38

Fault Tree Analysis

Liudong Xing[1] and Suprasad V. Amari[2]

[1] Department of Electrical and Computer Engineering, University of Massachusetts-Dartmouth, USA
[2] Relex Software Corporation, Greensburg, USA

Abstract: In this chapter, a state-of-the-art review of fault tree analysis is presented. Different forms of fault trees, including static, dynamic, and non-coherent fault trees, their applications and analyses will be discussed. Some advanced topics such as importance analysis, dependent failures, disjoint events, and multistate systems will also be presented.

38.1 Introduction

The fault tree analysis (FTA) technique was first developed in 1962 at Bell Telephone Laboratories to facilitate analysis of the launch control system of the intercontinental Minuteman missile [1]. It was later adopted, improved, and extensively applied by the Boeing Company. Today FTA has become one of the most widely used techniques for system reliability and safety studies. In particular, FTA has been used in analyzing safety systems in nuclear power plants, aerospace, and defense.

FTA is an analytical technique, whereby an undesired event (usually system or subsystem failure) is defined, and then the system is analyzed in the context of its environment and operation to find all combinations of basic events that will lead to the occurrence of the predefined undesired event [2]. The basic events represent basic causes for the undesired event; they can be events associated with component hardware failures, human errors, environmental conditions, or any other pertinent events that can lead to the undesired event. A fault tree thus provides a graphical representation of logical relationships between the undesired event and the basic fault events. From a system design perspective, FTA provides a logical framework for understanding the ways in which a system can fail, which is often as important as understanding how a system can operate successfully.

In this chapter, we first compare the FTA method with other existing analysis methods, in particular, reliability block diagrams, and then describe how to construct a fault tree model. Different forms of fault trees, including static, dynamic, and non-coherent fault trees and their applications will also be discussed. We then discuss different types of FTA as well as both classical and modern techniques used for FTA. We also discuss some advanced topics such as importance analysis, common-cause failures, generalized dependent failures, disjoint events, as well as application of fault trees in analyzing multistate systems and phased-mission systems. Some FTA software tools will be introduced at the end of this chapter.

38.2 A Comparison with Other Methods

System analysis methods can be classified into two generic categories: inductive methods and deductive methods. Induction constitutes reasoning from specific cases to a general conclusion. In an inductive system analysis, we postulate a particular fault or initiating event and attempt to find out the effect of this fault on the entire system failure. Examples of inductive system analysis include failure mode and effect analysis (FMEA) [2–4], failure mode effect and criticality analysis (FMECA) [2, 4, 5], preliminary hazards analysis (PHA) [2], fault hazard analysis (FHA) [2], and event tree analysis [4, 6]. In a deductive system analysis, we postulate a system failure and attempt to find out what modes of system or component behavior contribute to the system failure. In other words, the inductive methods are applied to determine what system states (usually failed states) are possible; the deductive methods are applied to determine how a particular system state can occur. FTA is an example of deductive method and is the principle subject of this chapter. Similar to FTA, we can also use reliability block diagrams (RBD) [7] to specify various combinations of component successes that lead to a specific state or performance level of a system. Therefore, RBD can also be viewed as a deductive method. In the following subsection, we give a brief comparison between fault trees and RBD.

38.2.1 Fault Trees *Versus* RBD

The most fundamental difference between fault trees and RBD is that an RBD is a success-oriented model, while a fault tree is failure-oriented. Specifically, in an RBD, one works in the "success space" and thus looks at system success combinations, whereas in a fault tree one works in the "failure space" and thus looks at system failure combinations. For practical applications in which the system failure depends only on combinations of its component failures, we may choose either a fault tree or an RBD to model the system structure. Both methods will produce the same results. But in most applications, particularly for safety critical systems, it is recommended to start by constructing a fault tree instead of an RBD because thinking in terms of failures will often reveal more potential failure causes than thinking from the function point of view [4].

In most cases, we may convert a fault tree to an RBD or *vice versa*. Particularly, the conversion is possible for all static coherent structures. In the conversion from a fault tree to an RBD, we start from the TOP event of the fault tree and replace the gates successively. A logic AND-gate is replaced by a parallel structure of the inputs of the gate, and an OR gate is replaced by a series structure of the inputs of the gate. In the conversion from an RBD to a fault tree, a parallel structure is represented as a fault tree where all the input events are connected through an AND-gate, and a series structure is represented as a fault tree where all the input events are connected through an OR-gate. Figure 38.1 shows the relationship between a fault tree and an RBD. Note that the events in the fault tree are failure events. Blocks in the RBD means the components represented by the blocks are functioning.

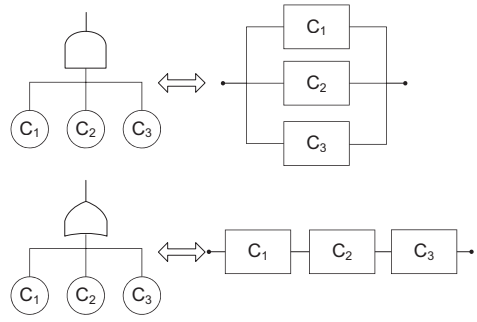

Figure 38.1. Conversion between RBDs and fault trees

Both FTA and RBD are evolutionary in nature, meaning that their modeling capabilities are enhanced as needed to support a wide range of scenarios. For example, introducing new gates, the FTA is enhanced to support sequence dependent failures. However, RBDs are not enhanced to support these modeling features. Similarly, there are some other enhancements to RBDs that are not available in FTA. Hence, it is not possible or practical to covert all fault trees into equivalent RBDs and *vice-versa*.

38.3 Fault Tree Construction

FTA is a deductive technique where we start with the failure scenario being considered, and decompose the failure symptom into its possible causes. Each possible cause is then investigated and further refined until the basic causes of the failure are understood. For more details, one can refer to [21, Chapter 8] or [32, Chapter 7]. The failure scenario to be analyzed is normally called the TOP event of the fault tree. The basic causes are the basic events of the fault tree. The fault tree should be completed in levels, and they should be built from top to bottom. However, various branches of a fault tree can be built to achieve different levels of granularity.

38.3.1 Important Definitions

The following concepts are critical for the proper selection and definition of fault tree events, and thus for the construction of fault trees:

- An *undesired event* constitutes the TOP event of a fault tree model constructed for a system. Careful selection of the undesired event is important to the success of FTA. The definition of the TOP event should be neither too general nor too specific. Here are several examples of undesired events that can be suitable for beginning FTA: overflow of a washing machine [8], no signal from the start relay of a fire detector system when a fire condition is present [4], car does not start when ignition key is turned [2], and loss of spacecraft in the space exploration [2].
- A *primary (or basic) failure* is a failure caused by natural aging of the component. For example, fatigue failure of a relay spring within its rated lifetime, and leakage of a valve seal within its pressure rating.
- A *secondary failure* is a failure induced by the exposure of the failed component to environmental and/or service stresses exceeding its intended ratings. The stresses may be shocks from mechanical, electrical, chemical, thermal, or radioactive energy sources. The stresses may be caused by neighboring components within the system, external environment, or system operators. For example, the component has been improperly designed, or selected, or installed for the application, and a failed component is overstressed or under-qualified for its burden.

38.3.2 Elements of Fault Trees

The main elements of a fault tree include:

- A TOP event: represents the undesired event, usually the system failure or accident.
- Basic events: represent basic causes for the undesired event, usually the failures of components that constitute the system, human errors, or environmental stresses. No further development of failure causes is required for basic events.
- Undeveloped events: represent fault events that are not examined further because information is unavailable or because its consequence is insignificant;
- Gates: are outcomes of one or a combination of basic events or other gates. The gate events are also referred to as intermediate events.

Readers may refer to [2, 8, 9, 21, and 32] for more details of these elements as well as their graphical representation in the fault tree model.

38.3.3 Construction Guidelines

To achieve a consistent analysis, the following steps are suggested for constructing a successful fault tree model:

1) Define the undesired event to be analyzed. The description of it should provide answers to the following questions:
   a. What: describe what type of undesired event is occurring (*e.g.*, fire, crash, or overflow).
   b. Where: describe where the undesired event occurs (*e.g.*, in a motor of an automobile).
   c. When: describe when the undesired event occurs (*e.g.*, when the power is applied, when a fire condition is present).

2) Define boundary conditions for the analysis, including
   a. Physical boundaries: define what constitutes the system, *i.e.* which parts of the system will be included in the FTA.
   b. Boundary conditions concerning environmental stresses: define what type of external stresses (*e.g.*, earthquake and bomb) should be included in the fault tree.
   c. Level of resolution: determine how far down in detail we should go to identify the potential reasons for a failed state.
3) Identify and evaluate fault events, *i.e.,* contributors to the undesired TOP event: if a fault event represents a primary failure, it is classified as a basic event; if the fault event represents a secondary failure, it is classified as an intermediate event that requires a further investigation to identify the prime causes.
4) Complete the gates: all inputs of a particular gate should be completely defined before further analysis of any one of them is undertaken (complete-the-gate rule) [2]. The fault tree should be developed in levels, and each level should be completed before any consideration is given to the next level.

38.3.4 Common Errors in Construction

Errors observed frequently in constructing fault trees are listed. The mistakes listed here are not intentional. Instead, they happen due to simple oversights, misconceptions, and/or lack of knowledge about the fault trees.

- *Ambiguous TOP event*: the definition of the undesired TOP event should be clear and unambiguous. If it is too general, the FTA can become unmanageable; if it is too specific, the FTA cannot provide a sufficiently broad view of the system.
- *Ignoring significant environment conditions*: another common mistake is to consider only failures of components that constitute the system and ignore external stresses, which sometimes can contribute significantly to the system failure.
- *Inconsistent fault tree event names*: the same name should be used for the same fault event or condition throughout the analysis.
- *Inappropriate level of detail/resolution*: the level of detail has a significant impact on the problem formulation. Avoid the formulations that are either too narrow or too broad. When determining the preferred level of resolution, we should remember that the detail in the fault tree should be comparable to the detail of the available information.

38.4 Different Forms

Fault trees can be broadly classified into coherent and noncoherent categories. Coherent fault trees do not use inverse gates, that is to say, the inclusion of inversion may lead to a noncoherent fault tree. Coherent trees can be further classified as static or dynamic trees depending on the sequence relationship between the input events. We describe these three types of fault trees in this section and their evaluation methods in Sections 38.6, 38.7, and 38.8, respectively.

38.4.1 Static Fault Trees

In a static fault tree, logical gates are restricted to static coherent gates, including AND, OR, and *K*-out-of-*N* gates. Static fault trees express the failure criteria of the system in terms of combinations of fault events. Moreover, the system failure is insensitive to the order of occurrence of component fault events [8].

