0 = 2.90\times10^{-8}$
	F12	2002	1	0	$pr[F11].q_m = 1.58\times10^{-7}$	$po^1[F9] = 8.89\times10^{-7}$	$po^0[F9] = 0.0$	$q_{2002}.po^1 + (1-q_{2002}).po^0 = 1.59\times10^{-8}$

Table 8.11: Results of the quantitative analysis applied to the BDDs in Figures 8.4 and 8.5

8.5 Results of the Application of the Fault Tree Analysis Strategy

The analysis strategy (program 'strategy.c') was applied to a set of 228 fault trees, whose summary details are shown in Appendix II. The calculation times were compared with those obtained for the construction and subsequent quantification of the BDDs directly from the original trees. As the choice of ordering scheme has such an effect on the number of **ite** calculations and the size of the resulting BDDs, the times were recorded for each of the eight available schemes. In the cases where more than one module was detected, each was ordered using the same scheme. The calculations were performed three times for each tree using the two methods and an average taken of the resulting calculation times in order to gain more accurate results.

Appendix IX shows the calculation times obtained using the two methods for the 1824 different cases. Applying the fault tree analysis strategy has the effect of both increasing and decreasing the analysis times, depending on which tree and ordering scheme is being used. In 1446 cases the analysis times actually increased. Although this seems a large proportion, the average increase in time was in fact only 0.145 seconds. This is probably due to the number of comparisons necessary in the Faunet reduction technique, which for small trees is not compensated for by the time saved in the BDD construction and quantification.

Although only 316 cases showed a decrease in analysis time, the average decrease for these was 15.48 seconds. This result does however include the times for the tree 'rando11', which has exceptionally large BDDs compared to the other trees. If the results for 'rando11' are excluded from the analysis, then the average decrease in analysis time is still 0.654 seconds.

The results for the largest tree in the set, 'rando11', are shown in Table 8.12.

Ordering scheme	1	2	3	4	5	6	7	8
Direct analysis times	629.16	3125.63	3145.01	174.12	306.98	234.95	940.13	108.74
Strategy analysis times	143.33	1624.34	1625.66	59.17	103.03	48.09	325.60	46.69
Difference	485.83	1501.29	1519.35	114.95	203.95	186.86	614.53	62.05

Table 8.12: Analysis times for the fault tree 'rando11'

These results demonstrate the substantial savings in analysis time that can be made when dealing with large fault trees. The time taken for the analysis using the third ordering scheme, modified priority depth-first, is reduced by over 25 minutes when the fault tree analysis strategy is implemented. The reduction in analysis time could be even more substantial for larger fault trees.

8.6 Conclusions

The fault tree analysis strategy has the potential to reduce the analysis times of large fault trees significantly and increase the likelihood of obtaining a BDD for any given fault tree. The results of the application of the analysis strategy have shown that although applying the technique slightly increases the analysis times for some trees, due to the comparisons necessary for Faunet reduction, this is countered by the savings in analysis time for larger trees. It is also possible that the Faunet reduction technique could be coded in a more efficient manner, thus reducing the time spent applying the methods.

A significant advantage of the analysis strategy is the possibility of analysing the modules of the trees separately. This is likely to be of particular use where the tree is too large to be dealt with as a whole but can be taken piece by piece and the quantitative analysis applied to the set of resulting BDDs.

The results were obtained using the same ordering scheme for each module of the original fault tree. As discussed in section 8.3, it is possible to use different schemes for the modules, depending on which best suits the module under consideration. If the optimal scheme can be selected on each occasion, it would lead to smaller BDDs and further reduce the analysis times. This is the topic discussed in the following chapter.

Chapter 9: Neural Networks

9.1 Introduction

This chapter investigates the use of neural networks as a technique for scheme selection within the fault tree analysis strategy described in the previous chapter. The work aims to develop a neural network model that is capable of selecting the optimal variable ordering scheme for any given fault tree. If such a network model can be identified, it would eliminate the need for trying several schemes until an appropriate one is found and could significantly reduce computation time.

The use of pattern recognition techniques, such as neural networks, for selecting the optimal variable ordering scheme for a particular fault tree based on its individual characteristics was proposed by Bartlett and Andrews[30]. Their analysis produced encouraging results, with the prediction of the correct scheme in up to 70% of cases. This investigation differs from the previous research, in that the reduced fault trees will be used to train and test the network.

The following section describes the basic elements of a neural network. Two specific models, known as the multi-layer perceptron and the radial basis function network are then described in detail and the results of their application to the ordering problem are discussed. The investigation uses the programs written by Bartlett[19] to perform neural network training and validation, with modifications where necessary.

9.2 Overview of Neural Networks

Neural networks offer a powerful framework for representing non-linear mappings from several input variables to several output variables. The general structure of a neural network model is shown in Figure 9.1. A layer of input units representing the characteristics of the system connects via some internal processing to a layer of output units, which each represent one of the possible variable ordering schemes. The exact nature of the processing depends on the type of neural network being used and is described in detail later in the chapter.

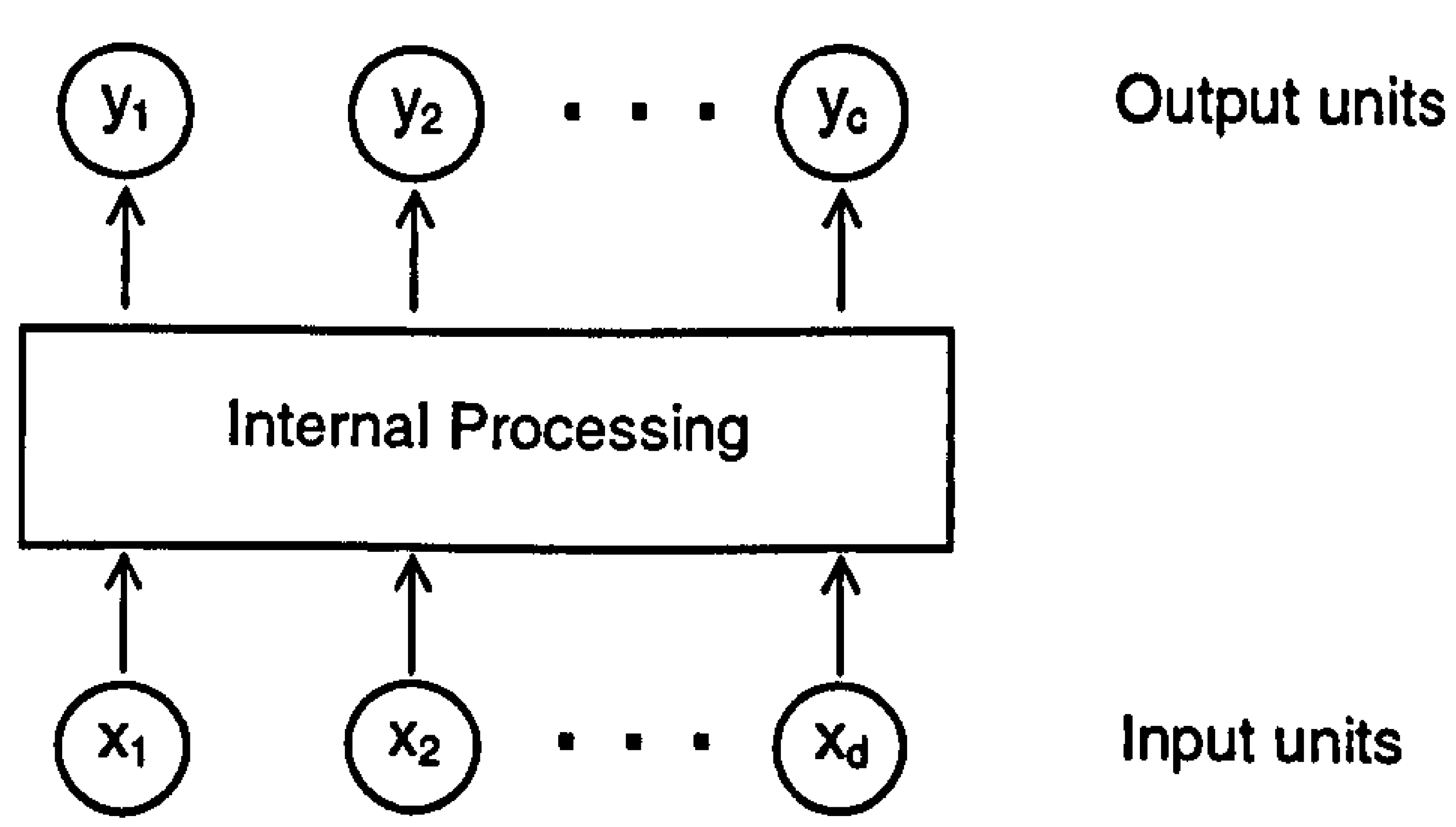


Figure 9.1: Basic structure of a neural network

The aim of the neural network technique is to optimise the internal parameters through some training process to produce an effective model for the problem. This is known as the training phase. The prediction phase tests the performance of the trained network, by using the calculated parameters to determine output responses for a set of validation data. These responses are compared with target responses for the data and determine whether or not it has been trained successfully. There are three techniques for training, which are described below.

9.2.1 Learning Techniques

Supervised learning is the most commonly used technique, in which desired values of the outputs (target values) are specified for each set of inputs. The parameters within the network are chosen so as to minimise the error between the target values and those actually attained by the network. This learning technique is used within the multi-layer perceptron model and is employed as part of the training process in the radial basis function network. In the *pattern* mode of training, the parameters are updated after each individual training case has been presented. In this investigation, however, the *batch* mode of training will be used, which updates the parameters only after the entire training set has been presented.

A second widely used technique is that of unsupervised learning, which does not provide the network with target output values for the inputs, but allows it to discover features of the training set and then group the data into classes that it regards as distinct. The radial basis function neural network uses an unsupervised learning technique during the training phase to determine the basis function parameters.

A third type of learning technique that will not be considered in this investigation is reinforcement learning. This is an unsupervised method in that target values are not specified, but is also supervised in that information is given as to whether the network response was good or bad.

9.3 Multi-Layer Perceptron

The multi-layer perceptron consists of a layer of input units, a layer of output units and one or more 'hidden' layers sandwiched between, as illustrated in Figure 9.2. The bias parameters that appear in each layer (except for the output layer) simply act like adding a constant to the equation. A network containing no hidden layers is referred to as a single-layer neural network. Although faster to train, it is limited in the range of functions it can represent and is therefore not considered in this analysis.

Weights connect each of the units in one layer to each unit in the next and primarily determine the behaviour of the network, as they control the influence each unit has in propagating the intermediate outcome to the output nodes. It is the aim of the training phase to determine optimal values for the weights, which are initially assigned random values between -0.5 and 0.5.

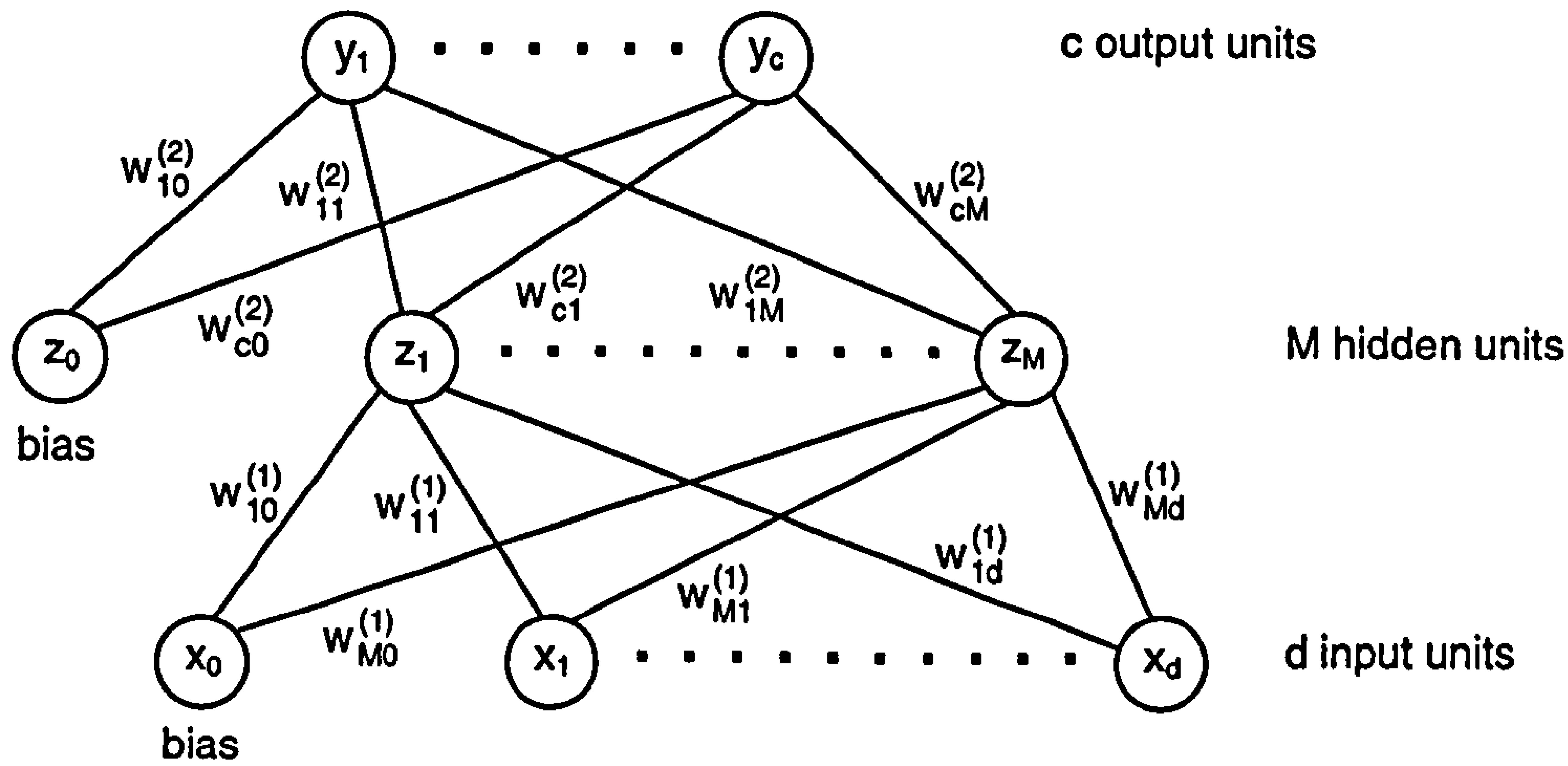


Figure 9.2: Multi-layer perceptron neural network

The training phase is an iterative process, which repeatedly applies two passes through the network, and terminates when either the error in the output units is minimised or the maximum number of iterations is reached. The two passes consist of a forward pass, where the current values of the weights are used to determine the values of the output units and a backward pass through the network, which adjusts the weights in order to minimise the difference between the target values and those actually obtained.

9.3.1 The Forward Pass

During the forward pass, the outputs of each unit are calculated layer by layer until the values of the output units are obtained. Considering the network shown in Figure 9.2, which has d input units, M hidden units and c output units, the weighted sum of the inputs to each of the hidden units is given by:

$$a_j = \sum_{i=0}^{d} w_{ji}^{(1)} x_i \qquad 9.1$$

where $w_{ji}^{(1)}$ denotes a weight in the first layer going from input i to hidden unit j and x_i is the value of the input i. The value of each bias unit is permanently set to 1. The output of each hidden unit is calculated by applying a non-linear activation function, g, to its input:

$$z_j = g(a_j) = g\left(\sum_{i=0}^{d} w_{ji}^{(1)} x_i\right) \qquad 9.2$$

The values of the output units are determined in the same way, by applying a non-linear activation function to the weighted sum of their inputs:

$$y_k = g(a_k) = g\left(\sum_{j=0}^{M} w_{kj}^{(2)} z_j\right)$$

$$= g\left(\sum_{j=0}^{M} w_{kj}^{(2)} g\left(\sum_{i=0}^{d} w_{ji}^{(1)} x_i\right)\right) \qquad 9.3$$

In this example the activation functions applied to the output units and the hidden layers are the same, though this is not always the case. The form of the activation function is now discussed.

9.3.1.1 The Activation Function

The activation function introduces non-linearities into the system and is applied to the net input of each unit in order to determine its output. The majority of networks use the logistic sigmoid activation function and although there are several popular alternatives, it is the one that will be used in this investigation. It is given by:

$$g(a) = \frac{1}{1+e^{-a}} \qquad 9.4$$

where a represents the value of the unit to be activated. Although the domain of this function is any real number, the range is bounded between 0 and 1 as shown in Figure 9.3.

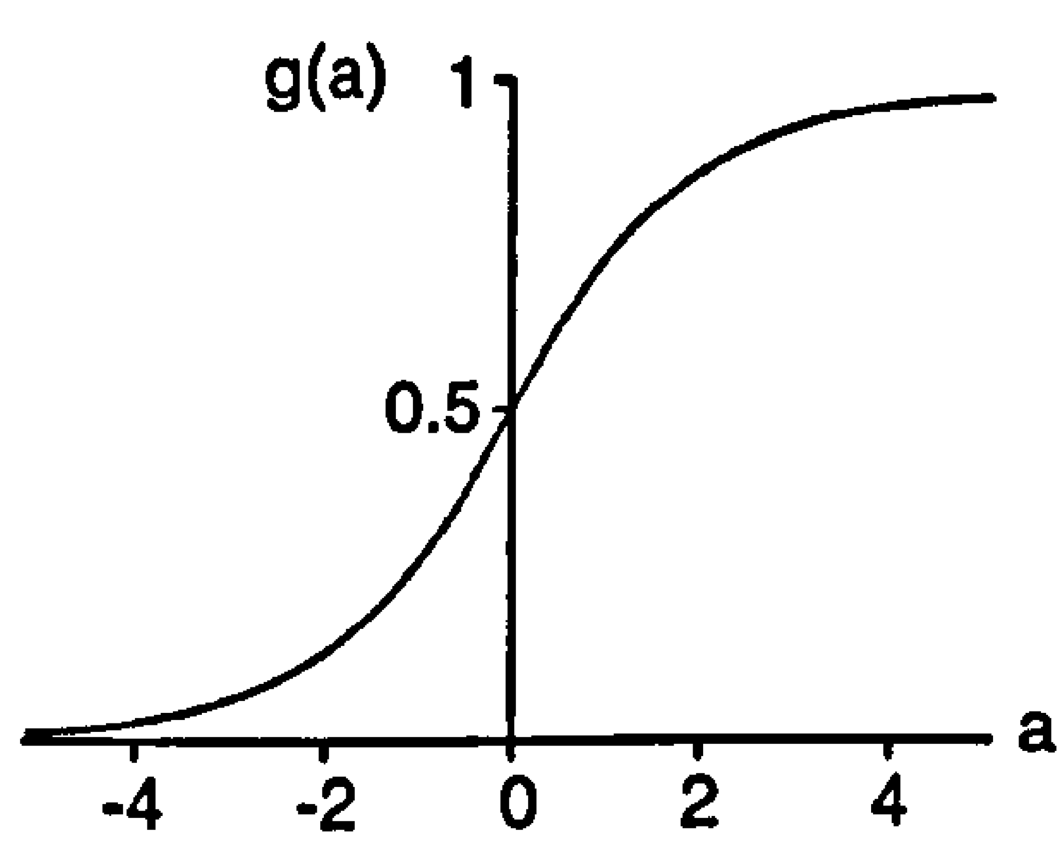


Figure 9.3: Logistic sigmoid activation function

Therefore the output of each unit will be in the range (0,1). For ease of comparison, the target values for the outputs are also scaled within this range, as are the values of the input units.

Two alternatives to this function are the Heaviside, or step, function and the hyperbolic tangent, given in Equations 9.5 and 9.6 respectively.

$$g(a) = \begin{cases} 0 & \text{when } a < 0 \\ 1 & \text{when } a \geq 0 \end{cases} \tag{9.5}$$

$$g(a) = \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \tag{9.6}$$

One of the advantages of the logistic sigmoid function is that the derivative is easily calculated as:

$$g'(a) = g(a).(1 - g(a)) \tag{9.7}$$

This is of importance during the backward pass through the network, which is described in the following section.

9.3.2 The Backward Pass

The aim of the backward phase is to minimise the error at the output nodes by making adjustments to the weights within the network. This process is undertaken in three stages. Firstly, the error between the target output values and those actually attained is calculated using the sum-of-squares error function. This is a differentiable function of the network weights and therefore the derivatives of the error with respect to each of the weights can be calculated. An efficient algorithm, known as error-back propagation is used for calculating the derivatives and forms the second stage of the process. Finally, the derivatives are used to calculate the adjustments to be made to the weights in order to minimise the error in the system. Each of these stages is now described in detail.

9.3.2.1 Calculating the Errors

For each training case, n, presented to the network, the sum-of-squares error function is given by:

$$E^{n} = \frac{1}{2}\sum_{k=1}^{c}(y_k - t_k)^2 \tag{9.8}$$

where y_k is the actual response of the output unit k and t_k is the target response for that unit, for the training case n under consideration. The superscript n is omitted from input and output variables from this point onwards for clarity.

If the calculated errors are above a predetermined value and the maximum number of iterations has not been exceeded, then training continues with the calculation of the error derivatives.

9.3.2.2 Calculation of the Error Derivatives

The derivatives of the error with respect to the output layer weights are given by:

$$\frac{\partial E^n}{\partial w_{kj}^{(2)}} = \delta_k z_j \tag{9.9}$$

where the δ_k are referred to as the 'errors' and are calculated for each output unit according to the following expression:

$$\delta_k = g'(a_k).(y_k - t_k) \tag{9.10}$$

The derivative of the logistic sigmoid activation function is given by Equation 9.7 and is applied to the weighted sum of the inputs to each output unit to give:

$$\begin{aligned} g'(a_k) &= g(a_k).(1 - g(a_k)) \\ &= y_k.(1 - y_k) \end{aligned} \tag{9.11}$$

The errors, δ_k, are therefore simply given by:

$$\delta_k = y_k.(1 - y_k).(y_k - t_k) \tag{9.12}$$

leading to the simple evaluation of each of the derivatives of the output layer weights. The derivatives of the error with respect to each of the hidden layer weights can be calculated once the values of δ_k are known for the output layer. The δ_j for the hidden units can be calculated from the back-propagation formula given by:

$$\delta_j = g'(a_j)\sum_k w_{kj}\delta_k \tag{9.13}$$

Substituting for $g'(a_j)$ gives:

$$\delta_j = z_j.(1 - z_j).\sum_k w_{kj}\delta_k \tag{9.14}$$

As for the output layer weights, the derivatives of the error with respect to each of the hidden layer weights is given by the product of the value of δ for the unit at the output end of the weight and the value of the unit at the input end of the weight:

$$\frac{\partial E^n}{\partial w_{ji}^{(1)}} = \delta_j x_i \tag{9.15}$$

The back-propagation formula shows that the value of δ for any hidden unit can be calculated by propagating the δ's backwards from units higher up in the network. The output layer is always considered first, as the values of δ are dependent only on the target and calculated values of the output units. Any size of network can be dealt with using this method. A full derivation of these results can be found in reference 35.

9.3.2.3 Computation of the Weight Adjustments

Once the error derivatives have been calculated, an optimisation algorithm is employed to find the minimum of the error function. Gradient descent is one such algorithm and is considered below.

The gradient of a function is the direction in which it increases most rapidly. Therefore the negative gradient gives the direction in which to move in order to decrease the function most rapidly. The gradient descent algorithm iteratively updates the weights by moving small distances in the weight space in the direction of greatest rate of decrease of error. The weights can be combined to form a single weight vector w, which is updated according to:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \Delta\mathbf{w}^{\tau} \qquad 9.16$$

$$\Delta\mathbf{w}^{\tau} = -\eta\nabla E\big|_{\mathbf{w}^{\tau}} \qquad 9.17$$

where τ labels the iteration step and $\nabla E\big|_{\mathbf{w}^{\tau}}$ gives the gradient of E in weight space, evaluated at $\mathbf{w}^{\tau}$. The parameter η is referred to as the learning rate and determines the step size taken. If η is chosen to be too small, the convergence towards the minimum will be slow; conversely if η is too large, the algorithm may continually overshoot (causing oscillatory motion) and never converge.

A modification to this method adds a momentum term, with the aim of smoothing out any oscillations. Each new search direction is now calculated as a weighted sum of the current gradient and the previous search direction. The modified gradient descent formula is given by

$$\Delta\mathbf{w}^{\tau} = -\eta\nabla E\big|_{\mathbf{w}^{\tau}} + \mu\Delta\mathbf{w}^{\tau-1} \qquad 9.18$$

where μ is the momentum term in the range $0 \le \mu \le 1$.

A disadvantage of this technique is that the learning rate and momentum term must both be selected by trial and error at the start of the process. However, the optimum values will depend on the particular problem and may also vary during the training. One approach for automatically updating the values when required is the bold driver technique[35]. This applies a multiplicative factor to the learning rate parameter, which depends on whether the error has actually increased or decreased after a given step. If the error increases, then the algorithm must have overshot the minimum, so the learning rate parameter was too large. The step is undone and repeated with a smaller learning rate parameter. This process continues until a decrease in the error is recorded. However, if the error decreases at a given step then the new values are accepted and the value of the learning rate parameter increased for the next step, as it may currently be too small. The learning rate is therefore updated according to Equation 9.19.

$$\eta_{new} = \begin{cases} \alpha.\eta_{old} & \text{if } \Delta E < 0 \\ \gamma.\eta_{old} & \text{if } \Delta E > 0 \end{cases} \qquad 9.19$$

The parameter α is chosen to be slightly greater than 1 (typically around $\alpha = 1.1$), in order to avoid subsequent error increases and γ is chosen to be significantly less than 1 (typically $\gamma = 0.5$) so that the algorithm quickly finds an error decrease.

Having calculated the optimal weights, the network is tested by comparing its responses to a set of test data with target responses for those patterns.

The following section describes the initial network architecture that was employed for the application of the multi-layer perceptron to the ordering problem.

9.3.3 Network Architecture for the Ordering Problem

9.3.3.1 Output Units

Each of the output units of the network represents one of the possible variable ordering schemes. In this investigation, eight ordering schemes were available for selection, so requiring eight output units. Previous work had used six ordering schemes[19], but it was felt that as these were all structural schemes, they did not adequately cover the range of possibilities and so adding in some weighted schemes would lead to improved results. The eight schemes used are:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Each of these schemes is described in detail in Chapter 5. Target values were calculated for each of the output units by determining the number of non-distinct nodes in the BDDs obtained using each of the schemes (Appendices VI and X). As the values of the output units result from the application of the logistic sigmoid activation function to their summed inputs, they are in the range (0,1), though they will never reach the extreme values of 0 and 1 due to the nature of the function. Therefore the number of BDD nodes is scaled from 0.0001 to 0.9999 to allow easy comparison between the target values and those actually attained by the network. The value of 0.9999 is given to the most desirable scheme (i.e. the one with the lowest number of non-distinct nodes) and 0.0001 is given to the scheme with the worst performance. The remaining schemes are scaled linearly between these values. Scaled target output values are shown for the tree 'trials1' in Table 9.1.

Scheme number	1	2	3	4	5	6	7	8
Number of non-distinct nodes	244	439	439	416	221	230	513	186
Scaled target values	0.8226	0.2264	0.2264	0.2967	0.8929	0.8654	0.0001	0.9999

Table 9.1: The scaled target outputs for the tree 'trials1'

Target values are obtained in this way for each tree in the training data set. In the prediction phase, the scheme corresponding to the unit with the largest value is deemed the optimal choice. The selected scheme is compared with the known best scheme for that tree to determine the network performance.

9.3.3.2 Input Units

Each input unit represents one characteristic of the fault tree. There are an infinite number of possibilities for the characteristics and so initially the eleven that had produced the best results in Bartlett's work[19] are used. It was expected that they would produce improved results in this investigation, as they are being used on the reduced fault trees. The characteristics are:

1. Percentage of 'AND' gates in the tree.
2. Percentage of different events that are repeated.
3. Percentage of the total events that are repeated.
4. Top gate type ('AND' gate - 1, 'OR' gate - 0).
5. Number of inputs to the top gate.
6. Number of levels in the tree.
7. Number of basic events in the tree.
8. Greatest number of gates in any level.
9. Number of gates with gate inputs only.
10. Number of gates with event inputs only.
11. The highest multiple of a repeated event.

As the characteristics can take a large range of values (i.e. top gate type can only be 0 or 1, whereas the number of basic events can run into hundreds), they are scaled across the whole set of training trees to be between 0.9999 and 0.0001. The tree with the largest value of a particular characteristic is given an input unit value of 0.9999; the tree with the lowest value is assigned 0.0001 for the unit representing the characteristic. The value of that unit for each of the other trees is scaled linearly in the range relative to the minimum and maximum values. The only exception is for characteristic four, which encodes the top gate type. The unit is given the value 1 for an 'AND' gate and 0 for an 'OR' gate.

The maximum and minimum values of each characteristic are also used to determine the input unit values of the test trees. Again, each characteristic is scaled relative to the extreme values. If a characteristic is found to have a value that is larger or smaller than the maximum or minimum obtained from the training trees, then it is assigned the value 1 or 0 respectively.

9.3.3.3 Training and Validation Data

The 228 fault trees that were used for the analysis of the reduction method in Chapter 6 were initially considered for the training and validation data. However, once the reduction procedure has been applied, 22 of these trees consisted only of a single event. The number of distinct and non-distinct nodes is therefore one for each tree and the number of if-then-else (**ite**) calculations is zero. These trees were not considered useful for the neural network analysis and were removed from the set. A further 42 trees that each had an identical number of non-distinct nodes, distinct nodes and **ite** calculations for every ordering scheme were also removed. This does leave some trees that have identical target values for each scheme when considering only one measure of BDD complexity, but simply means that there are no trees in the set for which no distinctions could be made between the ordering schemes for all three measures. As the number of trees is now significantly smaller, 72 randomly generated trees were identified as suitable for addition into the data set. The summary details of these trees are listed in Appendix II. The data set now consists of 236 trees, 216 of which are used as training trees and the remaining 20 as test trees.

The 20 test trees are chosen from the set, so that each ordering scheme is the optimal choice for approximately the same number of trees. The target schemes for the chosen test trees are shown in Table 9.2.

Tree	bddtest	lisab25	lisab27	lisab34	lisab36	lisab62	lisab70
Target scheme(s)	1, 5, 8	4	3	2	7, 8	1	4
Tree	lisab78	rand121	rand135	rand137	rand139	rand141	rand142
Target scheme(s)	3	4, 6	6	8	8	7	5
Tree	rand144	rand147	rand156	rand159	rando47	rando54	
Target scheme(s)	5	6	7	2, 3	1	2	

Table 9.2: Target schemes for the twenty test fault trees

9.3.3.4 Hidden Layers and Units

The number of hidden layers and the number of units within those layers are central to the multi-layer perceptron model. Using too few may not model the complexity of the problem, but using too many increases the training time dramatically. Masters[36] documents that one hidden layer is usually all that is needed, but that two are sometimes required. However, more than two hidden layers are never theoretically needed.

A guideline for choosing the number of hidden units in a two-layer network (i.e. one which has two layers of weights and therefore has one hidden layer) with d input units and c output units is the geometric pyramid rule, which says that the hidden layer would have $\sqrt{c.d}$ hidden units. Masters recommends using as few hidden units as possible, so starting with two and adding one at a time. Usually three to six are optimal and the Masters suggests that more than ten are almost never needed.

A full investigation was carried out, using one and two hidden layers. When one hidden layer is used, the number of units is incremented from two to nine. When two hidden layers are used, the investigation starts with two units in each and increases the number in the second layer by one each time to a total of six. The process is repeated for three units in the first layer and so on until there are six units in each layer. These figures were used, as including more units in the hidden layers was beyond the computing capabilities available.

9.3.3.5 Parameter Values

Using the enhanced gradient descent technique means that the initial values chosen for the momentum and learning rate parameters should be less critical. However, various values between 0.001 and 1.0 were chosen for each parameter to try and obtain the best possible network performance. The values of α and γ were set to 1.04 and 0.5 respectively throughout the investigation.

9.4 Results of the Multi-Layer Perceptron Investigation

Using the network architecture described in the preceding sections, 520 trials were performed with a single hidden layer and 1000 trials were performed with two hidden layers.

The investigation with the single hidden layer started by using two hidden units and increased the number by one each time until there were nine hidden units. The value of the learning rate parameter, η, varied between 0.01 and 0.15. For each value of η, the momentum, μ, was varied between 0.001 and 0.15. The best results obtained were 6/20 correct predictions on the test data set; the average number of correct predictions was 3.548/20. The results are shown in Table 9.3.

Number of correct predictions	0	1	2	3	4	5	6	7	8
Number of trials	0	1	28	213	244	31	3	0	0

Table 9.3: Results obtained using a single hidden layer in the network

The number of hidden layers was increased to two and the investigation started with each layer containing two hidden nodes. This was increased by one each time until both layers consisted of six hidden units. The value of the learning rate parameter varied between 0.01 and 0.10. For each value of η, the momentum was varied between 0.001 and 0.15. The greatest number of correct predictions was 7/20, which was obtained using six hidden units in each layer and with η and μ set to 0.6 and 0.5 respectively. The average number of correct predictions was again very poor at 3.903, though slightly better than the average obtained using one hidden layer. The results are shown in Table 9.4.

Number of correct predictions	0	1	2	3	4	5	6	7	8
Number of trials	0	1	17	74	896	11	0	1	0

Table 9.4: The results obtained using two hidden layers in the network

Given the poor results obtained using the current network architecture, an alternative was considered, which uses the number of **ite** calculations required to construct the BDDs to calculate the target values for the output units.

9.4.1 Using the Number of If-Then-Else Calculations for the Output Units

The number of non-distinct nodes in the BDD gives an indication of its final size, but it was thought that the number of **ite** calculations required to produce the BDD would give a better indication of the complexity of the BDD. Some BDDs can have very few non-terminal nodes, but require extensive calculations for their construction.

The number of **ite** calculations required to construct the BDD using each of the eight orderings was obtained for the reduced fault trees (Appendices VIII and X) and scaled in the same way as for the number of non-distinct nodes. A new set of test trees was chosen, as the optimal scheme choices are not the same when considering the number of **ite** calculations. The twenty test trees and their target schemes are shown in Table 9.5.

Tree	lisab17	lisab37	lisab47	lisab70	lisab75	rand100	rand135
Target scheme(s)	7	4, 8	5	4	3	5, 8	6
Tree	rand139	rand141	rand142	rand143	rand144	rand147	rand148
Target scheme(s)	2	1	8	7	5	6	2
Tree	rand149	rand151	rand155	rando63	rando68	rando77	
Target scheme(s)	1	8	4	3, 7	1, 2, 6	3	

Table 9.5: Target schemes for the test trees, when considering the **ite** calculations

A total of 256 trials were performed using one hidden layer. The learning rate and momentum parameters were varied between 0.01 - 0.10 and 0.001 - 0.2 respectively. In each case, the number of hidden units was incremented from two to nine. Table 9.6 shows the number of correct predictions obtained from the trials.

Number of correct predictions	0	1	2	3	4	5	6	7	8
Number of trials	0	0	45	94	107	7	2	1	0

Table 9.6: The results obtained using a single hidden layer in the network

The greatest number of correct predictions was 7/20, which was obtained with nine hidden nodes and with the learning rate and momentum parameters set to 0.07 and 0.01. The average number of correct predictions was just 3.336.

Two hidden layers were also considered. The learning rate and momentum parameters were assigned exactly the same set of values as for the single layer investigation and the number of units in the hidden layers was varied between two and six for each case. A total of 700 trials were conducted and the results are shown in Table 9.7. The greatest number of correct predictions was 6/20 and the average number was calculated as 3.681.

Number of correct predictions	0	1	2	3	4	5	6	7	8
Number of trials	1	0	11	242	414	19	13	0	0

Table 9.7: The results obtained using two hidden layers in the network

The results obtained for both one hidden layer and two hidden layers are slightly worse than those recorded when using the number of non-distinct nodes as the target values. However, the difference is minimal and neither set currently looks promising - in order to implement this technique, a success rate of more than 80% would be desired. A different network architecture is now considered, which halves the number of possible ordering schemes to four. This is discussed in the following section.

9.4.2 Reducing the Number of Output Units to Four

The reason for reducing the number of output units was because it was felt that perhaps the number of training trees was not sufficient to allow the network to differentiate between eight possible outcomes. Rather than simply choosing four schemes at random, they are grouped into pairs and each possible combination of pairs is considered. For example, schemes 1 and 2 are paired, 3 and 4 are paired and so on. Then, each combination of these pairs can be put as the output units of the network - this gives six possible sets of four schemes:

$$\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \{1, 2, 5, 6\}, \{3, 4, 7, 8\}, \{1, 2, 7, 8\}, \{3, 4, 5, 6\}$$

Each set can be analysed to see which scheme comes out as the best choice and the results compared to give an overall optimal scheme. For example, if the outcome using the first set is that scheme 2 is the best choice but when using the second set scheme 8 is the best choice, then the scheme selected using set 5 (which contains both) would compare these two schemes and differentiate between them. If necessary, each set of schemes could use a different network architecture to gain the best possible results.

The investigation started by considering the number of lte calculations obtained using the first set of four schemes, i.e. {1, 2, 3, 4}, as the output units. The test trees are again chosen so that each scheme is the optimal choice for a similar number of trees, as shown in Table 9.8.

Tree	lisab10	lisab17	lisab35	lisab44	lisab77	lisaba9	rand135
Target scheme(s)	4	2, 3	4	4	3	4	3
Tree	rand139	rand141	rand142	rand143	rand144	rand147	rand148
Target scheme(s)	2	1	3	2	1	1	2
Tree	rand149	rand150	rand153	rand154	rand155	rand161	
Target scheme(s)	1	1	2	2	4	3	

Figure 9.8: Target schemes for the test trees, using **ite** calculations and four output nodes

A total of 1023 trials were conducted, with 648 using one hidden layer and the remaining using two hidden layers. For the trials with one hidden layer, the learning rate parameter was varied between 0.005 and 0.75, whilst momentum values of between 0.005 and 0.9 were tried. The number of hidden units was varied between two and nine for each parameter setting. The best result was 10/20 correct predictions, achieved using eight hidden units and parameter values $\eta = 0.04$ and $\mu = 0.9$. The results for all the trials using one hidden layer are shown in Table 9.9.

Number of correct predictions	0	1	2	3	4	5	6	7	8	9	10	11
Number of trials	1	3	15	69	110	146	142	112	33	16	1	0

Table 9.9: The results obtained using one hidden layer in the network

The average number of correct predictions is 5.346/20. Although the results appear better than for the previous network architectures, there are fewer schemes from which to choose, and so the expected number of correct predictions is higher.

For the trials using two hidden layers, the learning rate parameter was varied between 0.1 and 0.8 and the momentum parameter was assigned values between 0.05 and 0.9. Of the 375 trials conducted, ten correct predictions were achieved in two cases. Both were obtained using $\eta = 0.5$, $\mu = 0.5$ with six units in the first hidden layer. The second layer consisted of three units in the first case and six units in the second case. The average number of correct predictions over all the trials was 5.696. The full set of results is shown in Table 9.10.

Number of correct predictions	0	1	2	3	4	5	6	7	8	9	10	11
Number of trials	0	0	1	4	26	126	163	38	11	4	2	0

Table 9.10: The results obtained using two hidden layers in the network

As the network is performing so poorly, it can be concluded the current architecture does not adequately describe the problem. Further sets of four output units are not therefore investigated at this stage. Instead, the choice of fault tree characteristics is examined.