38.4.2 Dynamic Fault Trees

In practice, the failure criteria of a system may depend on both the combinations of fault events and sequence of occurrence of input events. For example, consider a fault-tolerant system with one primary component and one standby spare connected with a switch controller (Figure 38.2) [10]. If the switch controller fails after the primary component fails and thus the standby is switched into active operation, then the system can continue to operate. However, if the switch controller fails

before the primary component fails, then the standby component cannot be activated, and the system fails when the primary component fails even though the spare is still operational. Systems with sequence dependence are modeled with dynamic fault trees (DFT). Dugan and Doyle [8] described several different types of sequence dependencies and corresponding dynamic fault tree gates. A brief description of them is given as follows.

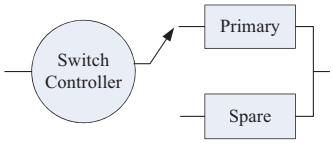

Figure 38.2. A standby sparing system

38.4.2.1 Functional Dependency (FDEP) Gate

A FDEP gate (Figure 38.3) has a single trigger input event and one or more dependent basic events. The trigger event can be either a basic event or the output of another gate in the fault tree. The occurrence of the trigger event forces the dependent basic events to occur. The separate occurrence of any of the dependent basic events has no effect on the trigger event. The FDEP gate has no logical output, thus it is connected to the fault tree through a dashed line.

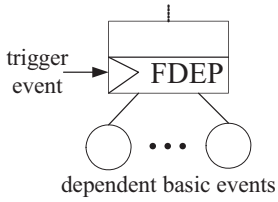

Figure 38.3. Functional dependency gate

For example, the FDEP gate can be used when communication is achieved through a network interface card (NIC), where the failure of the NIC (trigger event) makes the connected components inaccessible.

38.4.2.2 Cold Spare (CSP) Gate

A CSP gate (Figure 38.4) consists of one primary input event and one or more alternate input events. All the input events are basic events. The primary input represents the component that is initially powered on. The alternate inputs represent components that are initially un-powered and serve as replacements for the primary component. The output occurs after all the input events occur, *i.e.,* the primary component and all the spares have failed or become unavailable. As an example, the CSP gate can be used when a spare processor is shared between two active processors. The basic event representing the cold spare processor is the input event to two CSP gates. However, the spare is only available to one of the CSP gates, depending on which of the primary processors fails first.

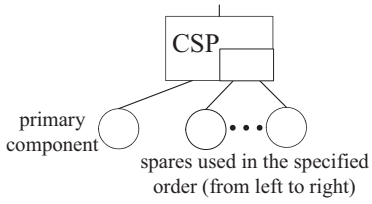

Figure 38.4. Cold spare gate

There are two variations of the CSP gate: hot spare (HSP) gate and warm spare (WSP) gate. The graphical layouts of these two gates are similar to Figure 38.4, only changing CSP to HSP and WSP respectively. In HSP, the spare components have the same failure rate before and after being switched into active use. In WSP, the spare components have reduced failure rate before being switched into active use. Note that the cold, warm, and hot spare gates not only model sparing behavior, but also affect the failure rates of basic events attached to them. As a result, basic events cannot be connected to spare gates of different types, because attenuation of the failure rate would not be defined.

Coppit *et al.* [11] suggest using a generic spare instead of a temperature (cold, warm, or hot) notion for the spare gate. The attenuation of failure rate of an unused, unfailed replica of a basic event

is dictated solely by a dormancy factor of the basic event. This change can provide more orthogonality between spare gates and basic events, and can remove the restriction on sharing of spares among spare gates. This design is implemented in Relex fault tree analysis software [12].

38.4.2.3 Priority-AND Gate

The priority-AND gate (Figure 38.5) is logically equivalent to a normal AND gate, with the extra condition that the input events must occur in a defined order. Specifically, the output occurs if both input events occur and the left input occurs before the right input. In other words, if any of the events has not occurred or if the right input occurs before the left input, the output does not occur.

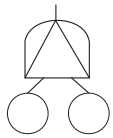

Figure 38.5. Priority-AND gate

As an example, the priority-AND gate can be used to describe one of the failure scenarios for the standby sparing system in Figure 38.2: if the switch controller fails before the primary component, the system fails when the primary component fails. Assume the cold spare is used in this example, and then the fault tree model for the entire system is shown in Figure 38.6.

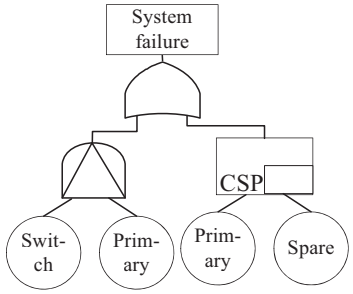

Figure 38.6. DFT of the standby sparing system

38.4.2.4 Sequence Enforcing (SEQ) Gate

The SEQ gate (Figure 38.7) forces all the input events to occur in a defined order: left-to-right order in which they appear under the gate. It is different from the priority-AND gate in that the SEQ gate only allows the events to occur in a specified order whereas the priority-AND gate detects whether the input events occur in a specified order, the events can occur in any order in practice, though.

Figure 38.7. Sequence enforcing gate

38.4.3 Noncoherent Fault Trees

A noncoherent fault tree is characterized by inverse gates besides logic gates used in coherent fault trees. In particular, it may have Exclusive-OR and NOT gates (Figure 38.8). A non-coherent fault tree is used to describe failure behavior of a noncoherent system, which can transit from a failed state to a good state by the failure of a component, or transit from a good state to a failed state by the repair of a component. The structure function of a noncoherent system does not increase monotonically with additional number of functioning components.

(a) NOT gate (b) Exclusive-OR gate

Figure 38.8. Noncoherent fault tree gates

Noncoherent systems are typically prevalent in systems with limited resources, multi-tasking and safety control applications. As an example, consider a *k*-to-*l*-out-of-*n* multiprocessor system where resources such as memory, I/O, and bus are shared among a number of processors [13]. If less than a certain number of processors *k* is being

used, the system will not work to its maximum capacity; on the other hand, if the number of processors being used exceeds *l*, the system efficiency also suffers due to the traffic congestion on a limited bandwidth bus. In FTA, we can consider the system has failed for these two extreme cases. Other examples of noncoherent systems include electrical circuits, traffic light systems, load balancing systems, protective control systems, liquid level control systems, pumping systems, and automatic power control systems [13–18].

In addition, noncoherent systems are often used to accurately analyze disjoint events [19], dependent events [20], and event trees [6]. The wide range of applications of noncoherent systems has gained the attention of reliability engineers working in safety-critical applications. As a result, several commercial reliability software vendors have extended the support of NOT logic from fault trees to reliability block diagrams [12].

38.5 Types of Fault Trees Analysis

Depending on the objectives of the analysis, FTA can be qualitative or quantitative. In the following subsections, possible results and analysis methods for qualitative and quantitative FTA will be discussed in detail.

38.5.1 Qualitative Analysis

Qualitative analysis usually consists of studying minimal cutsets. A cutset in a fault tree is a set of basic events whose occurrence leads to the occurrence of the TOP event. A minimal cutset is a cutset without redundancy. In other words, if any basic event is removed from a minimal cutset, it ceases to be a cutset.

To find the minimal cutsets of a fault tree, a top-down approach is applied. The algorithm starts at the top gate representing the TOP event of the fault tree and constructs the set of cutsets by considering the gates at each lower level. If the gate being considered is an AND gate, then all the inputs must occur to activate the gate. Thus, the AND gate will be replaced at the lower level by a list of all its inputs. If the gate being considered is an OR gate, then the occurrence of any input can activate the gate. Thus, the cutset being built is split into several cutsets, one containing each input to the OR gate.

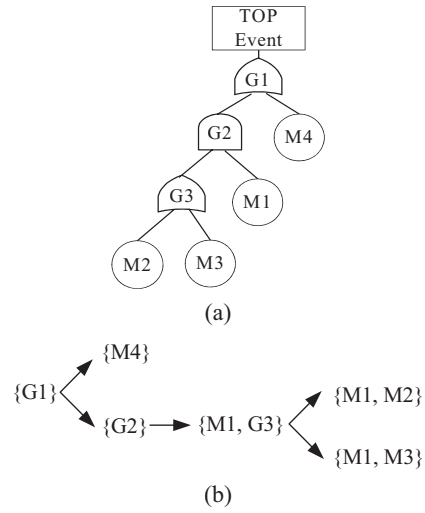

Figure 38.9. An example fault tree and its cutsets. **(a)** Fault tree, **(b)** minimal cutset generation

Consider a fault tree in Figure 38.9(a). Figure 38.9(b) shows its cutset generation. The top-down algorithm starts with the top gate G1. Since G1 is an OR gate, it is split into two sets, one containing each input to G1, that is, {G2} and {M4}. G2 is an AND gate, so it is replaced in the expansion by its two inputs {M1, G3}. Finally, the expansion of G3 splits the cutset {M1, G3} into two, yielding {M1, M2} and {M1, M3}. Therefore, there are three minimal cutsets for this example fault tree: C_1={M1, M2}, C_2={M1, M3}, and C_3={M4}.

Possible results from the qualitative analysis based on minimal cutsets include:

- All the unique combinations of component failures that may result in a critical event (system failure or some unsafe condition) in the system. Each combination is represented by a minimal cutset. For the fault tree in Figure 38.9(a), if both M1 and M2 fail, or both M1 and M3 fail, or M4 fails, the system fails.

- All single-point of failures for the system. A single-point of failure is any component whose failure by itself leads to the system failure. It is identified by a minimal cutset with only a single component. For the fault tree in Figure 38.9(a), M4 is a single-point of failure.
- Vulnerabilities resulting from particular component failures. The vulnerabilities can be identified by considering minimal cutsets that contain the component of interest. For the example system in Figure 38.9(a), once M1 fails, the system is vulnerable to the failure of either M2 or M3.

Those qualitative results can help to identify system hazards that might lead to failure or unsafe states so that proper preventive measures can be taken or reactive measures can be planned.

38.5.2 Quantitative Analysis

Quantitative analysis is used to determine the occurrence probability of the TOP event, given the occurrence probability (estimated or measured) of each basic event. Approaches for quantitative FTA can be broadly classified into three categories: state space oriented methods (see, *e.g.*, [22–26]), combinatorial methods (see, *e.g.*, [27–29]), and a modular solution that combines the previous two methods as appropriate (see, *e.g.*, [30, 31]).

The state space oriented approaches, which are based on Markov chains and/or Petri nets, are flexible and powerful in modeling complex dependencies among system components. However, they suffer from state explosion when modeling large-scale systems. Combinatorial methods can solve large fault trees efficiently. However, a widely held view among researchers is that combinatorial models are not able to model dependencies and thus cannot provide solutions to any dynamic fault tree.