9.4.3 Modified Fault Tree Characteristics

In order to be able to differentiate between fault trees, the characteristics should describe the features of each tree that make it unique. A modified set of key features was therefore chosen with the aim of being able to draw a representation of the tree using only the given characteristics data. There are many possibilities to consider, but ten were initially selected and are shown below:

1. Type of the top gate.
2. Number of levels in the fault tree.
3. Number of different basic events.
4. Total number of basic events.
5. Average number of event inputs to the gates.
6. Percentage of the different events that are repeated in the tree.
7. Number of different gates.
8. Total number of gates.
9. Percentage of 'AND' gates in the tree.
10. Percentage of different gates that are repeated in the tree.

The modified characteristics are calculated using the programs newchar_tr.c for the training trees and newchar_test.c for the test trees, which were written as part of the research. They are scaled in the same way as for the original characteristics, as described in section 9.3.3.2.

The initial investigation with the new characteristics uses eight output units, whose target values are calculated according to the number of **ite** calculations required to obtain the BDDs. The test trees are therefore the same as those used in section 9.4.1 and are shown with their target schemes in Table 9.5.

A total of 512 trials were conducted using one hidden layer, with the number of units ranging from two to nine. The parameter η was varied between 0.05 and 0.75 and μ was assigned values between 0.001 and 0.9. The results are shown in Table 9.11.

Number of correct predictions	0	1	2	3	4	5	6	7	8
Number of trials	4	12	42	159	222	64	8	1	0

Table 9.11: The results obtained using one hidden layer in the network

The greatest number of correct predictions is 7/20, which was obtained using six hidden units and parameter values $\eta = 0.5$ and $\mu = 0.5$. On average, the number of correct predictions was only 3.586.

1625 trials were conducted using two hidden layers in the network. The number of units in each hidden layer ranged from two up to six. Again, the parameter η was varied between 0.05 and 0.75 and μ was varied between 0.001 and 0.9. The results are shown in Table 9.12.

Number of correct predictions	0	1	2	3	4	5	6	7	8
Number of trials	0	4	65	717	794	44	1	0	0

Table 9.12: The results obtained using two hidden layers in the network

The greatest number of correct predictions is 6/20, which was obtained using five units in the first hidden layer, six units in the second hidden layer and parameter values of $\eta = 0.5$ and $\mu = 0.1$. Although more trials were performed than with one hidden layer, fewer resulted in five, six and seven correct predictions and on average only 3.500 correct predictions were made.

Finally, the modified characteristics were used with four output units in the network. The test trees are the same as those used in section 9.4.2 with the target schemes shown in Table 9.8.

A total of 520 trials were performed using one hidden layer, with the parameters varying in the ranges 0.05 - 0.75 for η and 0.001 - 0.9 for μ. The greatest number of correct predictions is 9/20, which was obtained from 15 trials. The average number of correct predictions is 5.763. Table 9.13 shows the results from all 520 trials.

Number of correct predictions	0	1	2	3	4	5	6	7	8	9	10
Number of trials	0	0	2	13	68	126	198	55	43	15	0

Table 9.13: The results obtained using one hidden layer in the network

The use of two hidden layers did not produce improved results, as can be seen from Table 9.14. A total of 1625 trials were conducted, with the same range of parameter values as for the one-layer investigation.

Number of correct predictions	0	1	2	3	4	5	6	7	8	9	10
Number of trials	0	0	0	8	44	578	885	93	16	1	0

Table 9.14: The results obtained using two hidden layers in the network

The greatest number of correct predictions was 9/20, which was obtained using five units in the first hidden layer, six units in the second hidden layer and parameter values of $\eta = 0.05$ and $\mu = 0.9$. The average over all the trials is 5.654, which is lower than obtained with one hidden layer.

9.4.4 Discussion of Results

Overall, the results from the multi-layer perceptron investigation have been very disappointing. When using eight output units the greatest number of correct predictions was seven out of twenty and when using four output units the best result was ten out of twenty correct predictions. In order to be a viable technique for scheme selection within the fault tree strategy, at least 80% accuracy would be required.

The chosen network architectures have been unable to reproduce the results previously obtained by Bartlett, where 14/20 correct predictions were attained. The main difference in the approaches is that the reduced fault trees have been used in this investigation. Although it was expected that this would lead to improved results, because unnecessary elements have been removed from the system, it could in fact have made it more difficult for the network to distinguish between the fault trees.

Another reason for the apparent disparity in the network performance could be the choice of test data. As explained in section 9.3.3.3, the test trees for each investigation are chosen so that each scheme appears as the best choice for approximately the same number of trees. However, the initial test data chosen by Bartlett is shown in Table 9.15.

Tree	1	2	3	4	5	6	7	8	9	10
Best scheme(s)	1-6	1	1-6	3	2	1,2	2	4	1,3	2

Tree	11	12	13	14	15	16	17	18	19	20
Best scheme(s)	1,2,4	3	2	3	1-6	1-3	1-6	2	3	3

Table 9.15: Target schemes for the set of twenty test trees in Bartlett's study

Having used six ordering schemes, it can be seen that the worst result would be 4/20 correct predictions as each scheme performs equally well on trees 1, 3, 15 and 17. If the network simply chooses the same scheme for each tree, then by selecting schemes 1, 2 and 3 it would correctly predict nine, twelve and eleven cases out of twenty respectively. The distribution of correct predictions for 200 trials shows that they range from the minimum possible up to 14/20 for one case. They are mainly grouped around 7-11 correct predictions, with 8 correct predictions obtained the greatest number of times (~38/200 trials). This distribution is therefore not dissimilar to the results obtained in the current investigation.

As the current neural network technique has not adequately modelled the ordering problem, a second method known as the radial basis function neural network is considered. A significant advantage of this network model is the fast training times, which allows for a more thorough analysis. The radial basis function network is described in the following sections.

9.5 Radial Basis Function Neural Network

The radial basis function neural network model again performs a non-linear mapping from a set of input units that represent the fault tree characteristics to a set of output units, which each represent a variable ordering scheme. Diagrammatically, it is very similar to the multi-layer perceptron, as shown in Figure 9.4:

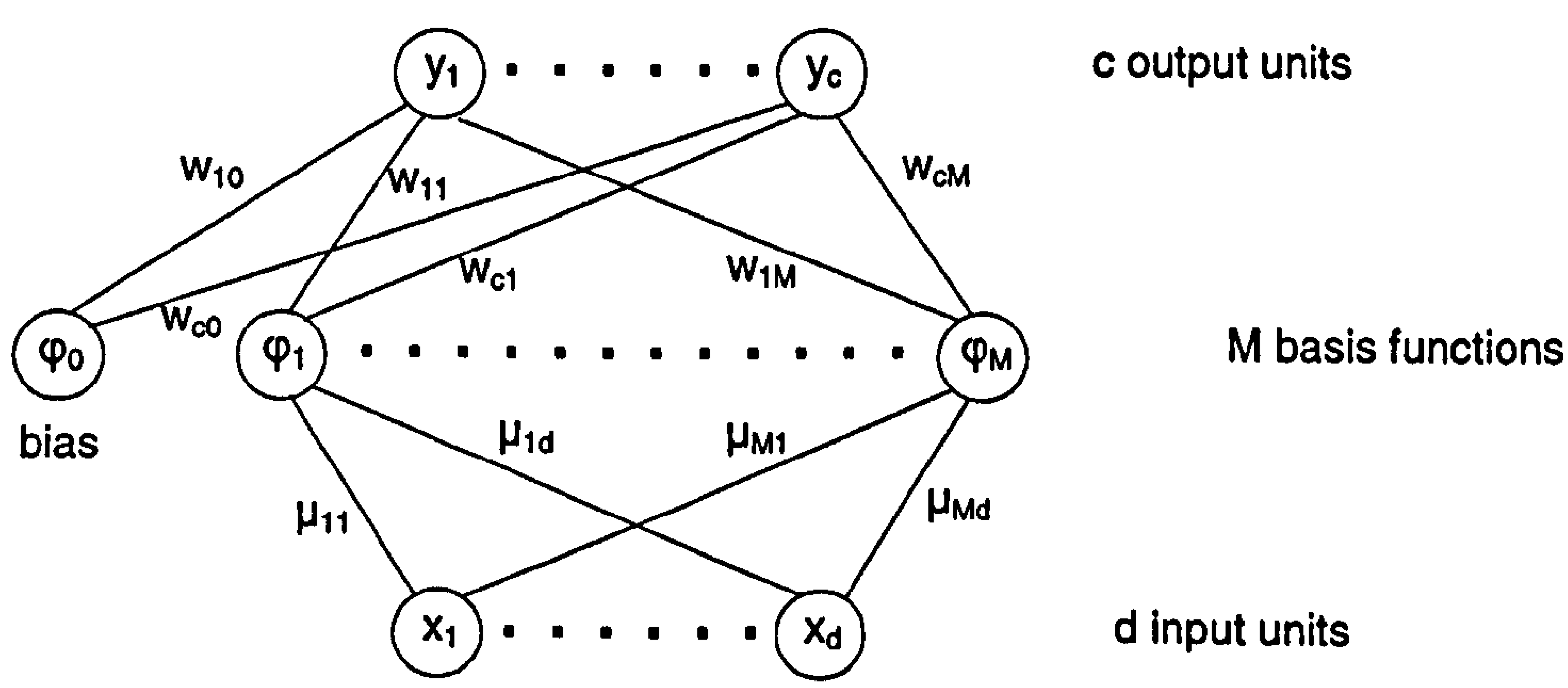


Figure 9.4: Radial basis function neural network

Unlike the multi-layer perceptron, which can have any number of hidden layers, the radial basis function network has only one hidden layer, which is made up of units known as basis functions. The outputs of the basis functions are determined by the distance between the input vector and a prototype vector.

As with the multi-layer perceptron model, connections run between every unit in one layer to every unit in the next. The connections between the units in the input layer and a basis function in the hidden layer represent the elements of the vector determining the centre of that basis function. The connections between the hidden layer and the output layer represent the weights of the network and control the influence of each basis function on the output units, in the same way as with the multi-layer perceptron model.

The training procedure is implemented in two stages. The first stage determines the values of the parameters governing the basis functions using unsupervised training methods. The second stage of training uses a supervised technique to calculate the values of the second layer weights. The parameters and weights calculated in training are subsequently used to progress through the network in the testing phase. The two training stages are described in detail in the following sections.

9.5.1 Training Stage One

The first stage of training uses unsupervised techniques to determine the parameters of the basis functions using only the input data. There are several non-linear basis functions that can be used; the one chosen for this investigation is the Gaussian function of the form:

$$\varphi_j(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) \qquad 9.20$$

where $\mathbf{x}$ is the d-dimensional input vector with elements x_i, $\boldsymbol{\mu}_j$ is the vector determining the centre of the basis function φ_j with elements μ_{ji} and σ is the width parameter, which controls the smoothness of the interpolating function. The Gaussian function has the property that $\varphi \to 0$ as $|x| \to \infty$. As it is a localised function, only a few hidden units will have significant outputs for any given input vector.

Another choice of localised basis function is:

$$\varphi(x) = (x^2 + \sigma^2)^{-\alpha}, \alpha > 0 \qquad 9.21$$

However, the basis function need not be localised and other choices are:

- the thin-plate spline function, $\varphi(x) = x^2 \ln(x)$ 9.22

- the function $\varphi(\mathbf{x}) = (x^2 + \sigma^2)^\beta,\ 0 < \beta < 1$ 9.23
- the cubic function, $\varphi(\mathbf{x}) = x^3$ 9.24
- and the linear function, $\varphi(\mathbf{x}) = x$ 9.25

which all have the property that $\varphi \to \infty$ as $|x| \to \infty$.

However, the Gaussian function will be used and the parameters that must be calculated during this training stage are therefore:

- The radial basis function centres.
- The width parameters.

The radial basis function centres are chosen as a random subset of the input vectors. This is one of the simplest possible methods of selecting the centres, but it is very fast and is a good starting point from which to work. Other methods can be found in reference 35. The number of centres can range from one to a maximum of the number of input vectors used, though there are typically many less than this maximum.

The width parameter of each radial basis function is given the same value, which is equal to the average distance between their centres. It is also possible to use multiples of this value, or indeed different parameters for each basis function. Again these alternative methods are discussed in detail in reference 35. Using the average distance between the centres ensures that the basis functions overlap to some degree and so give a relatively smooth representation of the distribution of the data set.

9.5.2 Training Stage Two

The second stage of training uses a supervised technique to calculate the optimum values for the final layer weights in the network. The value of each output unit is calculated as a weighted sum of its inputs, giving:

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\varphi_j(\mathbf{x}) \qquad 9.26$$

where w_{kj} is a weight going from unit j in the hidden layer to unit k in the output layer and M is the total number of basis functions. φ_0 denotes the bias, whose output is fixed at one.

The weights are optimised by minimising the error at the output units. The sum-of-squares error function for the network is given by:

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(y_k(\mathbf{x}^n) - t_k^n)^2 \qquad 9.27$$

where t_k^n is the target value for output unit k, when the network is presented with input vector $\mathbf{x}^n$. By substituting Equation 9.26 for $y_k(\mathbf{x}^n)$, this can be re-written as:

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}\left(\sum_{j=0}^{M} w_{kj}\varphi_j(\mathbf{x}^n) - t_k^n\right)^2 \qquad 9.28$$

Differentiating this expression with respect to the weights and setting the derivative equal to zero gives a set of equations of the form:

$$\sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_{kj}\varphi_j(\mathbf{x}^n) - t_k^n\right)\varphi_j(\mathbf{x}^n) = 0 \qquad 9.29$$

In order to solve these equations, they can be written in matrix notation as:

$$(\mathbf{\Phi}^T \mathbf{\Phi})\, \mathbf{W}^T = \mathbf{\Phi}^T \mathbf{T} \qquad 9.30$$

where $\mathbf{\Phi}$ is an N x M matrix with elements $\varphi_j(\mathbf{x}^n)$ in the n^{th} row and j^{th} column,

$\mathbf{W}$ is a c x M matrix with elements w_{kj} in the k^{th} row and j^{th} column,

$\mathbf{T}$ is an N x c matrix with elements t_k^n in the n^{th} row and k^{th} column.

Re-arranging for **W** gives:

$$\mathbf{W}^T = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{T} \qquad 9.31$$

Therefore, the calculation of the weights is a linear problem and they can be found very easily using Equation 9.31.

A non-linear activation function can be applied to the output units, but the calculation of the weights would then become a non-linear optimisation. One of the major advantages of using radial basis function networks is the possibility of avoiding the need for such an optimisation, resulting in much faster training times.

9.5.3 A Comparison of the Multi-Layer Perceptron and Radial Basis Function Models

The multi-layer perceptron and radial basis function models have very similar roles, in that they are both techniques for performing non-linear mappings between multi-dimensional spaces. However, the networks themselves have significant differences and employ different techniques for their analysis. Some of the main differences are outlined below.

- The outputs of the hidden units in the multi-layer perceptron are calculated by applying a non-linear activation function to the weighted sum of their inputs. In contrast, the outputs of the hidden units of the radial basis function network are generated depending on the distance between the input vector and a prototype vector and transformed using a localised basis function.

- Many hidden units contribute to the value of the output units in the multi-layer perceptron. This means that the training process to determine the weights is highly non-linear and can lead to very slow convergence times. The radial basis function network uses localised basis functions for the hidden units, which means that typically only a few will have significant outputs that contribute to the values of the output units and examples far from the decision boundaries have little influence on the network.

- The multi-layer perceptron can have many layers of hidden weights, leading to complex network architectures and long training times. The radial basis function network, however, has a simple structure, consisting of two layers of weights. The first layer represents the parameters of the basis functions and the second layer forms linear combinations of the outputs of the basis functions to generate the values of the output units.

- The weights in the multi-layer perceptron are determined simultaneously during a single training phase. However, with the radial basis function network, the training takes place in two stages. The first stage uses an unsupervised technique to determine the parameters of the radial basis functions. This is very fast, but means that the centres and width parameters of the basis functions are not necessarily optimal for the problem. The second stage determines the final layer weights using a fast linear supervised method. The training process is much faster than for the multi-layer perceptron as it does not require a non-linear optimisation.

9.6 Results of the Radial Basis Function Investigation

Each of the five network architectures investigated for the multi-layer perceptron is examined using the radial basis function network. This allows for a direct comparison of the two methods.

The number of radial basis function centres can range from one up to the number of training trees. The centres are chosen randomly with a random number sequence that is initiated with the use of a seed value. Throughout the network evaluation, seed values from 1 to 500 are used for each possible number of centres (1 - 216), which results in 108,000 trials. This should give a good indication of the network performance.

9.6.1 Initial Network Architecture

The initial network architecture is the same as for the multi-layer perceptron and is discussed in sections 9.3.3.1 - 9.3.3.3. Briefly, this comprises of eleven input units, each representing a fault tree characteristic and eight output units that represent the ordering schemes available for selection. The target values for the output units are determined by the number of non-distinct nodes in the BDDs constructed using each of the ordering schemes. 216 fault trees are used in the training phase and twenty trees are used in the predictive phase. The test data for the initial investigation is shown in Table 9.2.

The number of correct predictions was recorded for each trial and the results are shown in Table 9.16.

Number of correct predictions	0	1	2	3	4	5	6	7
Number of trials	3	71	310	59296	48159	153	8	0

Table 9.16: The results obtained using the initial network architecture

The greatest number of correct predictions is 6/20, which is lower than for the multi-layer perceptron model, which succeeded in predicting the correct scheme in seven cases. Given the large number of trials that were performed, a greater spread of results was predicted. From these results it is obvious that the current network architecture is not capable of modelling the variable ordering problem.

9.6.2 Using the Number of If-Then-Else Calculations for the Output Units

The second network uses the number of **ite** calculations required to obtain the BDDs as the target values for the output units. The results of the 108,000 trials are shown in Table 9.17.

Number of correct predictions	0	1	2	3	4	5	6	7	8
Number of trials	2	58	298	72791	34602	197	50	1	0

Table 9.17: The results obtained using the number of **ite** calculations and eight output units

The best result is 7/20 correct predictions, which matches that obtained using the multi-layer perceptron network. However, this was produced in only one trial. Over 99% of the trials resulted in three or four correct predictions.

9.6.3 Reducing the Number of Output Units to Four

The third network architecture uses four output units, whose target values are again determined by the number of ite calculations required to obtain the BDDs. The schemes used are 1, 2, 3 and 4. Table 9.18 shows the results obtained from the trials.

Number of correct predictions	0	1	2	3	4	5	6	7	8	9	10
Number of trials	0	2	14	84	9013	75653	22955	243	33	3	0

Table 9.18: The results obtained using four output units for schemes 1, 2, 3 and 4

The greatest number of correct predictions is 9/20, which is one less than the best result obtained using the multi-layer perceptron model. In order to check that the chosen four schemes are not simply a 'bad' combination, the trials were also conducted using schemes 5, 6, 7 and 8. The modified set of test fault trees is shown in Table 9.19.

Tree	lisab17	lisab22	lisab25	lisab47	lisab57	rand135	rand139
Target scheme(s)	7	5	7	5	5	6	7
Tree	rand141	rand142	rand143	rand144	rand146	rand147	rand149
Target scheme(s)	7	8	7	5	8	6	5
Tree	rand150	rand151	rand153	rand154	rand155	rand156	
Target scheme(s)	8	8	6	8	6	6	

Figure 9.19: Target schemes for the test trees using the four output schemes 5, 6, 7 and 8

The results of the trials using the modified set of output units are given in Table 9.20.

Number of correct predictions	0	1	2	3	4	5	6	7	8	9	10
Number of trials	0	2	18	93	193	107296	235	125	37	1	0

Table 9.20 The results obtained using four output units for schemes 5, 6, 7 and 8

They clearly show that no improvement has been made to the network performance, with over 99% of trials predicting only 5/20 correct ordering schemes.

9.6.4 Modified Fault Tree Characteristics

The final network architectures use the modified fault tree characteristics listed in section 9.4.3. The results obtained using eight and four output schemes are shown in Tables 9.21 and 9.22 respectively.

Number of correct predictions	0	1	2	3	4	5	6	7
Number of trials	0	43	285	70882	36499	253	38	0

Table 9.21: The results obtained using the modified characteristics and eight output units

Number of correct predictions	0	1	2	3	4	5	6	7	8	9	10
Number of trials	0	1	11	63	7817	63099	36570	344	78	14	3

Table 9.22: The results obtained using the modified characteristics and four output units

The new set of characteristics does not significantly change the network performance. Although ten correct predictions were made in three of the trials using four output units (whereas in the equivalent multi-layer perceptron investigation the best result was nine correct predictions), the network is clearly incapable of modelling the ordering problem sufficiently well.

Overall, the radial basis function network does not appear to perform as well as the multi-layer perceptron model. Although more trials were conducted, due to the faster training times, it proved impossible to establish a good network architecture. Alternative methods for selecting the basis function centres and setting the width parameter could be considered, but the performance did not seem promising enough to warrant further investigation at this stage.

9.7 Conclusions

The neural network techniques have proved unsuccessful in modelling the variable ordering problem. Neither the multi-layer perceptron nor the radial basis function models have been able to reproduce the best result of 14/20 correct predictions achieved in previous work in the area[19].

Numerous trials were conducted with both models, but the best result when using eight output units was 7/20 correct predictions. When the number of output units was reduced to four, the best result was 10/20 correct predictions, but this was simply due to fewer options being available for selection. Both models produced the same best result for each number of output units. These results and the number of trials conducted show conclusively that the neural network models used are not capable of predicting the most appropriate ordering schemes for fault trees.

Many features of the neural network models could be altered to try to improve the networks' performance. Several alternatives have been suggested throughout the chapter, but it is thought that the most likely reason for the poor performance of the network is that the chosen fault tree characteristics do not accurately represent the problem. There are an infinite number of choices for the characteristics and they need to be thoroughly reviewed before other, more detailed aspects of the models are examined. Another reason that the neural network approach has proved unsuccessful could be the non-unique way in which fault trees are written. Although the reduced trees have been restructured to a more concise format than the original trees, there are still numerous ways in which they can be constructed (for example, altering the order of gate inputs changes the tree structure, but maintains the underlying logic), that could result in different values for the characteristics. Consequently, the network models may not be able to distinguish between the different classes of fault tree accurately and so cannot predict a correct outcome for new input data.

It is concluded therefore, that the current neural network models do not provide a satisfactory mechanism for selecting the ordering schemes to be used within the fault tree analysis strategy described in the previous chapter. However, the techniques used within the strategy for reducing the fault tree complexity have been shown to be very successful and as such, further research will focus on extending the methods of fault tree simplification.

Chapter 10: Extending the Reduction Technique

10.1 Introduction

The Faunet reduction technique, discussed in Chapter 6, has been shown to reduce the size of a sample set of fault trees and their resulting BDDs significantly. However, structures were identified within the reduced fault trees that could be further simplified through the application of the absorption and idempotent laws to the fault tree logic.

This chapter describes how these laws can be incorporated into the reduction technique, by further manipulating the fault tree structure to give a more concise representation of the logic function. The aim of this work is to restructure the trees in such a way that they can be used to construct smaller BDDs, using fewer calculations, than are possible with either the original fault trees or those restructured using the Faunet reduction technique.

10.2 Application of the Absorption and Idempotent Laws to Fault Tree Structures

The Boolean laws of absorption are given as follows:

$$a+(a.b) = a \tag{10.1}$$

$$a.(a+b) = a \tag{10.2}$$

According to these laws, fault tree structures such as those shown in Figure 10.1 (obtained from the left-hand sides of Equations 10.1 and 10.2), where an event is repeated on consecutive levels of a fault tree branch, will simply reduce to a single event 'a'.

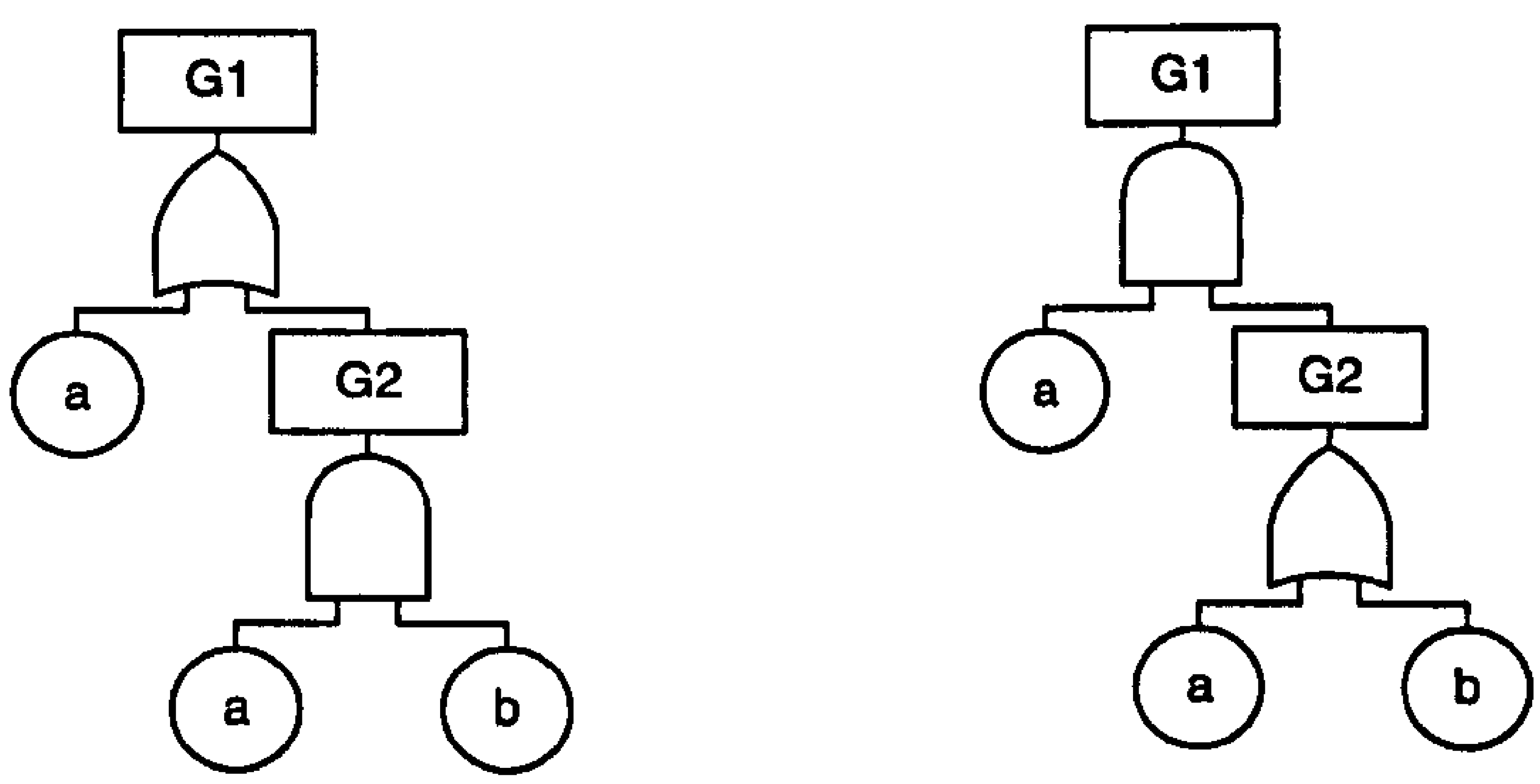


Figure 10.1: Fault tree segments that can be reduced to a single event 'a'

Further structures of this type can be examined by considering events that are repeated over any number of levels of a fault tree branch. Figure 10.1 shows the simplest possible case, with only one level between the occurrences of the repeated event, but in fact the fault tree can be simplified when a repeated event appears any number of levels down the tree.

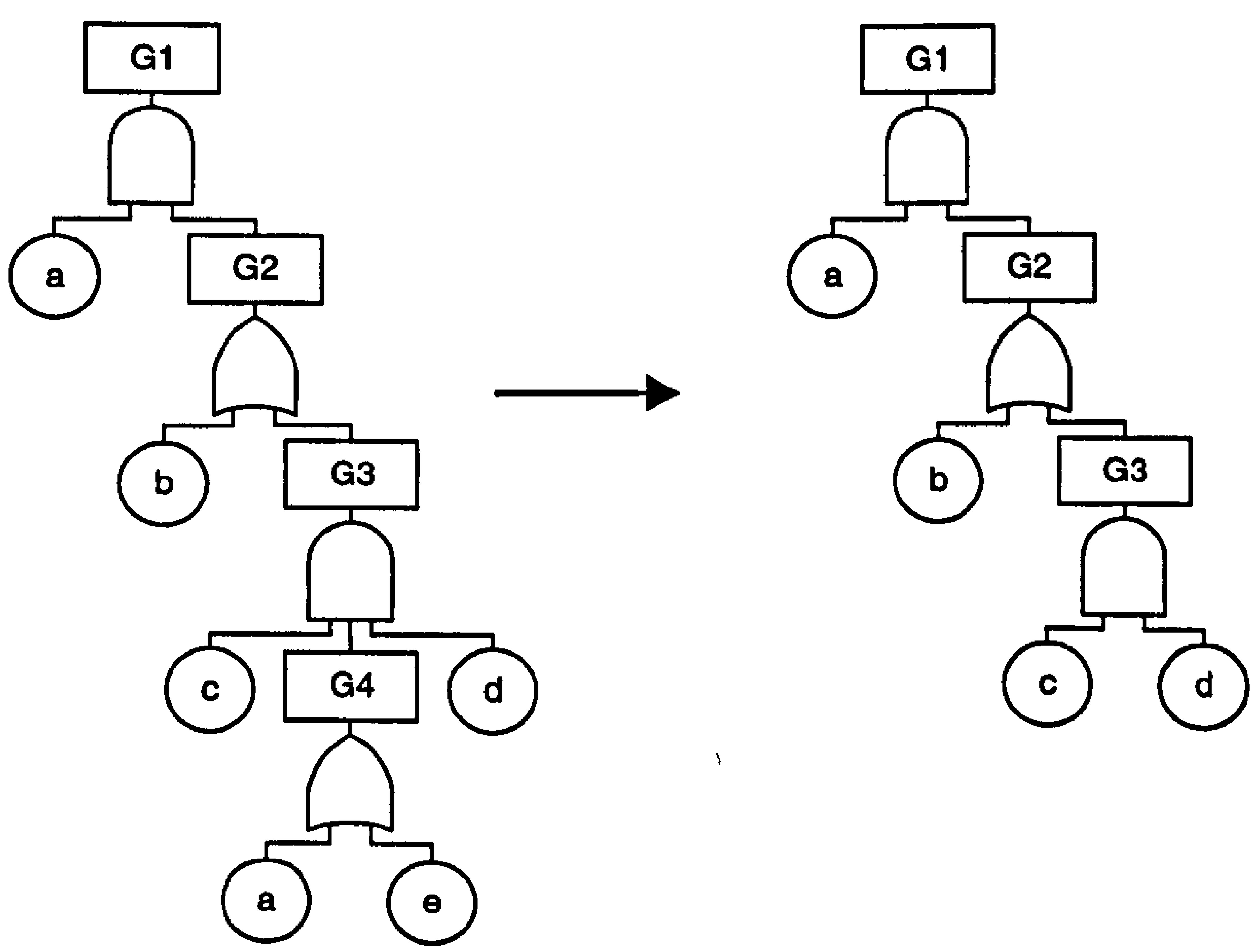
The simplification of the tree structure is based upon the application of the absorption and/or idempotent laws (i.e. a.a = a) to its underlying logic. The second law of absorption given by

Equation 10.2 is actually a combination of Equation 10.1 and the idempotent law and so the second tree segment shown in Figure 10.1 requires the use of both laws for its reduction to a single event. Some logic expressions (such as those of the trees shown in Figure 10.1) would only require the use of the absorption laws, some the idempotent laws and others require a combination of both. However, regardless of which laws would be necessary for the reduction of the logic expression, the way in which the tree structure is manipulated is dependent only on whether or not the two occurrences of the repeated event occur under the same gate type.

The two gates to which the repeated event is an input are referred to as the primary and secondary gates, where the primary gate is the one located further up the fault tree branch. The following sections describe the manipulation of the fault tree according to the types of the primary and secondary gates. In each case, the fault tree must have an alternating sequence of 'AND' and 'OR' gates (which it does after the contraction stage of the reduction process) before the technique can be applied.

10.2.1 Primary and Secondary Gates of Different Types

For fault tree branches that have primary and secondary gates of different types with an event in common, the structure is simplified by removing the whole of the secondary gate and its descendents. Figure 10.2(a) shows a tree with event 'a' common to gates G1 and G4. The tree is reduced by removing gate G4 and its descendants as described above. This results in the logically equivalent tree shown in Figure 10.2(b).



(a) The original fault tree segment (b) The simplified fault tree segment

Figure 10.2: Application of the absorption and idempotent laws for the case where the primary and secondary gates are of a different type

The reduction of the logic expression confirms this re-arrangement. For the original tree segment shown in Figure 10.2(a), G1 is given by:

$$G1 = a.(b + (c.d.(a + e)))$$
$$= a.b + a.c.d.a + a.c.d.e$$

Applying the idempotent law a.a = a to the second term reduces the expression giving

$$G1 = a.b + a.c.d + a.c.d.e$$

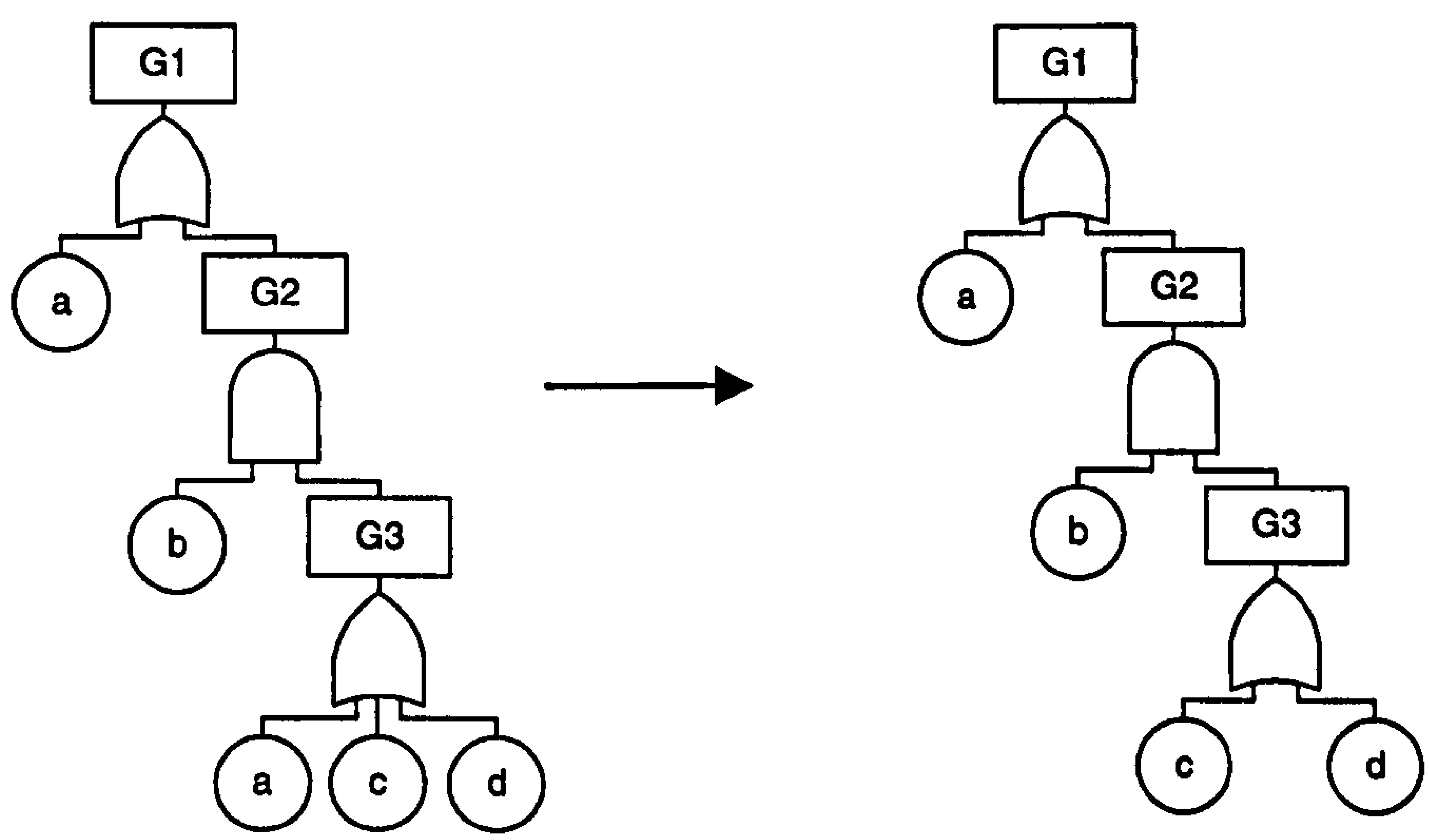
Finally, the absorption law removes the third term to give

$$G1 = a.b + a.c.d$$

which represents the simplified tree segment in Figure 10.2(b). The method is applied in exactly the same way for primary gates that are 'OR' gates.

10.2.2 Primary and Secondary Gates of the Same Gate Type

For fault tree branches that have primary and secondary gates of the same type with an event in common, the structure is simplified by deleting the occurrence of the event beneath the secondary gate. Figure 10.3(a) shows a tree with event 'a' repeated under gates G1 and G3. In order to simplify the tree, event 'a' is removed from the inputs to G3, which is the secondary gate. This simply removes any combinations of events that include 'a', as 'a' alone is sufficient to cause system failure.



(a) The original fault tree segment (b) The simplified fault tree segment

Figure 10.3: Application of the absorption law for the case where the primary and secondary gates are the same type

In this case, the logic expression can be reduced with the application of the absorption law:

$$G1 = a + b.(a + c + d)$$
$$= a + b.a + b.c + b.d$$
$$= a + b.c + b.d$$

This reduced logic expression represents the simplified tree structure shown in Figure 10.3(b).

The method is applied in exactly the same way to trees whose primary and secondary gates are 'AND' gates. The following section describes the one exception to this general method.

10.2.2.1 Special Case

There is one special case to consider, which occurs when the inputs to the secondary gate are a subset of the inputs to the primary gate and the gates are of the same type. In this instance, the fault tree branch is terminated from the gate above the secondary gate. An example of this special case, with primary and secondary gates of type 'AND' is shown in Figure 10.4.

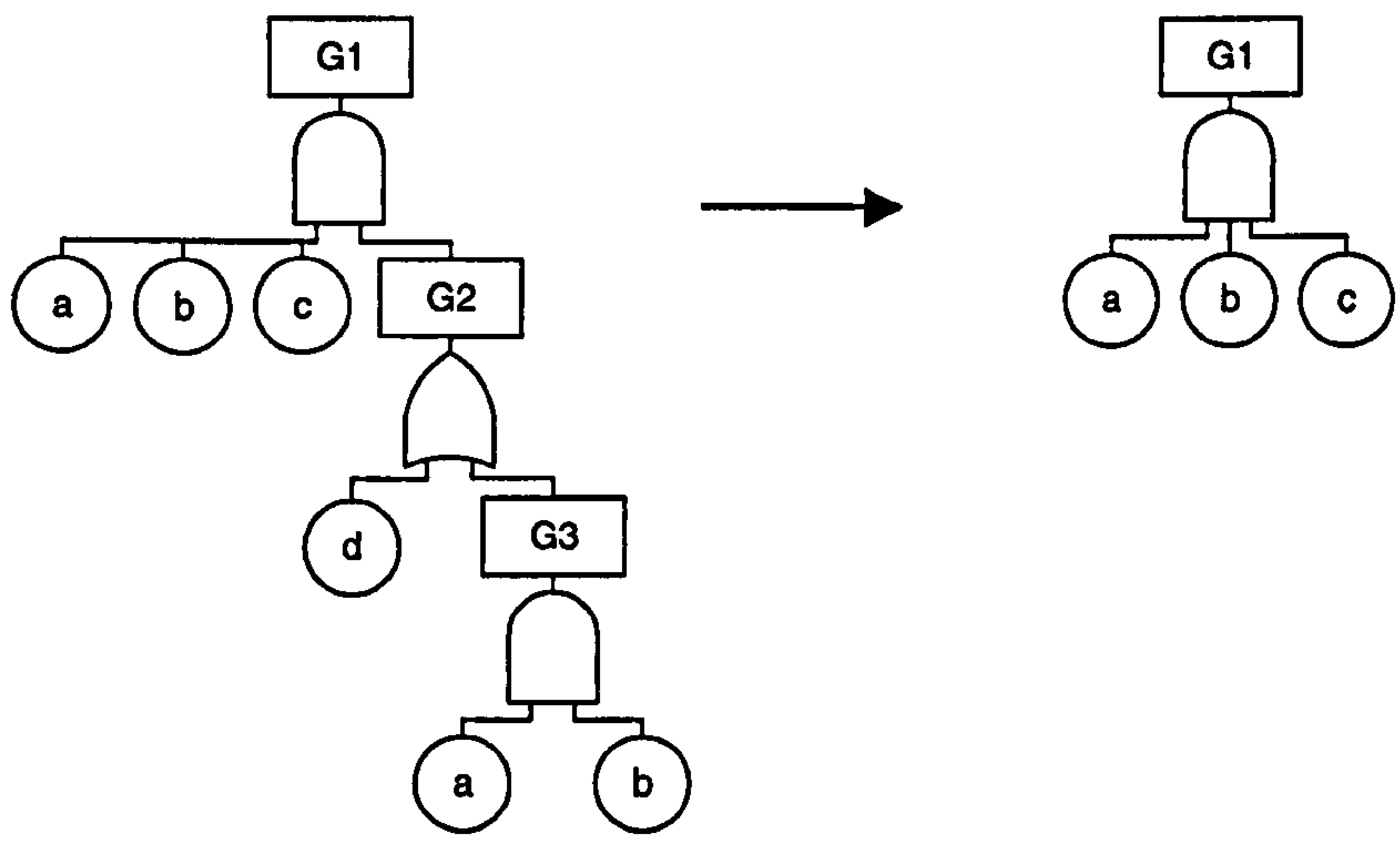


Figure 10.4: The special case, where the secondary gate is a subset of the primary gate

Event 'd' is irrelevant to the failure of the system, so the branch below and including gate G2 is removed. This special case must be accounted for separately, as the general method of dealing with primary and secondary gates of the same type would simply remove all the inputs to the secondary gate, but leave the gate above in place.

The application of the absorption and idempotent laws to the fault tree will be referred to as the absorption technique. The technique is applied throughout the tree, considering not only event inputs to the gates, but also gate inputs. Gates that are repeated on a fault tree branch are also subject to the absorption and idempotent laws and are treated in exactly the same way as the events. However, only the case where the primary and secondary gates are of the same type will apply, as otherwise the tree would not be an alternating sequence of gate types.

10.3 Implementation of the Absorption Technique

The absorption technique has been programmed as part of the research (extended.c) and is capable of dealing with any given fault tree structure. The implementation is described in this section with the aid of two worked examples, each covering different aspects of the technique.

10.3.1 Worked Example

Consider the fault tree shown in Figure 10.5.

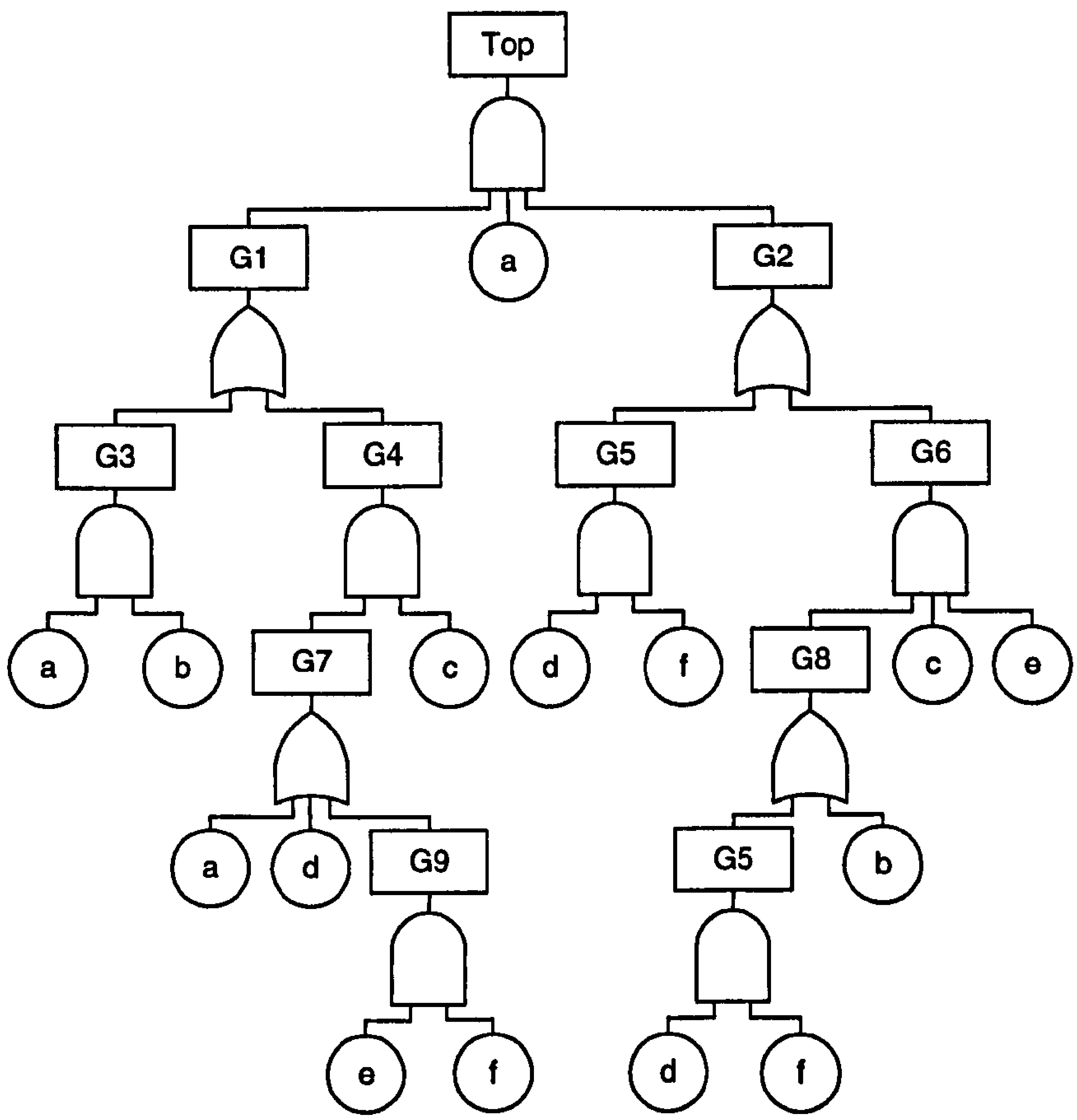


Figure 10.5: Example fault tree

The tree must be input to the program in the form of a data file, which represents the fault tree by listing each gate, together with its type ('AND' or 'OR') and inputs. It is this data that is subsequently manipulated by the program and converted back to a tree structure after the process is complete. The data for the fault tree shown in Figure 10.5 is given in Table 10.1.

Gate name	Type	Number of gate inputs	Number of event inputs	Inputs		
Top	AND	2	1	G1	G2	a
G1	OR	2	0	G3	G4	
G2	OR	2	0	G5	G6	
G3	AND	0	2	a	b	
G4	AND	1	1	G7	c	
G5	AND	0	2	d	f	
G6	AND	1	2	G8	c	e
G7	OR	1	2	G9	a	d
G8	OR	1	1	G5	b	
G9	AND	0	2	e	f	

Table 10.1: Data for the fault tree in Figure 10.5

Each column of the fault tree data is held in an array and in the program is converted to a numerical format for ease of manipulation. The absorption technique is applied to this tree in three stages.

Absorption Stage One

Starting at the head of the tree, a depth-first exploration is undertaken, which identifies inputs to the gates that occur more than once in the fault tree data (as they must occur at least twice if it is to appear further down the branch). This is achieved by referring to an array that holds the number of occurrences of each gate and event and which is updated as necessary as changes are made to the data. If an input is repeated in the data, its gate becomes known as the primary gate and a further depth-first exploration takes place through the branches beneath that gate to establish whether the event occurs again in its descendants. As explained in the previous sections, any subsequent changes to the tree data depend on whether the second occurrence (under the gate referred to as the secondary gate) is an input to an 'AND' gate or an 'OR' gate.

Of the inputs to gate Top, event 'a' occurs elsewhere in the data, so Top becomes the primary gate and the branches below are searched for any other occurrences of 'a'. Gate G3 is identified as having 'a' as an input and as it is of the same type (they are both 'AND' gates), this results in the removal of 'a' from the inputs to G3 and its number of occurrences is reduced by one. However, as gate G3 now has only one input it can be removed and its remaining input, event 'b', becomes an input to G1, the parent gate of G3. After this first stage, the fault tree is altered to give the new tree shown in Figure 10.6, with the corresponding data shown in Table 10.2.

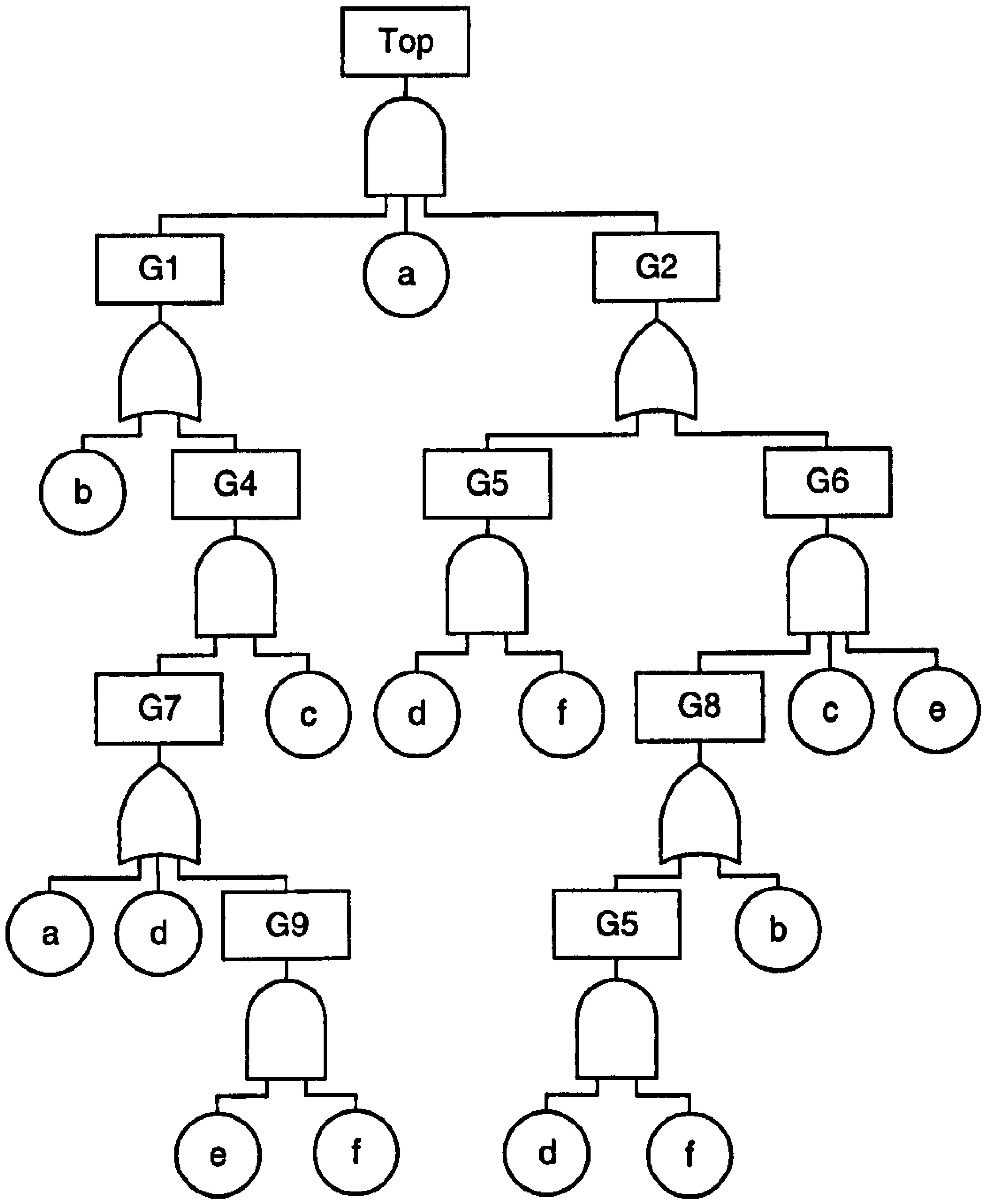


Figure 10.6: The fault tree after stage one of the absorption technique

Gate name	Type	Number of gate inputs	Number of event inputs	Inputs		
Top	AND	2	1	G1	G2	a
G1	OR	1	1	G4	b	
G2	OR	2	0	G5	G6	
G4	AND	1	1	G7	c	
G5	AND	0	2	d	f	
G6	AND	1	2	G8	c	e
G7	OR	1	2	G9	a	d
G8	OR	1	1	G5	b	
G9	AND	0	2	e	f	

Table 10.2: Data for the fault tree in Figure 10.6

Whenever absorption has taken place, the tree must be checked to ensure it still has an alternating sequence of gate types. It is obvious from this example that if event 'b' had been a gate, it would have been an 'OR' gate to maintain the alternating sequence after gate G3. The result would be two 'OR' gates in succession, not the alternating sequence that is required to continue with this method.

It is not possible to maintain the alternating sequence by allowing gates to have only one input (or indeed no inputs if a further absorption was to take place) and scanning the data to remove these gates after all possible absorptions have been applied, as this causes further problems. For example, if there was another occurrence of event 'b' higher up this branch (but obviously lower than the primary gate which had caused the first absorption to take place), which was under an 'AND' gate, then this would now cause the removal of the entire branch from G1 downwards. If however, gate G3 had remained, the primary and secondary gates would both be the same type and the result would be simply to remove 'b' from the inputs to G3. The consequence of this would be that the branch below and including G1 would remain, giving an incorrect fault tree structure. Therefore in order to avoid these problems, gates with only one input are removed and the contraction routine is performed after each stage has taken place.

Absorption Stage Two

Continuing through the branches below the primary gate Top, event 'a' also occurs as an input to gate G7. As this gate is a different type to Top, the branch from G7 downwards is removed. The data is updated by deleting the lines for gates G7 and G9, and G7 is removed from the list of inputs to gate G4. This leaves G4 with only one input, resulting in its subsequent removal and its remaining event 'c' becomes an input to gate G1. The occurrence array is also updated accordingly. Figure 10.7 shows the current fault tree and Table 10.3 shows the updated fault tree data.

Gate name	Type	Number of gate inputs	Number of event inputs	Inputs		
Top	AND	2	1	G1	G2	a
G1	OR	0	2	b	c	
G2	OR	2	0	G5	G6	
G5	AND	0	2	d	f	
G6	AND	1	2	G8	c	e
G8	OR	1	1	G5	b	

Table 10.3: Data for the fault tree in Figure 10.7

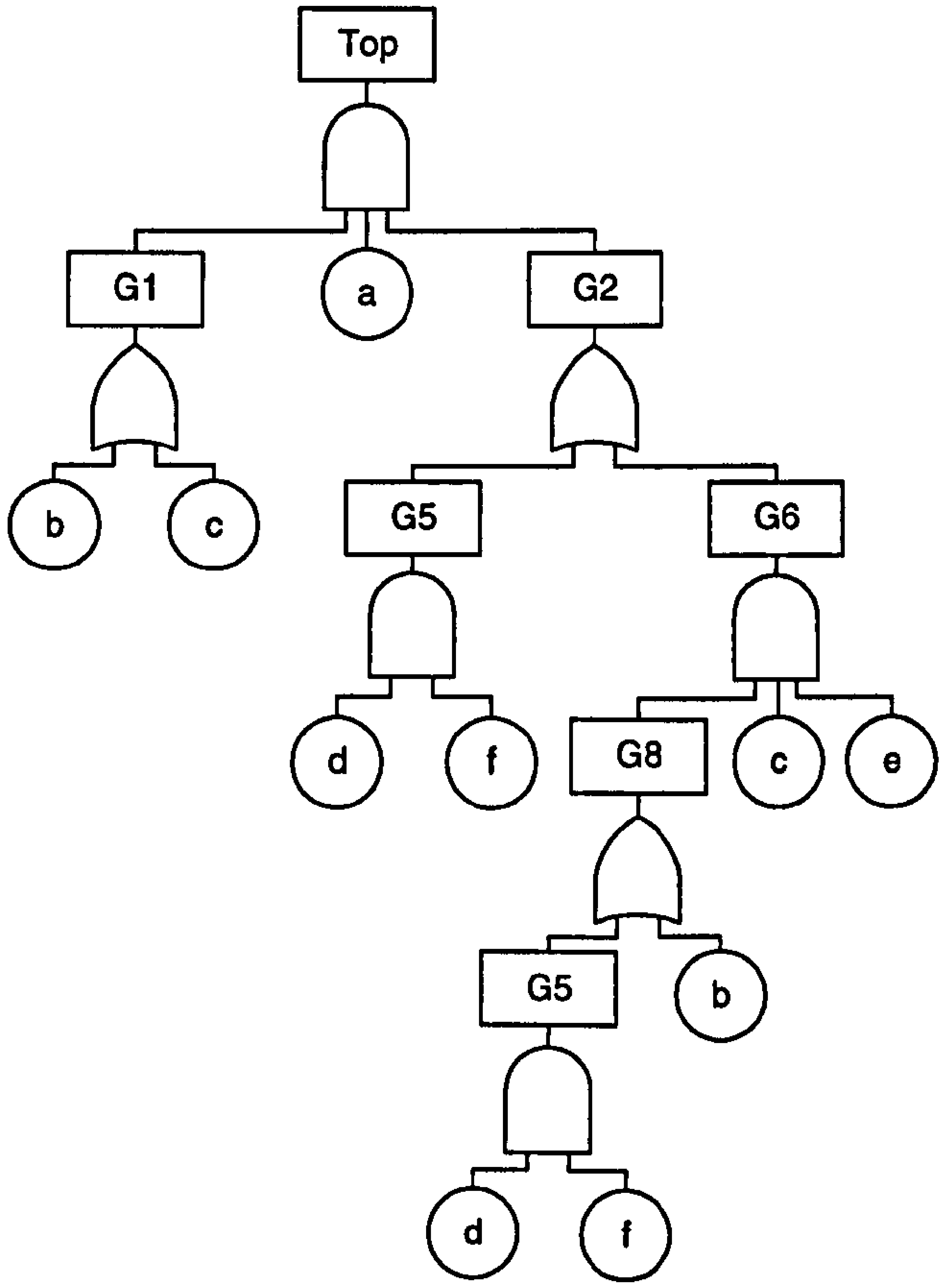


Figure 10.7: The fault tree after stage two of the absorption technique

Absorption Stage Three

Event 'a' now occurs only once in the data, so the depth-first exploration continues, with the aim of identifying inputs to gates that have more than one occurrence in the fault tree data. Gate G1 is considered next, but as it lies at the end of a branch no further analysis can take place. Of the inputs to gate G2, gate G5 is known to occur elsewhere in the fault tree, so the branches beneath G2 are examined. G2 is an 'OR' gate and as G5 also occurs under another 'OR' gate, G8, it is simply deleted as an input to the secondary gate. The line of data for G5 is not deleted as it occurs elsewhere in the tree, but the occurrence array is changed so that it has only one occurrence. G8 is left with the single input 'b', which now becomes an input to G6 and G8 is removed from the data. The updated fault tree and corresponding data is shown in Figure 10.8 and Table 10.4.

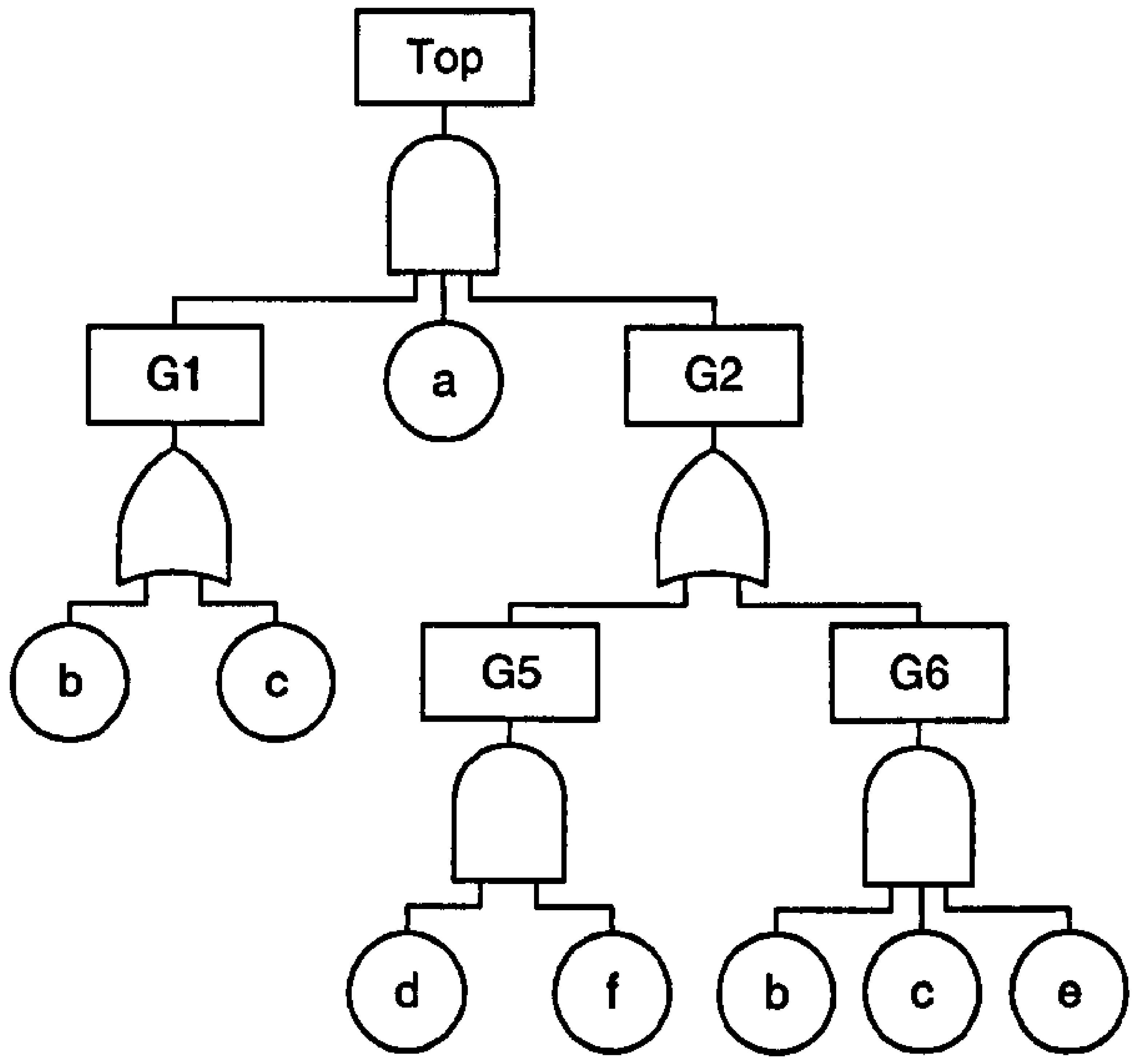


Figure 10.8: The fault tree after stage three of the absorption technique

Gate name	Type	Number of gate inputs	Number of event inputs	Inputs
Top	AND	2	1	G1 G2 a
G1	OR	0	2	b c
G2	OR	2	0	G5 G6
G5	AND	0	2	d f
G6	AND	0	3	b c e

Table 10.4: Data for the fault tree in Figure 10.8

This concludes the application of the absorption technique to this fault tree. Although they look very different, the fault trees in Figures 10.5 and 10.8 have exactly the same underlying logic, which can be shown by considering the minimal cut sets of both trees.

Considering the original fault tree, as shown in Figure 10.5, G1 and G2 can be written as:

$$G1 = a.b + c.(a + d + e.f)$$

$$G2 = d.f + c.e.(b + d.f)$$

Therefore the top event is given by:

$$
\begin{aligned}
Top &= a.G1.G2 \\
&= a.(a.b + c.a + c.d + c.e.f).(d.f + c.e.b + c.e.d.f) \\
&= a.b.d.f + a.c.d.f + a.b.c.e
\end{aligned}
$$

Now considering the modified fault tree shown in Figure 10.8, G1 and G2 are given by:

$$G1 = b + c$$

$$G2 = d.f + b.c.e$$

Top can therefore be written as:

$$\begin{aligned} Top &= a.G1.G2 \\ &= a.(b + c).(d.f + b.c.e) \\ &= a.b.d.f + a.c.d.f + a.b.c.e \end{aligned}$$

The minimal cut sets of the two fault trees are therefore identical.

10.3.2 Dealing with Repeated Gates Within the Fault Tree Structure

This section highlights the way in which the fault tree data is manipulated when gates occur more than once in the fault tree and require altering in different ways. This is an aspect that was not covered in the previous example and a second example fault tree, shown in Figure 10.9, is used to demonstrate the process.

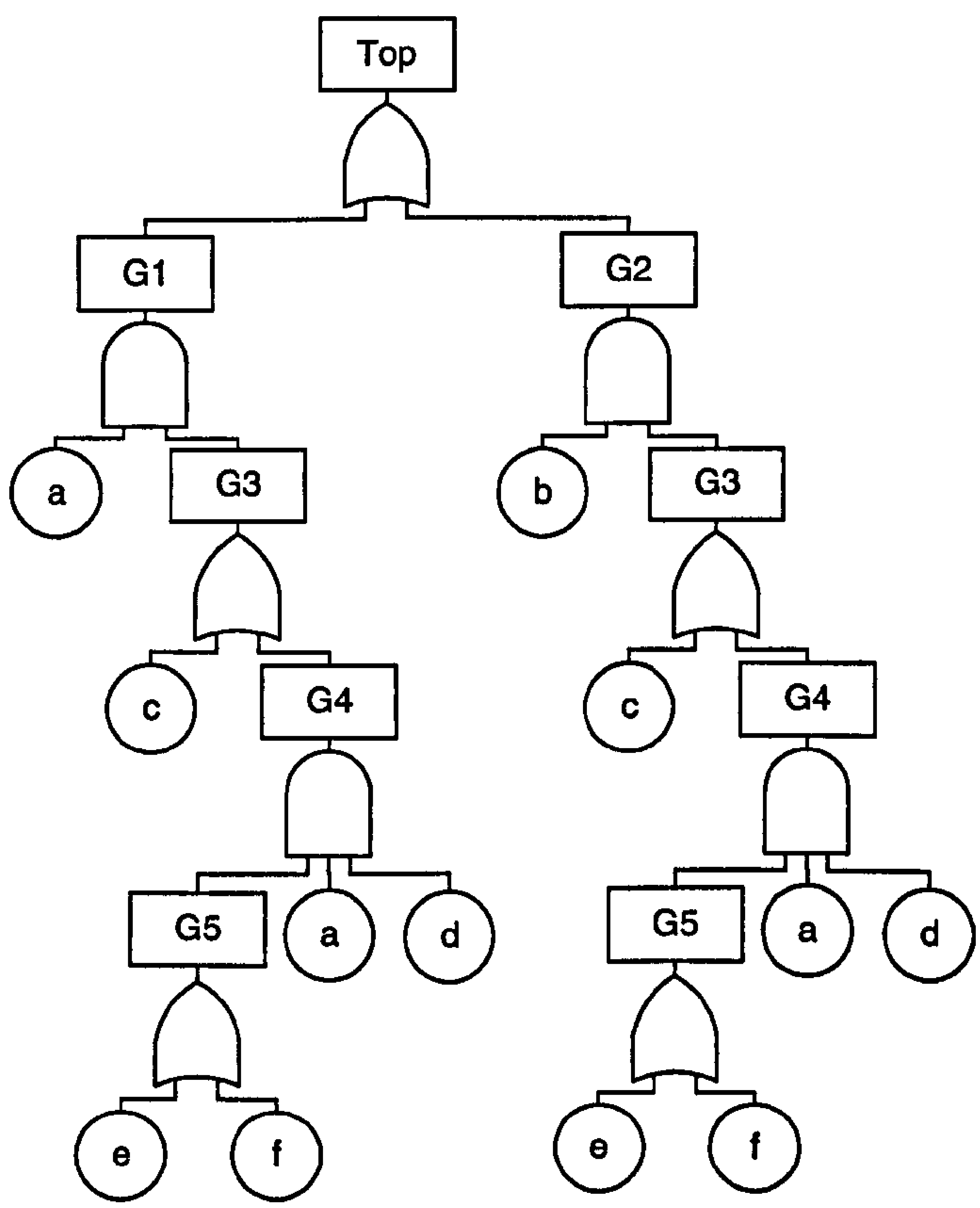


Figure 10.9: The second example fault tree

Again, it is the data that is manipulated within the program and the corresponding data for the tree in Figure 10.9 is shown in Table 10.5.

Gate name	Type	Number of gate inputs	Number of event inputs	Inputs
Top	OR	2	0	G1 G2
G1	AND	1	1	G3 a
G2	AND	1	1	G3 b
G3	OR	1	1	G4 c
G4	OR	1	2	G5 a d
G5	AND	0	2	e f

Table 10.5: Fault tree data for the example tree shown in Figure 10.9

On the left-hand branch of the tree, event 'a' appears as an input to both G1 and G4. In order to simplify, the absorption method would remove the second occurrence under gate G4. However, gate G4 occurs elsewhere in the fault tree and this occurrence cannot be simplified as 'a' does not appear as an input further up the branch. The fault tree data lists each gate just once, so the solution is to duplicate the data for gate G4 under a new gate name and apply the changes to the generated gate. This new gate name will need to be listed as the input to its parent gate, G3. However, as G3 appears twice in the tree and the other occurrence does not require alteration, it must also be duplicated and the modifications made to the new generated gate.

This method can be generalised as follows. A list is made of the gates encountered on the path through the tree from the primary gate to the secondary gate. In this case the path is G1, G3, G4. If the primary gate occurs more than once in the tree data, no further action is required, as the modifications will be valid for each repeated section. However, if any gate after the primary gate is repeated then duplicates are required of each gate from the repeated gate down to the secondary gate. As each gate is duplicated, the one preceding it in the list is altered so that it points to the correct gate input. The absorption method is then applied to the new secondary gate.

Gates G3 and G4 are therefore duplicated and are given the names G6 and G7 respectively. Input G4 to gate G6 now becomes input G7 and the input list for G1 is altered to include G6 instead of G3. The absorption method removes event 'a' from the inputs of gate G7.

The modified fault tree and data are shown in Figure 10.10 and Table 10.6. Although the example is actually very simple, with just a single application of the absorption technique, the method of re-arranging the data is important to avoid incorrect analysis.

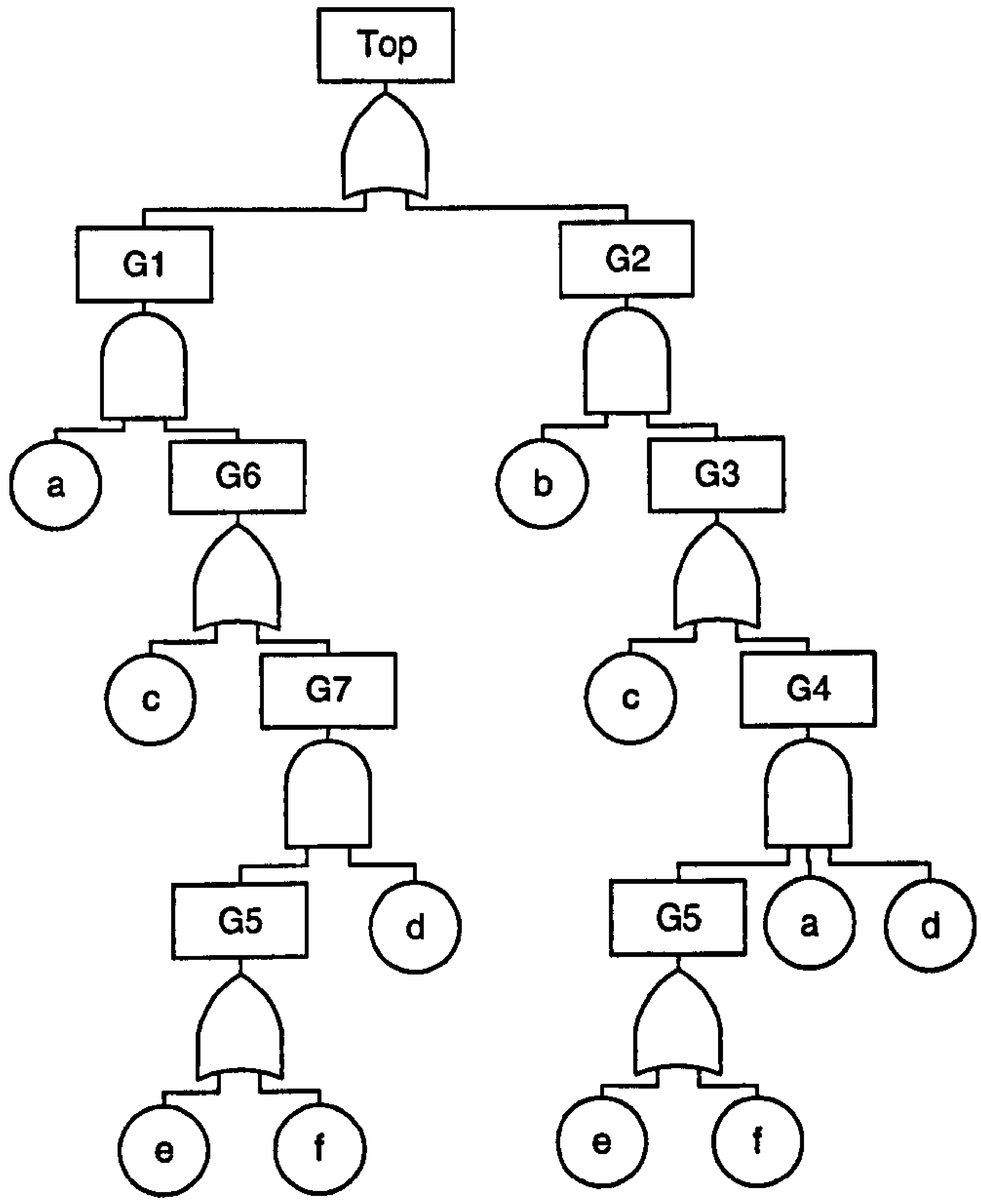


Figure 10.10: The fault tree after application of the absorption technique

Gate name	Type	Number of gate inputs	Number of event inputs	Inputs		
Top	OR	2	0	G1	G2	
G1	AND	1	1	G6	a	
G2	AND	1	1	G3	b	
G3	OR	1	1	G4	c	
G4	OR	1	2	G5	a	d
G5	AND	0	2	e	f	
G6	OR	1	1	G7	c	
G7	OR	1	1	G5	d	

Table 10.6: Fault tree data after application of the absorption technique

The previous examples have described how the absorption method was implemented and the following section considers its integration into the existing reduction technique.

10.4 Integration of the Absorption Stage into the Reduction Technique

The original three steps of the reduction technique are contraction, factorisation and extraction. The fourth stage of absorption was included after the final stage, but contraction was also first re-applied to ensure the required alternating sequence of gate types. As before, the stages are continually applied to the fault tree until no further changes are possible in the system. The new extended reduction technique was applied to the same 228 trees as the original reduction method (summary details for the trees are given in Appendix II) so that a direct comparison of its effectiveness in reducing the size of the resulting BDDs could be made. The results are given in the following section.

10.5 Results of the Application of the Extended Reduction Technique

The absorption stage was shown to contribute significantly to the reduction in fault tree size, with a total of 773 applications over the 228 trees. Table 10.7 shows the results obtained for all four stages.

Stage of the technique	Number of applications on the 228 trees
Contraction	127
Factorisation	3008
Extraction	254
Absorption	773

Table 10.7: Number of applications of each stage of the technique over the set of fault trees

Absorption is obviously a worthwhile addition to the reduction technique, with over three times as many applications as the extraction stage. The absorption method can be applied over any number of levels in the fault tree (the depth-first algorithm ensures this happens in practice) and was shown to occur over up to nine levels. The number of applications over the different levels are shown in Table 10.8.

Levels between absorption	1	2	3	4	5	6	7	8	9
Number of absorptions	210	144	168	100	74	44	17	13	3

Table 10.8: Analysis of the levels over which the absorption technique takes place

As the method is applied in a depth-first manner, more absorptions are likely over fewer levels as they have the ability to remove whole branches of the fault tree below, thus reducing the size of the tree and therefore the number of levels that can be explored.

The BDDs were obtained for both the original and reduced trees using the same eight ordering schemes that were used for analysing the original reduction method. These are described fully in Chapter 5, and are given below:

- Modified top-down.
- Modified depth-first.
- Modified priority depth-first.
- Depth-first, with number of leaves.
- Non-dynamic top-down weights.
- Dynamic top-down weights.
- Bottom-up weights.
- Event criticality.

Three measures of complexity were used to assess the resulting BDDs: the number of non-distinct BDD nodes, the number of distinct BDD nodes and the number of **ite** calculations required to construct the BDDs. The resulting values for the BDDs obtained from the reduced trees can be found in Appendices XI, XII and XIII.

125 of the 228 fault trees resulted in BDDs that had an identical number of non-distinct nodes for all eight ordering schemes. This is significantly more than the number obtained using the original reduction method, where 90 trees produced BDDs with an identical number of nodes using each scheme. Increases in the number of identical results were also seen for the distinct BDD nodes (126 trees compared with 90 using the original reduction technique) and the number of **ite** calculations (114 trees compared with 64 using the original reduction method). This suggests that the choice of ordering scheme becomes less critical when dealing with trees that have been restructured using the extended reduction technique.

The success of the extended reduction technique in reducing BDD complexity was evaluated by comparing the BDDs constructed from the reduced trees against those obtained using the original fault trees. As there are 228 trees with eight ordering schemes used for each, there are a total of 1824 cases to consider. The difference in the number of **ite** calculations and the number of non-distinct and distinct BDD nodes was calculated for each case, together with the percentage decrease. The results are discussed in the following sections.

10.5.1 Non-Distinct Nodes

Out of a total of 1824 cases, 1823 showed a decrease or no change in the number of non-distinct BDD nodes after reduction. 40 of these remained the same size, but this was mainly due to the fact that the BDDs obtained from the original trees were already minimal. Of the

1823 cases that decreased or stayed the same, there was an average decrease of 79.03% in the number of non-distinct nodes. This compares favourably with the average decrease of 46.72% obtained over 96.00% of the cases using the original reduction method.