The modular approach combines both combinatorial methods and state space oriented methods. Specifically, in the modular approach, independent subtrees are identified and the decision to use a state space oriented solution or a combinatorial solution is made for a subtree instead of for the fault tree as a whole. These independent subtrees are treated separately and their solutions are integrated to obtain the solution for the entire fault tree. The advantage of the modular approach is that it allows the use of state space oriented approach for those parts of a system that require them and the use of combinatorial methods for the more "well-behaved" parts (static parts) of the system, so that the efficiency of combinatorial solution can be retained where possible. In Section 38.7.2, an example of the modular approach that combines the use of the Markov chain solution for dynamic subtrees and binary decision diagrams based solution for static subtrees will be discussed in detail. The following three sections are devoted to the quantitative analysis techniques for static, dynamic, and noncoherent fault trees, respectively.

38.6 Static FTA Techniques

Quantitative analysis techniques for static fault trees using cutsets or binary decision diagrams will be discussed in this section.

38.6.1 Cutset Based Solutions

In Section 38.5.1, the top-down approach to generate the minimal cutsets from a static fault tree has been described. Each cutset represents a way in which the system can fail. So the system unreliability (denoted by U_{sys}) is simply the probability that all of the basic events in one or more minimal cutsets will occur. Let C_i represent a minimal cutset and there are n minimal cutsets for a system, thus we have:

$$U_{sys} = \Pr(\bigcup_{i=1}^{n} C_i) \cdot \tag{38.1}$$

Because the minimal cutsets are not generally disjoint, the probability of the union in (38.1) is not equal to the sum of the probabilities of the individual cutsets. Actually, for coherent systems, the sum of the individual cutsets gives an upper bound of the system unreliability since the intersection of the events from two minimal cutsets may be counted more than once. Several methods exist for the evaluation of (38.1) [10, 21, 33]. We describe two commonly used ones: inclusion-exclusion and sum of disjoint products.

38.6.1.1 Inclusion–Exclusion (I–E)

The I–E method is a generalization of the rule for computing the probability of the union of two events: $\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B)$. It is given by the sum of probabilities of cutsets taken one at a time, minus the sum of probabilities of the intersection of cutsets taken two at a time, plus the sum of probabilities of the intersection of cutsets taken three at a time, and so on, until reaching an term which contains the probability of the intersection of all the cutsets [8]. The equation for representing the above procedure is:

$$U_{sys} = \Pr\{\bigcup_{i=1}^{n} C_i\} = \sum_{i=1}^{n} \Pr(C_i) - \sum_{i<j} \Pr(C_i \cap C_j) + \sum_{i<j<k} \Pr(C_i \cap C_j \cap C_k) \mp \ldots \pm \Pr(\bigcap_{j=1}^{n} C_j) \tag{38.2}$$

Consider the example system in Figure 38.9, there are three minimal cutsets: C_1={M1, M2}, C_2={M1, M3}, and C_3={M4}. The system unreliability can be calculated as:

$$U_{sys} = \Pr\{C_1 \cup C_2 \cup C_3) = \sum_{i=1}^{3} \Pr(C_i) - \Pr(C_1 \cap C_2) - \Pr(C_1 \cap C_3) - \Pr(C_2 \cap C_3) + \Pr(C_1 \cap C_2 \cap C_3)$$

The evaluation of (38.2) gives the exact system unreliability. As each successive summation term is calculated and included into the sum, the result alternatively overestimates (if the term is added) or underestimates (if the term is subtracted) the desired system unreliability. Hence, lower and upper bounds on the system unreliability can be determined by using only a portion of the terms in (38.2).

38.6.1.2 Sum of Disjoint Products (SDP)

The basic idea of the SDP method is to take each minimal cutset and make it disjoint with each preceding cutset using Boolean algebra, as shown in (38.3):

$$\bigcup_{i=1}^{n} C_i = C_1 \cup (\overline{C_1} C_2) \cup (\overline{C_1}\,\overline{C_2} C_3) \cup \ldots \cup (\overline{C_1}\,\overline{C_2}\,\overline{C_3} \ldots \overline{C_{n-1}} C_n) \tag{38.3}$$

$\overline{C_i}$ represents the negation of the set C_i. Because the terms in the right-hand side of (38.3) are disjoint, the sum of probabilities of these individual terms gives the exact system unreliability, that is,

$$U_{sys} = \Pr(\bigcup_{i=1}^{n} C_i) = \Pr(C_1) + \Pr(\overline{C_1} C_2) + \ldots + \Pr(\overline{C_1}\,\overline{C_2} \ldots \overline{C_{n-1}} C_n) \tag{38.4}$$

Consider the example system in Figure 38.9, the system unreliability using the SDP method will be calculated as: $U_{sys} = \Pr(C_1) + \Pr(\overline{C_1} C_2) + \Pr(\overline{C_1}\,\overline{C_2} C_3)$. Similar to the I–E method, lower and upper bounds on the system unreliability can be obtained by using a portion of the terms in (38.4) [8].

38.6.2 Binary Decision Diagrams

Binary decision diagrams (BDD) were, at first, used in the circuit design and verification as an efficient method to manipulate Boolean expressions [34, 35]. It has recently been adapted to solve a static fault tree model for the system reliability analysis. It has been shown by many studies [36–42] that in most cases, the BDD based method requires less memory and computational time than other methods. Thus, it provides an efficient way to analyze large fault trees.

A BDD is a directed acyclic graph (DAG) based on Shannon decomposition. Let f be a Boolean expression on a set of Boolean variables X and x be a variable of X, then the Shannon decomposition and its if-then-else (*ite*) format is:

$$f = x \cdot f_{x=1} + \overline{x} \cdot f_{x=0} = x \cdot F_1 + \overline{x} \cdot F_2 = ite(x, F_1, F_2)$$

The BDD has two sink nodes, each labeled by a distinct logic value 0, 1, representing the system being operational or failed, respectively. Each non-sink node is associated with a Boolean variable x and has two outgoing edges called *then*-edge (or 1-edge) and *else*-edge (or 0-edge), respectively. The two edges represent the two corresponding expressions in the Shannon decomposition as shown in Figure 38.10. In other words, each non-sink node in the BDD encodes a Boolean expression, or an *ite* format. One of the key

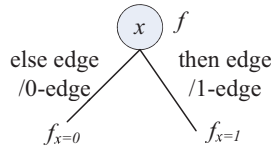

Figure 38.10. A non-sink node in BDD

characteristics of the BDD is the disjointness of $x \cdot f_{x=1}$ and $\bar{x} \cdot f_{x=0}$.

An ordered BDD (OBDD) is defined as a BDD with the constraint that variables are ordered and every source to sink path in the OBDD visits the variables in an ascending order. Further, a reduced OBDD (ROBDD) is an OBDD where each node represents a distinct Boolean expression. Two reduction rules will be introduced in Section 38.6.2.2 to obtain an ROBDD from an OBDD.

To perform a quantitative analysis of a static fault tree using the BDD method, we convert the fault tree to the BDD first, and then evaluate the resulted BDD to yield the system unreliability. In the following, we discuss the conversion and evaluation processes in detail.

38.6.2.1 Converting Fault Trees to BDDs

To construct an OBDD from a fault tree, the ordering of variables/components has to be selected first. The ordering strategy is very important for the OBDD generation, because the size of the OBDD will heavily depend on the order of input variables. A poor ordering can significantly affect the size of BDD, thus the reliability analysis solution time for large systems. Currently there is no exact procedure for determining the best way of ordering variables for a given fault tree structure. Fortunately, heuristics can usually be used to find a reasonable variable ordering [43].

After each variable is assigned a different order or index, a depth-first traversal of the fault tree is performed and the OBDD model is constructed in a bottom-up manner [44]. Specifically, the OBDDs are created for basic events first. Then these basic event OBDDs will be combined based on the logic operation of the current gate traversed. The resulted sub-OBDDs are further combined based on the logic operation of the traversed gate. The mathematical representation of the logic operation on two sub-OBDDs is described as follows.

Let $\Diamond$ represent any logic operation (AND/OR). Let the *ite* format for Boolean expressions G and H, representing two sub-OBDDs, be: $G = ite(x, G_{x=1}, G_{x=0}) = ite(x, G_1, G_2)$ and $H = ite(y, H_{x=1}, H_{x=0}) = ite(y, H_1, H_2)$.

Then:

$$G \Diamond H = ite(x, G_1, G_2) \Diamond ite(y, H_1, H_2) = \begin{cases} ite(x, G_1 \Diamond H_1, G_2 \Diamond H_2) & index(x) = index(y) \\ ite(x, G_1 \Diamond H, G_2 \Diamond H) & index(x) < index(y) \\ ite(y, G \Diamond H_1, G \Diamond H_2) & index(x) > index(y) \end{cases} \quad (38.5)$$

The same rules can be used for logic operation between sub-expressions until one of them becomes a constant expression '0' or '1'. Note that Boolean algebra ($1+x=1$, $0+x=x$, $1 \cdot x=x$, $0 \cdot x=0$) is applied to simplify the representation when one of the sub-OBDDs is a constant expression '0' or '1'.

To illustrate the fault tree to BDD conversion process, we present the construction of the OBDD from the fault tree in Figure 38.9 (a). Assume the variable ordering is M1<M2<M3<M4. Consider the subtree rooted at the OR gate G3; the first path traversed leads to the basic event M2. This means that the OR gate G3 will be applied once OBDDs are built for all the inputs of G3, that is, M2 and M3. Figure 38.11 shows the initial OBDDs for the two basic events M2 and M3 as well as the OBDD resulting from the application of the logic OR gate G3. M2 is the root of the resulted OBDD since it has a lower index than M3.

Figure 38.12 shows the OBDD resulting from the application of the logic AND gate G2 on the OBDD of Figure 38.11 and the OBDD for the basic event M1. M1 is the root of the resulted OBDD since it has a lower index than M2.

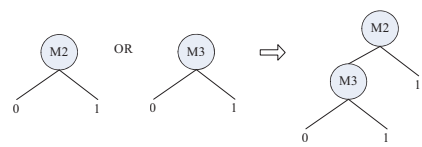

Figure 38.11. OBDD construction up to G3

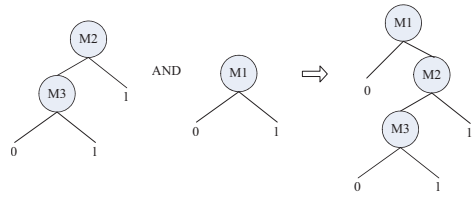

Figure 38.12. OBDD construction up to G2

Figure 38.13 shows the OBDD resulting from the application of OR gate G1 on the OBDDs which represent the inputs of G1, *i.e.,* the OBDD generated in Figure 38.12 and the OBDD for the basic event M4. Since G1 is the top gate of the fault tree, the OBDD in Figure 38.13 gives the full OBDD representing the entire fault tree of Figure 38.9 (a). This graph demonstrates that if both M1 and M2 fail; or if M1 fails, M2 does not fail, and M3 fails; or if M1 fails, M2 and M3 do not fail, and M4 fails; or if M1 does not fail and M4 fails, the entire system fails.