Only one case showed an increase in the number of non-distinct nodes after reduction. This was for the fault tree 'trials4', though a smaller BDD than had previously been possible was obtained through alternative orderings. The increase for the single case can be attributed to the change in the variable ordering obtained from the reduced tree, which obviously affects the resulting BDD (this is discussed in greater detail in Chapter 6). The smallest number of non-distinct nodes (i.e. the minimum obtained over all eight ordering schemes) therefore either increased for remained the same for all 228 fault trees, with an average decrease recorded of 75.29%. This again compares well with the results obtained for the original reduction technique, where an average decrease of 44.86% was obtained over 224 trees.

10.5.2 Distinct Nodes

In this category, 1809 cases (i.e. 99.18% of the total) showed a decrease or no change in the number of distinct nodes after reduction. The average decrease for these cases was 66.87%, which again compares well with the results obtained using the original reduction method, where an average decrease of 34.29% was obtained over 94.96% of cases.

A total of fifteen cases showed an increase, but this can again be attributed to the change in variable ordering that occurs after manipulation of the fault tree. These cases account for eleven different fault trees, of which reduction had a negative effect on two, as the minimum number of distinct nodes obtained over all the orderings was smaller before reduction than after reduction. However, the increase in the number of nodes was small - for the tree 'lisaba4' the minimum number of distinct nodes increased from 148 to 155; for the tree 'trials4' the minimum number increased from 101 to 104 distinct nodes. Of the remaining 226 trees, the average decrease in the minimum number of distinct nodes of 60.90% compares favourably with the average decrease of 32.47% obtained over 216 trees using the original reduction method.

10.5.3 Number of If-Then-Else Calculations

The number of **ite** calculations required to obtain the BDD is the measure that it is most advantageous to reduce. This is because the usual reason for being unable to obtain a BDD is the large number of calculations involved and the lack of computational resources for performing them.

Only in three cases did the number of **ite** calculations increase after reduction had taken place. Again this is attributed to the change in variable ordering that occurs after manipulation of the tree. The three cases involved two trees (lisab57 and nakashi), but for both a smaller number of **ite** calculations than was previously possible was obtained after reduction using alternative orderings. A total of 99.84% of cases either showed a decrease or no change in the number of **ite** calculations after reduction and the average decrease over these was 84.62%. Using the previous method of reduction, only a 40.87% average decrease was recorded, over 86.62% of cases.

The minimum attainable values of the number of **ite** calculations were also compared for each of the original and reduced trees. An average decrease of 74.16% was recorded over 228 trees; using the original reduction method, an average decrease of 40.39% over 201 trees was obtained.

None of the fault trees showed an increase in all three measures of BDD complexity and only two trees (benjiam and worrell) showed no improvement in any of the measures. This means that the extended reduction technique had a positive effect on 226 trees, as they each resulted in BDDs with at least one improved measure of complexity.

10.6 Conclusions

The application of the extended reduction technique to fault trees has been shown to significantly reduce the complexity of the resulting BDDs. The method has been analysed using three measures of BDD complexity (number of non-distinct nodes, number of distinct nodes and number of **ite** calculations) and has performed exceptionally well under each. It is also a substantial improvement on the original reduction method, which was itself deemed to have performed extremely well when first analysed. The additional stage of absorption obviously has additional benefit and the extended reduction method would be recommended for application to any fault tree before conversion to a BDD.

Chapter 11: Conclusions and Future Work

11.1 Summary

Fault Tree Analysis is used extensively for system reliability assessment, providing a clear visual representation of the causes of system failure. However, the conventional techniques for the quantitative analysis of fault trees can be computationally intensive and require the use of approximations, which inevitably leads to a loss of accuracy. The BDD technique has emerged as an alternative approach for performing the required analysis. The method is efficient and produces exact results, without the need for approximations. However, the structure of the BDD is very sensitive to the variable ordering used for its construction. A bad choice of ordering can result in a time-consuming construction process and a large BDD, which in turn can lead to increased analysis times.

The aim of this research was to develop techniques for the efficient construction of BDDs from fault trees. This was approached in two ways. One method was to explore the variable ordering issue and the problem of finding an ordering scheme that produces the smallest BDD for any fault tree structure. The second approach considered techniques for reducing the complexity of fault trees, with the aim of constructing smaller BDDs and making the choice of variable ordering scheme less critical.

The survey of ordering schemes conducted in Chapter 4 highlighted techniques that had not been fully explored and were considered worthy of further investigation. Eight schemes were chosen for a comparative study, which included four structural ordering schemes and four weighted methods. BDDs were constructed for 228 test trees, using the variable orderings determined by each of the schemes. In order to compare the performance of the schemes, three different measures of BDD complexity were considered: the number of non-distinct BDD nodes, the number of distinct BDD nodes and the number of **ite** calculations required to construct the BDD. The results showed that none of the schemes consistently outperformed the others, but that each scheme is relevant, as it generated a BDD complexity that could not be matched by any other scheme for at least one fault tree (and in many cases, several trees). It was also shown that even within a particular fault tree, different schemes work best depending on the measure used to assess the BDD complexity.

The structure of a fault tree can vary considerably whilst still satisfying the same logic function, and is rarely written in its most concise form. This can have a significant effect on the complexity of the resulting BDD. The Faunet reduction technique was considered as a method for optimising fault trees, before implementing the BDD construction process. A set of 228 test trees were restructured using this technique and its success was evaluated by comparing the complexity of the BDDs obtained from the reduced fault trees against those generated using the original trees. The BDDs were constructed using variable orderings obtained from the eight ordering schemes developed during the comparative study. Again, three measures of BDD complexity were considered: the number of non-distinct BDD nodes, the number of distinct BDD nodes and the number of **ite** calculations required to construct the BDD.

The reduction technique was shown to perform well according to each measure of BDD complexity, with average decreases of 46.72% over 96.00% of the 1824 cases for the number of non-distinct nodes, 34.29% over 94.96% of cases for the number of distinct nodes and 40.87% over 86.62% of cases for the number of **ite** calculations. The smallest attainable values of BDD complexity (i.e. the minimum obtained over all eight ordering schemes) were also compared for each of the original and reduced trees. Average decreases were recorded of 44.86% over 224 trees for the number of non-distinct nodes, 32.47% over 216 trees for the number of distinct nodes and 40.39% over 201 trees for the number of **ite** calculations. Only one tree recorded an increase in each measure of BDD complexity. Nine other trees showed no improvement in any of the measures, but reduction had a positive effect on the remaining 218 trees, which each produced BDDs with at least one improved complexity measure. The performance of the eight ordering schemes on the reduced trees was also assessed according to these measures and the results obtained suggested that the choice of ordering scheme becomes less critical when dealing with reduced trees. The Faunet reduction technique was therefore concluded to be an effective pre-processing tool for fault trees.

A fault tree analysis strategy was developed, which aims to increase the likelihood of obtaining a BDD for any given fault tree, by ensuring that the associated calculations are as efficient as possible. The method implements Faunet reduction, together with a second method of fault tree simplification, linear-time modularisation. This results in a set of concisely written subtrees, which are each converted to a BDD structure. The set of BDDs, which can encode both complex and modular events, fully represents the original fault tree. The appropriate quantitative analysis for the BDDs was developed, enabling the calculation of system parameters such as the unavailability and unconditional failure intensity. In addition, the methods for extracting the criticality functions of the basic events were demonstrated, which allow the system to be analysed in terms of its original components.

The analysis strategy was applied to a set of 228 fault trees, and the calculation times compared with those obtained for the construction and subsequent quantification of the BDDs directly from the trees. The results showed substantial savings in analysis time when dealing with large fault trees, but slight increases in analysis time when considering small trees. The increases were due to the number of comparisons necessary for the Faunet reduction technique. The strategy does therefore have the potential to substantially reduce the analysis times of large fault trees and increase the likelihood of obtaining a BDD for any given tree. A significant advantage is the possibility of analysing fault tree modules separately. This is likely to be of particular use where the tree is too large to be dealt with as a whole but can be analysed in pieces and the quantitative analysis applied afterwards to the set of BDDs.

Neural networks were considered as a method of selecting an appropriate variable ordering scheme based on the fault tree characteristics. The aim of this research was to develop a network model that could be used within the fault tree analysis strategy for selecting the best ordering scheme for each module. If the optimal scheme could be chosen on each occasion, it would lead to smaller BDDs and further reduce the analysis times. Two neural network models were considered: the multi-layer perceptron and the radial basis function. Numerous trials were conducted with both models using the reduced fault trees. The best result when choosing from eight ordering schemes was 7/20 correct predictions. When the number of ordering schemes was reduced to four, the best result was 10/20 correct predictions, but this was simply due to fewer options being available for selection. These results and the number of trials conducted show conclusively that the neural network models used were not capable of modelling the variable ordering problem and it was concluded that they were not satisfactory for selecting the ordering schemes to be used within the fault tree analysis strategy. Further research therefore focussed on extending the methods of fault tree simplification.

Structures were identified within the reduced fault trees (i.e. those that had been restructured using the Faunet reduction technique) that could be further simplified through the application of the absorption and idempotent laws to the fault tree logic. An additional stage was developed for the reduction technique that manipulates the fault tree structure to incorporate these laws. This extended reduction technique was applied to a set of 228 test trees. BDDs were obtained for both the original and reduced trees using variable orderings determined by eight different ordering schemes. The performance of the technique was evaluated by comparing the complexity of the BDDs obtained from the reduced trees against those obtained using the original fault trees. Three measures of BDD complexity were considered: the number of non-distinct BDD nodes, the number of distinct BDD nodes and the number of **ite** calculations required to construct the BDD.

Average decreases were calculated of 79.03% over 99.95% of the 1824 cases for the number of non-distinct nodes, 66.87% over 99.18% of cases for the number of distinct nodes and 84.62% over 99.84% of cases for the number of **ite** calculations. The smallest attainable values of BDD complexity were also compared for each of the original and reduced trees. Average decreases were recorded of 75.29% over 228 trees for the number of non-distinct nodes, 60.90% over 226 trees for the number of distinct nodes and 74.16% over 228 trees for the number of **ite** calculations. Only two trees showed no improvement in any of the measures, meaning that the extended reduction technique had a positive effect on 226 trees, as they each resulted in BDDs with at least one improved measure of complexity. The number of trees for which all eight ordering schemes produced identical results was significantly increased after the extended reduction technique had been applied (compared with both the original trees and those restructured using Faunet reduction), which demonstrates that the choice of ordering scheme becomes less critical when considering the reduced trees. This method was therefore shown to be beneficial in the BDD construction process, and is also a substantial improvement on the original reduction technique.

11.2 Conclusions

- The performance of any ordering scheme is dependent on the fault trees to which it is applied and varies according to the measure used to assess the complexity of the resulting BDDs. Even within a particular tree, different schemes work best depending on the measure used to evaluate BDD complexity.
- The fault tree analysis strategy resulted in substantial savings in analysis time for a particularly large fault tree and increases the likelihood of obtaining a BDD for any given tree. A significant advantage of the strategy is the ability to analyse a fault tree in several stages, if it is too large to be considered as a whole.
- The models considered for the neural network technique did not accurately represent the variable ordering problem and were therefore not satisfactory for inclusion within the fault tree analysis strategy. Further research is required before the neural network method can be used as a technique for selecting an appropriate ordering scheme for a fault tree.
- The extended reduction method is an effective pre-processing tool for fault trees, significantly reducing the size of resulting BDDs and the number of calculations required for their construction. The choice of variable ordering scheme also becomes less critical if reduction has been applied to the fault tree.

11.3 Future Work

11.3.1 Combine Structural and Weighted Ordering Techniques

Both structural and weighted schemes have been shown to be valuable in the construction of BDDs. An ordering scheme that combines these techniques, so that variables retain their neighbourhoods, but are also ordered according to their weighting within the tree could be beneficial for BDD construction.

11.3.2 Incorporate Extended Reduction into the Fault Tree Analysis Strategy

The fault tree analysis strategy was developed using the Faunet reduction technique and produced promising results. The strategy could be modified by incorporating the extended reduction method, which has been shown to result in significantly smaller BDDs than were obtained using the original reduction technique. This could result in improved analysis times. It would also be interesting to see the result of applying the strategy to trees that can not be analysed using other methods.

11.3.3 Develop Further Quantification Methods

The quantification methods developed for BDDs encoding complex and modular events enable the calculation of the system unavailability, system unconditional failure intensity and the event criticality functions. The methods could be extended to include the calculation of other performance indicators such as the system unreliability and basic event importance measures.

11.3.4 Extend the Neural Network Approach

There are many aspects of the multi-layer perceptron network that could be changed to try to fit the model to the ordering problem more successfully. For example, different activation functions could be applied, alternative optimisation algorithms could be implemented, or even pattern training could be used instead of batch training. Several features of the radial basis function model could also be altered, including the type of basis function, the choice of basis function centres and the way in which the width parameters are chosen. However, the choice of fault tree characteristics is thought to have the biggest influence on the success of the network, and these need to be reviewed in detail before more sophisticated network models are considered.

One approach for determining the important fault tree characteristics is to use an unsupervised training technique, which can identify the classes that the network itself regards as distinct. Discussion in reference 37 suggests that models capable of unsupervised training can be especially valuable in exploratory work. As the most significant fault tree features have not yet been found, the network itself could help in detecting them.

11.3.5 Analyse the Fault Tree Test Data

The results obtained throughout the thesis are dependent upon the fault trees used to test the methods. The fault tree test set consisted of a combination of trees obtained from industry and trees generated randomly. However this test data is not exhaustive, that is, it is unlikely to cover the full range of fault tree structures that can exist. Although a larger sample will lead to increased confidence in the results, including more fault trees in the data set may not provide a more rigorous assessment of the techniques, as the underlying features of the additional trees may be equivalent to those found in the existing set. Further work could therefore be undertaken to examine the structures of the test trees, in order to determine whether they could be classified according to particular characteristics. This could give an indication of whether the techniques examined within the thesis are more suited to one type of fault tree structure than another.

11.3.6 Optimise Non-Coherent Fault Trees

The work contained within this thesis has focussed on coherent fault tree structures, but the methods could be extended to consider non-coherent fault trees. Within such structures, both working and failed components can contribute to system failure and the techniques of reduction (both Faunet reduction and extended reduction) and modularisation could be modified to deal with these, so that smaller non-coherent BDDs can be constructed.

References

1. Andrews, J. D. and Moss, T. R. 'Reliability and Risk Assessment', Longman, 1993.
2. Watson, H. A. and Bell Telephone Laboratories 'Launch Control Safety Study', Bell Telephone Laboratories, Murray Hill, NJ USA, 1961.
3. Vesely, W. E. 'A Time Dependent Methodology for Fault Tree Evaluation', Nuclear Design and Engineering, **13**, pp337-360, 1970.
4. Rauzy, A. 'New Algorithms for Fault Tree Analysis', Reliability Engineering and System Safety, **40**, pp203-211, 1993.
5. Schneeweiss, W. G. 'Boolean Functions with Engineering Applications and Computer Programs', Springer-Verlag, 1989.
6. Birnbaum, Z. W. 'On the Importance of Different Components in a Multi-Component System', Multivariate Analyses II, P. R. Krishnaiah (Ed), Academic Press, 1969.
7. Fussell, J. B. 'How to Hand-Calculate System Reliability and Safety Characteristics', IEEE Trans. Reliability, **R-24**, No. 3, pp169-174, 1975.
8. Barlow, R. E. and Proschan, F. 'Importance of System Components and Fault Tree Events', Stochastic Processes and their Applications, **3**, pp153-173, 1975.
9. Beeson, S. and Andrews, J. D. 'Importance Measures for Non-Coherent Systems' Analysis', Accepted for Publication by IEEE Trans. Reliability, 2002.
10. Beeson, S. 'Non-Coherent Fault Tree Analysis', Doctoral Thesis, Loughborough University, 2002.
11. Dutuit, Y. and Rauzy, A. 'A Linear-Time Algorithm to find Modules of Fault Trees', IEEE Trans. Reliability, **45**, No. 3, pp422-425, 1996.
12. Lee, C. 'Representation of Switching Circuits by Binary Decision Diagrams', Bell Syst. Tech. Journal, No. 38, pp985-999, 1959.
13. Friedman, S. J. and Supowit, K. J. 'Finding the Optimal Variable Ordering for Binary Decision Diagrams', IEEE Trans. Computers, **39**, No. 5, pp710-713, 1990.
14. Bryant, R. E. 'Graph-Based Algorithms for Boolean Function Manipulation', IEEE Trans. Computers, **C-35**, No. 8, pp677-691, 1986.
15. Nikolskaia, M. 'Binary Decision Diagrams and Applications to Reliability Analysis', Doctoral Thesis, University of Bordeaux, 1999.
16. Fujita, M., Matsunaga, Y. and Kakuda, N. 'On the Variable Ordering of Binary Decision Diagrams for the Application of Multilevel Logic Synthesis', Proc. European Design Automation Conference, EDAC'91, pp50-54, 1991.

17. Ishiura, N., Sawada, H. and Yajima, S. 'Minimization of Binary Decision Diagrams Based on Exchange of Variables', Proc. IEEE International Conference on Computer Aided Minimization of Binary Decision Diagrams, pp472-475, Nov. 1991.

18. Bouissou, M., Bruyère, F. and Rauzy, A. 'Binary Decision Diagram Based Fault Tree Processing: A comparison of Variable Ordering Heuristics', Proc. ESREL'97, pp2045-2052, 1997.

19. Bartlett, L. M. 'Variable Ordering Heuristics for Binary Decision Diagrams', Doctoral Thesis, Loughborough University, 2000.

20. Sinnamon, R. M. and Andrews, J. D. 'Improved Efficiency in Qualitative Fault Tree Analysis', Proc. Advances in Reliability Technology Symposium, ARTS'96, Manchester, April 1996.

21. Sinnamon, R. M. 'Binary Decision Diagrams for Fault Tree Analysis', Doctoral Thesis, Loughborough University, 1996.

22. Sinnamon, R. M. and Andrews, J. D. 'New Approaches to Evaluating Fault Trees', Proc. ESREL'95, pp241-254, June 1995.

23. Bouissou, M. 'An Ordering Heuristic for Building Binary Decision Diagrams from Fault Trees', Proc. Reliability and Maintainability Symposium, ARMS'96, pp208-214, Jan. 1996.

24. Minato, S., Ishiura, N., Yajima, S. 'Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation', Proc. 27th ACM/IEEE Design Automation Conference, DAC'90, pp52-57, 1990.

25. Bartlett, L. M. and Andrews J. D. 'Efficient Basic Event Ordering Schemes for Fault Tree Analysis', Quality and Reliability Engineering International, **15**, pp95-101, 1999.

26. Platz, O. and Olsen J. V. 'FAUNET: A Program Package for Evaluation of Fault Trees and Networks', Research Establishment Risφ Report, No. 348, DK-4000 Roskilde, Denmark, Sept. 1976.

27. Group Aralia, 'Computation of Prime Implicants of a Fault Tree Within Aralia', Proc. Of ESREL'95, pp190-202, 1995.

28. Coudert, O. and Madre, J. C. 'METAPRIME - An Interactive Fault-Tree Analyzer', IEEE Trans. Reliability, **43**, No. 1, pp121-127, 1994.

29. Bartlett, L. M. and Andrews J. D. 'Efficient Basic Event Orderings for Binary Decision Diagrams', Proc. Reliability and Maintainability Symposium, ARMS'98, pp61-68, 1998.

30. Bartlett, L. M and Andrews, J. D. 'Selecting an Ordering Heuristic for the Fault Tree Binary Decision Diagram Conversion Process using Neural Networks', IEEE Trans. Reliability, **51**, No. 3, pp344-349, 2002.

31. Fussell, J. B. and Vesely W. E. 'A New Methodology for Obtaining Cut Sets for Fault Trees', Trans. Am. Nucl. Soc., **15**, pp262-263, 1972.

32. Sinnamon, R. M. and Andrews, J. D. 'Quantitative Fault Tree Analysis using Binary Decision Diagrams', European Journal of Automation, **30**, No. 3, pp1051-1071, 1996.

33. Reay, K. A. and Andrews, J. D. 'Modularised Binary Decision Diagrams for Fault Tree Analysis', Proc. 3rd Edinburgh Conference on Risk: Analysis, Assessment and Management, April 2002.

34. Reay, K. A. and Andrews, J. D. 'A Fault Tree Analysis Strategy Using Binary Decision Diagrams', Reliability Engineering and System Safety, **78**, pp45-56, 2002.

35. Bishop, C. M. 'Neural Networks for Pattern Recognition', Clarendon Press, Oxford, 1995.

36. Masters, T. 'Signal and Image Processing with Neural Networks: A C++ Sourcebook', Wiley, 1994.

37. Masters, T. 'Practical Neural Network Recipes in C++', Academic Press Inc. 1993.

Appendix I

Implementation of the Linear-Time Algorithm

The linear-time algorithm, which determines the modules of a fault tree, can be described in four steps and uses the following variables:

- visit1: step number of the first visit to a gate or event.
- visit2: step number of the second visit to a gate or event.
- last-visit : step number of the final visit to a gate or event.
- min: collected minimum of the variable visit1 for the descendants of a gate.
- max: collected maximum of the variable last-visit for the descendants of a gate.

The steps of the algorithm are as follows:

1. Set all the counters (as above) to zero.
2. Perform a depth-first traversal of the fault tree (detailed algorithm shown in Figure I.1), setting variables visit1, visit2 and last-visit for each gate and event.

   Note: For basic events, visit1 and visit2 are identical. Also, the subtree under any gate is never traversed more that once - if visit1 has already been set for a gate, then last-visit is simply updated and the traversal continues with the next gate.
3. Perform the second depth-first traversal (detailed algorithm shown in Figure I.2), finding for each gate, the maximum of the last visits and the minimum of the first visits of all the gates and events beneath it.
4. A gate heads a module iff:
   - max is less that the value of visit2 for that gate

   and
   - min is greater than the value of visit1 for that gate.

The program 'module.c' that implements the algorithm was written in the C programming language. It reads the fault tree data from a datafile of the form *.dat, performs the analysis and outputs a list of the gates and whether or not they head modules into a file of the form *.idm (identify modules).

```
df_setup (node, step)
{
    if (node is a gate)
    {
        step = step + 1

        if (node has already been visited)
        {
            set last-visit [node] = step
        }

        else (not been visited)
        {
            set visit1[node]  = step

            for (all inputs to gate)
            {
                call df_setup (input, step)
            }

            step = step + 1

            set:  visit2 [node]  = step
                  last-visit [node]  = step
        }
    }

    else (node is a basic event)
    {
        if (node has already been visited)
        {
            step = step + 1

            set last-visit [node]  = step
        }

        else (not been visited)
        {
            step = step + 1

            set:  visit1[node]  = step
                  visit2 [node]  = step
                  last-visit [node]  = step
        }
    }
}
```

Figure I.1: Algorithm to set the variables visit1, visit2 and last-visit for each gate and event

```
df_max_min (node)          [node is always a gate in this case]
{
    for (each input to the gate)
    {
        if (the input is a gate)
        {
            if (the max/min hasn't been found for this gate input)
            {
                call df_max_min (input)
            }
            if (no initial values assigned to max [node] and min [node])
            {
                max [node] = max [input]
                min [node] = min [input]
                if (last-visit [input] > max [node])
                {
                    max [node] = last-visit [input]
                }
                if (visit1[input] < min[node])
                {
                    min [node] = visit1[input]
                }
            }
            else
            {
                if (max [input] > max [node])
                {
                    max [node] = max [input]
                }
                if (last-visit [input] > max [node])
                {
                    max [node] = last-visit [input]
                }
                if (min [input] <  min [node])
                {
                    min [node] = min [input]
                }
                if (visit1[input] < min [node])
                {
                    min [node] = visit1[input]
                }
            }
        }
        else (input is an event)
        {
            if (no initial values assigned to max[node] and min[node])
            {
                max [node] = last-visit [input]
                min [node] = visit1[input]
            }
            else
            {
                if (last-visit [input] > max [node])
                {
                    max [node] = last-visit [input]
                }
                if (visit1[input] < min [node])
                {
                    min [node] = visit1 [input]
                }
            }
        }
    }
}
```

Figure I.2: algorithm to set the variables max and min for each gate.

Appendix II

Fault Tree Summary Details

Fault tree	Minimal cut sets	Top gate type	No. of levels[1]	No. of different events	Total no. of events in tree	No. of different gates	Used in chapters 5, 6, 8, 10	Used in chapter 9
aaaaaaa	2	AND	3	3	4	3	✓	
artqual	7	AND	5	7	11	5	✓	✓
arttree	2	OR	3	4	5	3	✓	
astolfo	27	OR	8	16	22	19	✓	✓
bddtest	9	OR	5	13	15	9	✓	✓
benjiam	43	AND	5	11	22	15	✓	✓
bpfeg03	8716	OR	6	63	63	20	✓	
bpfen05	7471	OR	6	61	61	17	✓	
bpfig05	7056	OR	6	60	60	17	✓	
bpfin05	416	OR	6	40	40	14	✓	
bpfpp02	3	OR	3	4	5	3	✓	
bpfsw02	84424	AND	7	40	44	21	✓	✓
ch8tree	5	AND	4	7	12	5	✓	✓
dre1019	63	OR	4	19	20	4	✓	
dre1032	75	OR	4	21	22	4	✓	
dre1057	2100	AND	5	32	33	7	✓	
dre1058	11934	AND	5	41	64	13	✓	✓
dre1059	36990	AND	7	57	80	17	✓	✓
dresden	11934	AND	7	57	144	17	✓	✓
emerh2o	13	OR	4	10	11	4	✓	
fatram2	6	AND	5	8	10	5	✓	✓
hpisf02	255	OR	6	72	80	19	✓	✓
hpisf03	71	OR	4	31	33	7	✓	✓
hpisf21	7777	OR	6	61	208	15	✓	✓
hpisf36	61	OR	4	30	34	8	✓	✓
jdtree1	4	AND	4	7	7	5	✓	
jdtree2	4	AND	4	7	7	5	✓	
jdtree3	36	AND	7	21	21	11	✓	
jdtree4	30	AND	7	20	21	11	✓	
jdtree5	10	OR	7	20	21	11	✓	
khictre	21	AND	5	22	74	19	✓	✓
lisa123	37	OR	7	27	39	15	✓	✓
lisab10	940	AND	7	48	80	27	✓	✓

[1] Number of levels counts the top event as being on level 1.

Fault tree	Minimal cut sets	Top gate type	No. of levels	Different events	Total events	Different gates	Chapters 5, 6, 8, 10	Chapter 9
lisab25	35	OR	6	26	37	15	✓	✓
lisab28	66	OR	6	22	22	9	✓	
lisab30	17	OR	7	32	45	19	✓	✓
lisab31	164	AND	6	47	94	31	✓	✓
lisab34	14	AND	4	14	23	8	✓	✓
lisab35	136	AND	5	40	57	19	✓	✓
lisab36	52	OR	6	39	130	46	✓	✓
lisab42	10	OR	5	21	23	7	✓	
lisab44	12	OR	4	20	33	10	✓	✓
lisab51	11	OR	5	19	21	8	✓	✓
lisab52	139	AND	6	38	94	31	✓	✓
lisab53	15	OR	4	9	10	5	✓	
lisab54	14	OR	4	15	19	6	✓	✓
lisab57	170	AND	5	28	46	18	✓	✓
lisab59	3096	AND	5	49	49	16	✓	
lisab60	19	AND	4	16	23	7	✓	✓
lisab78	503	AND	5	39	49	16	✓	✓
lisab86	383	AND	7	40	50	21	✓	✓
lisaba4	827	OR	7	44	63	26	✓	✓
lisaba9	85	OR	6	41	46	17	✓	✓
modtree	2	AND	4	5	7	4	✓	
nakashi	20	AND	7	16	29	21	✓	✓
newtre2	3	OR	4	7	9	5	✓	
newtre3	2	OR	4	5	6	4	✓	
newtree	3	OR	4	6	7	4	✓	
rand100	8	OR	7	27	53	19	✓	✓
rand101	2	AND	4	7	8	3	✓	
rand102	1	AND	4	7	9	3	✓	
rand103	13	OR	6	23	29	13	✓	✓
rand104	9	OR	7	22	41	16	✓	✓
rand105	96	OR	6	33	37	15	✓	✓
rand106	8	AND	7	37	76	31	✓	✓
rand107	5	OR	3	8	9	2	✓	
rand108	35	AND	7	35	76	32	✓	✓
rand109	203	OR	7	56	68	27	✓	✓
rand110	8	OR	8	30	61	24	✓	✓
rand111	22	OR	6	22	47	19	✓	✓
rand112	1	AND	3	5	7	2	✓	
rand113	1	AND	7	11	28	12	✓	✓
rand114	2	AND	5	9	12	4	✓	✓

Fault tree	Minimal cut sets	Top gate type	No. of levels	Different events	Total events	Different gates	Chapters 5, 6, 8, 10	Chapter 9
rand115	46	OR	6	29	46	21	✓	✓
rand116	15	AND	6	33	68	24	✓	✓
rand117	11	OR	5	17	23	10	✓	✓
rand118	52	OR	6	39	47	19	✓	✓
rand119	84	AND	6	30	37	14	✓	✓
rand120	58	AND	6	39	47	20	✓	✓
rand121	80	AND	7	37	50	18	✓	✓
rand122	4	OR	3	5	6	2	✓	
rand123	12	AND	6	17	23	9	✓	✓
rand124	27	OR	7	24	30	12	✓	✓
rand125	13	OR	5	14	19	6	✓	✓
rand126	59	OR	6	37	53	25	✓	✓
rand127	43	AND	7	28	31	12	✓	✓
rand128	52	AND	6	35	68	24	✓	✓
rand129	1	AND	6	20	26	8	✓	✓
rand130	5	OR	7	23	40	13	✓	✓
rand131	2	AND	4	7	10	4	✓	✓
rand132	67	AND	7	39	84	31	✓	✓
rand133	4	AND	3	7	9	3	✓	
rand134	60	OR	7	56	98	34	✓	✓
rand135	24	AND	7	33	64	24	✓	✓
rand136	1	AND	3	4	6	2	✓	
rand137	15	AND	7	21	26	10	✓	✓
rand138	2	OR	7	18	31	10	✓	✓
rand139	53	AND	7	29	50	21	✓	✓
rand140	5	OR	4	9	11	3	✓	
rand141	8	OR	8	30	61	24	✓	✓
rand142	410	AND	7	46	97	32	✓	✓
rand143	8	OR	7	28	40	17	✓	✓
rand144	41	OR	6	48	85	29	✓	✓
rand145	47	OR	6	33	34	11	✓	✓
rand146	15	AND	7	21	26	10	✓	✓
rand147	30	AND	7	43	90	36	✓	✓
rand148	8	OR	6	27	31	12	✓	✓
rand149	18	OR	6	57	64	22	✓	✓
rand150	114	OR	6	44	74	29	✓	✓
rand151	36	OR	6	28	32	10	✓	✓
rand152	1	AND	3	2	3	2	✓	
rand153	3	AND	7	21	47	16	✓	✓
rand154	1	OR	7	21	30	11	✓	✓

Fault tree	Minimal cut sets	Top gate type	No. of levels	Different events	Total events	Different gates	Chapters 5, 6, 8, 10	Chapter 9
rand155	52	AND	6	33	47	20	✓	✓
rand156	20	AND	6	22	28	10	✓	✓
rand158	9	AND	7	71	123	49	✓	✓
rando10	4	OR	3	6	8	2	✓	
rando11	6391	AND	6	94	143	48	✓	✓
rando12	68	AND	6	68	98	32	✓	✓
rando13	73	OR	6	56	140	46	✓	✓
rando14	1	AND	4	7	9	3	✓	
rando15	5	OR	4	5	18	5	✓	✓
rando16	76	OR	8	46	84	31	✓	✓
rando17	1	AND	3	6	7	2	✓	
rando18	24	OR	7	85	178	62	✓	✓
rando19	764	AND	6	53	133	51	✓	✓
rando20	122	OR	8	47	143	52	✓	✓
rando21	5	OR	5	11	11	5	✓	
rando22	423	AND	7	64	128	46	✓	✓
rando23	9	OR	7	39	56	19	✓	✓
rando24	4	OR	3	7	8	2	✓	
rando25	6	OR	5	16	33	14	✓	✓
rando26	3	OR	5	8	15	6	✓	✓
rando27	100	AND	8	46	115	45	✓	✓
rando28	1	OR	9	35	50	17	✓	✓
rando29	22	OR	7	38	67	25	✓	✓
rando30	195	AND	6	41	45	17	✓	✓
rando31	5	OR	9	36	120	47	✓	✓
rando32	5	OR	4	6	15	4	✓	
rando33	11	AND	5	32	63	17	✓	✓
rando34	35	OR	8	36	61	24	✓	✓
rando35	8	AND	6	24	51	19	✓	✓
rando36	10	OR	6	29	37	15	✓	✓
rando37	29	AND	6	30	74	27	✓	✓
rando38	9	OR	6	21	26	11	✓	✓
rando39	51	AND	7	26	66	27	✓	✓
rando40	9	OR	5	17	22	8	✓	✓
rando41	1	AND	5	8	12	4	✓	
rando42	2	AND	5	17	24	9	✓	✓
rando43	22	OR	5	27	31	10	✓	✓
rando44	436	OR	7	59	68	27	✓	✓
rando45	16	AND	6	28	60	22	✓	✓
rando46	10	OR	6	41	69	22	✓	✓

Fault tree	Minimal cut sets	Top gate type	No. of levels	Different events	Total events	Different gates	Chapters 5, 6, 8, 10	Chapter 9
rando47	15	OR	5	42	62	20	✓	✓
rando48	16	AND	5	21	42	16	✓	✓
rando49	4	AND	6	16	24	10	✓	✓
rando50	1	AND	5	8	12	4	✓	
rando51	3	OR	4	5	9	3	✓	
rando52	41	OR	11	34	80	33	✓	✓
rando53	2	AND	6	21	35	13	✓	✓
rando54	269	AND	9	34	39	13	✓	✓
rando55	9	AND	7	23	41	15	✓	✓
rando56	3	OR	5	9	15	6	✓	✓
rando57	2	AND	5	8	17	5	✓	
rando58	3	AND	6	17	28	10	✓	✓
rando59	99	OR	5	42	60	23	✓	✓
rando60	22	OR	7	70	87	36	✓	✓
rando61	15	AND	7	20	51	19	✓	✓
rando62	7	AND	6	18	35	13	✓	✓
rando63	9	AND	7	23	41	15	✓	✓
rando64	31	OR	7	35	45	19	✓	✓
rando65	13	OR	5	15	25	11	✓	✓
rando66	5	AND	8	26	51	17	✓	✓
rando67	1	AND	3	4	6	2	✓	
rando68	5	OR	5	8	19	6	✓	✓
rando69	6	OR	4	8	12	4	✓	
rando70	27	AND	7	24	28	12	✓	✓
rando71	2	AND	4	6	10	4	✓	✓
rando72	2	OR	4	6	14	5	✓	✓
rando73	80	AND	6	34	65	22	✓	✓
rando74	2	OR	4	6	8	3	✓	
rando75	4	AND	7	17	28	12	✓	✓
rando76	24	AND	6	32	45	15	✓	✓
rando77	27	OR	7	37	79	31	✓	✓
rando78	2	AND	6	30	38	17	✓	✓
rando79	4	OR	4	7	12	4	✓	
rando80	22	AND	5	26	29	9	✓	✓
rando81	4	OR	3	6	8	2	✓	
rando82	5	OR	7	13	27	9	✓	✓
rando83	39	AND	6	21	30	14	✓	✓
rando84	52	OR	6	39	47	19	✓	✓
rando85	7	OR	7	26	40	13	✓	✓
rando86	1	AND	4	7	9	3	✓	

Fault tree	Minimal cut sets	Top gate type	No. of levels	Different events	Total events	Different gates	Chapters 5, 6, 8, 10	Chapter 9
rando87	15	OR	6	22	29	11	✓	✓
rando88	29	OR	5	22	25	11	✓	✓
rando89	21	OR	8	41	61	24	✓	✓
rando90	2	AND	3	3	3	2	✓	
rando91	106	AND	6	58	98	32	✓	✓
rando92	58	AND	8	64	130	41	✓	✓
rando93	16	AND	7	40	55	19	✓	✓
rando94	1	AND	4	7	9	3	✓	
rando95	31	AND	7	22	31	11	✓	✓
rando96	5	AND	4	8	9	3	✓	
rando97	2	OR	5	5	6	4	✓	
rando98	283	OR	6	52	69	22	✓	✓
rando99	28	AND	6	40	77	26	✓	✓
random1	5	OR	4	6	12	6	✓	✓
random2	2	OR	3	5	7	2	✓	
random3	235	OR	8	49	61	24	✓	✓
random4	5	OR	3	5	9	2	✓	
random6	93	OR	6	49	122	45	✓	✓
random7	1	AND	4	5	8	3	✓	
random8	4	AND	6	15	21	7	✓	✓
random9	2	AND	5	9	17	5	✓	
relcour	6	AND	3	6	6	3	✓	
rstree1	3	AND	5	5	6	4	✓	
rstree2	3	AND	6	6	7	5	✓	
rstree3	6	AND	6	8	10	8	✓	✓
rstree4	4	OR	4	5	10	5	✓	
rstree5	2	OR	3	4	6	3	✓	
rstree6	4	OR	3	6	8	3	✓	
rstree7	8	AND	5	10	13	8	✓	✓
trials1	39	AND	10	16	66	27	✓	✓
trials2	5	OR	8	14	32	22	✓	✓
trials3	1	AND	10	25	44	20	✓	✓
trials4	49	OR	13	21	85	39	✓	✓
usatree	2	AND	3	4	5	3	✓	
worrell	10	AND	5	8	13	9	✓	✓
lisa100	313	OR	7	63	79	31		✓
lisa102	200063	AND	7	110	137	54		✓
lisa104	4	AND	6	20	28	10		✓
lisa107	6	OR	5	11	15	6		✓
lisa108	1	AND	5	10	17	6		✓

Fault tree	Minimal cut sets	Top gate type	No. of levels	Different events	Total events	Different gates	Chapters 5, 6, 8, 10	Chapter 9
lisa109	20	AND	7	20	50	21		✓
lisa110	32	OR	7	52	87	36		✓
lisa111	46	OR	6	50	55	17		✓
lisa112	4769	AND	6	81	90	32		✓
lisa113	79	AND	7	60	75	25		✓
lisa115	37	OR	5	25	35	12		✓
lisa116	6	OR	5	14	23	10		✓
lisa118	45505	OR	8	77	96	37		✓
lisa119	15	AND	4	14	16	7		✓
lisa121	72	OR	6	41	46	21		✓
lisa122	10	OR	6	23	38	12		✓
lisa124	1112	AND	6	65	81	28		✓
lisab11	2	AND	6	21	35	13		✓
lisab13	8	OR	8	31	61	24		✓
lisab14	1633	OR	6	84	140	46		✓
lisab15	8113	AND	9	98	122	49		✓
lisab17	1054	OR	7	68	76	27		✓
lisab22	493	AND	6	72	143	48		✓
lisab26	3	OR	5	9	15	6		✓
lisab27	285	AND	8	62	77	26		✓
lisab33	2	OR	5	11	15	6		✓
lisab37	64	AND	4	30	33	10		✓
lisab39	1	AND	4	5	10	4		✓
lisab40	3	AND	4	13	16	7		✓
lisab45	1	AND	6	10	25	9		✓
lisab47	3	AND	6	12	24	10		✓
lisab48	4	OR	4	17	29	8		✓
lisab50	2	AND	5	14	24	10		✓
lisab56	3	OR	6	17	29	11		✓
lisab61	14	OR	6	40	57	22		✓
lisab62	74	OR	6	39	43	17		✓
lisab63	6	OR	5	18	23	8		✓
lisab64	7	OR	7	40	67	21		✓
lisab66	33	OR	7	40	79	31		✓
lisab67	1118	OR	7	77	96	38		✓
lisab69	46	OR	5	30	33	14		✓
lisab70	88	OR	7	48	53	19		✓
lisab71	3	OR	5	14	23	8		✓
lisab72	34	OR	7	51	85	34		✓
lisab74	68122	AND	7	122	135	46		✓