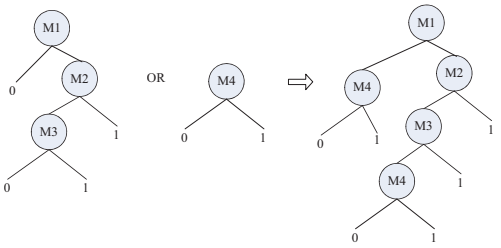

Figure 38.13. OBDD construction up to G1

38.6.2.2 Generating a Reduced OBDD (ROBDD)

As the OBDD is built, the following two reduction rules can be applied to ensure that the OBDD that results is minimal for the chosen ordering:

- Rule#1: isomorphic subtrees are merged since two isomorphic subtrees encode the same Boolean expression. Thus at least one is superfluous and the isomorphic sub-OBDDs can be merged as one sub-OBDD (Figure 38.14). For example, the two sub-BDDs rooted at node M4 in Figure 38.13 are isomorphic and can be merged. Figure 38.15 shows the ROBDD for the fault tree of Figure 38.9 (a) after applying this reduction rule to the OBDD in Figure 38.13.

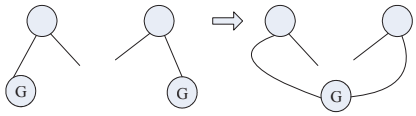

Figure 38.14. Rule#1: merging isomorphic sub-OBDDs

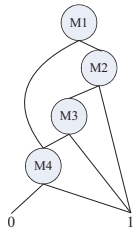

Figure 38.15. An example ROBDD

- Rule#2: deletion of useless nodes. A node encoding a function of form $(x \wedge G) \vee (\bar{x} \wedge G)$ is superfluous and thus can be deleted from the model because the function is simply equivalent to *G* (Figure 38.16).

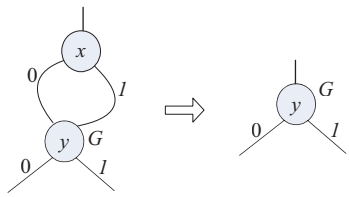

Figure 38.16. Rule#2: deleting useless nodes

38.6.2.3 Calculating System Unreliability

The final BDD model must be evaluated to obtain the system unreliability. Observing the BDD (Figure 38.15) generated for the fault tree in Figure 38.9(a), it is easy to find that each non-sink node in the BDD represents a component that can fail, and each path from the root to a leaf/sink node represents a disjoint combination of component failures and non-failures. If a path leads from a node to its then-edge (or right branch), then the failure of the component should be considered for that path. If a path leads from a node to its else-edge (or left branch), then the non-failure of the component should be considered for that path. If the sink node for a path is labeled with a "1", then the path leads to system failure; if the sink is labeled with a "0" then the path represents the

system being functioning. The probabilities associated with the then-edges on each path are the failure probabilities of corresponding components; the probabilities associated with the else-edges on each path are the operational probabilities of the corresponding components. Because all the paths are disjoint, the system unreliability is given by the sum of the probabilities for all the paths from the root to a sink node labeled "1", or the system reliability is given by the sum of the probabilities for all the paths from the root to a sink node labeled "0".

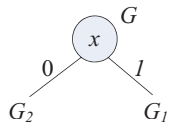

Figure 38.17. An ROBDD branch

The recursive algorithm for evaluating the ROBDD is described as follows. Consider a ROBDD branch G rooted at node x in Figure 38.17. The 1-edge of node x is associated with the failure probability of the component $q(x)$. The 0-edge is associated with the operational probability of the component $1\text{-}q(x)$. The unreliability concerning the sub-BDD G is calculated as: $U(G) = q(x)U(G_1)+[1\text{-}q(x)]U(G_2)$. When x is the rood node of the entire system BDD, $U(G)$ gives the entire system unreliability. The exit condition of this recursive algorithm is: if $G = 0$, then $U(G) = 0$; if $G = 1$, then $U(G) = 1$.

38.6.2.4 Variations of BDDs

Recently the BDD model has been combined with the multistate concept to analyze the reliability of systems subject to imperfect coverage behaviour [45–47], where an uncovered component fault can lead to extensive damage to the entire system despite the presence of fault-tolerant mechanisms [8]. There are three states for a system with imperfect coverage and its components: operation, covered failure and uncovered failure [8]. Readers may refer to Chapter 22 for detailed discussion on imperfect fault coverage. In the multistate BDD based method, each state of the component is represented using a Boolean variable indicating whether the component is in that particular state and the system BDD is generated using these Boolean variables. Because statistical-dependence exists among variables representing different states of the same component, special treatments are needed to address the dependence when applying the traditional Boolean algebra for BDD evaluation [46, 47]. In [42, 48] a similar similar idea was applied to the analysis of general multistate systems in which both the system and its components may exhibit three or more than three performance levels (or states) varying from perfect operation to complete failure. However, the disadvantage of the BDD-based method is that many Boolean variables must be dealt with and dependence among variables representing different states of the same component must be addressed.

To decrease the number of variables involved in the generation and evaluation of the system model, Xing and Dugan first adapted multiple-valued decision diagrams (MDD) [49] for the reliability analysis of fault tolerant systems with imperfect coverage [50, 51]. In their work, a MDD is a directed acyclic graph with three sink nodes each labeled by a distinct logic value 0, 1, 2, representing the system being in the operation, covered failure, and uncovered failure state, respectively. Each non-sink node representing a three-state component is labelled by a ternary-valued variable and has three outgoing edges; one corresponding to each logic value or component state. According to the special characteristic of an uncovered failure, *i.e.*, it leads to the entire system failure [8], a set of special *ternary-valued algebra* rules was developed for performing the logic AND/OR operation on the two basic events. Xing and Dugan showed that the MDD-based method provides smaller models and a much neater and simpler evaluation algorithm for analyzing systems subject to imperfect coverage than the BDD-based method. However, the MDD and rules developed in [50, 51] apply only to systems subject to imperfect coverage, in which both the system and its components have the same set of three states (operation, covered failure, and uncovered failure), and once a component is in the uncovered failure state, the entire system is in that state too. They cannot apply to the general multistate systems in

which the component states may not be consistent with the system states, and characteristics of each state can be nondeterministic.

Recently, a new modelling approach called multistate multivalued decision diagrams (MMDD) has been proposed, which provides an efficient and effective means for analyzing general large-scale multistate systems [52]. Different from the MDD model used in [50, 51], which can have more than two sink nodes representing the system being in each of the multiple states, a MMDD model can have two and only two sink nodes representing the system being or not being in a specific state. Results of case studies in [52] show that the MMDD based method provides smaller models in terms of the number of nodes and much neater and simpler generation and evaluation processes than the BDD-based approach proposed in [42]. Moreover, similar to the BDD-based method, the MMDD model can implicitly represent the sum of disjoint products, each of which indicates a disjoint combination of component states that cause the system to be in a specific state.

38.7 Dynamic FTA Techniques

38.7.1 Markov Chains

Dynamic fault trees (DFT) extend traditional FTA to include dynamic system behavior such as sequence dependence and shared pool of resources. The DFT model includes special purpose gates (dynamic gates described in Section 38.4.2) to incorporate the dynamic behavior into Markov chains, which are used for the solution to the system unreliability analysis.

The two main concepts in the Markov model are system states and state transitions. The state of a system represents a specific combination of system parameters that describe the system at any given instant of time. For representing the system reliability, each state of the Markov model generally represents a distinct combination of faulty and fault free components. The state transitions govern the changes of a state that occur within a system. As time passes and failures occur, the system goes from one state to another until one of the absorbing states (usually the system failure states) is reached. The state transitions are characterized by parameters such as failure rates, fault coverage factors, and repair rates [53].

Solving a Markov model consists of solving a set of differential equations: $\boldsymbol{AP}(t) = \boldsymbol{P}'(t)$. The specific form is:

$$\begin{bmatrix} -\alpha_{11} & \alpha_{21} & \alpha_{31} & \dots & \alpha_{n1} \\ \alpha_{12} & -\alpha_{22} & \alpha_{32} & \dots & \alpha_{n2} \\ \alpha_{13} & \alpha_{23} & -\alpha_{33} & \dots & \alpha_{n3} \\ \dots & \dots & \dots & \dots & \dots \\ \alpha_{1n} & \alpha_{2n} & \alpha_{3n} & \dots & -\alpha_{nn} \end{bmatrix} \bullet \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \\ \dots \\ P_n(t) \end{bmatrix} = \begin{bmatrix} P_1'(t) \\ P_2'(t) \\ P_3'(t) \\ \dots \\ P_n'(t) \end{bmatrix}$$

where $\alpha_{jk}, j \neq k$ is the transition rate from state j to state k. The diagonal element α_{jj} in the matrix $\boldsymbol{A}$ is the sum of departure rates from state j, that is, $\alpha_{jj} = \sum_{k=1,k\neq j}^{n} \alpha_{jk}$. Thus, the sum of each column of $\boldsymbol{A}$ is 0. $P_i(t)$ is the probability of system being in state i at time t, and n represents the number of states presented in the Markov model.

To solve the differential equations, Laplace transform is typically applied [4]. The solution includes the probability of the system being in each state. The system unreliability can be calculated by adding the probability of being each failure state: $\sum_{\forall F} P_{F_i}(t)$, where $P_{F_i}(t)$ is the probability of system being in the failure state F_i at time t.

Markov model has more power as a solution method than the combinatorial methods in that it can solve system with dynamic and dependent behaviors. However, Markov model has the significant disadvantage that its size grows exponentially as the size of the system increases. This rapid growth of the number of states may lead to intractable models. Therefore, many researchers made efforts to the approximate bounding methods where only a portion of the state space of Markov chain is generated [54, 55]. In addition, Markov model assumes *exponential* time-to-failure distribution, whereas, combinatorial methods can be applied to any arbitrary failure distribution. Since Markov and combinatorial approaches both have their *pros* and *cons*, a dynamic fault tree modular approach has been proposed to combine both solutions in the system reliability analysis (refer to the following section for details).

38.7.2 The Modular Approach

Gulati and Dugan [30] presented an exciting hybrid approach, called the modular approach, for the efficient analysis of both static and dynamic fault trees. It provides a combination of BDD solution for static subtrees and Markov chain solution for dynamic subtrees coupled with the detection of independent subtrees. The modular approach allows the use of Markov models for dynamic parts of a system that require them, and use of combinatorial methods for static parts of the system to retain the efficiency of combinatorial solutions where possible.

Specifically, in the modular approach, the fault tree is divided into independent subtrees (subtrees that share no input events) using a fast and efficient algorithm [56]. These independent subtrees are further identified as static or dynamic depending on the relationships between the input events. Static subtree gates express the failure criteria in terms of combinations of events. Dynamic subtree gates express the failure criteria in terms of both combinations of events and order of occurrence of input events.

As an example, the modular approach is applied to the fault tree in Figure 38.18 [57]. The fault tree is divided into four independent subtrees: two static subtrees and two dynamic subtrees, as indicated in Figure 38.18. The static subtrees can be solved using the combinatorial BDD-based method. The dynamic subtrees can be solved using the Markov chain based method.

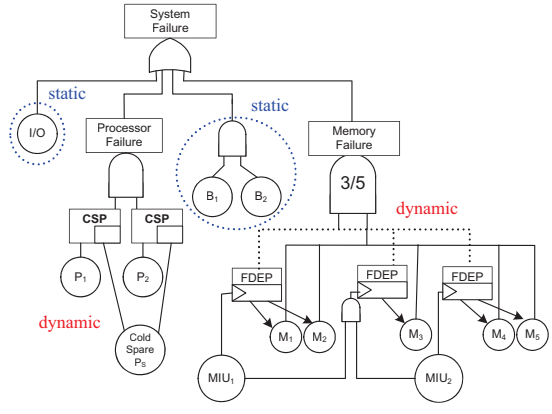

Figure 38.18. The modular approach [57]

Note that modularization is a recursive process as subtrees might themselves contain independent subtrees [30]. Solutions of various independent subtrees are integrated using a relatively straightforward and recursive algorithm to obtain the solution to the entire system.

38.8 Noncoherent FTA Techniques

38.8.1 Prime Implicants

The traditional approach to analyzing a non-coherent system is using prime implicants in the place of minimal cutsets [58]. A prime implicant in a fault tree is a minimal set of basic events whose occurrence or non-occurrence leads to the occurrence of the TOP event (system being unavailable). Similar to the FTA using cutsets, either I–E or SDP method can be applied to obtain the system unavailability based on prime implicants. We use two examples to illustrate the prime implicant based method for the analysis of noncoherent fault trees.

In the first example, we consider a noncoherent fault tree containing an Exclusive OR gate with two inputs: x and y. There are two prime implicants: $I_1 = \{x, \bar{y}\}$ and $I_2 = \{\bar{x}, y\}$. Applying the I-E method, we obtain the expression of system unavailability as:

$$\begin{aligned} \Pr(I_1 \cup I_2) &= \Pr(I_1) + \Pr(I_2) - \Pr(I_1 I_2) \\ &= \Pr(x\bar{y}) + \Pr(\bar{x}y) - \Pr(x\bar{y}\bar{x}y) \\ &= \Pr(x)\Pr(\bar{y}) + \Pr(\bar{x})\Pr(y) - 0 \\ &= \Pr(x)(1-\Pr(\bar{y})) + (1-\Pr(\bar{x}))\Pr(y) \end{aligned} \tag{38.6}$$

Note that $\Pr(I_1 I_2)$ is zero since $I_1 I_2$ contains disjoint events: x and $\bar{x}$; y and $\bar{y}$.

In the second example, we consider the traffic light system used at the crossing of two mono-directional roads (Figure 38.19) [14, 59]. Assume the light functions properly and is RED for road 1 and GREEN for road 2. We define three basic events: event a –car A fails to stop; event b – car B fails to stop; and event c – car C fails to continue. The system has three prime implicants:

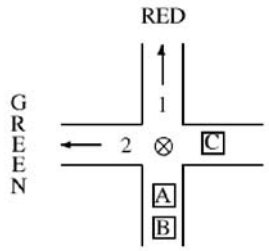

Figure 38.19. Traffic light system

- $I_1 = \{a, \bar{c}\}$: the accident occurs when car *A* fails to stop (*a*) and car *C* moving towards road 2 is crossing ($\bar{c}$);
- $I_2 = \{\bar{a}, b\}$: the accident occurs when car *A* acts properly and stops ($\bar{a}$) and car *B* fails to stop (*b*);
- $I_3 = \{b, \bar{c}\}$: the accident occurs when car *B* fails to stop (*b*) and car *C* continues through the light ($\bar{c}$), no matter what car *A* does.

Define the probability of an event *a* occurring as q_a, and not occurring as p_a. Applying the I–E method, the expression for computing the occurrence probability of an accident is:

$$\begin{aligned}
&\Pr(I_1 \cup I_2 \cup I_3) \\
&= \Pr(I_1) + \Pr(I_2) + \Pr(I_3) - \Pr(I_1 I_2) - \Pr(I_1 I_3) \\
&\quad - \Pr(I_2 I_3) + \Pr(I_1 I_2 I_3) \\
&= \Pr(a\bar{c}) + \Pr(\bar{a}b) + \Pr(b\bar{c}) - 0 - \Pr(a\bar{c}b\bar{c}) \\
&\quad - \Pr(\bar{a}bb\bar{c}) + 0 \\
&= q_a p_c + p_a q_b + q_b p_c - q_a q_b p_c - p_a q_b p_c
\end{aligned} \tag{38.7}$$

38.8.2 Importance Measures

Considerable research efforts have been expended in the component importance analysis for coherent systems and many different importance measures have been proposed for coherent system analysis [4, 60, 61]. However, these measures cannot be directly applied to the analysis of noncoherent systems. In [62, 63] it was proposed to extend four commonly used importance measures for noncoherent systems: 1) Birnbaum's measure [64], 2) component criticality measure [4], 3) Fussell–Vesely measure [65], and 4) initiator and enabler measure [66]. Because Birnbaum's measure is central to the other three importance measures, we discuss it in detail in this section. Readers may refer to [63] for details on the definitions and applications of the other three measures.

Birnbaum's measure of component importance is defined as the probability that a component is critical to system failure, or the probability that the system is residing in a critical state for a component such that its failure causes the system failure [64]. Define:

- $B_i(\underline{q}) \equiv$ Birnbaum's measure of component *i*.
- $q_i(t) \equiv$ the probability that a component *i* is not working at time *t*, it can be either unreliability for a non-repairable system or unavailability for a repairable system.
- $p_i(t) \equiv$ the probability that a component *i* is working at time *t*, *i.e.*, $1 - q_i(t)$.
- $\underline{q} \equiv$ a vector of component unavailability or unreliability for all other components except *i*.
- $Q_{sys}(1_i, \underline{q}) \equiv$ the probability that the system fails with component *i* failed.
- $Q_{sys}(0_i, \underline{q}) \equiv$ the probability that the system fails with component *i* functioning.
- $Q_{sys}(t) \equiv$ the probability that the system fails at time *t*.

The Birnbaum's measure can be expressed as:

$$B_i(\underline{q}) = Q_{sys}(1_i, \underline{q}(t)) - Q_{sys}(0_i, \underline{q}(t)) = \frac{\partial Q_{sys}(t)}{\partial q_i(t)} \tag{38.8}$$

When dealing with a coherent system, the system failure can only be caused by component failures. Therefore, a component in a coherent system can only be *failure-critical*. However, when dealing with a noncoherent system, the system failure can be caused, not only by the failure of a component (referred to as an event *i*), but also by the repair of the component (referred to as event $\bar{i}$). Thus, a component in a noncoherent system can be *failure-critical* or *repair-critical*. These two criticalities must be considered separately because a component can exist in only one state at any time.

Birnbaum's measure for a noncoherent system is given by:

$$B_i(\underline{q}) = B_i^F(\underline{q}) + B_i^R(\underline{q}) \tag{38.9}$$

where $B_i^F(\underline{q})$ represents the component *failure-criticality*, specifically, the probability that the system is in a working state such that the failure of component *i* would cause the system failure; $B_i^R(\underline{q})$ represents the component *repair-criticality*, specifically, the probability that the system is in a working state such that the repair of component *i* would cause the system failure. It has been shown that the failure and repair criticalities can be calculated separately by differentiating $Q_{sys}(t)$ with respect to q_i and p_i, respectively [62]:

$$B_i^F(\underline{q}) = \frac{\partial Q_{sys}(t)}{\partial q_i(t)}, \quad B_i^R(\underline{q}) = \frac{\partial Q_{sys}(t)}{\partial p_i(t)} \tag{38.10}$$

For example, consider the traffic light system in Section 38.8.1. The system unavailability is given in (38.7): $Q_{sys}(t) = q_a p_c + p_a q_b + q_b p_c - q_a q_b p_c - p_a q_b p_c$. According to (38.10), the failure criticality and repair criticality for event *a* are:

$$B_a^F(\underline{q}) = \frac{\partial Q_{sys}(t)}{\partial q_a(t)} = p_c - q_b p_c = p_c(1-q_b) = p_c p_b$$

$$B_a^R(\underline{q}) = \frac{\partial Q_{sys}(t)}{\partial p_a(t)} = q_b - q_b p_c = q_b(1-p_c) = q_b q_c$$

According to (38.9), the Birnbaum's measure of event *a* is: $B_a(\underline{q}) = B_a^F(\underline{q}) + B_a^R(\underline{q}) = p_b p_c + q_b q_c$.

38.8.3 Failure Frequency

Perhaps, the first paper on frequency calculations of noncoherent systems is by Inagaki and Henley [58]. Their method is similar to the method proposed by Vesely for coherent system analysis [67]. For noncoherent systems, prime implicants will be used in the place of minimal cut sets for the failure frequency calculation. According to the method proposed in [58], the expected number of failures within $[t, t + \Delta t]$ is:

$$N(t,t+\Delta t) = \Pr\left\{\bigcup_{i=1}^{p}\theta_i\right\} - \Pr\left\{B\bigcup_{i=1}^{p}\theta_i\right\} \tag{38.11}$$

$$\text{where } B \equiv \bigcup_{i=1}^{p} d_i \text{ and } B\bigcup_{i=1}^{p}\theta_i \equiv B\bigcap\left(\bigcup_{i=1}^{p}\theta_i\right)$$

If Δt is small and is equivalent to the time unit, $N(t,t+\Delta t)$ is equivalent to the failure frequency denoted by $\omega(t)$. It should be noted that $\omega \equiv \omega(t) = \lim_{\Delta t \to 0} \frac{N(t,t+\Delta t)}{\Delta t}$.