Fault tree	Minimal cut sets	Top gate type	No. of levels	Different events	Total events	Different gates	Chapters 5, 6, 8, 10	Chapter 9
lisab75	1	AND	6	29	41	14		✓
lisab76	898	OR	7	58	96	38		✓
lisab77	130	OR	6	59	74	29		✓
lisab80	2	AND	4	7	10	4		✓
lisab82	33540	AND	6	85	94	31		✓
lisab83	61	OR	6	33	47	19		✓
lisab85	4	OR	5	12	15	6		✓
lisab87	93726	AND	7	96	137	54		✓
lisab88	28	AND	5	49	70	25		✓
lisab89	84	AND	7	61	76	32		✓
lisab91	7598	AND	7	62	77	32		✓
lisab94	5	OR	3	7	10	3		✓
lisab95	1	AND	5	18	25	7		✓
lisaba1	1054	OR	7	68	76	27		✓
lisaba2	66083	AND	6	114	143	48		✓
lisaba3	5396	AND	8	84	105	40		✓
lisaba5	228	AND	8	57	82	29		✓
lisaba6	990	AND	6	56	62	22		✓
lisaba7	1054	OR	7	68	76	27		✓
lisaba8	3344	OR	6	100	125	45		✓
rand159	13	OR	7	34	67	25		✓
rand161	114	AND	6	38	62	22		✓
rand163	716	OR	8	58	96	37		✓
rand164	4374	AND	7	58	77	32		✓
rand165	2072	AND	7	98	109	40		✓
rand166	262	OR	7	55	79	31		✓
rand167	256	OR	7	37	44	19		✓

Appendix III

Number of Non-Distinct Nodes in BDDs Obtained from the Original Fault Trees

Key to ordering schemes[1]:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
aaaaaaa	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
artqual	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
arttree	5	5	5	**4**	**4**	**4**	**4**	**4**
astolfo	128	131	**107**	123	128	125	131	130
bddtest	**38**	**38**	**38**	59	62	60	**38**	62
benjiam	87	**76**	**76**	80	87	84	80	83
bpfeg03	290934	104687	219063	316983	321123	316983	**95675**	364508
bpfen05	151974	**49337**	99003	53343	151563	150543	52619	151568
bpfig05	144054	**47987**	94863	142623	143643	142623	50135	142628
bpfin05	5316	**2915**	5282	5282	5282	5282	2963	5287
bpfpp02	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
bpfsw02	112553	**110698**	**110698**	110799	112553	112553	110799	112553
ch8tree	12	**11**	**11**	14	12	14	14	12
dre1019	**69**	**69**	**69**	**69**	**69**	**69**	73	**69**
dre1032	87	87	87	**81**	**81**	**81**	87	**81**
dre1057	2478	2487	2468	2303	2310	2310	2712	**2300**
dre1058	26237	30373	**22602**	24956	26189	22628	29232	23132
dre1059	65085	126229	119848	119408	64813	**56952**	125796	61036
dresden	838653	27379	23037	**22376**	787373	22628	2344652	631388
emerh2o	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
fatram2	11	11	11	11	**10**	**10**	11	11
hpisf02	225258	180757	180757	**137120**	267413	168046	530539	1105399

[1] For each fault tree, the ordering scheme(s) resulting in the fewest non-distinct BDD nodes is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
hpisf03	202	202	202	202	202	202	**182**	**182**
hpisf21	15155	**10593**	**10593**	11535	26825	26419	11505	62377
hpisf36	178	150	178	210	210	210	**132**	**132**
jdtree1	12	**10**	**10**	**10**	12	12	**10**	12
jdtree2	12	**10**	**10**	**10**	12	12	**10**	12
jdtree3	79	**71**	**71**	**71**	79	81	**71**	79
jdtree4	67	**59**	**59**	**59**	67	67	**59**	67
jdtree5	76	**70**	**70**	**70**	76	76	**70**	76
khictre	1244	**982**	1244	1364	1364	1364	**982**	999
lisa123	346	360	360	280	336	**234**	430	307
lisab10	14113	18490	24243	9828	8612	6719	23411	**4975**
lisab25	164	181	167	**149**	154	150	155	164
lisab28	201	156	190	160	171	**150**	190	162
lisab30	145	85	85	91	121	91	141	**77**
lisab31	6641	92082	92082	16757	5416	9295	51869	**5339**
lisab34	35	39	39	39	38	36	55	**34**
lisab35	17368	44339	44339	14332	14859	19581	31806	**12710**
lisab36	1553	698	708	4486	2724	3298	**450**	**450**
lisab42	23	**17**	**17**	**17**	**17**	**17**	24	**17**
lisab44	170	172	172	138	164	136	106	**98**
lisab51	104	95	95	87	103	91	**74**	91
lisab52	5376	33585	33585	29644	3961	20021	45209	**3360**
lisab53	25	25	25	**21**	22	22	25	**21**
lisab54	61	**43**	55	55	55	55	55	59
lisab57	1144	1751	1793	2359	**1063**	1134	1859	1193
lisab59	77222	**35962**	41272	105105	60994	110922	43242	114954
lisab60	101	66	66	97	60	97	97	**48**
lisab78	3694	2561	**2528**	6994	3687	6933	3959	4872
lisab86	5458	5464	4637	4310	5190	4394	4349	**3326**
lisaba4	9814	13425	13737	12478	**6170**	6811	12055	6330
lisaba9	5055	**2863**	**2863**	3850	6145	3850	3962	4435
modtree	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
nakashi	687	536	448	806	476	583	481	**375**
newtre2	9	9	9	9	9	9	**8**	10
newtre3	7	**6**	**6**	**6**	**6**	**6**	**6**	**6**
newtree	9	9	9	**8**	9	9	**8**	10
rand100	21	21	21	21	22	22	**14**	22
rand101	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rand102	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand103	125	106	106	**90**	112	110	103	106
rand104	81	104	104	90	94	88	**37**	88
rand105	1001	1117	1000	700	847	970	**525**	930
rand106	19	6769	6769	24	**16**	21	22	**17**
rand107	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand108	249	1978	1939	1139	**134**	520	1579	136
rand109	11133	14224	14224	**2766**	6656	4956	3661	3401
rand110	37	46	46	45	44	48	**22**	45
rand111	227	303	286	210	205	230	156	**115**
rand112	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand113	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rand114	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rand115	920	1131	1131	1022	809	620	1146	**587**
rand116	1182	837	1003	2172	846	2040	2660	**392**
rand117	43	32	**31**	**31**	**31**	**31**	**31**	32
rand118	1683	1014	1022	593	947	682	744	**396**
rand119	296	295	391	**263**	269	266	295	269
rand120	3925	5362	4839	3642	3607	4111	5536	**3077**
rand121	315	164	164	146	156	**142**	882	148
rand122	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand123	20	22	22	27	**17**	**17**	22	**17**
rand124	206	178	166	166	167	168	**125**	144
rand125	24	30	24	**21**	**21**	**21**	24	**21**
rand126	2024	5910	2552	**1496**	2106	1856	1820	3120
rand127	285	**218**	**218**	272	285	272	272	292
rand128	1833	**593**	630	1245	1762	1472	1266	931
rand129	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand130	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand131	8	8	8	8	8	9	8	**7**
rand132	4891	12858	12858	24215	4393	14958	22951	**3909**
rand133	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand134	**550**	4792	5470	51222	673	10732	29655	661
rand135	365	1255	1157	1101	395	**349**	1420	421
rand136	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand137	143	139	139	139	149	147	139	**99**
rand138	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand139	869	769	906	1273	645	1192	693	**624**
rand140	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand141	37	46	46	45	44	48	**22**	45

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand142	**59072**	74796	74322	115444	59462	111488	176698	62046
rand143	270	201	243	171	229	229	**127**	146
rand144	5617	39898	37150	15208	2309	9489	65099	**1985**
rand145	499	303	303	400	504	384	**193**	444
rand146	143	139	139	139	149	147	139	**99**
rand147	3017	160475	168581	6307	**2761**	36930	11385	5460
rand148	40	35	**30**	32	40	32	32	32
rand149	168	87	143	143	213	144	**84**	213
rand150	39213	108764	85856	62216	56768	57628	76449	**37284**
rand151	258	**148**	250	250	258	258	**148**	233
rand152	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand153	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
rand154	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand155	1051	1695	1888	1439	894	863	2484	**790**
rand156	44	**40**	**40**	**40**	**40**	**40**	41	**40**
rand158	39	30	30	26	25	26	26	**24**
rando10	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando11	**3.87×10^{7}**	1.02×10^{9}	1.05×10^{9}	1.35×10^{8}	3.93×10^{7}	1.31×10^{8}	6.98×10^{8}	4.54×10^{7}
rando12	9285	9739	9739	36182	7326	8415	35957	**6186**
rando13	1580	1963	1963	2456	**726**	3731	31647	767
rando14	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando15	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando16	533	**463**	474	1078	574	689	586	600
rando17	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando18	3625	31214	31214	768	2644	1273	**741**	2823
rando19	13171	402926	228764	312218	**9529**	26081	2318684	10784
rando20	**10126**	71152	81451	68892	14275	18437	232578	17137
rando21	36	**21**	35	**21**	36	**21**	**21**	24
rando22	61382	271889	271889	131606	15023	50787	361842	**14324**
rando23	151	159	159	159	138	150	159	**127**
rando24	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando25	24	**16**	**16**	59	22	34	37	19
rando26	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando27	**656**	8830	8830	6284	784	708	2867	669
rando28	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rando29	587	934	737	1216	819	**494**	1169	1233
rando30	12733	7929	9897	11787	12385	8160	**3800**	9866
rando31	**11**	**11**	**11**	1925	29	53	**11**	53
rando32	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando33	25	73	73	186	18	**17**	49	22
rando34	220	230	287	292	208	250	356	**196**
rando35	139	178	143	149	63	75	149	**57**
rando36	62	61	61	43	46	**42**	**42**	44
rando37	249	690	690	776	**156**	547	358	225
rando38	66	58	58	53	69	57	**33**	57
rando39	737	1400	1400	2252	**606**	2085	6824	997
rando40	54	36	36	48	39	**33**	48	**33**
rando41	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando42	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando43	94	106	106	**81**	94	91	85	94
rando44	367520	76890	**74190**	182720	234022	105551	212337	227519
rando45	132	159	159	264	99	**83**	182	97
rando46	20	**16**	**16**	**16**	23	21	51	32
rando47	1127	1530	1530	2558	**1094**	1930	2990	1229
rando48	87	**34**	38	**34**	56	**34**	98	38
rando49	26	**18**	**18**	**18**	26	26	**18**	26
rando50	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando51	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando52	563	503	451	747	559	431	347	**319**
rando53	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando54	1643	**920**	1577	1577	1579	1704	1705	1171
rando55	**24**	26	26	**24**	25	25	**24**	30
rando56	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rando57	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rando58	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
rando59	19360	31329	31329	21834	21162	19823	**11829**	17903
rando60	857	**383**	587	479	543	459	880	433
rando61	103	71	62	265	77	119	257	**42**
rando62	20	22	**11**	**11**	**11**	**11**	22	**11**
rando63	**24**	26	26	**24**	25	25	**24**	30
rando64	2632	1548	1409	1251	1598	1355	2153	**1108**
rando65	101	103	110	108	98	122	129	**93**
rando66	149	121	121	265	132	**113**	167	141
rando67	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando68	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rando69	**8**	**8**	**8**	**8**	**8**	9	**8**	**8**
rando70	100	79	79	79	84	97	99	**75**
rando71	7	7	7	7	7	7	7	**6**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando72	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando73	948	1481	1775	1775	836	1015	5802	**425**
rando74	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando75	16	16	16	16	**15**	16	16	**15**
rando76	404	233	229	303	311	323	**188**	288
rando77	266	358	358	414	**192**	345	562	213
rando78	**5**	**5**	8	8	**5**	**5**	**5**	**5**
rando79	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando80	123	119	119	**118**	121	**118**	145	**118**
rando81	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando82	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rando83	239	327	316	221	**210**	221	265	214
rando84	1683	1014	1022	593	947	682	744	**396**
rando85	**17**	18	18	26	18	18	**17**	27
rando86	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando87	22	23	23	20	**19**	**19**	21	**19**
rando88	822	**492**	679	787	636	812	598	662
rando89	236	238	286	284	229	277	451	**183**
rando90	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando91	33477	90989	80909	247443	29078	62748	49396	**26183**
rando92	**6679**	19287	20840	55703	10353	27182	8900	9657
rando93	76	144	96	66	47	**46**	131	**46**
rando94	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando95	89	**68**	86	86	89	76	86	76
rando96	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rando97	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando98	24786	28363	28363	23523	18087	26370	**12027**	13815
rando99	716	4427	4371	3876	**692**	1389	3159	805
random1	**6**	**6**	**6**	7	**6**	**6**	7	**6**
random2	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
random3	2377	4961	3371	2845	2681	3109	**2291**	2647
random4	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
random6	46584	1520239	1519663	635059	**41579**	49951	1042345	75836
random7	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
random8	36	**30**	35	35	36	**30**	**30**	**30**
random9	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
relcour	**9**	**9**	**9**	**9**	**9**	**9**	10	**9**
rstree1	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rstree2	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rstree3	14	**11**	**11**	14	14	14	**11**	14
rstree4	**5**	**5**	**5**	**5**	**5**	**5**	6	**5**
rstree5	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rstree6	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rstree7	29	33	33	**16**	18	**16**	22	**16**
trials1	807	800	1099	822	624	495	858	**344**
trials2	**12**	14	**12**	**12**	**12**	**12**	**12**	**12**
trials3	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
trials4	311	637	537	597	**264**	275	759	307
usatree	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
worrell	19	**17**	**17**	**17**	18	**17**	19	**17**

Appendix IV

Number of Distinct Nodes in BDDs Obtained from the Original Fault Trees

Key to ordering schemes[1]:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
aaaaaaa	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
artqual	9	**8**	9	9	9	9	9	**8**
arttree	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
astolfo	40	26	26	**25**	40	27	**25**	48
bddtest	32	**25**	**25**	34	52	37	**25**	40
benjiam	47	34	34	**32**	47	39	**32**	47
bpfeg03	101	**63**	**63**	**63**	93	**63**	**63**	70
bpfen05	90	**61**	**61**	**61**	82	**61**	**61**	85
bpfig05	88	**60**	**60**	**60**	81	**60**	**60**	63
bpfin05	45	**40**	**40**	**40**	**40**	**40**	**40**	42
bpfpp02	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
bpfsw02	150	**61**	**61**	62	150	150	62	150
ch8tree	10	**9**	**9**	10	10	10	10	10
dre1019	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**
dre1032	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**
dre1057	43	**32**	**32**	**32**	43	43	**32**	**32**
dre1058	164	186	90	107	190	103	**70**	72
dre1059	232	385	404	403	282	167	333	**152**
dresden	327	273	87	**80**	378	103	430	164
emerh2o	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
fatram2	11	11	11	11	**10**	**10**	11	11
hpisf02	414	**96**	**96**	98	361	357	134	419

[1] For each fault tree, the ordering scheme(s) resulting in the fewest distinct BDD nodes is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
hpisf03	45	**42**	**42**	**42**	45	45	**42**	46
hpisf21	220	**196**	**196**	249	429	406	210	1021
hpisf36	42	44	**40**	**40**	**40**	**40**	**40**	42
jdtree1	10	**7**	**7**	**7**	10	10	**7**	10
jdtree2	10	**7**	**7**	**7**	10	10	**7**	10
jdtree3	37	**21**	**21**	**21**	37	35	**21**	37
jdtree4	31	**19**	**19**	**19**	31	31	**19**	31
jdtree5	35	**20**	**20**	**20**	35	32	**20**	35
khictre	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
lisa123	82	40	40	**33**	75	44	84	124
lisab10	820	534	400	302	372	277	476	**243**
lisab25	65	64	69	55	62	56	**49**	66
lisab28	30	**22**	**22**	**22**	30	27	**22**	30
lisab30	44	37	37	34	40	34	36	**33**
lisab31	332	506	506	245	362	**176**	422	330
lisab34	22	**20**	**20**	**20**	23	**20**	28	23
lisab35	645	449	449	**207**	546	592	596	342
lisab36	134	114	110	324	162	267	**79**	98
lisab42	19	**17**	**17**	**17**	**17**	**17**	18	**17**
lisab44	**32**	**32**	**32**	41	33	41	**32**	41
lisab51	33	27	27	30	42	36	**24**	36
lisab52	598	778	778	659	423	524	623	**350**
lisab53	11	11	11	**9**	11	11	**9**	**9**
lisab54	25	22	**20**	**20**	**20**	**20**	**20**	22
lisab57	137	118	**110**	220	131	125	139	175
lisab59	133	**49**	**49**	**49**	172	81	**49**	144
lisab60	34	29	29	**25**	35	**25**	**25**	30
lisab78	313	**111**	149	**111**	196	140	195	167
lisab86	202	230	199	**160**	207	198	261	195
lisaba4	420	278	294	246	276	273	**148**	203
lisaba9	167	59	59	57	127	56	**55**	107
modtree	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
nakashi	147	**47**	65	62	118	71	49	111
newtre2	**6**	**6**	**6**	**6**	**6**	**6**	**6**	8
newtre3	6	**4**	**4**	**4**	**4**	**4**	**4**	**4**
newtree	**6**	**6**	**6**	**6**	**6**	**6**	**6**	8
rand100	13	13	13	13	15	15	**11**	15
rand101	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rand102	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand103	52	34	34	**27**	51	46	38	49
rand104	35	31	31	32	40	34	**19**	34
rand105	81	60	61	52	74	55	**49**	95
rand106	14	877	877	**13**	**13**	17	**13**	16
rand107	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand108	95	219	199	147	**80**	92	216	86
rand109	239	**127**	137	**127**	254	167	220	240
rand110	27	28	28	29	29	29	**13**	30
rand111	79	99	91	66	74	71	**47**	62
rand112	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand113	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rand114	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rand115	176	111	111	168	161	112	111	**94**
rand116	98	112	142	158	82	188	172	**63**
rand117	27	29	27	29	27	27	**25**	30
rand118	179	91	93	84	121	83	**70**	96
rand119	47	30	52	**29**	34	32	30	34
rand120	395	**82**	84	150	373	198	95	372
rand121	56	44	44	47	47	**39**	135	43
rand122	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand123	18	**17**	**17**	19	**17**	**17**	**17**	**17**
rand124	29	**21**	23	23	27	31	**21**	27
rand125	18	20	22	**17**	**17**	**17**	22	**17**
rand126	109	117	110	138	121	115	**100**	115
rand127	66	**29**	**29**	31	63	33	31	50
rand128	157	**70**	71	93	154	126	85	91
rand129	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand130	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand131	8	8	8	8	8	9	8	**7**
rand132	467	587	587	1084	**376**	845	1062	561
rand133	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand134	122	330	332	614	118	315	453	**109**
rand135	64	108	103	134	70	**57**	120	88
rand136	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand137	40	**32**	**32**	**32**	35	**32**	**32**	49
rand138	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand139	122	**76**	77	191	121	190	113	146
rand140	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand141	27	28	28	29	29	29	**13**	30

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand142	2056	2150	2139	2199	2133	1462	2411	**1339**
rand143	33	38	20	**17**	26	26	19	30
rand144	188	402	334	284	**177**	214	377	211
rand145	46	37	37	**34**	43	**34**	37	43
rand146	40	**32**	**32**	**32**	35	**32**	**32**	49
rand147	**234**	1474	1750	351	247	531	485	239
rand148	24	20	**14**	15	21	15	15	15
rand149	33	**21**	**21**	**21**	36	24	**21**	36
rand150	**785**	1896	1418	1634	997	960	1348	992
rand151	34	**25**	**25**	**25**	34	34	**25**	34
rand152	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand153	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
rand154	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand155	226	146	115	**97**	158	106	172	174
rand156	29	**22**	**22**	**22**	**22**	**22**	28	**22**
rand158	27	**18**	**18**	21	20	21	21	**18**
rando10	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando11	33318	55089	55097	**10447**	23334	16969	22213	12047
rando12	413	298	310	412	295	267	361	**226**
rando13	219	**76**	**76**	149	113	129	346	123
rando14	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando15	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando16	136	75	75	225	122	131	**62**	131
rando17	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando18	303	737	737	**64**	259	113	**64**	233
rando19	1076	1865	1832	3695	746	721	6392	**639**
rando20	737	1376	1076	1188	658	659	2121	**523**
rando21	16	**11**	**11**	**11**	16	**11**	**11**	16
rando22	1584	2726	2726	1394	1049	1615	3254	**878**
rando23	38	**31**	**31**	**31**	37	37	**31**	35
rando24	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando25	16	13	13	28	17	21	18	**12**
rando26	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando27	109	187	187	270	**90**	99	217	94
rando28	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rando29	124	183	174	172	124	**96**	115	108
rando30	295	**86**	93	94	272	185	90	249
rando31	**11**	**11**	**11**	145	17	25	**11**	25
rando32	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando33	18	31	31	48	16	**13**	19	16
rando34	64	62	58	58	57	66	**44**	48
rando35	45	51	54	55	**26**	27	30	29
rando36	28	17	17	**16**	20	**16**	**16**	18
rando37	71	179	179	203	73	226	**70**	97
rando38	24	19	19	**16**	32	22	17	22
rando39	140	170	170	347	**135**	259	361	237
rando40	27	**19**	**19**	24	25	**19**	24	**19**
rando41	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando42	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando43	31	28	28	**22**	35	30	**22**	31
rando44	734	245	239	385	470	**236**	461	476
rando45	54	50	50	52	47	**35**	60	40
rando46	20	**16**	**16**	**16**	23	21	29	30
rando47	128	154	154	81	77	**64**	86	70
rando48	34	25	**23**	25	28	26	38	27
rando49	21	**16**	**16**	**16**	25	18	**16**	18
rando50	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando51	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando52	106	**55**	124	130	115	129	86	112
rando53	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando54	68	**45**	68	68	70	68	69	61
rando55	24	25	25	**21**	25	25	24	26
rando56	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rando57	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rando58	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
rando59	557	239	239	182	452	293	**126**	235
rando60	98	**34**	37	**34**	73	59	42	57
rando61	41	37	33	57	35	44	53	**30**
rando62	13	12	**11**	**11**	**11**	**11**	12	**11**
rando63	24	25	25	**21**	25	25	24	26
rando64	172	91	101	93	198	104	**56**	103
rando65	35	33	**21**	29	33	38	32	38
rando66	59	**32**	**32**	47	54	46	56	53
rando67	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando68	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rando69	**8**	**8**	**8**	**8**	**8**	9	**8**	**8**
rando70	42	35	35	35	43	37	**33**	48
rando71	7	7	7	7	7	7	7	**6**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando72	2	2	2	2	2	2	2	2
rando73	136	114	127	127	108	105	240	64
rando74	2	2	2	2	2	2	2	2
rando75	13	13	13	13	11	13	13	11
rando76	71	30	39	39	39	45	39	40
rando77	72	82	82	60	48	48	132	41
rando78	5	5	8	8	5	5	5	5
rando79	4	4	4	4	4	4	4	4
rando80	31	26	26	29	31	29	36	29
rando81	4	4	4	4	4	4	4	4
rando82	6	6	6	6	6	6	6	6
rando83	62	54	58	34	46	34	58	47
rando84	179	91	93	84	121	83	70	96
rando85	15	15	15	17	15	15	15	22
rando86	2	2	2	2	2	2	2	2
rando87	19	20	20	16	15	15	15	15
rando88	64	31	49	37	41	38	52	45
rando89	34	28	30	34	33	33	39	32
rando90	3	3	3	3	3	3	3	3
rando91	490	1353	1341	861	432	458	1381	338
rando92	326	444	492	484	313	304	303	291
rando93	43	47	46	29	34	31	45	31
rando94	2	2	2	2	2	2	2	2
rando95	45	35	43	43	45	45	43	40
rando96	7	7	7	7	7	7	7	7
rando97	3	3	3	3	3	3	3	3
rando98	301	192	192	250	290	240	184	145
rando99	151	343	342	290	153	176	216	135
random1	6	6	6	7	6	6	7	6
random2	2	2	2	2	2	2	2	2
random3	133	124	130	136	135	132	80	146
random4	5	5	5	5	5	5	5	5
random6	1086	4817	4817	5067	1594	1341	5684	1485
random7	4	4	4	4	4	4	4	4
random8	18	14	14	14	18	14	14	18
random9	6	6	6	6	6	6	6	6
relcour	6	6	6	6	6	6	6	6
rstree1	4	4	4	4	4	4	4	4
rstree2	4	4	4	4	4	4	4	4

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rstree3	11	8	8	11	11	11	8	11
rstree4	5	5	5	5	5	5	5	5
rstree5	2	2	2	2	2	2	2	2
rstree6	4	4	4	4	4	4	4	4
rstree7	19	14	14	16	17	16	17	16
trials1	127	95	125	82	139	66	76	108
trials2	12	14	12	12	12	11	10	12
trials3	2	2	2	2	2	2	2	2
trials4	122	171	151	137	124	111	138	101
usatree	4	4	4	4	4	4	4	4
worrell	16	15	15	15	15	13	14	15

Appendix V

Number of If-Then-Else Calculations Required to Construct BDDs from the Original Fault Trees

Key to ordering schemes[1]:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
aaaaaaa	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
artqual	16	**15**	16	16	16	16	16	**15**
arttree	7	7	7	**6**	**6**	**6**	**6**	**6**
astolfo	56	59	56	**42**	58	47	60	67
bddtest	39	43	43	43	52	**38**	43	40
benjiam	76	75	75	71	76	**67**	75	101
bpfeg03	224	268	207	**196**	217	**196**	278	207
bpfen05	202	252	194	249	198	**181**	259	203
bpfig05	198	248	191	**177**	194	**177**	254	182
bpfin05	120	175	**103**	**103**	**103**	**103**	167	108
bpfpp02	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
bpfsw02	177	**111**	**111**	114	177	177	114	177
ch8tree	18	**17**	**17**	19	18	19	19	18
dre1019	**33**	**33**	**33**	**33**	**33**	**33**	37	**33**
dre1032	39	39	39	**33**	**33**	**33**	39	**33**
dre1057	65	73	63	60	59	59	76	**57**
dre1058	273	419	254	299	306	205	298	**203**
dre1059	299	598	631	630	333	**230**	540	251
dresden	864	1823	811	**744**	1220	810	1821	780
emerh2o	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**

[1] For each fault tree, the ordering scheme(s) requiring the fewest ite calculations to construct the BDD is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
fatram2	18	18	18	18	18	18	18	**17**
hpisf02	581	**223**	**223**	269	593	575	402	920
hpisf03	96	**91**	**91**	**91**	96	96	94	100
hpisf21	769	**621**	**621**	852	975	832	816	2717
hpisf36	124	126	118	**112**	116	**112**	126	132
jdtree1	10	**9**	**9**	**9**	10	10	**9**	10
jdtree2	10	**9**	**9**	**9**	10	10	**9**	10
jdtree3	46	**38**	**38**	**38**	46	44	**38**	46
jdtree4	52	**46**	**46**	**46**	52	52	**46**	52
jdtree5	46	**38**	**38**	**38**	46	44	**38**	46
khictre	207	**184**	207	204	204	204	**184**	200
lisa123	170	160	160	145	186	**132**	176	245
lisab10	1150	1041	880	687	667	689	1021	**571**
lisab25	204	**192**	216	202	203	193	197	209
lisab28	42	47	40	**37**	39	44	40	48
lisab30	255	216	214	207	231	**192**	239	220
lisab31	849	1289	1289	994	808	730	1195	**721**
lisab34	**58**	60	60	60	63	61	67	61
lisab35	772	534	534	**434**	682	703	694	544
lisab36	1053	1277	1142	1316	1121	1257	1059	**1017**
lisab42	44	**41**	**41**	**41**	**41**	**41**	42	**41**
lisab44	**101**	103	103	108	102	114	102	111
lisab51	57	**51**	**51**	54	61	58	62	56
lisab52	930	1287	1287	1007	742	808	1159	**654**
lisab53	**16**	**16**	**16**	**16**	17	17	**16**	**16**
lisab54	63	**56**	58	58	58	58	60	58
lisab57	191	189	**183**	376	184	189	267	243
lisab59	182	129	126	136	223	164	**125**	216
lisab60	59	53	53	**51**	59	52	**51**	52
lisab78	414	279	313	242	282	**239**	332	260
lisab86	313	386	304	286	318	301	417	**272**
lisaba4	684	746	767	671	434	469	**402**	438
lisaba9	266	164	164	130	194	**126**	131	202
modtree	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
nakashi	229	**98**	128	125	168	115	103	157
newtre2	**12**	**12**	**12**	**12**	**12**	**12**	13	13
newtre3	7	**6**	**6**	**6**	**6**	**6**	**6**	**6**
newtree	**10**	**10**	**10**	11	**10**	**10**	11	11
rand100	187	191	191	189	**173**	175	180	**173**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand101	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
rand102	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
rand103	133	115	115	**102**	132	124	121	138
rand104	280	324	328	316	262	274	**168**	269
rand105	143	143	129	115	137	**113**	128	172
rand106	679	2050	2050	796	624	554	963	**530**
rand107	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
rand108	369	580	558	499	351	419	564	**321**
rand109	522	**337**	343	455	529	415	543	658
rand110	**390**	547	490	486	451	459	395	457
rand111	325	370	327	282	314	**272**	277	287
rand112	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
rand113	81	81	81	81	**73**	**73**	81	**73**
rand114	24	24	24	24	**23**	**23**	24	**23**
rand115	314	286	286	326	301	227	**226**	234
rand116	521	383	432	801	474	744	761	**355**
rand117	63	62	60	60	58	58	**57**	59
rand118	271	**166**	182	173	209	180	167	**166**
rand119	114	119	124	**97**	103	101	122	100
rand120	493	**211**	246	272	473	313	279	460
rand121	215	205	205	199	201	**196**	283	207
rand122	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand123	55	61	61	57	56	**51**	61	57
rand124	**91**	101	108	94	99	100	112	102
rand125	52	55	53	**49**	**49**	**49**	53	**49**
rand126	288	259	259	265	295	**238**	273	298
rand127	97	67	67	**63**	89	64	**63**	86
rand128	660	455	**439**	644	647	695	624	440
rand129	56	56	56	55	55	55	55	**53**
rand130	207	178	178	**172**	178	176	208	179
rand131	14	14	14	14	14	16	14	**13**
rand132	765	817	817	1550	**607**	1195	1525	888
rand133	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
rand134	546	719	739	1487	550	1028	1123	**526**
rand135	319	322	317	515	326	**269**	390	345
rand136	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rand137	81	**75**	**75**	**75**	78	**75**	**75**	86
rand138	**101**	**101**	**101**	**101**	103	103	**101**	103
rand139	320	**226**	273	511	324	515	267	365

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand140	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
rand141	**390**	547	490	486	451	459	395	457
rand142	2923	2643	2632	2965	3013	2150	3318	**1902**
rand143	166	141	149	**140**	151	146	**140**	156
rand144	555	1065	966	961	**505**	630	1084	572
rand145	80	88	88	**72**	81	76	92	87
rand146	81	**75**	**75**	**75**	78	**75**	**75**	86
rand147	621	2689	2956	755	**619**	1096	1022	654
rand148	78	62	**61**	68	73	68	72	69
rand149	215	244	231	221	229	**198**	244	243
rand150	**1189**	2334	1816	2053	1301	1332	1679	1298
rand151	84	90	**80**	**80**	85	84	91	89
rand152	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand153	175	167	167	**163**	**163**	**163**	**163**	170
rand154	123	84	119	119	93	93	122	**81**
rand155	314	307	297	213	238	**196**	448	250
rand156	170	130	130	130	130	**124**	178	130
rand158	11389	7473	7473	7942	8785	**6677**	7127	7869
rando10	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
rando11	33678	56142	56138	**11312**	23727	17480	24658	12557
rando12	747	595	605	887	567	561	721	**531**
rando13	998	1107	1107	1654	**892**	1213	1725	938
rando14	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
rando15	**26**	**26**	**26**	**26**	27	**26**	**26**	27
rando16	1895	1790	1553	1815	1876	1703	**1484**	1881
rando17	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
rando18	12975	26709	25905	38686	9984	22662	36197	**9819**
rando19	1772	3287	3249	4943	**1401**	1610	8248	1445
rando20	2605	2960	2735	3006	2439	2644	6408	**2383**
rando21	17	17	**16**	17	17	17	17	20
rando22	3662	6136	6136	4453	2668	3238	7618	**2237**
rando23	317	361	361	354	266	279	360	**235**
rando24	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
rando25	85	**74**	**74**	125	86	105	92	**74**
rando26	**32**	**32**	**32**	**32**	**32**	**32**	**32**	**32**
rando27	2273	2588	2586	2248	2579	2123	**1964**	2289
rando28	245	242	236	235	**231**	233	238	241
rando29	449	587	449	450	428	**374**	415	386
rando30	351	**163**	176	174	327	269	186	307

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando31	1547	**1420**	1491	2206	1491	1535	1431	1668
rando32	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**
rando33	180	172	172	201	182	**167**	201	177
rando34	397	379	443	447	375	389	386	**318**
rando35	378	474	426	421	296	297	**277**	284
rando36	141	**129**	136	146	147	140	142	138
rando37	388	554	554	590	**387**	572	399	437
rando38	99	96	96	95	99	92	**88**	92
rando39	389	351	351	791	**346**	593	684	433
rando40	45	**41**	**41**	42	50	44	42	**41**
rando41	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
rando42	**51**	52	55	54	55	54	52	59
rando43	101	106	106	104	103	101	113	**99**
rando44	979	388	**380**	571	599	469	674	677
rando45	246	238	238	299	228	**209**	245	216
rando46	687	595	595	522	522	498	**441**	497
rando47	244	284	284	245	202	206	243	**199**
rando48	144	139	139	137	138	**131**	144	143
rando49	65	**62**	**62**	**62**	68	**62**	**62**	**62**
rando50	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
rando51	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
rando52	1138	1074	1154	1220	1109	1161	1239	**956**
rando53	107	**95**	118	115	113	113	**95**	115
rando54	133	**128**	130	130	133	132	130	139
rando55	191	191	191	217	193	192	189	**187**
rando56	**31**	**31**	**31**	**31**	**31**	**31**	**31**	36
rando57	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**
rando58	104	107	107	106	**95**	**95**	103	**95**
rando59	820	493	493	444	696	548	**411**	520
rando60	355	378	405	378	330	**291**	452	296
rando61	165	173	168	234	162	196	230	**161**
rando62	108	**107**	110	113	109	112	**107**	109
rando63	191	191	191	217	193	192	189	**187**
rando64	519	382	363	341	562	**329**	356	337
rando65	62	64	**56**	61	63	75	94	73
rando66	248	243	243	282	247	**231**	270	242
rando67	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rando68	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
rando69	**17**	**17**	**17**	**17**	**17**	18	**17**	**17**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando70	94	**86**	**86**	**86**	87	88	87	91
rando71	17	17	17	17	17	17	17	**16**
rando72	**17**	**17**	**17**	**17**	18	18	**17**	18
rando73	325	333	330	332	283	270	489	**261**
rando74	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
rando75	**58**	**58**	**58**	**58**	**58**	59	**58**	59
rando76	226	**159**	208	190	181	188	238	203
rando77	514	470	**465**	529	536	559	688	533
rando78	**107**	109	120	115	**107**	**107**	112	113
rando79	**16**	**16**	17	17	**16**	**16**	**16**	**16**
rando80	59	**54**	**54**	59	60	59	65	59
rando81	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
rando82	60	60	60	60	60	62	**59**	60
rando83	97	111	105	**75**	84	**75**	110	82
rando84	271	**166**	182	173	209	180	167	**166**
rando85	221	213	213	205	**200**	**200**	219	204
rando86	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
rando87	55	59	59	55	**52**	**52**	54	**52**
rando88	93	71	92	72	**69**	74	88	71
rando89	987	866	833	822	910	845	**621**	970
rando90	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando91	899	1772	1758	1408	838	857	1829	**789**
rando92	7078	9862	11801	13736	5392	5029	**3656**	6782
rando93	390	332	**290**	388	339	343	321	307
rando94	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
rando95	134	**95**	124	124	134	120	124	114
rando96	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
rando97	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
rando98	755	611	611	714	649	639	654	**450**
rando99	**505**	1091	1034	921	541	664	864	585
random1	20	20	20	**19**	**19**	**19**	**19**	20
random2	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
random3	601	696	604	632	574	531	**452**	595
random4	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
random6	**1978**	6846	6845	7631	2791	2686	8041	3081
random7	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
random8	72	**66**	68	68	72	**66**	**66**	**66**
random9	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
relcour	**7**	**7**	**7**	**7**	**7**	**7**	8	**7**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rstree1	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
rstree2	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
rstree3	**14**	15	15	15	15	15	15	15
rstree4	**10**	**10**	**10**	**10**	**10**	**10**	11	**10**
rstree5	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rstree6	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
rstree7	41	31	31	**30**	32	**30**	**30**	**30**
trials1	302	293	310	252	312	**224**	260	268
trials2	67	68	67	67	70	70	**65**	66
trials3	116	118	112	114	111	103	114	**100**
trials4	450	588	511	522	400	388	469	**358**
usatree	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
worrell	29	28	28	28	28	27	**24**	27

Appendix VI

Number of Non-Distinct Nodes in BDDs Obtained from Fault Trees Restructured Using the Faunet Reduction Method

Key to ordering schemes[1]:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
aaaaaaa	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
artqual	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
arttree	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
astolfo	24	**23**	**23**	27	24	27	**23**	30
bddtest	**32**	35	35	35	**32**	36	35	**32**
benjiam	87	**76**	**76**	80	87	84	80	83
bpfeg03	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfen05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfig05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfin05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfpp02	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfsw02	19	19	**14**	**14**	19	**14**	19	15
ch8tree	9	**8**	**8**	**8**	9	10	10	**8**
dre1019	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1032	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1057	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1058	30	**26**	**26**	**26**	30	28	**26**	30
dre1059	256	312	261	261	232	**214**	312	216
dresden	540	**160**	**160**	**160**	540	441	550	543
emerh2o	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