Although the method proposed in [58] produces correct results for noncoherent systems, it is unnecessarily complex. The evaluation of (38.11) involves an *NP* problem within each step of another *NP* problem. Therefore, even for the simple example problem considered in [58], a complex procedure is required to solve it. In this section, we describe a simple rule-based method proposed in [59]. The method converts the expression for system unavailability *U* obtained using the calculation procedure of [58] into an expression for ω. The general form of the expression for *U* is the sum of products form: $U = \sum_{i=1}^{m} c_i T_i$, where *m* is the number of product terms, T_i is the product of component availabilities and unavailabilities, and c_i is an integer coefficient that can be negative or positive. For example, in (38.7), the terms are:

i	1	2	3	4	5
c_i	1	1	1	-1	-1
T_i	$q_a p_c$	$p_a q_b$	$q_b p_c$	$q_a q_b p_c$	$p_a q_b p_c$

Each term T_i is in the form of $q_j q_k \ldots p_m p_n$. The general form of T_i is: $T_i = \prod_{j \in F_i} q_j \prod_{k \in S_i} p_k$, where F_i and S_i are the set of component indices corresponding to the unavailability and availability terms in T_i, respectively.

The rule for converting *U* into ω is to multiply every term $c_i T_i$ with the effective rate term $R_i = \sum_{j \in F_i} \alpha_j + \sum_{k \in S_i} \beta_k$, where $\alpha_i = \frac{\omega_i}{q_i} = \frac{p_i \lambda_i}{q_i}$ and $\beta_i = \frac{\nu_i}{p_i} = \frac{q_i \mu_i}{p_i}$. If the system is in the steady-state, then $\alpha_i = \mu_i$ and $\beta_i = \lambda_i$. Refer to [59] for the proof. In simple terms, if T_i is in the form of $q_j q_k \ldots p_m p_n$, then multiply that term with $(\alpha_j + \alpha_k + \ldots + \beta_m + \beta_n)$. Hence, we have: $\omega = c_i T_i R_i$. For example, the R_i terms for (38.7) are:

i	c_i	T_i	R_i
1	1	$q_a p_c$	$\alpha_a + \beta_c$
2	1	$p_a q_b$	$\beta_a + \alpha_b$
3	1	$q_b p_c$	$\alpha_b + \beta_c$
4	-1	$q_a q_b p_c$	$\alpha_a + \alpha_b + \beta_c$
5	-1	$p_a q_b p_c$	$\beta_a + \alpha_b + \beta_c$

Therefore, the failure frequency of the traffic light system (Figure 38.19) is:

$$\omega = q_a p_c(\alpha_a + \beta_c) + p_a q_b(\beta_a + \alpha_b) + q_b p_c(\alpha_b + \beta_c) - q_a q_b p_c(\alpha_a + \alpha_b + \beta_c) - p_a q_b p_c(\beta_a + \alpha_b + \beta_c)$$

38.9 Advanced Topics

38.9.1 Component Importance Analysis

Results from fault tree reliability analysis have been key contributors to system design and tuning activities. However, reliability analysis tells only part of the story; in particular, reliability analysis gives very little information about each individual component's contribution to the entire system failure. Follow-up questions such as "How does a change in one component's reliability affect the entire system reliability?", "How can the entire system reliability be best improved given limited resources?" have to be answered. These and similar questions can be best answered using results of component importance analysis (also called sensitivity analysis) [68].

The importance analysis helps the designer to identify which components contribute most to the system reliability and thus these components would be good candidates for efforts leading to improving the entire system reliability. From the maintenance point of view, the analysis would, by means of a list, tell the repairperson in which order to check the components that may have caused the system failure. Ideally speaking, the maintenance-oriented importance analysis [61] will rank the component whose repair will hasten the system recovery the most, the highest. Section 38.8.2 presents the component importance analysis for noncoherent systems. Xing [72] considers the importance analysis of components in a generalized phased-mission system subject to modular imperfect coverage. Here, we discuss the component importance analysis in the general term.

Two classes of component importance measures have been proposed for the case where the support model is a fault tree: structural-importance (SI) measures and reliability-importance (RI) measures. The SI measures assess the importance of a component to the system operation or reliability by virtue of its position in the fault tree structure, without considering the reliability of the component [70]. Thus, they can be used even if the component reliability is unknown or subject to changes. However, the SI measures cannot distinguish between components that occupy similar structural positions but have drastically different component reliabilities. On the other hand, the RI measures consider both the position of the component in the system and the reliability of the component in question. Thus, the RI measures can generally provide more useful information for generating the ranked list than the SI measures.

Xing [61] studied seven different RI measures, including conditional probability (CP) [71], risk achievement worth (RAW) [60], risk reduction worth (RRW) [60, 70], diagnostic importance factor (DIF), Birnbaum's measure (BM) [4], the criticality importance factor (CIF) [4], and the improvement potential (IP). Refer to [61] for their mathematical definitions as well as physical interpretations. The study in [61] compared the performance of these measures in assisting the system design and maintenance activities. Results of the study show that CP, RAW, and BM may induce misleading conclusions in terms of guiding the system maintenance, though some of these measures serve a good indicator for selecting components that are the best candidates for efforts leading to improving the entire system reliability. RRW, CIF, and IP generally induce reasonable conclusions. However, they give the same importance result for all components in a parallel structure irrespective of the (drastic) difference among the component reliabilities. In addition, the CIF and IP measures become impractical for large dynamic systems because they must be solved using Markov approaches which suffer from the

well-known state explosion problem. Furthermore, the computation of both CIF and IP measures involves the assessment of BM measure that involves simultaneously solving a set of differential equations (the number of equations is the same as the number of states present in the Markov model) for the state occupation probabilities and a much larger set of partial differential equations for the component importance analysis [73]. The solutions to those equations are computationally intensive.

Based on the experimental results obtained in [61], the DIF measure is the most informative and appropriate measure for the maintenance-oriented importance analysis among the nine measures. The DIF measure generally produces the ranking that is consistent with those produced by using the RRW, CIF, and IP measures; it accounts for the effects of exceptionally unreliable component; it can always distinguish components that occupy similar structural positions (for both series and parallel structures) but have different reliabilities.

38.9.2 Common Cause Failures

Common cause failures (CCF) are multiple dependent component failures within a system that are a direct result of a shared root cause or common cause (CC), such as sabotage, flood, earthquake, lightening, power outage, sudden changes in environment, design weaknesses, or human errors. According to [74], CCF are defined as "A subset of dependent events in which two or more component fault states exist at the same time, or in a short time interval, and are direct results of a shared cause." CCF typically occur in systems designed with redundancy techniques, which are characterized by the use of *s*-identical components [75]. It is critical to consider CCF in the system reliability analysis because failure to consider CCF can lead to overestimated system reliability [76, 77].

Considerable research efforts have been expended in the study of CCF for the system reliability modeling and analysis. However, most of these CCF models suffer from various limitations, such as being concerned with a specific system structure [78, 79]; applicable only to systems with exponential time-to-failure distributions [80–82]; being subject to combinatorial explosion as the redundancy level of the system increases [83, 84]; limiting analysis to components being affected by at most a single common-cause [75, 77]; having a single CC that affects all components of a system [79, 85]; or defining CC as being *s*-independent or mutually exclusive [86]. Xing [39] proposed a generic CCF model that addressed these restrictions of the existing CCF models in the reliability analysis of computer network systems by allowing for multiple CC that can affect different subsets of system components, and which can occur *s*-dependently.

Xing [87] utilized the generalized CCF model of [39] and incorporated this CCF model into dynamic fault trees using a new dynamic gate, called CCF gate, for the reliability analysis of hierarchical systems subject to CCF. Moreover, an efficient decomposition and aggregation (EDA) approach was proposed for incorporating CCF into the reliability analysis of hierarchical systems. The basic idea of the EDA approach is to decompose an original reliability problem into a number of reduced reliability problems according to the total probability theorem. The effects of CCF are factored out through the reduction. The reduced problems are represented in a dynamic fault tree model by the CCF gate, which is modeled after the FDEP gate [8]. These reduced problems can be solved using any reliability evaluation approaches that ignore CCF; for example, an efficient one is the BDD based method (Section 38.6.2). The final reliability measure is obtained by aggregating the results of each reduced problem.

Specifically, the EDA approach can be applied in the following three steps:

Step 1: Building common-cause event (CCE) space. Assume the system is subject to *m* common-causes (CC). The *m* CC partition the event space into the 2^m disjoint subsets, each called a CCE:

$$CCE_1 = \overline{CC_1} \cap \overline{CC_2} \cap \ldots \cap \overline{CC_m},$$
$$CCE_2 = CC_1 \cap \overline{CC_2} \cap \ldots \cap \overline{CC_m},$$
$$\ldots\ldots,$$
$$CCE_{2^m} = CC_1 \cap CC_2 \cap \ldots \cap CC_m.$$

A space called "CCE space" (denoted by Ω_{CCE}) is built over this set of collectively exhaustive and mutually exclusive CCE that can occur in the system, *i.e.*, $\Omega_{CCE} = \{CCE_1, CCE_2, ..., CCE_{2^m}\}$. If $\Pr(CCE_j)$ denotes the occurrence probability of CCE_j, then we have $\sum_{j=1}^{2^m} \Pr(CCE_j) = 1$ and $CCE_i \cap CCE_j = \phi$ for any $i \neq j$. Define a common-cause group (CCG) as a set of components that are cause to fail due to the same elementary CC. Let S_{CCE_i} denote the set of components affected by CCE_i, then S_{CCE_i} is simply the union of CCG whose corresponding CC occur. For example, define $CCE_i = \overline{CC}_1 \cap \overline{CC}_2 \cap CC_3$ as a CCE in a system with three elementary CC; S_{CCE_i} is then equal to CCG_3 because CC_3 is the only active elementary CC. For another example, consider $CCE_j = \overline{CC}_1 \cap CC_2 \cap CC_3$, S_{CCE_i} is then equal to $CCG_2 \cup CCG_3$ because both CC_2 and CC_3 are active elementary CC.

Step 2: Generating and solving reduced problems. Based on the CCE space built in step 1 and the total probability theorem, the system unreliability can be calculated as:

$$U_{sys} = \sum_{i=1}^{2^m} [\Pr(\text{system fails} \mid CCE_i) \bullet \Pr(CCE_i)] = \sum_{i=1}^{2^m} [U_i \bullet \Pr(CCE_i)] \quad (38.12)$$

As defined in (38.12), U_i is a conditional probability that the system fails conditioned on the occurrence of CCE_i. The evaluation of U_i is actually a reduced reliability evaluation problem in which the set of components affected by CCE_i do not appear. Specifically, in the system DFT model, each basic event (the failure of a component) that appears in S_{CCE_i} will be replaced by a constant logic value "1" (*true*). After the replacement, a Boolean reduction can be applied to the system DFT to generate a simpler DFT in which all the components of S_{CCE_i} do not appear. Most importantly, the evaluation of the reduced DFT can proceed without further consideration of CCF. The studies in [87] showed that most of DFT after reduction become trivial to solve. In addition, given the fact that systems are usually subject to a small number (m) of root common causes, and considering the parallel processing capability of modern computing systems, even though there are 2^m reduced problems involved in the EDA approach, the overall solution complexity is still low.

Step 3: Integrating for the final result. After obtaining the results for all the reduced problems in (38.12), we integrate them with the occurrence probabilities of CCE, *i.e.*, $\Pr(CCE_i)$ to obtain the final unreliability of the system subject to CCF.

Advantages offered by the EDA approach include: 1) it enables the analysis of multiple CC that can affect different subsets of components within the system, and which may occur *s*-dependently; 2) it allows reliability engineers to use their favorite software package that ignores CCF for computing reliability, and adjust the input and output of the program slightly to produce the system reliability considering CCF. Due to the separation of CCF from the solution combinatorics, the EDA approach has higher computational efficiency and is easier to implement than other potential methods such as Markov methods, which can accommodate CCF by expanding the state space and number of transitions, worsening the state explosion problem [30].

38.9.3 Dependent Failures

In FTA, a common assumption made is that all system components fail independently. However, this is not necessarily true in practical systems. CCF and failure dependence described in the FDEP gate are two examples of dependent failures.

In general, there are two types of dependencies: *positive dependence* and *negative dependence* [4]. *Positive dependence* occurs if the failure of one component leads to an increased tendency for another component to fail. For example, when several components share a common load, the failure of one component may lead to an increased load on the remaining components and thus may

lead to an increased likelihood of failure. N*egative dependence* occurs if the failure of a component leads to a reduced tendency for another component to fail. For example, if an electrical fuse fails open such that downstream circuit is disconnected, the load on the electrical devices in this circuit is removed and thus their likelihood of failure is reduced.

In probability theory, we say that two events E1 and E2 are independent if $Pr(E1 \cap E2) = Pr(E1) \cdot Pr(E2)$ or $Pr(E1|E2) = Pr(E1)$ and $Pr(E2|E1) = Pr(E2)$, meaning that the occurrence of one event has no influence on the occurrence of the other event. A component has a positive dependence when $Pr(E1|E2) > Pr(E1)$ and $Pr(E2|E1) > Pr(E2)$, such that $Pr(E1 \cap E2) > Pr(E1) \cdot Pr(E2)$. A component has a negative dependence when $Pr(E1|E2) < Pr(E1)$ and $Pr(E2|E1) < Pr(E2)$, such that $Pr(E1 \cap E2) < Pr(E1) \cdot Pr(E2)$.

Besides CCF discussed in Section 38.9.2 and functional dependence discussed in Section 38.4.2.1, another type of dependent failure we would like to briefly mention here is cascading failures, also called propagating failures. According to [4], cascading failures are "multiple failures initiated by the failure of one component in the system that results in a chain reaction or domino effect." Cascading failures are common in power grids when one of the elements fails (completely or partially) and shifts its load to nearby elements in the system. Those nearby elements are then pushed beyond their capacity so they become compromised and shift their load onto other elements [88]. Cascading failures may be modeled and analyzed by event trees and fault trees.

38.9.4 Disjoint Events

Disjoint events, also referred to as mutually exclusive events, are events that cannot occur at the same time. For example, two failure modes of a relay: "stuck-open" and "stuck-closed" cannot occur simultaneously. This event dependence can be easily modeled using Markov chains. However, due to the well-known state explosion problem, the Markov chain solution is only practical for small systems. Another alternative is to approximate the mutually exclusive events in a fault tree by stochastically independent events. Thus cutsets containing more than one of mutually exclusive events can occur, leading to incorrect quantitative reliability evaluation, although the errors are usually insignificant. Twigg *et al*. [19] proposed an accurate method to model the mutually exclusive events by converting each of the mutually exclusive events to a subtree that is constructed from ordinary and stochastically independent events as well as logic AND, OR, and NOT gates. Next, we review the basics of the approach through an example fault tree with two disjoint events from [19].

Consider the fault tree in Figure 38.20. Events D1 and D2 are two disjoint events representing two disjoint failure modes of a component D. B and C represent two independent component failure events. Figure 38.20 actually models the two disjoint failure events as independent events, which means that the two failure events D1 and D2 may occur at the same time, leading to errors in reliability calculation. This can also be seen from the cutsets generation.

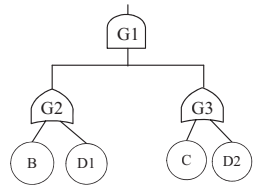

Figure 38.20. Fault tree without modeling disjoint

Apply the top-down approach described in Section 38.5.1, we obtain the minimal cutsets for this fault tree as: {D1, C}, {D2, B}, {B,C}, and {D1, D2}. The minimal cutset {D1, D2} representing the simultaneous occurrence of the two disjoint failure events appears because the dependence between those two events has not been modeled in the fault tree analysis. In the solution of [19], each disjoint event in the original fault tree will be replaced with a disjoint subtree as shown in Figure 38.21.

Specifically, the basic event D1 is replaced with a subtree encoding the Boolean expression of $D1=A \cap A1$, and D2 is replaced with a subtree encoding the Boolean expression of $D2=A \cap \overline{A1}$, where A and A1 are independent arbitrary events

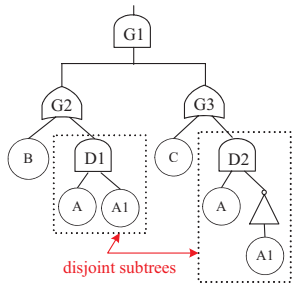

Figure 38.21. Fault tree with modeling of disjoint dependence using disjoint subtrees

and $A = D1 \cup D2$. The minimal cutsets, more accurately, the prime implicants, generated from the new fault tree (Figure 38.21): {B, C}, {A, A1, C}, {A, $\overline{A1}$, B} will be used in the system reliability calculation. Note that the set {A, A1, $\overline{A1}$} was also generated from the top-down approach, but since both A1 and $\overline{A1}$ occur in this set, this set can be automatically removed in the cutset generation.

In general, give a set of *n* mutually exclusive events {D_1, D_2, …, D_n}, each event D_i with probability of π_i. To construct *n* disjoint subtrees with the equivalent occurrence probability π_i, we introduce *n* stochastically independent events: {A, A_1, …, A_{n-1}}, where $A = \bigcup_{i=1}^{n} D_i$, and {$A_1$, …, A_{n-1}} are arbitrary events. The disjoint sets {D_1, D_2, …, D_n} are constructed by subdividing A using A_1, …, A_{n-1} consecutively:

$$
\begin{aligned}
&D_1 = A \cap A_1,\\
&D_2 = A \cap \overline{A_1} \cap A_2,\\
&\ldots\ldots,\\
&D_{n-1} = A \cap \overline{A_1} \cap \ldots \cap \overline{A_{n-2}} \cap A_{n-1},\\
&D_n = A \cap \overline{A_1} \cap \ldots \cap \overline{A_{n-2}} \cap \overline{A_{n-1}}.
\end{aligned}
$$

In particular, the value of *n* is 2 for the example fault tree in Figure 38.20. In general, each disjoint event D_k is converted to a subtree encoding the Boolean function of $D_k = A \cap \overline{A_1} \cap \ldots \cap \overline{A_{k-1}} \cap A_k$. Apparently, the subtree requires one AND gate and (*k*-1) NOT gates. To decrease the number of gates, Morgan's law is applied to D_k:

$$
\begin{aligned}
D_k &= A \cap (\overline{A_1} \cap \ldots \cap \overline{A_{k-1}}) \cap A_k\\
&= A \cap \overline{(A_1 \cup A_2 \cup \ldots \cup A_{k-1})} \cap A_k
\end{aligned}
$$

which requires only three gates: one AND gate, one OR gate, and one NOT gate.

To ensure the subtrees have the same occurrence probabilities as the corresponding disjoint events, Twigg *et al.* [19] derived the probabilities of each independent event in the set {A, A_1, …, A_{n-1}} as:

$$\alpha = \Pr(A) = \Pr(\bigcup_{i=1}^{n} D_i) = \sum_{i=1}^{n} \Pr(D_i) = \sum_{i=1}^{n} \pi_i,$$

$$p_1 = \Pr(A_1) = \frac{\pi_1}{\alpha},$$

$$p_2 = \Pr(A_2) = \frac{\pi_2}{\alpha(1-p_1)} = \frac{\pi_2}{1-\pi_1},$$

……

$$p_k = \Pr(A_k) = \frac{\pi_k}{\pi_{k-1}}\left\{\frac{p_{k-1}}{1-p_{k-1}}\right\} = \frac{\pi_k}{\alpha - \sum_{j=1}^{k-1}\pi_j}.$$

These probabilities are used in the quantitative evaluation of the system unreliability using prime implicants method (Section 38.8.1).

38.9.5 Multistate Systems

A multistate system is a system in which both the system and its components may exhibit multiple performance levels (or states) varying from perfect operation to complete failure [89]. Examples abound in real applications such as communication networks, computer systems, circuits, power systems, and fluid transmission systems [36, 42, 90, 91]. Analyzing the probability of the system being in each state, and thus the reliability of a multistate system is essential to the design and tuning of dependable multistate systems. The difficulty and challenge in analysis arise from the non-binary state property of the system and its components.

Due to the wide use of fault tree in the analysis of systems in other applications, the traditional fault trees have been adapted to model and analyze multistate systems. And the adapted fault trees are called multistate fault trees (MFT) [42]. Similar to the traditional fault tree, a MFT provides a mathematical and graphical representation of the

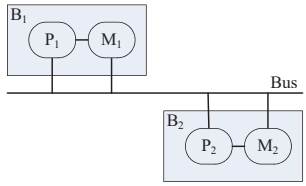

Figure 38.22. An example multistate system

combination of events that can cause the system to occupy a specific state. The quantitative analysis of MFT will be used to determine the probability of system being in that specific state, given the occurrence probabilities of basic events. Each basic event in the MFT represents a component being in a specific state. Also, each MFT consists of a top event representing the system being in a state S_j. The top event is resolved into a combination of events that can cause the occurrence of S_j by means of AND, OR, and *K*-out-of-*N* logic gates.