[1] For each fault tree, the ordering scheme(s) resulting in the fewest non-distinct BDD nodes is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
fatram2	9	9	9	9	9	9	9	10
hpisf02	159	137	137	140	171	140	130	172
hpisf03	14	14	14	14	14	14	14	14
hpisf21	30	41	41	38	33	41	32	31
hpisf36	14	14	14	14	14	14	14	14
jdtree1	1	1	1	1	1	1	1	1
jdtree2	1	1	1	1	1	1	1	1
jdtree3	1	1	1	1	1	1	1	1
jdtree4	6	6	6	6	6	6	6	6
jdtree5	4	4	4	4	4	4	4	4
khictre	36	30	30	33	39	33	30	30
lisa123	207	227	227	171	180	123	205	164
lisab10	11160	14901	18023	7476	7180	4883	20756	3913
lisab25	90	93	82	77	86	78	87	84
lisab28	1	1	1	1	1	1	1	1
lisab30	69	57	57	57	65	61	61	49
lisab31	5798	83846	83846	14033	5369	6963	47793	4656
lisab34	25	20	32	25	23	25	32	23
lisab35	1572	2739	2935	2170	1584	2273	2739	679
lisab36	1553	698	708	2930	2576	3102	450	450
lisab42	6	6	6	6	6	6	6	6
lisab44	138	41	128	128	141	125	41	74
lisab51	17	16	16	16	17	16	18	21
lisab52	4957	28897	28897	26738	3169	18701	34655	2772
lisab53	1	1	1	1	1	1	1	1
lisab54	22	22	19	19	21	19	19	20
lisab57	1438	1626	1309	1582	1188	1466	1523	957
lisab59	1	1	1	1	1	1	1	1
lisab60	35	57	57	33	33	34	47	32
lisab78	603	539	423	538	508	538	532	432
lisab86	1132	2269	1954	1173	943	1188	1104	872
lisaba4	2694	4921	5584	5054	2211	1971	6384	1764
lisaba9	48	46	46	36	36	37	36	39
modtree	2	2	2	2	2	2	2	2
nakashi	501	359	318	342	360	455	359	304
newtre2	3	3	3	3	3	3	3	3
newtre3	3	3	3	3	3	3	3	3
newtree	1	1	1	1	1	1	1	1
rand100	18	18	18	18	19	19	13	19

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand101	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand102	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand103	31	33	29	25	25	25	**23**	**23**
rand104	69	70	70	70	68	66	**31**	66
rand105	29	34	28	28	**27**	28	32	**27**
rand106	19	22	24	3495	18	37	5780	**17**
rand107	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand108	249	1978	1555	1139	**133**	499	1579	142
rand109	775	1169	1269	535	547	443	**404**	533
rand110	30	35	35	35	34	38	**20**	36
rand111	214	277	266	188	186	220	135	**106**
rand112	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rand113	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rand114	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand115	497	659	659	655	419	389	750	**312**
rand116	1189	1003	1003	2172	858	1900	2660	**391**
rand117	**13**	15	14	14	**13**	**13**	15	**13**
rand118	260	184	177	188	236	199	154	**132**
rand119	25	24	24	**22**	25	**22**	24	**22**
rand120	238	242	207	**203**	218	206	327	232
rand121	176	84	84	**78**	107	**78**	453	100
rand122	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand123	12	13	13	16	**10**	16	13	16
rand124	23	19	**18**	**18**	**18**	**18**	22	**18**
rand125	15	19	14	14	**13**	**13**	14	**13**
rand126	238	430	267	236	196	186	**175**	220
rand127	11	**10**	**10**	**10**	11	**10**	**10**	**10**
rand128	3586	**417**	**417**	3045	1569	2388	960	526
rand129	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand130	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand131	6	6	6	6	6	7	6	**5**
rand132	4901	12858	12858	23901	3920	9231	22637	**3813**
rand133	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rand134	1918	2659	3020	35046	330	1428	18819	**326**
rand135	365	1255	1157	981	395	**349**	1420	421
rand136	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rand137	69	67	67	67	70	72	56	**43**
rand138	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand139	501	461	544	645	358	733	523	**341**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand140	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand141	30	35	35	35	34	38	**20**	36
rand142	43467	44476	43955	77267	**41534**	64968	119851	42920
rand143	31	42	36	**28**	31	30	**28**	31
rand144	5141	27724	25960	10268	**2459**	5138	6895	2996
rand145	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rand146	69	67	67	67	70	72	56	**43**
rand147	6692	60575	50517	54288	2262	**1493**	6842	1655
rand148	8	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rand149	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
rand150	30959	78516	61194	60240	39610	81592	64501	**24546**
rand151	13	12	12	12	12	15	12	**11**
rand152	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand153	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
rand154	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand155	733	1018	1052	1044	663	903	1507	**525**
rand156	23	22	22	22	23	22	**20**	23
rand158	35	29	29	24	24	24	24	**22**
rando10	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando11	**3944979**	1.24×10^8	1.24×10^8	2.39×10^7	9066673	1.91×10^7	1.16×10^8	7290272
rando12	3969	2838	2838	3393	2152	2007	9122	**1932**
rando13	1235	1963	1963	2456	**702**	4955	31647	767
rando14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rando15	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando16	409	**238**	252	796	352	421	300	396
rando17	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando18	3349	24217	24217	684	2350	1256	**618**	2681
rando19	44265	207893	207893	237216	10274	**9413**	602967	14141
rando20	**9780**	85872	81451	99481	9886	18437	232578	13786
rando21	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rando22	47936	220656	221672	118924	15661	59043	388687	**13776**
rando23	106	111	111	111	91	105	200	**68**
rando24	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando25	20	16	16	16	20	16	37	**15**
rando26	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando27	**656**	8830	8830	6284	784	708	2867	669
rando28	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rando29	475	293	293	479	553	**265**	569	467
rando30	121	112	**103**	**103**	121	107	155	146

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando31	**11**	**11**	**11**	1925	29	53	**11**	53
rando32	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando33	24	72	70	60	**18**	23	68	20
rando34	174	167	185	185	163	202	270	**158**
rando35	104	142	117	117	74	71	149	**57**
rando36	22	25	25	**18**	21	**18**	22	22
rando37	460	544	585	702	**183**	221	283	231
rando38	13	13	13	13	13	13	**11**	14
rando39	736	1290	1290	2252	**606**	2085	6824	997
rando40	38	**23**	24	36	38	27	38	**23**
rando41	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando42	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando43	14	17	17	17	14	16	**13**	14
rando44	1543	1000	957	1341	1487	**903**	1453	1506
rando45	132	159	159	264	99	**83**	182	97
rando46	14	**12**	**12**	**12**	13	15	**12**	18
rando47	**474**	706	706	1322	589	626	1106	562
rando48	77	**32**	**32**	**32**	49	**32**	91	38
rando49	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
rando50	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando51	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando52	411	571	510	540	279	315	299	**242**
rando53	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando54	39	**35**	38	38	39	42	38	41
rando55	26	29	26	22	26	**21**	26	26
rando56	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rando57	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando58	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
rando59	3836	3624	3425	2593	3158	2685	**1281**	2058
rando60	83	291	309	69	**63**	**63**	109	65
rando61	111	88	67	265	94	75	257	**42**
rando62	14	16	**9**	**9**	**9**	**9**	16	**9**
rando63	26	29	26	22	26	**21**	26	26
rando64	216	244	222	187	171	187	208	**151**
rando65	46	42	42	59	42	39	44	**38**
rando66	92	185	185	185	108	**89**	184	141
rando67	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando68	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rando69	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando70	61	47	47	59	56	55	74	49
rando71	6	6	6	6	6	6	6	5
rando72	2	2	2	2	2	2	2	2
rando73	1019	1480	1816	1816	553	1146	7349	424
rando74	2	2	2	2	2	2	2	2
rando75	14	14	14	14	13	14	14	13
rando76	59	39	49	51	49	51	39	53
rando77	329	378	378	399	186	294	559	213
rando78	4	4	7	7	4	4	7	4
rando79	3	3	3	3	3	3	3	3
rando80	8	8	8	8	8	8	8	8
rando81	3	3	3	3	3	3	3	3
rando82	6	6	6	6	6	6	6	6
rando83	169	228	211	182	177	182	244	168
rando84	260	184	177	188	236	199	154	132
rando85	14	15	15	21	15	14	14	22
rando86	1	1	1	1	1	1	1	1
rando87	9	11	11	11	9	11	11	9
rando88	35	34	31	34	29	30	26	23
rando89	94	102	110	110	95	111	182	81
rando90	1	1	1	1	1	1	1	1
rando91	29915	62303	62303	85085	16890	24765	41973	23606
rando92	6594	14779	17238	88624	6503	21162	7544	7909
rando93	47	57	57	43	37	34	62	35
rando94	1	1	1	1	1	1	1	1
rando95	49	40	48	48	48	40	48	37
rando96	3	3	3	3	3	3	3	3
rando97	3	3	3	3	3	3	3	3
rando98	572	1031	1031	1031	566	748	949	785
rando99	522	3139	2539	2837	515	2623	2817	547
random1	6	7	7	6	6	6	7	6
random2	2	2	2	2	2	2	2	2
random3	299	410	388	388	277	387	895	273
random4	5	5	5	5	5	5	5	5
random6	50601	1515090	1514514	630016	35450	155781	1034143	70875
random7	4	4	4	4	4	4	4	4
random8	9	9	9	9	9	9	9	9
random9	4	4	4	4	4	4	4	4
relcour	1	1	1	1	1	1	1	1

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rstree1	3	3	3	3	3	3	3	3
rstree2	3	3	3	3	3	3	3	3
rstree3	10	7	7	10	10	8	10	8
rstree4	4	4	4	4	4	4	4	4
rstree5	2	2	2	2	2	2	2	2
rstree6	3	3	3	3	3	3	3	3
rstree7	15	17	17	10	13	10	14	12
trials1	244	439	439	416	221	230	513	186
trials2	11	12	12	12	10	11	12	12
trials3	2	2	2	2	2	2	2	2
trials4	242	491	496	698	210	291	760	262
usatree	1	1	1	1	1	1	1	1
worrell	19	17	17	17	18	17	19	17

Appendix VII

Number of Distinct Nodes in BDDs Obtained from Fault Trees Restructured Using the Faunet Reduction Method

Key to ordering schemes[1]:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
aaaaaaa	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
artqual	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
arttree	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
astolfo	16	17	17	**14**	16	18	17	19
bddtest	26	**22**	**22**	**22**	26	25	**22**	26
benjiam	47	34	34	**32**	47	39	**32**	47
bpfeg03	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfen05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfig05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfin05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfpp02	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfsw02	17	14	**13**	**13**	17	**13**	14	14
ch8tree	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
dre1019	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1032	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1057	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1058	24	**18**	**18**	**18**	24	21	**18**	24
dre1059	89	94	91	91	70	57	94	**51**
dresden	134	**23**	**23**	**23**	134	63	39	137
emerh2o	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

[1] For each fault tree, the ordering scheme(s) resulting in the fewest distinct BDD nodes is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
fatram2	9	9	9	9	9	9	9	10
hpisf02	77	24	24	34	67	34	33	86
hpisf03	11	11	11	11	11	11	11	11
hpisf21	26	22	22	24	24	25	25	24
hpisf36	11	11	11	11	11	11	11	11
jdtree1	1	1	1	1	1	1	1	1
jdtree2	1	1	1	1	1	1	1	1
jdtree3	1	1	1	1	1	1	1	1
jdtree4	6	6	6	6	6	6	6	6
jdtree5	4	4	4	4	4	4	4	4
khictre	15	11	11	11	17	11	11	11
lisa123	75	36	36	29	80	38	67	58
lisab10	780	522	385	290	346	269	448	246
lisab25	54	48	55	45	52	46	43	53
lisab28	1	1	1	1	1	1	1	1
lisab30	29	28	28	28	27	27	23	23
lisab31	407	500	500	242	301	172	415	299
lisab34	18	16	20	16	19	16	20	19
lisab35	329	339	362	194	295	347	339	164
lisab36	134	114	110	301	145	256	79	96
lisab42	6	6	6	6	6	6	6	6
lisab44	35	29	37	35	41	32	29	33
lisab51	13	12	12	12	13	12	12	18
lisab52	583	751	751	656	385	522	551	339
lisab53	1	1	1	1	1	1	1	1
lisab54	18	15	15	15	17	15	13	16
lisab57	110	108	118	102	96	120	120	159
lisab59	1	1	1	1	1	1	1	1
lisab60	26	20	20	23	24	24	22	22
lisab78	178	69	105	106	153	106	83	89
lisab86	148	164	141	107	150	104	146	143
lisaba4	328	204	222	179	200	198	208	191
lisaba9	36	18	18	18	21	18	18	26
modtree	2	2	2	2	2	2	2	2
nakashi	138	43	64	60	120	55	43	105
newtre2	3	3	3	3	3	3	3	3
newtre3	3	3	3	3	3	3	3	3
newtree	1	1	1	1	1	1	1	1
rand100	12	12	12	12	14	14	10	14

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand101	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand102	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand103	24	**19**	21	24	24	24	22	22
rand104	33	29	29	29	33	30	**17**	30
rand105	23	24	20	20	22	20	**16**	22
rand106	14	**13**	**13**	606	15	25	782	16
rand107	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand108	97	219	192	147	**81**	91	216	90
rand109	147	85	95	**83**	143	109	105	147
rand110	26	29	29	29	25	26	**12**	27
rand111	77	96	88	66	70	68	**46**	60
rand112	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rand113	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rand114	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand115	126	106	106	113	112	94	98	**87**
rand116	89	142	142	158	81	168	172	**61**
rand117	**12**	14	14	14	13	13	14	13
rand118	79	51	**49**	56	70	59	**49**	56
rand119	17	14	14	**13**	17	**13**	14	**13**
rand120	98	52	**43**	72	91	73	81	109
rand121	42	33	33	**32**	35	34	86	33
rand122	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand123	11	**10**	**10**	12	**10**	12	**10**	12
rand124	15	**11**	13	13	13	13	13	13
rand125	12	13	14	14	**11**	**11**	14	**11**
rand126	74	79	74	90	66	59	69	**58**
rand127	10	9	9	9	10	9	**8**	**8**
rand128	134	**59**	**59**	88	144	98	77	74
rand129	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand130	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand131	6	6	6	6	6	7	6	**5**
rand132	473	584	584	1128	**347**	591	1106	559
rand133	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rand134	188	253	255	579	100	146	424	**84**
rand135	64	108	103	128	70	**57**	120	88
rand136	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rand137	23	**21**	**21**	**21**	27	**21**	29	27
rand138	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand139	98	65	**63**	150	102	155	106	121

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand140	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand141	26	29	29	29	25	26	**12**	27
rand142	1892	1789	1732	1876	1595	**1248**	1992	1299
rand143	16	21	14	**12**	16	14	**12**	16
rand144	**179**	381	313	275	197	231	403	217
rand145	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rand146	23	**21**	**21**	**21**	27	**21**	29	27
rand147	391	1656	1481	1276	257	**206**	425	228
rand148	7	**5**	6	6	6	6	6	6
rand149	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
rand150	**981**	1843	1341	1908	1211	1554	1273	1040
rand151	11	9	9	9	9	10	9	**8**
rand152	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand153	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
rand154	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand155	190	134	102	**76**	131	83	155	122
rand156	17	**14**	**14**	**14**	17	**14**	**14**	18
rand158	23	**17**	**17**	20	20	20	20	**17**
rando10	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando11	16878	46572	46572	7965	13456	**7171**	16680	8829
rando12	309	214	214	242	275	221	320	**198**
rando13	179	**76**	**76**	149	126	137	384	123
rando14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rando15	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando16	122	65	65	205	101	98	**63**	103
rando17	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando18	287	594	594	**62**	212	108	63	184
rando19	1496	1811	1811	3565	920	950	5094	**791**
rando20	711	1348	1076	1148	635	663	2121	**490**
rando21	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rando22	1537	2648	2565	1376	1045	1657	3418	**803**
rando23	31	**25**	**25**	**25**	31	31	29	32
rando24	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando25	14	13	13	13	15	13	18	**12**
rando26	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando27	109	187	187	270	**90**	99	217	94
rando28	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rando29	154	**63**	**63**	158	123	71	89	93
rando30	44	38	**34**	**34**	44	39	35	49

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando31	**11**	**11**	**11**	145	17	25	**11**	25
rando32	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando33	17	30	28	24	**16**	**16**	25	**16**
rando34	56	49	50	50	50	53	**41**	45
rando35	32	50	45	45	27	**25**	30	29
rando36	17	13	13	**12**	16	**12**	14	16
rando37	108	193	202	200	90	104	**59**	100
rando38	11	11	11	11	11	11	**9**	12
rando39	**139**	160	160	347	**139**	259	357	237
rando40	19	15	15	16	19	**14**	20	18
rando41	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando42	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando43	11	11	11	11	11	12	**9**	11
rando44	244	112	**106**	184	257	169	186	226
rando45	54	50	50	52	47	**35**	60	40
rando46	14	**12**	**12**	**12**	13	15	**12**	18
rando47	72	120	120	78	64	**58**	73	67
rando48	33	24	**22**	**22**	29	23	35	25
rando49	11	11	11	11	11	11	11	**9**
rando50	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando51	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando52	101	86	123	126	112	125	**85**	109
rando53	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rando54	28	**19**	27	27	28	28	27	29
rando55	22	23	22	**19**	23	21	22	23
rando56	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rando57	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando58	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
rando59	363	177	172	120	392	148	**118**	177
rando60	34	65	63	**21**	26	26	35	29
rando61	43	33	34	57	39	35	52	**30**
rando62	11	10	**9**	**9**	**9**	**9**	10	**9**
rando63	22	23	22	**3**	23	21	22	23
rando64	86	**50**	56	62	73	62	55	64
rando65	30	**19**	**19**	32	29	26	23	26
rando66	45	46	46	46	47	**40**	53	53
rando67	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando68	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rando69	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando70	26	22	22	20	26	23	27	31
rando71	6	6	6	6	6	6	6	5
rando72	2	2	2	2	2	2	2	2
rando73	110	113	125	125	86	97	239	63
rando74	2	2	2	2	2	2	2	2
rando75	12	12	12	12	10	12	12	10
rando76	34	19	24	23	22	23	19	24
rando77	73	85	85	57	47	43	131	41
rando78	4	4	7	7	4	4	7	4
rando79	3	3	3	3	3	3	3	3
rando80	8	8	8	8	8	8	8	8
rando81	3	3	3	3	3	3	3	3
rando82	6	6	6	6	6	6	6	6
rando83	56	45	49	31	42	31	49	42
rando84	79	51	49	56	70	59	49	56
rando85	13	13	13	15	14	13	13	19
rando86	1	1	1	1	1	1	1	1
rando87	9	11	11	11	9	11	11	9
rando88	23	14	17	14	21	15	18	19
rando89	26	22	24	24	27	27	49	27
rando90	1	1	1	1	1	1	1	1
rando91	674	1250	1250	486	513	408	1436	335
rando92	276	446	456	568	288	283	291	261
rando93	32	34	34	22	30	27	35	28
rando94	1	1	1	1	1	1	1	1
rando95	31	26	30	30	30	32	30	29
rando96	3	3	3	3	3	3	3	3
rando97	3	3	3	3	3	3	3	3
rando98	172	126	126	126	151	148	154	171
rando99	117	315	261	256	122	223	254	126
random1	6	7	7	6	6	6	7	6
random2	2	2	2	2	2	2	2	2
random3	92	94	87	87	91	84	47	78
random4	5	5	5	5	5	5	5	5
random6	1046	4773	4773	5059	1583	1871	5676	1481
random7	4	4	4	4	4	4	4	4
random8	9	9	9	9	9	9	9	9
random9	4	4	4	4	4	4	4	4
relcour	1	1	1	1	1	1	1	1

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rstree1	3	3	3	3	3	3	3	3
rstree2	3	3	3	3	3	3	3	3
rstree3	9	6	6	8	9	8	8	8
rstree4	4	4	4	4	4	4	4	4
rstree5	2	2	2	2	2	2	2	2
rstree6	3	3	3	3	3	3	3	3
rstree7	13	11	11	10	12	10	12	11
trials1	87	98	98	83	88	75	84	94
trials2	10	11	11	9	10	10	11	12
trials3	2	2	2	2	2	2	2	2
trials4	115	162	140	172	101	108	160	99
usatree	1	1	1	1	1	1	1	1
worrell	16	15	15	15	15	13	14	15

Appendix VIII

Number of If-Then-Else Calculations Required to Construct BDDs from Fault Trees Restructured Using the Faunet Reduction Method

Key to ordering schemes[1]:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
aaaaaaa	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
artqual	10	**9**	10	10	10	10	10	**9**
arttree	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
astolfo	**19**	26	26	20	**19**	21	26	23
bddtest	27	31	31	31	27	**26**	31	27
benjiam	76	75	75	70	76	**67**	75	101
bpfeg03	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
bpfen05	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
bpfig05	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
bpfin05	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
bpfpp02	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
bpfsw02	27	**18**	22	22	27	22	**18**	24
ch8tree	16	**14**	**14**	**14**	16	17	17	**14**
dre1019	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
dre1032	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
dre1057	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
dre1058	23	25	25	25	23	**21**	25	23
dre1059	90	134	131	131	75	65	134	**62**
dresden	273	230	230	230	271	**142**	255	258
emerh2o	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

[1] For each fault tree, the ordering scheme(s) requiring the fewest ite calculations to construct the BDD is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
fatram2	11	11	11	11	11	11	11	**10**
hpisf02	107	**47**	**47**	60	96	60	63	113
hpisf03	19	**17**	**17**	**17**	19	**17**	**17**	19
hpisf21	58	**50**	**50**	65	54	52	56	53
hpisf36	19	**17**	**17**	**17**	19	**17**	**17**	19
jdtree1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
jdtree2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
jdtree3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
jdtree4	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
jdtree5	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
khictre	31	**24**	28	26	31	26	28	**24**
lisa123	146	130	130	115	156	**100**	146	131
lisab10	1095	1012	847	654	629	647	943	**562**
lisab25	204	**152**	188	204	208	198	192	216
lisab28	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
lisab30	191	180	179	179	173	**165**	173	181
lisab31	1235	1605	1605	1182	1051	**890**	1483	950
lisab34	72	**61**	82	81	72	81	82	70
lisab35	418	418	438	329	363	403	418	**234**
lisab36	1058	1331	1209	1293	1063	1201	1095	**984**
lisab42	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
lisab44	160	158	149	142	161	**131**	158	141
lisab51	23	**21**	**21**	**21**	23	**21**	22	25
lisab52	943	1282	1282	1033	726	840	1087	**685**
lisab53	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
lisab54	36	**30**	33	33	34	33	32	31
lisab57	158	218	234	168	**141**	171	232	219
lisab59	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
lisab60	39	40	40	38	37	39	40	**36**
lisab78	228	170	204	181	196	179	180	**137**
lisab86	214	275	213	214	224	209	288	**195**
lisaba4	504	562	606	520	346	**327**	636	357
lisaba9	67	44	44	42	51	**41**	42	51
modtree	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
nakashi	207	**91**	118	113	179	107	**91**	151
newtre2	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
newtre3	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
newtree	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
rand100	184	188	188	186	**170**	172	176	**170**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand101	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rand102	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand103	74	66	74	72	72	69	**63**	70
rand104	272	290	292	292	222	232	**182**	223
rand105	55	52	53	53	54	53	**48**	50
rand106	553	921	892	2376	587	1107	3076	**546**
rand107	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rand108	365	566	498	490	333	365	544	**316**
rand109	296	**205**	214	273	280	274	308	315
rand110	**353**	522	471	466	444	452	374	448
rand111	365	422	341	**298**	354	355	302	300
rand112	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand113	67	67	67	67	**64**	**64**	65	**64**
rand114	15	15	15	15	**14**	**14**	15	**14**
rand115	257	270	270	260	244	**200**	209	217
rand116	509	434	434	778	473	707	751	**351**
rand117	**24**	27	26	26	**24**	**24**	27	**24**
rand118	129	**93**	100	108	121	109	101	103
rand119	51	**47**	**47**	**47**	50	**47**	55	**47**
rand120	155	**119**	125	147	149	147	163	164
rand121	164	153	153	**147**	157	152	198	170
rand122	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rand123	**30**	33	33	31	31	32	33	31
rand124	60	**56**	68	68	69	68	57	69
rand125	32	33	32	32	**30**	**30**	32	**30**
rand126	198	170	170	167	169	**165**	178	183
rand127	16	15	15	15	16	15	**14**	**14**
rand128	627	353	**352**	663	561	554	495	386
rand129	34	**32**	**32**	**32**	**32**	**32**	34	**32**
rand130	190	164	164	157	161	161	189	**156**
rand131	11	11	11	11	11	13	11	**10**
rand132	792	846	846	1608	**542**	957	1583	880
rand133	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rand134	1130	675	696	2292	652	698	1731	**634**
rand135	371	340	335	609	375	**312**	408	418
rand136	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rand137	52	51	51	51	54	50	54	**48**
rand138	**123**	124	**123**	**123**	124	124	124	124
rand139	262	**186**	220	421	273	439	239	304

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand140	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
rand141	**353**	522	471	466	444	452	374	448
rand142	2852	2270	2219	2682	2238	1974	2924	**1881**
rand143	124	115	128	124	125	121	**105**	123
rand144	541	1004	914	949	**533**	605	926	564
rand145	9	9	9	9	9	9	9	**8**
rand146	52	51	51	51	54	50	54	**48**
rand147	890	2888	2682	2519	652	**530**	963	648
rand148	30	**22**	32	32	32	32	32	32
rand149	**64**	92	92	91	73	91	106	79
rand150	**1258**	2236	1686	2308	1451	1817	1584	1270
rand151	33	32	32	32	32	32	32	**30**
rand152	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rand153	167	159	160	169	159	**156**	169	164
rand154	97	**54**	96	96	72	83	96	55
rand155	265	266	255	**168**	200	177	397	186
rand156	93	85	85	85	93	**79**	111	94
rand158	6699	6466	6707	7156	6837	6226	6144	**4741**
rando10	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rando11	17216	47508	47508	8621	13881	**7699**	19305	9301
rando12	563	442	442	445	496	**413**	560	449
rando13	1310	1252	1252	2153	**1167**	1519	2040	1339
rando14	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
rando15	30	30	30	30	27	28	30	**25**
rando16	1838	1574	1389	1709	1694	1584	**1339**	1671
rando17	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando18	12890	26332	25528	39037	10872	22645	36439	**9592**
rando19	2703	3493	3493	4923	1746	1976	6101	**1603**
rando20	2997	4875	4746	5695	**2930**	3109	14427	3191
rando21	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
rando22	3779	6137	5930	4483	2723	3341	8047	**2129**
rando23	254	291	291	289	192	223	**153**	177
rando24	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
rando25	74	68	68	68	73	71	89	**65**
rando26	**36**	**36**	**36**	**36**	**36**	37	**36**	**36**
rando27	2273	2588	2586	2248	2579	2123	**1964**	2289
rando28	212	215	206	205	**191**	206	212	200
rando29	416	**303**	**303**	466	393	318	362	349
rando30	56	54	**51**	**51**	56	55	64	59

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando31	1538	**1441**	1511	2105	1514	1571	1464	1686
rando32	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**
rando33	175	162	161	209	168	**158**	212	174
rando34	346	308	365	370	313	322	332	**268**
rando35	377	558	443	443	325	328	**307**	310
rando36	99	99	99	100	93	100	**78**	92
rando37	736	768	777	853	605	718	**444**	633
rando38	78	78	78	78	78	78	**65**	71
rando39	386	**339**	**339**	796	350	601	677	442
rando40	28	27	**25**	26	30	26	29	28
rando41	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
rando42	**35**	36	39	39	36	37	36	39
rando43	**33**	36	36	36	**33**	**33**	38	**33**
rando44	316	177	**172**	290	332	234	294	315
rando45	246	238	238	299	228	**209**	245	216
rando46	554	497	497	513	446	457	495	**418**
rando47	**178**	227	227	295	187	185	294	202
rando48	123	113	112	112	112	**107**	129	108
rando49	55	54	54	54	55	54	55	**42**
rando50	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
rando51	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rando52	1269	1389	1350	1353	1194	1314	1370	**1144**
rando53	98	**83**	107	106	104	104	**83**	106
rando54	53	**47**	52	52	53	54	52	48
rando55	168	168	**166**	190	170	183	**166**	170
rando56	**29**	**29**	**29**	**29**	**29**	**29**	**29**	34
rando57	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**
rando58	89	89	89	89	80	**79**	89	**79**
rando59	588	354	348	**300**	597	324	361	363
rando60	212	225	225	256	201	213	**197**	217
rando61	204	197	197	280	188	183	276	**180**
rando62	95	**83**	85	102	95	102	88	95
rando63	168	168	**166**	190	170	183	**166**	170
rando64	280	243	231	221	247	**217**	227	236
rando65	63	54	54	71	59	55	55	**53**
rando66	205	265	265	265	206	**198**	247	222
rando67	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
rando68	**32**	**32**	34	34	34	**32**	34	34
rando69	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando70	58	53	53	53	56	53	62	55
rando71	15	15	15	15	15	15	15	14
rando72	19	19	18	18	19	19	19	17
rando73	312	336	333	333	256	266	490	259
rando74	5	5	5	5	5	5	5	5
rando75	53	53	53	53	53	54	53	54
rando76	122	88	127	114	98	110	88	111
rando77	537	470	465	527	533	547	681	536
rando78	82	59	97	80	77	71	94	75
rando79	13	13	13	13	13	13	13	13
rando80	13	13	13	13	13	13	12	12
rando81	7	7	7	7	7	7	7	7
rando82	50	50	50	50	49	47	50	47
rando83	84	93	89	66	76	66	94	73
rando84	129	93	100	108	121	109	101	103
rando85	156	149	149	150	154	153	155	154
rando86	2	2	2	2	2	2	2	2
rando87	33	34	34	34	32	34	34	31
rando88	34	24	32	24	26	23	28	25
rando89	804	720	680	671	742	693	515	848
rando90	0	0	0	0	0	0	0	0
rando91	961	1578	1578	811	822	714	1770	724
rando92	6585	10211	11343	13415	4818	5856	3721	5590
rando93	245	213	213	265	220	224	186	197
rando94	2	2	2	2	2	2	2	2
rando95	100	69	91	91	88	77	91	84
rando96	4	4	4	4	4	4	4	4
rando97	7	7	7	7	7	7	7	7
rando98	456	404	404	404	423	466	476	494
rando99	425	1020	818	833	450	655	832	469
random1	20	20	20	21	20	21	20	20
random2	5	5	5	5	5	5	5	5
random3	445	513	475	469	431	435	245	423
random4	13	13	13	13	13	13	13	13
random6	2077	6949	6947	7829	2925	2947	8247	3338
random7	10	10	10	10	10	10	10	10
random8	49	49	49	49	50	50	49	50
random9	25	25	25	25	25	25	25	25
relcour	0	0	0	0	0	0	0	0

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rstree1	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rstree2	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rstree3	**10**	**10**	**10**	11	**10**	**10**	11	12
rstree4	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
rstree5	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
rstree6	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
rstree7	25	**19**	**19**	21	22	21	20	23
trials1	271	308	308	277	240	**215**	292	289
trials2	79	**72**	74	83	74	75	**72**	**72**
trials3	93	96	98	98	94	90	101	**88**
trials4	427	578	535	593	386	423	471	**382**
usatree	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
worrell	29	28	28	28	28	27	**24**	27

Appendix IX

Comparison of Analysis Times for the Fault Tree Strategy and a Direct BDD Analysis Technique

Key to ordering schemes:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Method	Times using ordering scheme							
		1	2	3	4	5	6	7	8
aaaaaaa	Direct	0.063	0.067	0.060	0.067	0.083	0.090	0.067	0.063
	Strategy	0.067	0.063	0.063	0.067	0.087	0.090	0.067	0.067
artqual	Direct	0.067	0.063	0.060	0.070	0.083	0.087	0.070	0.063
	Strategy	0.083	0.090	0.090	0.083	0.110	0.110	0.090	0.090
arttree	Direct	0.067	0.060	0.067	0.063	0.090	0.080	0.070	0.063
	Strategy	0.063	0.067	0.060	0.067	0.087	0.087	0.070	0.063
astolfo	Direct	0.067	0.070	0.063	0.070	0.090	0.097	0.070	0.070
	Strategy	0.090	0.090	0.090	0.083	0.113	0.110	0.090	0.093
bddtest	Direct	0.060	0.067	0.063	0.067	0.087	0.087	0.070	0.070
	Strategy	0.087	0.090	0.083	0.087	0.110	0.110	0.093	0.090
benjiam	Direct	0.063	0.067	0.067	0.067	0.087	0.090	0.070	0.073
	Strategy	0.090	0.090	0.090	0.090	0.110	0.110	0.090	0.100
bpfeg03	Direct	0.350	0.193	0.277	0.370	0.407	0.403	0.193	0.427
	Strategy	0.060	0.070	0.070	0.067	0.093	0.090	0.070	0.070
bpfen05	Direct	0.227	0.140	0.170	0.223	0.247	0.253	0.147	0.233
	Strategy	0.063	0.070	0.067	0.070	0.090	0.090	0.070	0.070
bpfig05	Direct	0.217	0.140	0.167	0.213	0.240	0.240	0.143	0.223
	Strategy	0.067	0.070	0.067	0.067	0.090	0.093	0.067	0.070
bpfin05	Direct	0.077	0.083	0.077	0.077	0.100	0.103	0.083	0.083
	Strategy	0.067	0.067	0.063	0.067	0.090	0.090	0.063	0.070
bpfpp02	Direct	0.060	0.067	0.063	0.070	0.087	0.087	0.067	0.067
	Strategy	0.067	0.063	0.067	0.063	0.090	0.087	0.063	0.070
bpfsw02	Direct	0.180	0.170	0.167	0.167	0.203	0.207	0.170	0.187
	Strategy	0.110	0.113	0.110	0.110	0.160	0.160	0.117	0.117

Tree	Method	1	2	3	4	5	6	7	8
ch8tree	Direct	0.063	0.063	0.063	0.063	0.090	0.087	0.063	0.067
	Strategy	0.087	0.087	0.087	0.087	0.110	0.110	0.087	0.090
dre1019	Direct	0.067	0.063	0.067	0.063	0.090	0.090	0.067	0.067
	Strategy	0.063	0.067	0.063	0.067	0.090	0.083	0.067	0.070
dre1032	Direct	0.073	0.063	0.067	0.067	0.087	0.090	0.067	0.073
	Strategy	0.060	0.070	0.060	0.070	0.087	0.090	0.067	0.070
dre1057	Direct	0.070	0.070	0.070	0.070	0.090	0.093	0.077	0.070
	Strategy	0.063	0.067	0.067	0.067	0.087	0.090	0.070	0.063
dre1058	Direct	0.120	0.173	0.113	0.107	0.153	0.133	0.130	0.110
	Strategy	0.133	0.130	0.130	0.137	0.203	0.210	0.140	0.140
dre1059	Direct	0.163	0.320	0.327	0.330	0.203	0.170	0.300	0.153
	Strategy	0.090	0.103	0.093	0.097	0.120	0.110	0.100	0.100
dresden	Direct	1.143	1.753	0.383	0.340	1.473	0.417	3.753	0.877
	Strategy	0.120	0.113	0.113	0.100	0.147	0.123	0.120	0.120
emerh2o	Direct	0.063	0.063	0.067	0.063	0.087	0.090	0.063	0.067
	Strategy	0.067	0.063	0.067	0.063	0.090	0.087	0.067	0.067
fatram2	Direct	0.063	0.067	0.063	0.063	0.090	0.090	0.070	0.063
	Strategy	0.130	0.127	0.127	0.133	0.210	0.200	0.140	0.133
hpisf02	Direct	0.437	0.250	0.250	0.220	0.500	0.413	0.610	1.463
	Strategy	0.097	0.093	0.093	0.097	0.113	0.120	0.093	0.100
hpisf03	Direct	0.070	0.070	0.067	0.070	0.093	0.097	0.070	0.073
	Strategy	0.087	0.090	0.087	0.083	0.110	0.113	0.083	0.093
hpisf21	Direct	0.357	0.243	0.247	0.403	0.563	0.460	0.377	3.873
	Strategy	0.093	0.090	0.093	0.093	0.117	0.113	0.097	0.093
hpisf36	Direct	0.083	0.077	0.070	0.070	0.097	0.097	0.077	0.077
	Strategy	0.090	0.087	0.087	0.090	0.113	0.110	0.090	0.093
jdtree1	Direct	0.067	0.063	0.060	0.070	0.083	0.087	0.070	0.063
	Strategy	0.063	0.063	0.063	0.063	0.090	0.087	0.070	0.063
jdtree2	Direct	0.067	0.060	0.063	0.067	0.090	0.080	0.070	0.067
	Strategy	0.067	0.060	0.063	0.067	0.090	0.087	0.063	0.070
jdtree3	Direct	0.063	0.070	0.060	0.070	0.087	0.087	0.067	0.070
	Strategy	0.063	0.070	0.060	0.070	0.090	0.080	0.070	0.070
jdtree4	Direct	0.070	0.063	0.070	0.060	0.090	0.093	0.070	0.070
	Strategy	0.090	0.083	0.090	0.090	0.110	0.110	0.090	0.090
jdtree5	Direct	0.060	0.070	0.060	0.070	0.090	0.090	0.060	0.070
	Strategy	0.083	0.090	0.087	0.083	0.110	0.110	0.090	0.090
khictre	Direct	0.083	0.077	0.090	0.080	0.113	0.100	0.090	0.080
	Strategy	0.133	0.130	0.130	0.140	0.207	0.207	0.140	0.140
lisa123	Direct	0.073	0.077	0.073	0.080	0.100	0.093	0.080	0.097
	Strategy	0.100	0.093	0.100	0.093	0.120	0.117	0.097	0.097

Tree	Method	1	2	3	4	5	6	7	8
lisab10	Direct	0.610	0.517	0.380	0.250	0.260	0.273	0.493	0.197
	Strategy	0.583	0.520	0.380	0.260	0.267	0.273	0.457	0.213
lisab25	Direct	0.080	0.080	0.083	0.080	0.100	0.103	0.083	0.080
	Strategy	0.103	0.097	0.103	0.100	0.130	0.127	0.103	0.110
lisab28	Direct	0.063	0.067	0.067	0.067	0.090	0.090	0.063	0.070
	Strategy	0.067	0.063	0.067	0.063	0.087	0.090	0.067	0.070
lisab30	Direct	0.087	0.080	0.080	0.083	0.103	0.107	0.083	0.087
	Strategy	0.100	0.100	0.100	0.097	0.123	0.120	0.103	0.103
lisab31	Direct	0.320	0.850	0.857	0.463	0.320	0.300	0.710	0.250
	Strategy	0.633	1.313	1.313	0.653	0.520	0.423	1.110	0.427
lisab34	Direct	0.067	0.063	0.067	0.063	0.090	0.090	0.070	0.067
	Strategy	0.090	0.090	0.087	0.093	0.110	0.113	0.090	0.093
lisab35	Direct	0.290	0.200	0.190	0.163	0.273	0.247	0.240	0.187
	Strategy	0.150	0.147	0.150	0.150	0.153	0.170	0.147	0.113
lisab36	Direct	0.470	0.687	0.563	0.770	0.553	0.713	0.497	0.470
	Strategy	0.497	0.797	0.667	0.743	0.527	0.663	0.547	0.453
lisab42	Direct	0.067	0.063	0.067	0.067	0.087	0.090	0.067	0.067
	Strategy	0.177	0.170	0.170	0.173	0.300	0.300	0.183	0.187
lisab44	Direct	0.070	0.070	0.067	0.067	0.093	0.093	0.070	0.070
	Strategy	0.100	0.093	0.100	0.093	0.120	0.120	0.100	0.097
lisab51	Direct	0.067	0.063	0.070	0.067	0.087	0.090	0.070	0.067
	Strategy	0.087	0.090	0.083	0.087	0.110	0.110	0.090	0.090
lisab52	Direct	0.373	0.843	0.847	0.453	0.280	0.340	0.683	0.213
	Strategy	0.403	0.860	0.853	0.500	0.297	0.390	0.630	0.253
lisab53	Direct	0.067	0.063	0.067	0.063	0.087	0.090	0.063	0.067
	Strategy	0.067	0.063	0.063	0.067	0.087	0.093	0.063	0.070
lisab54	Direct	0.067	0.067	0.063	0.067	0.090	0.087	0.070	0.067
	Strategy	0.087	0.087	0.087	0.087	0.110	0.110	0.090	0.090
lisab57	Direct	0.077	0.077	0.080	0.077	0.100	0.103	0.097	0.097
	Strategy	0.100	0.103	0.107	0.103	0.123	0.123	0.110	0.110
lisab59	Direct	0.147	0.107	0.110	0.163	0.167	0.203	0.113	0.200
	Strategy	0.070	0.063	0.067	0.067	0.087	0.090	0.070	0.067
lisab60	Direct	0.067	0.063	0.070	0.063	0.087	0.093	0.070	0.067
	Strategy	0.090	0.087	0.090	0.087	0.110	0.113	0.087	0.090
lisab78	Direct	0.150	0.093	0.107	0.093	0.127	0.113	0.110	0.100
	Strategy	0.110	0.100	0.110	0.100	0.127	0.123	0.107	0.100
lisab86	Direct	0.107	0.130	0.110	0.100	0.130	0.127	0.147	0.103
	Strategy	0.103	0.123	0.110	0.107	0.133	0.130	0.123	0.107
lisaba4	Direct	0.267	0.293	0.293	0.240	0.167	0.183	0.137	0.153
	Strategy	0.203	0.223	0.240	0.193	0.167	0.160	0.263	0.147

Tree	Method	1	2	3	4	5	6	7	8
lisaba9	Direct	0.093	0.080	0.080	0.077	0.107	0.100	0.080	0.090
	Strategy	0.133	0.130	0.130	0.133	0.207	0.203	0.140	0.143
modtree	Direct	0.070	0.060	0.060	0.070	0.083	0.087	0.073	0.060
	Strategy	0.087	0.087	0.090	0.083	0.110	0.110	0.090	0.090
nakashi	Direct	0.093	0.070	0.070	0.070	0.100	0.090	0.070	0.080
	Strategy	0.103	0.090	0.097	0.090	0.123	0.117	0.093	0.097
newtre2	Direct	0.060	0.060	0.070	0.060	0.090	0.090	0.060	0.070
	Strategy	0.090	0.083	0.087	0.087	0.110	0.110	0.090	0.087
newtre3	Direct	0.060	0.070	0.063	0.060	0.090	0.090	0.060	0.070
	Strategy	0.110	0.110	0.107	0.110	0.153	0.157	0.117	0.110
newtree	Direct	0.060	0.070	0.060	0.070	0.087	0.090	0.067	0.063
	Strategy	0.067	0.067	0.067	0.063	0.087	0.090	0.067	0.063
rand100	Direct	0.077	0.073	0.080	0.080	0.097	0.100	0.080	0.080
	Strategy	0.100	0.100	0.100	0.100	0.120	0.127	0.100	0.103
rand101	Direct	0.063	0.060	0.070	0.063	0.090	0.087	0.067	0.067
	Strategy	0.087	0.083	0.087	0.083	0.117	0.117	0.087	0.090
rand102	Direct	0.063	0.067	0.063	0.063	0.087	0.087	0.067	0.070
	Strategy	0.087	0.083	0.087	0.087	0.110	0.110	0.090	0.087
rand103	Direct	0.070	0.067	0.070	0.070	0.093	0.097	0.070	0.073
	Strategy	0.110	0.110	0.110	0.113	0.160	0.160	0.113	0.117
rand104	Direct	0.090	0.100	0.107	0.103	0.110	0.113	0.077	0.093
	Strategy	0.113	0.117	0.120	0.120	0.127	0.133	0.100	0.110
rand105	Direct	0.077	0.073	0.073	0.077	0.093	0.097	0.073	0.080
	Strategy	0.097	0.083	0.090	0.090	0.113	0.117	0.090	0.090
rand106	Direct	0.227	1.710	1.730	0.300	0.220	0.210	0.407	0.170
	Strategy	0.190	0.410	0.390	2.103	0.230	0.547	3.663	0.207
rand107	Direct	0.067	0.063	0.067	0.060	0.090	0.083	0.067	0.070
	Strategy	0.087	0.087	0.087	0.087	0.110	0.107	0.090	0.090
rand108	Direct	0.103	0.190	0.180	0.150	0.130	0.153	0.190	0.107
	Strategy	0.140	0.210	0.180	0.170	0.150	0.157	0.203	0.130
rand109	Direct	0.173	0.120	0.120	0.143	0.197	0.167	0.173	0.230
	Strategy	0.120	0.110	0.110	0.113	0.143	0.143	0.127	0.127
rand110	Direct	0.117	0.173	0.150	0.150	0.153	0.163	0.117	0.140
	Strategy	0.137	0.190	0.170	0.170	0.183	0.187	0.140	0.163
rand111	Direct	0.103	0.117	0.100	0.093	0.127	0.113	0.097	0.097
	Strategy	0.137	0.157	0.133	0.120	0.163	0.160	0.120	0.123
rand112	Direct	0.067	0.063	0.063	0.067	0.090	0.083	0.067	0.067
	Strategy	0.087	0.087	0.083	0.090	0.110	0.110	0.087	0.087
rand113	Direct	0.067	0.070	0.067	0.067	0.090	0.087	0.067	0.070
	Strategy	0.087	0.090	0.090	0.090	0.110	0.117	0.090	0.093

Tree	Method	1	2	3	4	5	6	7	8
rand114	Direct	0.063	0.067	0.060	0.067	0.087	0.090	0.067	0.067
	Strategy	0.080	0.090	0.087	0.083	0.110	0.110	0.097	0.090
rand115	Direct	0.103	0.100	0.097	0.103	0.120	0.110	0.087	0.090
	Strategy	0.110	0.123	0.113	0.117	0.130	0.130	0.103	0.110
rand116	Direct	0.157	0.110	0.130	0.320	0.167	0.303	0.303	0.110
	Strategy	0.177	0.150	0.147	0.333	0.183	0.310	0.317	0.133
rand117	Direct	0.070	0.063	0.067	0.067	0.087	0.090	0.067	0.070
	Strategy	0.087	0.083	0.090	0.090	0.110	0.110	0.090	0.090
rand118	Direct	0.093	0.077	0.077	0.077	0.107	0.100	0.080	0.080
	Strategy	0.093	0.093	0.100	0.090	0.120	0.113	0.100	0.093
rand119	Direct	0.070	0.073	0.070	0.070	0.097	0.093	0.077	0.070
	Strategy	0.110	0.107	0.110	0.110	0.160	0.160	0.117	0.113
rand120	Direct	0.163	0.090	0.097	0.100	0.180	0.130	0.107	0.153
	Strategy	0.100	0.090	0.100	0.093	0.120	0.120	0.103	0.103
rand121	Direct	0.083	0.080	0.083	0.080	0.100	0.103	0.097	0.083
	Strategy	0.097	0.093	0.103	0.097	0.120	0.123	0.103	0.103
rand122	Direct	0.067	0.060	0.063	0.067	0.087	0.087	0.070	0.063
	Strategy	0.090	0.083	0.090	0.087	0.110	0.110	0.083	0.090
rand123	Direct	0.063	0.070	0.067	0.067	0.090	0.087	0.067	0.073
	Strategy	0.090	0.087	0.083	0.090	0.110	0.110	0.090	0.090
rand124	Direct	0.067	0.070	0.070	0.070	0.090	0.090	0.073	0.070
	Strategy	0.090	0.090	0.090	0.090	0.110	0.110	0.097	0.090
rand125	Direct	0.067	0.067	0.067	0.063	0.090	0.090	0.067	0.070
	Strategy	0.090	0.090	0.083	0.087	0.110	0.113	0.090	0.090
rand126	Direct	0.093	0.090	0.090	0.090	0.120	0.117	0.090	0.107
	Strategy	0.103	0.100	0.100	0.100	0.120	0.123	0.100	0.103
rand127	Direct	0.070	0.067	0.063	0.070	0.093	0.090	0.070	0.070
	Strategy	0.130	0.130	0.130	0.130	0.207	0.203	0.140	0.140
rand128	Direct	0.227	0.140	0.133	0.220	0.240	0.290	0.210	0.140
	Strategy	0.247	0.130	0.130	0.270	0.223	0.240	0.170	0.150
rand129	Direct	0.067	0.063	0.067	0.063	0.090	0.090	0.070	0.070
	Strategy	0.093	0.090	0.083	0.087	0.113	0.110	0.090	0.090
rand130	Direct	0.077	0.073	0.070	0.077	0.093	0.100	0.080	0.080
	Strategy	0.097	0.100	0.097	0.093	0.120	0.120	0.100	0.100
rand131	Direct	0.067	0.067	0.063	0.060	0.090	0.090	0.060	0.070
	Strategy	0.083	0.087	0.090	0.090	0.107	0.110	0.090	0.090
rand132	Direct	0.257	0.330	0.330	0.823	0.210	0.560	0.790	0.307
	Strategy	0.293	0.370	0.370	0.947	0.213	0.413	0.910	0.323
rand133	Direct	0.063	0.063	0.063	0.067	0.087	0.087	0.067	0.067
	Strategy	0.087	0.087	0.083	0.087	0.107	0.110	0.090	0.090

Tree	Method	1	2	3	4	5	6	7	8
rand134	Direct	0.167	0.250	0.263	0.947	0.187	0.490	0.603	0.163
	Strategy	0.560	0.240	0.260	2.140	0.250	0.280	1.223	0.220
rand135	Direct	0.097	0.103	0.097	0.163	0.123	0.113	0.123	0.110
	Strategy	0.133	0.130	0.130	0.233	0.160	0.147	0.153	0.157
rand136	Direct	0.063	0.060	0.067	0.063	0.087	0.087	0.070	0.067
	Strategy	0.087	0.090	0.083	0.087	0.110	0.110	0.087	0.087
rand137	Direct	0.063	0.070	0.063	0.070	0.090	0.090	0.070	0.070
	Strategy	0.090	0.087	0.090	0.090	0.113	0.117	0.090	0.090
rand138	Direct	0.073	0.067	0.067	0.070	0.090	0.093	0.070	0.070
	Strategy	0.090	0.093	0.097	0.090	0.120	0.113	0.097	0.093
rand139	Direct	0.103	0.083	0.093	0.167	0.127	0.193	0.093	0.113
	Strategy	0.120	0.100	0.110	0.153	0.140	0.180	0.120	0.120
rand140	Direct	0.067	0.063	0.063	0.063	0.087	0.090	0.063	0.067
	Strategy	0.090	0.080	0.090	0.087	0.110	0.110	0.087	0.093
rand141	Direct	0.117	0.177	0.150	0.150	0.160	0.163	0.117	0.137
	Strategy	0.133	0.193	0.177	0.170	0.187	0.183	0.137	0.170
rand142	Direct	3.777	3.043	3.053	3.793	4.087	2.100	4.553	1.600
	Strategy	3.657	2.310	2.173	3.160	2.300	1.890	3.670	1.620
rand143	Direct	0.073	0.070	0.077	0.073	0.090	0.103	0.070	0.080
	Strategy	0.100	0.090	0.093	0.097	0.117	0.120	0.093	0.097
rand144	Direct	0.187	0.580	0.483	0.417	0.180	0.240	0.640	0.190
	Strategy	0.207	0.543	0.463	0.437	0.223	0.270	0.430	0.230
rand145	Direct	0.070	0.070	0.070	0.070	0.090	0.090	0.070	0.073
	Strategy	0.110	0.107	0.107	0.110	0.160	0.157	0.113	0.117
rand146	Direct	0.067	0.070	0.063	0.067	0.090	0.093	0.070	0.070
	Strategy	0.087	0.087	0.093	0.087	0.113	0.113	0.093	0.090
rand147	Direct	0.193	3.073	3.733	0.233	0.217	0.550	0.420	0.210
	Strategy	0.373	3.500	2.947	2.530	0.263	0.213	0.397	0.243
rand148	Direct	0.070	0.070	0.060	0.067	0.090	0.090	0.070	0.070
	Strategy	0.090	0.083	0.090	0.087	0.110	0.113	0.087	0.090
rand149	Direct	0.090	0.087	0.087	0.087	0.110	0.107	0.090	0.097
	Strategy	0.093	0.097	0.093	0.090	0.117	0.117	0.100	0.090
rand150	Direct	0.560	2.143	1.267	1.493	0.697	0.733	1.023	0.650
	Strategy	0.657	2.020	1.127	2.020	0.857	1.403	0.967	0.663
rand151	Direct	0.070	0.067	0.070	0.070	0.090	0.090	0.070	0.070
	Strategy	0.090	0.083	0.090	0.087	0.113	0.110	0.090	0.090
rand152	Direct	0.063	0.067	0.060	0.063	0.090	0.087	0.063	0.070
	Strategy	0.087	0.087	0.087	0.090	0.107	0.110	0.090	0.090
rand153	Direct	0.073	0.077	0.073	0.077	0.097	0.100	0.077	0.080
	Strategy	0.093	0.097	0.100	0.093	0.120	0.123	0.100	0.103

Tree	Method	1	2	3	4	5	6	7	8
rand154	Direct	0.073	0.063	0.070	0.070	0.090	0.093	0.073	0.070
	Strategy	0.090	0.087	0.090	0.090	0.117	0.110	0.090	0.093
rand155	Direct	0.100	0.103	0.107	0.083	0.110	0.100	0.157	0.090
	Strategy	0.117	0.120	0.117	0.103	0.123	0.127	0.163	0.103
rand156	Direct	0.073	0.070	0.067	0.070	0.093	0.090	0.080	0.077
	Strategy	0.090	0.087	0.090	0.090	0.113	0.113	0.093	0.093
rand158	Direct	66.047	27.763	27.747	28.287	36.320	21.090	25.050	27.840
	Strategy	22.430	20.013	21.520	23.117	22.000	18.113	18.663	9.360
rando10	Direct	0.067	0.060	0.067	0.063	0.093	0.087	0.063	0.067
	Strategy	0.087	0.087	0.090	0.087	0.110	0.110	0.083	0.090
rando11	Direct	629.157	3125.633	3145.013	174.117	306.983	234.953	940.133	108.740
	Strategy	143.327	1624.337	1625.660	59.173	103.027	48.087	325.597	46.693
rando12	Direct	0.273	0.197	0.197	0.217	0.203	0.213	0.273	0.177
	Strategy	0.200	0.160	0.157	0.170	0.200	0.180	0.210	0.173
rando13	Direct	0.417	0.517	0.517	1.097	0.357	0.590	1.220	0.370
	Strategy	0.727	0.743	0.747	1.980	0.603	0.957	1.773	0.783
rando14	Direct	0.067	0.070	0.063	0.067	0.087	0.083	0.070	0.067
	Strategy	0.087	0.093	0.083	0.087	0.110	0.110	0.087	0.090
rando15	Direct	0.063	0.060	0.067	0.063	0.090	0.087	0.067	0.067
	Strategy	0.087	0.087	0.083	0.087	0.113	0.110	0.093	0.090
rando16	Direct	1.230	1.230	0.860	1.347	1.320	1.173	0.807	1.337
	Strategy	1.227	0.997	0.740	1.240	1.120	1.033	0.703	1.090
rando17	Direct	0.070	0.063	0.067	0.060	0.090	0.087	0.063	0.070
	Strategy	0.083	0.090	0.083	0.087	0.110	0.110	0.090	0.090
rando18	Direct	68.500	333.773	320.223	785.390	42.000	241.683	672.627	44.570
	Strategy	68.570	324.997	308.940	784.617	51.273	241.943	686.420	41.037
rando19	Direct	1.260	4.960	4.647	5.827	0.847	1.170	30.990	0.843
	Strategy	3.047	5.387	5.370	7.413	1.337	1.753	18.933	1.103
rando20	Direct	2.850	3.770	3.140	3.863	2.370	2.953	17.447	2.270
	Strategy	3.843	10.467	9.880	14.373	3.597	4.053	93.960	4.103
rando21	Direct	0.063	0.070	0.060	0.070	0.080	0.090	0.070	0.063
	Strategy	0.067	0.063	0.063	0.067	0.087	0.087	0.067	0.063
rando22	Direct	5.697	16.223	16.257	8.810	2.913	4.743	30.040	2.127
	Strategy	6.057	16.177	15.113	9.023	3.063	5.227	33.590	1.957
rando23	Direct	0.097	0.103	0.100	0.097	0.113	0.113	0.100	0.090
	Strategy	0.107	0.113	0.113	0.113	0.120	0.130	0.100	0.100
rando24	Direct	0.067	0.063	0.063	0.063	0.087	0.090	0.063	0.067
	Strategy	0.087	0.087	0.087	0.083	0.110	0.110	0.090	0.087
rando25	Direct	0.067	0.067	0.067	0.070	0.090	0.090	0.070	0.070
	Strategy	0.090	0.087	0.090	0.093	0.113	0.117	0.090	0.093

Tree	Method	1	2	3	4	5	6	7	8
rando26	Direct	0.063	0.067	0.063	0.063	0.087	0.090	0.070	0.063
	Strategy	0.090	0.090	0.083	0.087	0.110	0.110	0.090	0.090
rando27	Direct	1.913	2.510	2.523	2.147	2.540	1.710	1.437	1.930
	Strategy	1.920	2.520	2.520	2.167	2.550	1.727	1.453	1.947
rando28	Direct	0.103	0.090	0.080	0.090	0.110	0.110	0.087	0.093
	Strategy	0.110	0.107	0.103	0.107	0.127	0.130	0.107	0.107
rando29	Direct	0.123	0.177	0.127	0.130	0.147	0.133	0.120	0.117
	Strategy	0.143	0.117	0.117	0.160	0.163	0.143	0.133	0.133
rando30	Direct	0.130	0.080	0.090	0.087	0.147	0.127	0.083	0.117
	Strategy	0.090	0.090	0.090	0.090	0.113	0.117	0.090	0.093
rando31	Direct	0.930	0.823	0.880	2.130	0.910	0.950	0.807	1.153
	Strategy	0.940	0.887	0.957	1.973	0.960	1.013	0.890	1.193
rando32	Direct	0.060	0.067	0.063	0.063	0.087	0.093	0.067	0.063
	Strategy	0.087	0.090	0.087	0.083	0.110	0.110	0.090	0.090
rando33	Direct	0.083	0.073	0.077	0.080	0.103	0.100	0.080	0.080
	Strategy	0.100	0.100	0.100	0.107	0.123	0.120	0.110	0.103
rando34	Direct	0.120	0.117	0.130	0.133	0.137	0.143	0.110	0.107
	Strategy	0.133	0.127	0.140	0.137	0.147	0.153	0.130	0.117
rando35	Direct	0.103	0.130	0.120	0.117	0.113	0.117	0.093	0.090
	Strategy	0.133	0.207	0.147	0.153	0.147	0.147	0.120	0.123
rando36	Direct	0.073	0.073	0.077	0.073	0.097	0.093	0.080	0.077
	Strategy	0.093	0.090	0.093	0.093	0.113	0.117	0.090	0.093
rando37	Direct	0.110	0.173	0.170	0.187	0.133	0.200	0.120	0.130
	Strategy	0.280	0.330	0.327	0.400	0.243	0.310	0.163	0.240
rando38	Direct	0.070	0.070	0.070	0.067	0.093	0.090	0.070	0.070
	Strategy	0.090	0.087	0.087	0.093	0.113	0.110	0.093	0.090
rando39	Direct	0.123	0.110	0.120	0.350	0.130	0.247	0.277	0.140
	Strategy	0.150	0.133	0.130	0.380	0.157	0.273	0.300	0.167
rando40	Direct	0.070	0.060	0.073	0.060	0.090	0.090	0.070	0.063
	Strategy	0.110	0.110	0.103	0.113	0.157	0.160	0.113	0.117
rando41	Direct	0.067	0.060	0.070	0.060	0.090	0.083	0.067	0.070
	Strategy	0.090	0.087	0.090	0.087	0.110	0.107	0.087	0.090
rando42	Direct	0.060	0.073	0.060	0.070	0.090	0.103	0.063	0.067
	Strategy	0.090	0.087	0.087	0.087	0.113	0.110	0.090	0.090
rando43	Direct	0.070	0.070	0.070	0.073	0.090	0.090	0.070	0.077
	Strategy	0.090	0.087	0.087	0.090	0.110	0.110	0.090	0.090
rando44	Direct	0.813	0.197	0.187	0.353	0.433	0.283	0.427	0.450
	Strategy	0.133	0.107	0.100	0.123	0.150	0.137	0.123	0.133
rando45	Direct	0.093	0.083	0.087	0.093	0.110	0.103	0.087	0.090
	Strategy	0.107	0.110	0.103	0.120	0.130	0.127	0.110	0.110

Tree	Method	1	2	3	4	5	6	7	8
rando46	Direct	0.247	0.200	0.200	0.157	0.190	0.183	0.137	0.163
	Strategy	0.203	0.177	0.180	0.187	0.180	0.190	0.180	0.153
rando47	Direct	0.087	0.100	0.100	0.090	0.100	0.107	0.090	0.087
	Strategy	0.107	0.110	0.113	0.120	0.123	0.133	0.120	0.110
rando48	Direct	0.073	0.073	0.070	0.073	0.093	0.100	0.070	0.077
	Strategy	0.097	0.093	0.090	0.097	0.113	0.120	0.093	0.097
rando49	Direct	0.067	0.067	0.067	0.063	0.090	0.090	0.067	0.070
	Strategy	0.087	0.090	0.083	0.090	0.110	0.117	0.090	0.090
rando50	Direct	0.063	0.063	0.067	0.063	0.087	0.090	0.063	0.067
	Strategy	0.083	0.087	0.087	0.087	0.110	0.113	0.090	0.087
rando51	Direct	0.063	0.067	0.063	0.063	0.087	0.087	0.067	0.067
	Strategy	0.090	0.087	0.087	0.087	0.113	0.107	0.090	0.087
rando52	Direct	0.560	0.537	0.567	0.630	0.550	0.600	0.680	0.417
	Strategy	0.690	0.903	0.767	0.770	0.637	0.750	0.820	0.580
rando53	Direct	0.070	0.070	0.070	0.070	0.090	0.093	0.070	0.073
	Strategy	0.090	0.090	0.090	0.093	0.120	0.113	0.093	0.093
rando54	Direct	0.080	0.070	0.073	0.077	0.093	0.097	0.080	0.077
	Strategy	0.133	0.130	0.133	0.133	0.210	0.207	0.140	0.140
rando55	Direct	0.073	0.080	0.073	0.080	0.103	0.100	0.080	0.077
	Strategy	0.097	0.100	0.097	0.100	0.120	0.123	0.100	0.100
rando56	Direct	0.070	0.060	0.067	0.063	0.090	0.090	0.063	0.070
	Strategy	0.090	0.087	0.083	0.090	0.110	0.110	0.087	0.093
rando57	Direct	0.060	0.070	0.063	0.063	0.090	0.087	0.070	0.063
	Strategy	0.090	0.087	0.087	0.090	0.110	0.110	0.087	0.087
rando58	Direct	0.067	0.070	0.070	0.063	0.097	0.090	0.070	0.070
	Strategy	0.090	0.097	0.090	0.090	0.110	0.113	0.090	0.093
rando59	Direct	0.343	0.190	0.180	0.160	0.293	0.223	0.130	0.187
	Strategy	0.227	0.140	0.137	0.130	0.250	0.160	0.140	0.140
rando60	Direct	0.110	0.113	0.127	0.113	0.133	0.133	0.133	0.110
	Strategy	0.107	0.117	0.113	0.117	0.133	0.137	0.107	0.117
rando61	Direct	0.080	0.073	0.077	0.083	0.100	0.100	0.090	0.077
	Strategy	0.103	0.103	0.100	0.120	0.127	0.127	0.123	0.103
rando62	Direct	0.070	0.067	0.070	0.070	0.093	0.097	0.070	0.073
	Strategy	0.090	0.090	0.090	0.093	0.117	0.110	0.093	0.097
rando63	Direct	0.070	0.080	0.080	0.080	0.100	0.100	0.077	0.080
	Strategy	0.097	0.093	0.100	0.103	0.120	0.123	0.100	0.107
rando64	Direct	0.163	0.120	0.120	0.110	0.200	0.133	0.120	0.113
	Strategy	0.113	0.110	0.110	0.107	0.133	0.127	0.110	0.113
rando65	Direct	0.060	0.070	0.060	0.070	0.090	0.090	0.070	0.070
	Strategy	0.087	0.093	0.087	0.090	0.110	0.113	0.090	0.090

Tree	Method	1	2	3	4	5	6	7	8
rando66	Direct	0.080	0.083	0.080	0.090	0.110	0.100	0.090	0.090
	Strategy	0.103	0.107	0.107	0.113	0.123	0.127	0.107	0.110
rando67	Direct	0.060	0.060	0.070	0.060	0.090	0.093	0.060	0.070
	Strategy	0.093	0.083	0.087	0.087	0.110	0.110	0.083	0.090
rando68	Direct	0.060	0.073	0.060	0.067	0.083	0.090	0.070	0.060
	Strategy	0.090	0.087	0.083	0.090	0.110	0.113	0.090	0.090
rando69	Direct	0.070	0.060	0.067	0.063	0.090	0.087	0.063	0.073
	Strategy	0.087	0.083	0.087	0.087	0.110	0.110	0.090	0.090
rando70	Direct	0.067	0.063	0.073	0.070	0.090	0.090	0.070	0.070
	Strategy	0.087	0.087	0.093	0.087	0.117	0.113	0.090	0.093
rando71	Direct	0.060	0.070	0.060	0.067	0.083	0.090	0.067	0.063
	Strategy	0.083	0.087	0.083	0.083	0.110	0.110	0.090	0.090
rando72	Direct	0.070	0.063	0.060	0.070	0.087	0.083	0.070	0.067
	Strategy	0.087	0.083	0.093	0.087	0.110	0.110	0.083	0.090
rando73	Direct	0.103	0.107	0.103	0.110	0.120	0.117	0.167	0.093
	Strategy	0.130	0.130	0.130	0.137	0.140	0.140	0.193	0.127
rando74	Direct	0.067	0.063	0.060	0.067	0.087	0.087	0.070	0.060
	Strategy	0.083	0.087	0.087	0.087	0.110	0.110	0.087	0.090
rando75	Direct	0.070	0.063	0.067	0.067	0.087	0.093	0.067	0.070
	Strategy	0.130	0.130	0.130	0.133	0.207	0.203	0.140	0.143
rando76	Direct	0.080	0.077	0.077	0.077	0.097	0.100	0.087	0.080
	Strategy	0.090	0.090	0.097	0.093	0.117	0.113	0.090	0.100
rando77	Direct	0.157	0.143	0.143	0.167	0.183	0.200	0.250	0.170
	Strategy	0.187	0.167	0.173	0.190	0.207	0.220	0.267	0.197
rando78	Direct	0.070	0.070	0.067	0.070	0.093	0.097	0.070	0.073
	Strategy	0.090	0.090	0.090	0.090	0.113	0.117	0.090	0.093
rando79	Direct	0.063	0.067	0.063	0.067	0.090	0.087	0.067	0.067
	Strategy	0.087	0.090	0.080	0.093	0.110	0.110	0.090	0.090
rando80	Direct	0.067	0.067	0.063	0.067	0.090	0.090	0.067	0.073
	Strategy	0.107	0.110	0.103	0.113	0.160	0.163	0.110	0.117
rando81	Direct	0.063	0.063	0.067	0.060	0.090	0.083	0.063	0.070
	Strategy	0.087	0.083	0.083	0.087	0.110	0.110	0.087	0.090
rando82	Direct	0.060	0.070	0.063	0.067	0.090	0.090	0.067	0.067
	Strategy	0.087	0.090	0.090	0.087	0.113	0.110	0.090	0.090
rando83	Direct	0.073	0.070	0.070	0.070	0.090	0.090	0.070	0.073
	Strategy	0.093	0.090	0.090	0.090	0.120	0.110	0.090	0.097
rando84	Direct	0.090	0.077	0.080	0.083	0.100	0.107	0.077	0.083
	Strategy	0.097	0.093	0.090	0.100	0.117	0.113	0.097	0.093
rando85	Direct	0.077	0.080	0.077	0.080	0.100	0.100	0.080	0.083
	Strategy	0.097	0.093	0.097	0.093	0.120	0.117	0.100	0.097

Tree	Method	1	2	3	4	5	6	7	8
rando86	Direct	0.067	0.063	0.063	0.067	0.083	0.087	0.073	0.063
	Strategy	0.087	0.087	0.083	0.087	0.110	0.113	0.087	0.090
rando87	Direct	0.067	0.067	0.063	0.070	0.090	0.090	0.067	0.067
	Strategy	0.090	0.083	0.090	0.087	0.117	0.113	0.087	0.090
rando88	Direct	0.070	0.070	0.070	0.067	0.090	0.090	0.070	0.070
	Strategy	0.087	0.090	0.083	0.087	0.110	0.110	0.090	0.090
rando89	Direct	0.420	0.343	0.317	0.313	0.387	0.347	0.197	0.420
	Strategy	0.317	0.280	0.243	0.247	0.303	0.280	0.183	0.350
rando90	Direct	0.060	0.063	0.070	0.063	0.087	0.090	0.060	0.070
	Strategy	0.067	0.063	0.067	0.063	0.090	0.087	0.063	0.073
rando91	Direct	0.350	1.050	1.020	0.707	0.337	0.383	1.113	0.307
	Strategy	0.440	0.913	0.903	0.397	0.370	0.317	1.107	0.300
rando92	Direct	24.307	44.423	63.450	85.157	13.987	11.917	5.643	22.197
	Strategy	21.543	48.793	60.767	85.187	11.577	16.103	5.900	15.203
rando93	Direct	0.120	0.110	0.093	0.120	0.130	0.130	0.110	0.107
	Strategy	0.110	0.107	0.107	0.113	0.130	0.133	0.100	0.110
rando94	Direct	0.063	0.060	0.067	0.063	0.090	0.087	0.067	0.067
	Strategy	0.083	0.087	0.087	0.083	0.113	0.110	0.090	0.090
rando95	Direct	0.073	0.067	0.067	0.073	0.093	0.093	0.077	0.077
	Strategy	0.097	0.090	0.090	0.090	0.113	0.113	0.093	0.097
rando96	Direct	0.060	0.067	0.063	0.060	0.090	0.090	0.060	0.070
	Strategy	0.103	0.107	0.110	0.110	0.157	0.153	0.117	0.110
rando97	Direct	0.060	0.067	0.063	0.063	0.087	0.090	0.063	0.067
	Strategy	0.083	0.090	0.087	0.083	0.110	0.110	0.090	0.087
rando98	Direct	0.317	0.233	0.243	0.280	0.260	0.260	0.243	0.160
	Strategy	0.170	0.153	0.150	0.157	0.180	0.190	0.173	0.187
rando99	Direct	0.140	0.510	0.460	0.380	0.180	0.233	0.337	0.173
	Strategy	0.147	0.473	0.330	0.347	0.180	0.260	0.343	0.167
random1	Direct	0.063	0.063	0.067	0.063	0.087	0.087	0.067	0.067
	Strategy	0.090	0.083	0.087	0.090	0.110	0.110	0.087	0.090
random2	Direct	0.063	0.063	0.067	0.060	0.093	0.090	0.060	0.070
	Strategy	0.087	0.087	0.083	0.087	0.110	0.110	0.087	0.093
random3	Direct	0.190	0.260	0.197	0.203	0.210	0.190	0.143	0.200
	Strategy	0.160	0.187	0.170	0.173	0.177	0.180	0.117	0.170
random4	Direct	0.063	0.067	0.063	0.070	0.080	0.090	0.070	0.060
	Strategy	0.083	0.087	0.083	0.090	0.110	0.107	0.093	0.087
random6	Direct	1.823	20.283	20.283	23.823	3.727	3.540	27.337	3.927
	Strategy	2.103	21.253	21.240	25.497	4.097	4.007	28.730	4.720
random7	Direct	0.070	0.060	0.060	0.073	0.083	0.087	0.070	0.060
	Strategy	0.087	0.087	0.087	0.083	0.110	0.110	0.090	0.087

Tree	Method	1	2	3	4	5	6	7	8
random8	Direct	0.070	0.063	0.067	0.067	0.090	0.087	0.067	0.070
	Strategy	0.090	0.093	0.083	0.087	0.113	0.110	0.090	0.093
random9	Direct	0.060	0.070	0.060	0.070	0.090	0.083	0.070	0.067
	Strategy	0.090	0.087	0.083	0.090	0.110	0.110	0.087	0.090
relcour	Direct	0.063	0.067	0.060	0.070	0.080	0.090	0.070	0.060
	Strategy	0.067	0.063	0.063	0.067	0.090	0.083	0.063	0.070
rstree1	Direct	0.067	0.063	0.060	0.070	0.087	0.083	0.070	0.063
	Strategy	0.080	0.090	0.087	0.087	0.110	0.110	0.090	0.087
rstree2	Direct	0.070	0.060	0.067	0.063	0.090	0.083	0.070	0.067
	Strategy	0.087	0.090	0.087	0.083	0.110	0.110	0.093	0.087
rstree3	Direct	0.067	0.063	0.067	0.060	0.090	0.090	0.060	0.070
	Strategy	0.087	0.087	0.087	0.087	0.110	0.110	0.090	0.087
rstree4	Direct	0.060	0.070	0.060	0.063	0.087	0.090	0.063	0.067
	Strategy	0.087	0.087	0.083	0.090	0.107	0.110	0.090	0.090
rstree5	Direct	0.063	0.067	0.063	0.067	0.083	0.087	0.067	0.070
	Strategy	0.087	0.087	0.090	0.080	0.110	0.113	0.090	0.090
rstree6	Direct	0.063	0.067	0.063	0.067	0.083	0.087	0.073	0.063
	Strategy	0.080	0.090	0.093	0.080	0.110	0.110	0.090	0.090
rstree7	Direct	0.067	0.063	0.063	0.060	0.090	0.090	0.063	0.067
	Strategy	0.083	0.087	0.090	0.083	0.110	0.110	0.093	0.090
trials1	Direct	0.100	0.090	0.097	0.087	0.127	0.103	0.090	0.090
	Strategy	0.110	0.120	0.117	0.113	0.133	0.127	0.113	0.120
trials2	Direct	0.070	0.067	0.063	0.070	0.090	0.090	0.070	0.070
	Strategy	0.090	0.090	0.090	0.090	0.113	0.113	0.093	0.093
trials3	Direct	0.070	0.070	0.067	0.073	0.093	0.090	0.077	0.073
	Strategy	0.090	0.090	0.090	0.097	0.117	0.117	0.093	0.093
trials4	Direct	0.133	0.180	0.153	0.140	0.143	0.140	0.150	0.113
	Strategy	0.150	0.197	0.183	0.203	0.160	0.173	0.167	0.140
usatree	Direct	0.063	0.067	0.060	0.063	0.090	0.087	0.063	0.067
	Strategy	0.067	0.060	0.063	0.067	0.090	0.083	0.070	0.067
worrell	Direct	0.067	0.067	0.063	0.067	0.083	0.093	0.067	0.063
	Strategy	0.083	0.090	0.083	0.087	0.110	0.110	0.093	0.090

Appendix X

BDD Complexities for Additional Reduced Trees Used in the Neural Network Investigation

Key to ordering schemes:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Number of Non-Distinct BDD Nodes[1]

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
lisa100	5301	7590	7423	11067	4565	3935	16787	**3810**
lisa102	1247485	4082011	4011371	2177558	716199	1238861	2378816	**670400**
lisa104	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
lisa107	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
lisa108	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
lisa109	100	190	190	373	101	**66**	332	103
lisa110	5066	10027	9130	6714	5555	6683	6830	**4409**
lisa111	41	**36**	**36**	**36**	41	37	**36**	41
lisa112	824	**751**	**751**	907	920	779	1068	815
lisa113	896	1292	1291	1270	702	643	1481	**557**
lisa115	104	85	81	81	58	101	57	**33**
lisa116	13	9	9	9	7	**6**	9	7
lisa118	68485	314912	216236	161438	42531	47349	218787	**38486**
lisa119	10	**7**	**7**	10	10	**7**	10	10
lisa121	69	53	53	78	73	73	**49**	60
lisa122	89	108	108	108	89	**56**	102	81
lisa124	6683	29271	28489	13552	**5556**	7747	12646	5831
lisab11	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
lisab13	30	35	35	35	34	38	**20**	36

[1] For each fault tree, the ordering scheme(s) resulting in the fewest non-distinct BDD nodes is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
lisab14	393722	$2.03x10^7$	$2.02x10^7$	$7.50x10^7$	277891	6432290	$3.62x10^7$	**173643**
lisab15	61882	97222	96640	82408	41131	36451	100806	**27763**
lisab17	330	591	591	720	**285**	609	602	**285**
lisab22	40778	279612	279612	616536	**34342**	273027	1089847	56122
lisab26	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
lisab27	1192	1145	**1028**	1417	1281	1148	1174	1166
lisab33	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
lisab37	10	10	10	**9**	**9**	10	10	**9**
lisab39	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
lisab40	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
lisab45	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
lisab47	**11**	**11**	**11**	**11**	**11**	**11**	12	13
lisab48	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
lisab50	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
lisab56	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
lisab61	194	136	136	136	151	160	126	**96**
lisab62	**54**	68	68	68	72	68	68	70
lisab63	29	25	25	25	25	**20**	**20**	25
lisab64	**12**	**12**	**12**	**12**	**12**	**12**	13	**12**
lisab66	184	796	796	304	**150**	225	1520	**150**
lisab67	4374	**3907**	**3907**	4892	4375	6942	5446	4196
lisab69	27	**23**	**23**	29	27	**23**	36	24
lisab70	251	264	207	**191**	243	245	214	242
lisab71	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
lisab72	2737	2011	2011	1911	2639	2583	6550	**1697**
lisab74	181810	216333	148790	**140998**	184136	151587	315425	217746
lisab75	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
lisab76	95319	86954	86954	114005	106220	69643	146792	**57580**
lisab77	4286	2333	2263	4785	3302	4261	4510	**1364**
lisab80	6	6	6	6	6	7	6	**5**
lisab82	2990	3048	3048	2856	2767	2896	3331	**2285**
lisab83	438	359	359	184	185	196	**107**	164
lisab85	7	**6**	7	7	**6**	7	7	**6**
lisab87	2835107	$6.67x10^7$	$6.26x10^7$	$2.21x10^7$	**1729805**	$1.16x10^7$	6148359	1791517
lisab88	167	147	147	124	109	187	364	**84**
lisab89	1822	7427	6425	688	1387	2467	**625**	1105
lisab91	737	1190	1190	1190	**589**	1242	1003	1104
lisab94	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
lisab95	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
lisaba1	330	591	591	720	**285**	609	602	**285**
lisaba2	2537083	1.86×10^7	1.86×10^7	5634287	2746780	5287903	2193175	**1492771**
lisaba3	3672	10529	10153	9659	4903	5015	7844	**3272**
lisaba5	1939	3345	1651	2673	1446	4240	2519	**1326**
lisaba6	156	663	647	210	126	199	207	**117**
lisaba7	330	591	591	720	**285**	609	602	**285**
lisaba8	1131891	1073774	947778	1488698	890146	901370	1267894	**708655**
rand159	488	**168**	**168**	793	393	172	205	208
rand161	1399	2921	2411	3723	**687**	1605	3779	974
rand163	4996	15007	11938	11170	4974	6438	395942	**4240**
rand164	8519	8530	7202	7202	8006	**5978**	7691	6771
rand165	20967	**11242**	11877	22707	16700	24927	12000	12977
rand166	53355	110373	60192	75445	17007	33731	160937	**16499**
rand167	354	459	402	375	366	397	**325**	329

Number of If-Then-Else Calculations Required for BDD Construction[2]

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
lisa100	1163	522	**516**	760	796	541	856	648
lisa102	9048	**1915**	2017	2609	3470	2438	3567	2398
lisa104	143	144	143	143	**139**	**139**	151	**139**
lisa107	**14**	**14**	**14**	**14**	**14**	**14**	**14**	15
lisa108	23	23	23	23	**22**	**22**	23	**22**
lisa109	304	**268**	**268**	349	305	292	299	303
lisa110	962	1082	1027	957	1266	**951**	989	1560
lisa111	52	**46**	**46**	**46**	52	48	**46**	52
lisa112	256	**146**	148	231	228	176	202	264
lisa113	832	1010	1028	1443	667	473	**386**	545
lisa115	80	71	69	69	58	66	**50**	45
lisa116	**36**	48	48	48	45	44	48	45
lisa118	1694	2089	1574	1732	973	844	**681**	819
lisa119	13	**12**	**12**	13	13	13	13	13
lisa121	49	57	57	47	52	**45**	58	50
lisa122	112	105	105	105	112	**100**	104	103
lisa124	1010	671	671	744	877	**554**	580	673

[2] For each fault tree, the ordering scheme(s) requiring the fewest **ite** calculations to construct the BDD is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
lisab11	98	**83**	107	106	104	104	**83**	106
lisab13	629	802	793	783	776	761	**552**	768
lisab14	2684	24420	24410	63029	**1968**	10536	19969	**1677**
lisab15	2586	2963	2978	3153	**925**	1065	4651	**674**
lisab17	136	116	116	145	133	129	**111**	128
lisab22	**4171**	10032	10032	9049	4246	6749	11680	6410
lisab26	**29**	**29**	**29**	**29**	**29**	**29**	**29**	34
lisab27	456	607	572	422	407	**224**	266	393
lisab33	21	**20**	21	21	21	21	21	**20**
lisab37	20	20	20	**18**	19	20	20	**18**
lisab39	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**12**
lisab40	12	12	12	12	12	12	**11**	12
lisab45	59	59	61	61	**57**	**57**	61	**57**
lisab47	50	54	54	54	**45**	49	47	46
lisab48	50	**46**	**46**	57	50	50	**46**	48
lisab50	**54**	**54**	**54**	**54**	56	55	**54**	56
lisab56	50	51	51	51	48	**46**	51	49
lisab61	171	192	192	192	**164**	172	240	167
lisab62	58	**51**	**51**	73	63	73	73	59
lisab63	28	26	26	26	27	**23**	**23**	28
lisab64	668	909	826	814	681	**620**	646	672
lisab66	482	858	847	558	**449**	497	928	486
lisab67	**381**	417	417	487	471	404	544	516
lisab69	**27**	**27**	**27**	**27**	30	**27**	34	29
lisab70	94	109	98	**67**	85	71	78	89
lisab71	44	41	41	41	**40**	41	41	43
lisab72	611	494	494	535	615	**476**	813	583
lisab74	4677	741	710	709	2788	**706**	879	4697
lisab75	79	78	**77**	85	81	81	84	80
lisab76	3548	8641	8641	2737	3393	3015	**2068**	2399
lisab77	604	446	379	520	464	551	**358**	275
lisab80	**11**	**11**	**11**	**11**	**11**	13	**11**	10
lisab82	367	**224**	**224**	330	391	390	317	336
lisab83	379	268	268	211	234	216	**165**	191
lisab85	16	**15**	16	16	**15**	16	16	**15**
lisab87	18986	61622	61684	19717	**10205**	13825	10332	7107
lisab88	303	309	309	390	357	337	**274**	326
lisab89	500	408	591	398	468	480	**305**	345
lisab91	691	470	470	470	620	469	**342**	557

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
lisab94	11	11	11	11	**10**	**10**	11	**10**
lisab95	54	**39**	59	59	58	58	**39**	58
lisaba1	136	116	116	145	133	129	**111**	128
lisaba2	22453	8224	8224	8083	17263	11381	**3296**	16936
lisaba3	**602**	1218	1163	1168	662	672	922	913
lisaba5	420	528	442	415	**388**	547	494	403
lisaba6	76	104	97	108	**74**	92	80	66
lisaba7	136	116	116	145	133	129	**111**	128
lisaba8	12047	3119	3235	1959	4519	**1710**	3740	4328
rand159	799	**307**	**307**	889	615	317	359	445
rand161	516	513	488	1027	**338**	527	1120	333
rand163	1073	1652	1502	1478	1065	**1045**	3340	1508
rand164	1595	1710	1035	1035	1090	1041	**1009**	1226
rand165	2395	1049	1042	1367	1382	1446	**567**	3033
rand166	1047	1273	1139	1465	**715**	1209	1222	729
rand167	175	151	144	133	150	136	**128**	144

Appendix XI

Number of Non-Distinct Nodes in BDDs Obtained from Fault Trees Restructured Using the Extended Reduction Method

Key to ordering schemes[1]:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
aaaaaaa	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
artqual	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
arttree	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
astolfo	**21**	**21**	**21**	**21**	**21**	**21**	**21**	27
bddtest	**32**	35	35	35	**32**	36	35	**32**
benjiam	87	**76**	**76**	80	87	84	80	83
bpfeg03	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfen05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfig05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfin05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfpp02	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfsw02	19	19	**14**	**14**	19	**14**	19	15
ch8tree	8	**7**	8	8	8	8	8	**7**
dre1019	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1032	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1057	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1058	30	**26**	**26**	**26**	30	28	**26**	30
dre1059	256	312	261	261	232	**214**	312	216
dresden	453	160	160	160	453	**117**	550	127
emerh2o	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

[1] For each fault tree, the ordering scheme(s) resulting in the fewest non-distinct BDD nodes is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
fatram2	**9**	**9**	**9**	**9**	**9**	**9**	**9**	10
hpisf02	159	137	137	140	171	140	**130**	172
hpisf03	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
hpisf21	**30**	41	41	38	33	41	32	31
hpisf36	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
jdtree1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
jdtree2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
jdtree3	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
jdtree4	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
jdtree5	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
khictre	36	**30**	**30**	33	39	33	**30**	**30**
lisa123	206	226	226	170	188	**122**	204	180
lisab10	4267	4629	4385	**2313**	3264	2686	8260	2380
lisab25	63	65	63	**57**	64	58	65	59
lisab28	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
lisab30	25	**19**	**19**	**19**	22	21	25	20
lisab31	917	1219	1628	1499	733	865	1196	**636**
lisab34	25	**20**	32	25	23	25	32	23
lisab35	1717	2443	2619	1425	1396	1925	2443	**668**
lisab36	348	367	347	267	257	274	299	**212**
lisab42	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
lisab44	18	18	18	18	18	18	**16**	18
lisab51	17	**16**	**16**	**16**	17	**16**	18	21
lisab52	3502	4092	4028	4420	2192	3202	4447	**1740**
lisab53	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
lisab54	21	21	**18**	**18**	20	**18**	**18**	21
lisab57	582	815	615	704	629	582	774	**575**
lisab59	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
lisab60	**23**	25	25	25	**23**	25	25	**23**
lisab78	602	538	422	537	502	537	673	**417**
lisab86	1132	2269	1954	1173	943	1188	1104	**872**
lisaba4	2011	3805	3056	2980	1461	1470	1977	**1174**
lisaba9	14	**13**	**13**	**13**	14	14	**13**	14
modtree	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
nakashi	501	359	318	367	360	455	359	**304**
newtre2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
newtre3	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
newtree	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
rand100	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand101	1	1	1	1	1	1	1	1
rand102	1	1	1	1	1	1	1	1
rand103	20	20	19	19	20	19	22	21
rand104	1	1	1	1	1	1	1	1
rand105	28	33	27	27	28	27	31	26
rand106	1	1	1	1	1	1	1	1
rand107	1	1	1	1	1	1	1	1
rand108	119	206	203	152	109	169	193	110
rand109	385	350	374	393	301	393	310	323
rand110	6	6	6	6	6	6	6	6
rand111	90	73	73	70	81	68	54	45
rand112	1	1	1	1	1	1	1	1
rand113	1	1	1	1	1	1	1	1
rand114	1	1	1	1	1	1	1	1
rand115	233	153	153	244	183	163	149	125
rand116	142	176	176	186	121	191	220	99
rand117	10	12	12	12	11	11	12	11
rand118	132	104	100	106	120	110	130	107
rand119	1	1	1	1	1	1	1	1
rand120	238	242	207	203	218	206	327	232
rand121	32	25	25	32	32	32	32	34
rand122	1	1	1	1	1	1	1	1
rand123	5	5	5	5	5	5	5	5
rand124	1	1	1	1	1	1	1	1
rand125	6	6	6	6	6	6	6	6
rand126	78	78	78	72	78	87	91	76
rand127	1	1	1	1	1	1	1	1
rand128	101	127	96	98	101	100	95	103
rand129	1	1	1	1	1	1	1	1
rand130	1	1	1	1	1	1	1	1
rand131	1	1	1	1	1	1	1	1
rand132	3446	3773	3703	3565	1757	2158	3565	2144
rand133	1	1	1	1	1	1	1	1
rand134	48	63	63	50	52	52	63	48
rand135	85	77	81	81	84	81	82	86
rand136	1	1	1	1	1	1	1	1
rand137	55	55	55	55	58	59	45	34
rand138	1	1	1	1	1	1	1	1
rand139	125	135	107	93	113	92	146	121

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand140	1	1	1	1	1	1	1	1
rand141	6	6	6	6	6	6	6	6
rand142	25750	29765	32074	38739	20834	32786	35941	**15231**
rand143	1	1	1	1	1	1	1	1
rand144	91	92	91	91	81	80	117	**72**
rand145	7	7	7	7	7	7	7	7
rand146	55	55	55	55	58	59	45	**34**
rand147	2493	3153	2845	5815	1458	2104	1607	**1363**
rand148	1	1	1	1	1	1	1	1
rand149	1	1	1	1	1	1	1	1
rand150	15725	21580	21498	**7049**	12428	7999	8393	8659
rand151	1	1	1	1	1	1	1	1
rand152	1	1	1	1	1	1	1	1
rand153	1	1	1	1	1	1	1	1
rand154	1	1	1	1	1	1	1	1
rand155	463	453	**429**	485	464	475	696	471
rand156	1	1	1	1	1	1	1	1
rand158	1	1	1	1	1	1	1	1
rando10	1	1	1	1	1	1	1	1
rando11	**2089234**	8685527	8685527	6546439	4025841	6055138	8054312	3178689
rando12	678	1026	1010	1016	558	534	1010	**526**
rando13	113	209	209	209	103	104	182	**99**
rando14	1	1	1	1	1	1	1	1
rando15	1	1	1	1	1	1	1	1
rando16	68	68	68	66	75	69	102	**60**
rando17	1	1	1	1	1	1	1	1
rando18	82	85	74	86	75	63	**55**	73
rando19	10875	31473	32304	29924	8402	21327	37153	**7150**
rando20	2686	5734	5666	5217	**2415**	2837	6664	2512
rando21	1	1	1	1	1	1	1	1
rando22	16787	22785	23439	66645	12061	24061	96609	**10012**
rando23	1	1	1	1	1	1	1	1
rando24	1	1	1	1	1	1	1	1
rando25	1	1	1	1	1	1	1	1
rando26	1	1	1	1	1	1	1	1
rando27	18	18	18	18	**16**	**16**	18	**16**
rando28	1	1	1	1	1	1	1	1
rando29	235	152	186	299	172	177	186	**138**
rando30	121	112	**103**	**103**	121	107	155	146

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando31	1	1	1	1	1	1	1	1
rando32	1	1	1	1	1	1	1	1
rando33	6	6	6	6	6	6	6	6
rando34	23	22	22	22	23	22	23	23
rando35	14	14	14	14	12	12	11	12
rando36	1	1	1	1	1	1	1	1
rando37	45	42	43	43	41	39	41	37
rando38	1	1	1	1	1	1	1	1
rando39	201	156	156	212	211	200	419	210
rando40	34	23	32	32	34	32	34	23
rando41	1	1	1	1	1	1	1	1
rando42	1	1	1	1	1	1	1	1
rando43	1	1	1	1	1	1	1	1
rando44	928	589	565	759	811	545	759	808
rando45	41	44	44	49	36	39	42	32
rando46	1	1	1	1	1	1	1	1
rando47	44	51	53	53	40	39	64	38
rando48	16	16	16	15	15	14	15	13
rando49	1	1	1	1	1	1	1	1
rando50	1	1	1	1	1	1	1	1
rando51	1	1	1	1	1	1	1	1
rando52	43	51	38	38	37	33	50	37
rando53	1	1	1	1	1	1	1	1
rando54	18	18	18	18	18	19	18	19
rando55	10	10	10	10	10	10	10	11
rando56	1	1	1	1	1	1	1	1
rando57	1	1	1	1	1	1	1	1
rando58	1	1	1	1	1	1	1	1
rando59	2188	1600	1511	845	1071	846	898	1026
rando60	13	11	13	13	11	11	13	11
rando61	19	21	16	16	18	16	16	17
rando62	1	1	1	1	1	1	1	1
rando63	10	10	10	10	10	10	10	11
rando64	80	81	81	81	87	81	86	83
rando65	34	27	27	36	30	28	27	26
rando66	22	29	29	29	27	27	23	22
rando67	1	1	1	1	1	1	1	1
rando68	1	1	1	1	1	1	1	1
rando69	1	1	1	1	1	1	1	1

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando70	56	44	54	54	51	54	63	46
rando71	1	1	1	1	1	1	1	1
rando72	1	1	1	1	1	1	1	1
rando73	1	1	1	1	1	1	1	1
rando74	1	1	1	1	1	1	1	1
rando75	1	1	1	1	1	1	1	1
rando76	10	10	10	10	10	10	10	10
rando77	79	45	45	83	70	61	52	41
rando78	1	1	1	1	1	1	1	1
rando79	1	1	1	1	1	1	1	1
rando80	7	7	7	7	7	7	7	7
rando81	1	1	1	1	1	1	1	1
rando82	1	1	1	1	1	1	1	1
rando83	39	41	34	34	34	31	35	34
rando84	132	104	100	106	120	110	130	107
rando85	1	1	1	1	1	1	1	1
rando86	1	1	1	1	1	1	1	1
rando87	1	1	1	1	1	1	1	1
rando88	35	34	31	34	29	30	26	23
rando89	1	1	1	1	1	1	1	1
rando90	1	1	1	1	1	1	1	1
rando91	1901	1200	1200	1280	1465	1419	1138	1416
rando92	668	651	643	1042	535	896	547	567
rando93	9	9	9	9	9	9	9	9
rando94	1	1	1	1	1	1	1	1
rando95	47	38	46	46	38	38	46	35
rando96	1	1	1	1	1	1	1	1
rando97	1	1	1	1	1	1	1	1
rando98	302	385	385	361	442	361	341	370
rando99	98	84	84	79	97	100	99	84
random1	1	1	1	1	1	1	1	1
random2	1	1	1	1	1	1	1	1
random3	84	97	92	92	84	92	133	88
random4	1	1	1	1	1	1	1	1
random6	7509	15525	12328	22094	5199	4159	9807	5638
random7	1	1	1	1	1	1	1	1
random8	1	1	1	1	1	1	1	1
random9	1	1	1	1	1	1	1	1
relcour	1	1	1	1	1	1	1	1

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rstree1	1	1	1	1	1	1	1	1
rstree2	1	1	1	1	1	1	1	1
rstree3	8	7	7	7	8	7	7	8
rstree4	1	1	1	1	1	1	1	1
rstree5	1	1	1	1	1	1	1	1
rstree6	1	1	1	1	1	1	1	1
rstree7	15	17	17	10	13	10	14	12
trials1	241	375	375	375	186	167	446	134
trials2	11	11	11	11	10	11	11	11
trials3	1	1	1	1	1	1	1	1
trials4	279	560	497	425	215	342	618	255
usatree	1	1	1	1	1	1	1	1
worrell	19	17	17	17	18	17	19	17

Appendix XII

Number of Distinct Nodes in BDDs Obtained from Fault Trees Restructured Using the Extended Reduction Method

Key to ordering schemes[1]:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
aaaaaaa	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
artqual	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
arttree	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
astolfo	**15**	**15**	**15**	**15**	**15**	**15**	**15**	18
bddtest	26	**22**	**22**	**22**	26	25	**22**	26
benjiam	47	34	34	**32**	47	39	**32**	47
bpfeg03	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfen05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfig05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfin05	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfpp02	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
bpfsw02	17	14	**13**	**13**	17	**13**	14	14
ch8tree	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
dre1019	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1032	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1057	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
dre1058	24	**18**	**18**	**18**	24	21	**18**	24
dre1059	89	94	91	91	70	57	94	**51**
dresden	87	**23**	**23**	**23**	87	26	39	32
emerh2o	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

[1] For each fault tree, the ordering scheme(s) resulting in the fewest distinct BDD nodes is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
fatram2	9	9	9	9	9	9	9	10
hpisf02	77	24	24	34	67	34	33	86
hpisf03	11	11	11	11	11	11	11	11
hpisf21	26	22	22	24	24	25	25	24
hpisf36	11	11	11	11	11	11	11	11
jdtree1	1	1	1	1	1	1	1	1
jdtree2	1	1	1	1	1	1	1	1
jdtree3	1	1	1	1	1	1	1	1
jdtree4	1	1	1	1	1	1	1	1
jdtree5	1	1	1	1	1	1	1	1
khictre	15	12	11	11	17	11	11	11
lisa123	74	35	35	28	59	37	66	56
lisab10	532	228	165	185	269	201	376	244
lisab25	40	39	45	35	42	36	39	42
lisab28	1	1	1	1	1	1	1	1
lisab30	17	15	15	15	16	15	19	16
lisab31	290	123	144	140	224	166	132	157
lisab34	18	16	20	16	19	16	20	19
lisab35	308	331	354	139	273	318	331	194
lisab36	85	93	66	53	68	57	80	96
lisab42	1	1	1	1	1	1	1	1
lisab44	16	15	15	15	16	15	14	16
lisab51	13	12	12	12	13	12	12	18
lisab52	481	361	354	497	418	414	279	353
lisab53	1	1	1	1	1	1	1	1
lisab54	17	14	14	14	16	14	12	17
lisab57	104	105	111	107	103	102	97	107
lisab59	1	1	1	1	1	1	1	1
lisab60	19	16	16	16	19	18	16	19
lisab78	177	68	104	105	142	105	99	86
lisab86	148	164	141	107	150	104	146	143
lisaba4	315	188	198	155	191	213	195	166
lisaba9	13	10	10	11	13	13	10	13
modtree	1	1	1	1	1	1	1	1
nakashi	138	43	64	58	120	55	43	105
newtre2	1	1	1	1	1	1	1	1
newtre3	1	1	1	1	1	1	1	1
newtree	1	1	1	1	1	1	1	1
rand100	1	1	1	1	1	1	1	1

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand101	1	1	1	1	1	1	1	1
rand102	1	1	1	1	1	1	1	1
rand103	19	15	18	18	19	18	17	19
rand104	1	1	1	1	1	1	1	1
rand105	22	23	19	19	22	19	15	21
rand106	1	1	1	1	1	1	1	1
rand107	1	1	1	1	1	1	1	1
rand108	75	82	71	70	70	72	73	71
rand109	125	71	81	94	130	91	89	111
rand110	6	6	6	6	6	6	6	6
rand111	50	40	41	39	44	40	30	32
rand112	1	1	1	1	1	1	1	1
rand113	1	1	1	1	1	1	1	1
rand114	1	1	1	1	1	1	1	1
rand115	85	51	51	77	69	54	58	48
rand116	49	49	49	48	48	49	79	47
rand117	10	12	12	12	11	11	12	11
rand118	68	45	41	48	61	51	38	54
rand119	1	1	1	1	1	1	1	1
rand120	98	52	43	72	91	73	81	109
rand121	24	18	18	22	24	23	22	27
rand122	1	1	1	1	1	1	1	1
rand123	5	5	5	5	5	5	5	5
rand124	1	1	1	1	1	1	1	1
rand125	6	6	6	6	6	6	6	6
rand126	38	32	32	37	46	31	58	43
rand127	1	1	1	1	1	1	1	1
rand128	44	42	35	37	44	41	38	54
rand129	1	1	1	1	1	1	1	1
rand130	1	1	1	1	1	1	1	1
rand131	1	1	1	1	1	1	1	1
rand132	443	456	436	414	290	305	414	376
rand133	1	1	1	1	1	1	1	1
rand134	39	22	22	35	40	38	22	37
rand135	42	36	36	36	40	36	27	45
rand136	1	1	1	1	1	1	1	1
rand137	19	19	19	19	23	19	25	22
rand138	1	1	1	1	1	1	1	1
rand139	73	42	36	36	62	39	37	58

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand140	1	1	1	1	1	1	1	1
rand141	6	6	6	6	6	6	6	6
rand142	1466	1173	1187	961	1192	798	902	1224
rand143	1	1	1	1	1	1	1	1
rand144	45	34	32	32	50	45	57	51
rand145	7	7	7	7	7	7	7	7
rand146	19	19	19	19	23	19	25	22
rand147	311	478	487	309	240	212	294	198
rand148	1	1	1	1	1	1	1	1
rand149	1	1	1	1	1	1	1	1
rand150	846	1010	654	318	883	411	382	790
rand151	1	1	1	1	1	1	1	1
rand152	1	1	1	1	1	1	1	1
rand153	1	1	1	1	1	1	1	1
rand154	1	1	1	1	1	1	1	1
rand155	142	102	70	90	117	94	90	115
rand156	1	1	1	1	1	1	1	1
rand158	1	1	1	1	1	1	1	1
rando10	1	1	1	1	1	1	1	1
rando11	12470	11851	11851	5951	7105	3319	5538	8645
rando12	196	159	155	165	184	182	172	146
rando13	58	40	40	40	54	46	68	45
rando14	1	1	1	1	1	1	1	1
rando15	1	1	1	1	1	1	1	1
rando16	43	44	43	41	46	45	52	44
rando17	1	1	1	1	1	1	1	1
rando18	46	57	45	44	41	34	32	38
rando19	860	825	833	1197	767	1092	1082	624
rando20	368	470	404	391	363	292	550	360
rando21	1	1	1	1	1	1	1	1
rando22	1472	1162	1099	2254	870	1013	2172	785
rando23	1	1	1	1	1	1	1	1
rando24	1	1	1	1	1	1	1	1
rando25	1	1	1	1	1	1	1	1
rando26	1	1	1	1	1	1	1	1
rando27	17	17	17	17	16	16	17	16
rando28	1	1	1	1	1	1	1	1
rando29	70	45	48	66	61	53	48	58
rando30	44	38	34	34	44	39	35	49

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando31	1	1	1	1	1	1	1	1
rando32	1	1	1	1	1	1	1	1
rando33	6	6	6	6	6	6	6	6
rando34	21	19	21	21	22	21	19	22
rando35	13	13	13	13	12	12	11	12
rando36	1	1	1	1	1	1	1	1
rando37	34	28	32	32	35	30	34	32
rando38	1	1	1	1	1	1	1	1
rando39	94	54	54	93	93	89	90	91
rando40	19	15	16	16	19	16	20	18
rando41	1	1	1	1	1	1	1	1
rando42	1	1	1	1	1	1	1	1
rando43	1	1	1	1	1	1	1	1
rando44	203	107	101	166	227	159	166	177
rando45	30	31	31	23	25	28	25	22
rando46	1	1	1	1	1	1	1	1
rando47	32	23	33	33	32	31	28	30
rando48	14	12	12	11	14	11	11	12
rando49	1	1	1	1	1	1	1	1
rando50	1	1	1	1	1	1	1	1
rando51	1	1	1	1	1	1	1	1
rando52	28	27	24	24	28	24	26	28
rando53	1	1	1	1	1	1	1	1
rando54	16	16	16	16	16	16	16	18
rando55	10	10	10	10	10	10	10	11
rando56	1	1	1	1	1	1	1	1
rando57	1	1	1	1	1	1	1	1
rando58	1	1	1	1	1	1	1	1
rando59	333	142	141	68	210	78	68	211
rando60	12	11	11	11	11	11	11	11
rando61	17	16	15	15	16	15	15	16
rando62	1	1	1	1	1	1	1	1
rando63	10	10	10	10	10	10	10	11
rando64	55	36	36	36	45	36	25	51
rando65	25	15	15	24	24	22	17	21
rando66	18	19	19	19	20	20	14	18
rando67	1	1	1	1	1	1	1	1
rando68	1	1	1	1	1	1	1	1
rando69	1	1	1	1	1	1	1	1

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando70	22	20	18	18	22	18	25	29
rando71	1	1	1	1	1	1	1	1
rando72	1	1	1	1	1	1	1	1
rando73	1	1	1	1	1	1	1	1
rando74	1	1	1	1	1	1	1	1
rando75	1	1	1	1	1	1	1	1
rando76	9	9	9	9	9	9	9	9
rando77	40	19	19	39	31	23	22	19
rando78	1	1	1	1	1	1	1	1
rando79	1	1	1	1	1	1	1	1
rando80	7	7	7	7	7	7	7	7
rando81	1	1	1	1	1	1	1	1
rando82	1	1	1	1	1	1	1	1
rando83	28	19	21	21	26	23	17	26
rando84	68	45	41	48	61	51	38	54
rando85	1	1	1	1	1	1	1	1
rando86	1	1	1	1	1	1	1	1
rando87	1	1	1	1	1	1	1	1
rando88	23	14	17	14	21	15	18	19
rando89	1	1	1	1	1	1	1	1
rando90	1	1	1	1	1	1	1	1
rando91	296	139	139	165	229	146	160	175
rando92	166	142	143	115	163	107	97	166
rando93	9	9	9	9	9	9	9	9
rando94	1	1	1	1	1	1	1	1
rando95	29	24	28	28	30	30	28	27
rando96	1	1	1	1	1	1	1	1
rando97	1	1	1	1	1	1	1	1
rando98	139	96	96	90	172	90	88	130
rando99	57	31	31	43	63	50	35	56
random1	1	1	1	1	1	1	1	1
random2	1	1	1	1	1	1	1	1
random3	49	49	44	44	49	44	41	55
random4	1	1	1	1	1	1	1	1
random6	486	362	484	528	574	278	372	414
random7	1	1	1	1	1	1	1	1
random8	1	1	1	1	1	1	1	1
random9	1	1	1	1	1	1	1	1
relcour	1	1	1	1	1	1	1	1

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rstree1	1	1	1	1	1	1	1	1
rstree2	1	1	1	1	1	1	1	1
rstree3	8	6	7	7	8	7	7	8
rstree4	1	1	1	1	1	1	1	1
rstree5	1	1	1	1	1	1	1	1
rstree6	1	1	1	1	1	1	1	1
rstree7	13	11	11	10	12	10	12	11
trials1	74	78	78	78	77	62	77	84
trials2	11	11	11	11	10	11	11	11
trials3	1	1	1	1	1	1	1	1
trials4	119	152	136	116	104	114	133	105
usatree	1	1	1	1	1	1	1	1
worrell	16	15	15	15	15	13	14	15

Appendix XIII

Number of If-Then-Else Calculations Required to Construct BDDs from Fault Trees Restructured Using the Extended Reduction Method

Key to ordering schemes[1]:

1. Modified top-down.
2. Modified depth-first.
3. Modified priority depth-first.
4. Depth-first, with number of leaves.
5. Non-dynamic top-down weights.
6. Dynamic top-down weights.
7. Bottom-up weights.
8. Event criticality.

Fault tree	Ordering scheme							
	1	2	3	4	5	6	7	8
aaaaaaa	0	0	0	0	0	0	0	0
artqual	8	7	8	8	8	8	8	7
arttree	0	0	0	0	0	0	0	0
astolfo	16	21	21	21	16	16	21	22
bddtest	27	31	31	31	27	26	31	27
benjiam	76	75	75	70	76	67	75	101
bpfeg03	0	0	0	0	0	0	0	0
bpfen05	0	0	0	0	0	0	0	0
bpfig05	0	0	0	0	0	0	0	0
bpfin05	0	0	0	0	0	0	0	0
bpfpp02	0	0	0	0	0	0	0	0
bpfsw02	27	18	22	22	27	22	18	24
ch8tree	11	9	11	11	11	11	11	9
dre1019	0	0	0	0	0	0	0	0
dre1032	0	0	0	0	0	0	0	0
dre1057	0	0	0	0	0	0	0	0
dre1058	23	25	25	25	23	21	25	23
dre1059	90	134	131	131	75	65	134	62
dresden	131	91	91	91	131	76	97	89
emerh2o	0	0	0	0	0	0	0	0

[1] For each fault tree, the ordering scheme(s) requiring the fewest ite calculations to construct the BDD is (are) shown in bold.

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
fatram2	11	11	11	11	11	11	11	10
hpisf02	107	47	47	60	96	60	63	113
hpisf03	19	17	17	17	19	17	17	19
hpisf21	58	50	50	65	54	52	56	53
hpisf36	19	17	17	17	19	17	17	19
jdtree1	0	0	0	0	0	0	0	0
jdtree2	0	0	0	0	0	0	0	0
jdtree3	0	0	0	0	0	0	0	0
jdtree4	0	0	0	0	0	0	0	0
jdtree5	0	0	0	0	0	0	0	0
khictre	32	26	29	27	32	27	29	24
lisa123	139	123	123	108	127	93	139	116
lisab10	624	367	304	316	346	308	598	348
lisab25	69	60	67	65	67	66	60	57
lisab28	0	0	0	0	0	0	0	0
lisab30	27	24	24	24	27	25	32	29
lisab31	356	263	301	230	275	244	278	211
lisab34	30	23	30	35	29	35	30	27
lisab35	381	401	422	253	330	366	401	252
lisab36	123	131	103	89	98	93	114	134
lisab42	0	0	0	0	0	0	0	0
lisab44	18	19	19	19	18	19	17	18
lisab51	23	21	21	21	23	21	22	25
lisab52	698	631	624	736	625	614	537	513
lisab53	0	0	0	0	0	0	0	0
lisab54	26	22	23	23	25	23	24	24
lisab57	133	194	202	197	127	137	185	129
lisab59	0	0	0	0	0	0	0	0
lisab60	25	26	26	26	25	28	26	23
lisab78	210	145	178	170	173	168	152	126
lisab86	214	275	213	214	224	209	288	195
lisaba4	362	283	299	255	235	267	318	206
lisaba9	13	13	13	14	13	13	13	13
modtree	0	0	0	0	0	0	0	0
nakashi	207	91	118	117	179	107	91	151
newtre2	0	0	0	0	0	0	0	0
newtre3	0	0	0	0	0	0	0	0
newtree	0	0	0	0	0	0	0	0
rand100	0	0	0	0	0	0	0	0

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand101	0	0	0	0	0	0	0	0
rand102	0	0	0	0	0	0	0	0
rand103	28	26	27	27	28	27	25	27
rand104	0	0	0	0	0	0	0	0
rand105	30	35	30	30	30	30	31	27
rand106	0	0	0	0	0	0	0	0
rand107	0	0	0	0	0	0	0	0
rand108	182	157	165	195	168	189	172	163
rand109	183	120	129	141	176	141	153	157
rand110	8	8	8	8	8	8	7	7
rand111	88	81	81	78	85	80	57	67
rand112	0	0	0	0	0	0	0	0
rand113	0	0	0	0	0	0	0	0
rand114	0	0	0	0	0	0	0	0
rand115	118	80	80	115	97	81	95	78
rand116	87	84	84	88	86	87	117	74
rand117	16	19	19	19	17	17	19	17
rand118	97	72	78	88	94	80	78	90
rand119	0	0	0	0	0	0	0	0
rand120	155	119	125	147	149	147	163	164
rand121	31	24	24	33	29	28	33	30
rand122	0	0	0	0	0	0	0	0
rand123	6	6	6	6	6	6	6	5
rand124	0	0	0	0	0	0	0	0
rand125	8	7	8	8	8	8	7	7
rand126	60	58	58	60	63	56	91	64
rand127	0	0	0	0	0	0	0	0
rand128	68	86	77	68	68	65	74	80
rand129	0	0	0	0	0	0	0	0
rand130	0	0	0	0	0	0	0	0
rand131	0	0	0	0	0	0	0	0
rand132	648	614	595	606	401	454	606	506
rand133	0	0	0	0	0	0	0	0
rand134	50	42	42	55	52	50	42	47
rand135	60	52	60	60	58	60	59	58
rand136	0	0	0	0	0	0	0	0
rand137	36	36	36	36	36	35	37	33
rand138	0	0	0	0	0	0	0	0
rand139	101	82	91	79	97	79	82	90

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rand140	0	0	0	0	0	0	0	0
rand141	8	8	8	8	8	8	7	7
rand142	2031	1412	1449	1301	1552	1151	1233	1407
rand143	0	0	0	0	0	0	0	0
rand144	65	72	73	73	69	70	76	71
rand145	9	9	9	9	9	9	9	8
rand146	36	36	36	36	36	35	37	33
rand147	456	607	598	627	346	317	404	305
rand148	0	0	0	0	0	0	0	0
rand149	0	0	0	0	0	0	0	0
rand150	1120	1302	1026	499	1088	682	535	949
rand151	0	0	0	0	0	0	0	0
rand152	0	0	0	0	0	0	0	0
rand153	0	0	0	0	0	0	0	0
rand154	0	0	0	0	0	0	0	0
rand155	191	180	168	167	167	146	224	158
rand156	0	0	0	0	0	0	0	0
rand158	0	0	0	0	0	0	0	0
rando10	0	0	0	0	0	0	0	0
rando11	12626	12455	12455	6447	7298	3677	7266	8851
rando12	293	292	287	326	267	281	274	265
rando13	98	92	92	92	90	83	115	83
rando14	0	0	0	0	0	0	0	0
rando15	0	0	0	0	0	0	0	0
rando16	96	97	96	93	93	85	95	85
rando17	0	0	0	0	0	0	0	0
rando18	70	91	79	78	58	55	52	58
rando19	1092	1174	1181	1463	935	1317	1395	795
rando20	545	794	669	658	492	451	902	468
rando21	0	0	0	0	0	0	0	0
rando22	1725	1590	1506	3297	1123	1226	3025	1034
rando23	0	0	0	0	0	0	0	0
rando24	0	0	0	0	0	0	0	0
rando25	0	0	0	0	0	0	0	0
rando26	0	0	0	0	0	0	0	0
rando27	27	27	27	27	22	22	27	22
rando28	0	0	0	0	0	0	0	0
rando29	125	84	91	151	90	82	91	78
rando30	56	54	51	51	56	55	64	59

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Schemo 7	Schemo 8
rando31	0	0	0	0	0	0	0	0
rando32	0	0	0	0	0	0	0	0
rando33	7	6	7	7	7	7	7	6
rando34	34	31	32	32	34	32	31	31
rando35	18	18	18	18	14	14	13	14
rando36	0	0	0	0	0	0	0	0
rando37	59	54	57	57	52	49	56	48
rando38	0	0	0	0	0	0	0	0
rando39	181	103	103	197	159	197	180	159
rando40	26	22	24	24	26	24	27	23
rando41	0	0	0	0	0	0	0	0
rando42	0	0	0	0	0	0	0	0
rando43	0	0	0	0	0	0	0	0
rando44	257	156	151	250	276	209	250	239
rando45	40	42	42	45	35	37	41	32
rando46	0	0	0	0	0	0	0	0
rando47	57	67	69	69	54	53	63	51
rando48	20	16	16	20	17	17	20	16
rando49	0	0	0	0	0	0	0	0
rando50	0	0	0	0	0	0	0	0
rando51	0	0	0	0	0	0	0	0
rando52	42	41	39	39	44	41	51	44
rando53	0	0	0	0	0	0	0	0
rando54	20	20	20	20	20	21	20	19
rando55	13	13	13	13	13	13	13	12
rando56	0	0	0	0	0	0	0	0
rando57	0	0	0	0	0	0	0	0
rando58	0	0	0	0	0	0	0	0
rando59	447	218	215	186	287	166	221	301
rando60	17	14	17	17	14	14	17	14
rando61	28	24	25	25	27	25	25	25
rando62	0	0	0	0	0	0	0	0
rando63	13	13	13	13	13	13	13	12
rando64	67	54	54	54	59	54	55	62
rando65	36	23	23	41	29	27	25	26
rando66	18	25	25	25	20	20	22	18
rando67	0	0	0	0	0	0	0	0
rando68	0	0	0	0	0	0	0	0
rando69	0	0	0	0	0	0	0	0

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rando70	35	31	32	32	33	32	40	35
rando71	0	0	0	0	0	0	0	0
rando72	0	0	0	0	0	0	0	0
rando73	0	0	0	0	0	0	0	0
rando74	0	0	0	0	0	0	0	0
rando75	0	0	0	0	0	0	0	0
rando76	13	13	13	13	13	13	13	13
rando77	90	60	60	87	79	64	75	59
rando78	0	0	0	0	0	0	0	0
rando79	0	0	0	0	0	0	0	0
rando80	9	9	9	9	9	9	8	8
rando81	0	0	0	0	0	0	0	0
rando82	0	0	0	0	0	0	0	0
rando83	36	37	36	36	33	31	34	30
rando84	97	72	78	88	94	80	78	90
rando85	0	0	0	0	0	0	0	0
rando86	0	0	0	0	0	0	0	0
rando87	0	0	0	0	0	0	0	0
rando88	34	24	32	24	26	23	28	25
rando89	0	0	0	0	0	0	0	0
rando90	0	0	0	0	0	0	0	0
rando91	428	236	236	396	349	333	258	388
rando92	227	222	221	187	214	169	227	242
rando93	11	10	11	11	11	11	10	10
rando94	0	0	0	0	0	0	0	0
rando95	48	41	47	47	38	38	47	37
rando96	0	0	0	0	0	0	0	0
rando97	0	0	0	0	0	0	0	0
rando98	174	137	137	131	201	131	128	167
rando99	86	67	67	77	88	81	77	87
random1	0	0	0	0	0	0	0	0
random2	0	0	0	0	0	0	0	0
random3	71	79	75	75	71	75	103	73
random4	0	0	0	0	0	0	0	0
random6	620	507	622	715	685	412	549	539
random7	0	0	0	0	0	0	0	0
random8	0	0	0	0	0	0	0	0
random9	0	0	0	0	0	0	0	0
relcour	0	0	0	0	0	0	0	0

Fault tree	Scheme 1	Scheme 2	Scheme 3	Scheme 4	Scheme 5	Scheme 6	Scheme 7	Scheme 8
rstree1	0	0	0	0	0	0	0	0
rstree2	0	0	0	0	0	0	0	0
rstree3	10	8	9	9	10	9	9	8
rstree4	0	0	0	0	0	0	0	0
rstree5	0	0	0	0	0	0	0	0
rstree6	0	0	0	0	0	0	0	0
rstree7	25	19	19	21	22	21	20	23
trials1	159	178	178	178	143	112	177	147
trials2	18	15	15	15	14	15	15	15
trials3	0	0	0	0	0	0	0	0
trials4	241	336	318	286	201	233	269	203
usatree	0	0	0	0	0	0	0	0
worrell	29	28	28	28	28	27	24	27