As an example, consider a multistate computer system that consists of two boards B_1 and B_2 (Figure 38.22) [42]. Each board has a processor and a memory. The two memories (M_1 and M_2) can be shared by both processors (P_1 and P_2) through a common bus. Each board can be considered as a single component with four mutually exclusive and complete states: $B_{i,4}$ (both P and M are functional), $B_{i,3}$ (M is functional, but P is down), $B_{i,2}$ (P is functional but M is down), and $B_{i,1}$ (both P and M are down). Note that $B_{i,j}$ represent the board B_i being in state j, where $i = 1, 2$ and $j = 1, 2, 3, 4$. The entire computer system has three states, which are defined as: S_3 (at least one processor and both memories are functional), S_2 (at least one processor and exactly one memory are functional), and S_1 (no processor or no memory is functional). For illustration purpose, Figure 38.23 shows the MFT for the computer system being in state S_3. Clearly, the system is in state S_3 if the board B_1 is in state 4 and the board B_2 is in state 3 or state 4; or if the board B_1 is in state 3 and the board B_2 is in state 4.

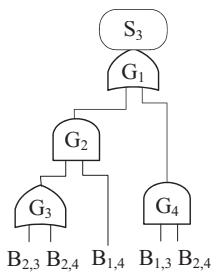

Figure 38.23. MFT of the example system in S_3

Various approaches have been proposed for the analysis of multistate systems; examples include universal moment generating function based methods [91], BDD based methods [38, 42, 92], and MDD based methods [50–52]. Note that among the work, the methods proposed in [38, 50, 51, 92] can only apply to the analysis of multistate systems with multiple failure modes along with a single operational mode, for example, systems subject to imperfect coverage. They cannot directly apply to the general multistate systems, which may contain the states of perfect operation and complete failure, as well as multiple degraded performance levels between those two states. The details of all those approaches for multistate system analysis are outside the scope of this chapter. Readers may refer to the references indicated above for more details.

38.9.6 Phased-mission Systems

A phased-mission system (PMS) is a system used in the mission characterized by multiple, consecutive, and non-overlapping operational phases [38]. During each mission phase, the system has to accomplish a specified task. Since the tasks may differ from phase to phase, the system may be subject to different stresses as well as different reliability requirements. Thus, system configuration, success/failure criteria, and component failure parameters may also change from phase to phase. In the fault tree analysis, the representation of structure functions of a PMS usually requires multiple different fault trees, one for each phase. Further complicating analysis are the statistical dependencies that exist across the phases for a given component

Extensive research has been conducted in the reliability analysis of PMS [38, 41, 46, 51, 72, 92]. Similar to the fault tree analysis methods for non-PMS (Section 38.5.2), the PMS analysis approaches can be classified into three groups: state space oriented approaches based on Markov

chains and/or Petri nets, combinatorial methods, and a modular approach. Readers may refer to Chapter 23 for a state-of-the-art review of these various phased-mission analysis techniques.

38.10 FTA Software Tools

Various software tools have been developed based on the fault trees models. NUREG-0492 [2] summarized available computer software for fault tree analysis and categorized them into five groups. Most of the software codes described in NUREG-0492 were developed in 1970s. In this section, we introduce two software tools that are commonly used by industries and academic research: Galileo dynamic fault tree analysis tool [93] and Relex fault tree analysis software [12]. For details on other available software packages, refer to [94].

Galileo is a dynamic fault tree modeling and analysis tool developed at the University of Virginia [93, 95]. Galileo combines the innovative dynamic fault tree analysis methodology, *i.e.,* the modular approach (Section 38.7.2) with a rich user interface built using package-oriented programming. The important modeling and analysis features of Galileo include: 1) automatic modularization of fault trees and independent solution of modules: efficient BDD based method for static subtrees and Markov chains for dynamic subtrees; 2) multiple time-to-failure distributions (fixed probability, exponential, lognormal, Weibull); 3) imperfect fault coverage modeling in both static and dynamic subtrees; 4) phased mission modeling and analysis; and 5) component importance analysis, *i.e.* sensitivity analysis.

Relex fault tree analysis software [12] supports both quantitative and qualitative analyses, providing computation flexibility based on users' requirements. Relex fault tree analysis tool can compute system unreliability, unavailability, failure frequency, and the number of failures. In addition, it incorporates a minimal cutset (MCS) engine that can quickly determine the minimal cutsets and support interactive, on-screen cutset highlighting. It is the only commercial software package that supports the exact analysis of dynamic fault trees. Relex fault tree analysis tool also supports Lambda-Tau calculations, various importance measures, and noncoherent fault trees.

References

[1] Watson HA. Launch control safety study. Bell Telephone Laboratories, Murray Hill, NJ, USA, 1961.

[2] Vesely WE, Goldberg FF, Roberts NH, Haasl DF. Fault tree handbook. U.S. Nuclear Regulatory Commission, Washington DC, 1981.

[3] Auda DJ, Nuwer K. Effective failure mode effects analysis facilitation. Tutorial Notes of the Annual Reliability and Maintainability Symposium, Alexandria, VA.; Jan. 24–27, 2005.

[4] Rausand M, Hoyland A. system reliability theory: models, statistical methods, and applications (2nd Edition). Wiley Inter-Science, New York, 2003.

[5] Bowles JB, Bonnell RD. Failure modes, effects, and criticality analysis. Tutorial Notes of the Annual Reliability and Maintainability Symposium 1997.

[6] Andrews JD, Dunnett SJ. Event-tree analysis using binary decision diagrams. IEEE Transactions on Reliability 2000; 49(2): 230–238.

[7] IEC61078, Analysis techniques for dependability – Reliability block diagram method. International Electrotechnical Commission, Geneva, 1991.

[8] Dugan JB, Doyle SA. New results in fault-tree analysis. Tutorial Notes of the Annual Reliability and Maintainability Symposium 1997.

[9] NASA, Fault tree handbook with aerospace applications, NASA Office of Safety and Mission Assurance, Washington DC, 2002.

[10] Henley EJ, Kumamoto H. Probabilistic risk assessment. IEEE Press, New York, 1992.

[11] Coppit D, Sullivan KJ, Dugan JB. Formal semantics of models for computational engineering: A case study on dynamic fault trees. Proceedings of the International Symposium on Software Reliability Engineering 2000; 270–282.

[12] Relex software, www.relex.com

[13] Pham H. Optimal design of a class of noncoherent systems. IEEE Transactions on Reliability 1991; 40(3): 361–363.

[14] Amendola A, Contini S. About the definition of coherency in binary system reliability analysis. In: Apostolakis G, Garribba S, Volta G, Editors. Synthesis and analysis methods for safety and reliability studies. Plenum Press, New York, 1978; 79–84.

[15] Jackson PS. Comment on probabilistic evaluation of prime implicants and top-events for non-coherent systems. IEEE Transactions on Reliability 1982; R-31: 172–173.
[16] Jackson PS. On the s-importance of elements and implicants of non-coherent systems. IEEE Transactions on Reliability 1983; R-32: 21–25.
[17] Johnson BD, Matthews RH. Non-coherent structure theory: a review and its role in fault tree analysis. UKAAE, SRD R245, 1983; October.
[18] Wolfram S. Mathematica – A system for doing mathematics by computer. Addison-Wesley, Reading, MA, 1991.
[19] Twigg DW, Ramesh AV, Sandadi UR, Sharma TC. Modeling mutually exclusive events in fault trees. Proceedings of the Annual Reliability and Maintainability Symposium 2000; 8–13.
[20] Twigg DW, Ramesh AV, Sharma TC. Modeling event dependencies using disjoint sets in fault trees. Proceedings of the 18th International System Safety Conference 2000; 275–279.
[21] Misra KB. Reliability analysis and prediction: a methodology oriented treatment. Elsevier, Amsterdam, 1992.
[22] Bobbio A, Franceschinis G, Gaeta R, Portinale L. Exploiting Petri nets to support fault tree based dependability analysis. Proceedings of the 8th International Workshop on Petri Nets and Performance Models 1999; 146 – 155.
[23] Dugan JB, Trivedi KS, Sometherman MK, Geist RM. The hybrid automated reliability predictor. AIAA Journal of Guidance, Control and Dynamics 1991; 9(3): 554–563.
[24] Dugan JB, Bavuso SJ, Boyd MA. Fault trees and Markov models for reliability analysis of fault tolerant systems. Reliability Engineering and System Safety 1993; 39: 291–307.
[25] Hura GS, Atwood JW. The use of Petri nets to analyze coherent fault trees. IEEE Transactions on Reliability 1988; R-37: 469–474.
[26] Malhotra M, Trivedi KS. Dependability modeling using Petri nets. IEEE Transactions on Reliability 1995; R-44: 428–440.
[27] Coudert O, Madre JC. Fault tree analysis: 10^{20} prime implicants and beyond. Proceedings of the Annual Reliability and Maintainability Symposium 1993; 240–245.
[28] Doyle SA, Dugan JB. Analyzing fault tolerance using DREDD. Proceedings of the 10th Computing in Aerospace Conference 1995.
[29] Sinnamon R, Andrews JD. Fault tree analysis and binary decision diagrams. Proceedings of the Annual Reliability and Maintainability Symposium 1996; 215–222.
[30] Gulati R, Dugan JB. A modular approach for analyzing static and dynamic fault trees. Proceedings of the Annual Reliability and Maintainability Symposium 1997.
[31] Sahner R, Trivedi KS, Puliafito A. Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package. Kluwer, Dordrecht, 1996.
[32] Misra KB. New trends in system reliability evaluation. Elsevier, 1993.
[33] Shooman ML. Probabilistic reliability: an engineering approach (2nd Edition). McGraw-Hill, New York, 1990.
[34] Brace K, Rudell R, Bryant R. Efficient implementation of a BDD package. Proceedings of the 27th ACM/IEEE Design Automation Conference 1990; 40–45.
[35] Bryant R. Graph based algorithm for boolean function manipulation. IEEE Transactions on Computers 1986; 35: 677–691.
[36] Chang YR, Amari SV, Kuo SY. OBDD-based evaluation of reliability and importance measures for multistate systems subject to imperfect fault coverage. IEEE Transactions Dependable and Secure Computing 2005; 2(4): 336–347.
[37] Kuo S, Lu S, Yeh F. Determining terminal-pair reliability based on edge expansion diagrams using OBDD. IEEE Transactions on Reliability 1999; 48(3): 234–246.
[38] Xing L, Dugan JB. Analysis of generalized phased-mission systems reliability, performance and sensitivity. IEEE Transactions on Reliability 2002; 51(2): 199–211.
[39] Xing L. Fault-tolerant network reliability and importance analysis using binary decision diagrams. Proceedings of the 50th Annual Reliability and Maintainability Symposium, Los Angeles, CA, 2004.
[40] Yeh F, Lu S, Kuo S. OBDD-based evaluation of k-terminal network reliability. IEEE Transactions on Reliability 2002; 51(4): 443–451.
[41] Zang X, Sun H, Trivedi KS. A BDD-based algorithm for reliability analysis of phased-mission systems. IEEE Transactions on Reliability 1999; 48(1): 50–60.
[42] Zang X, Wang D, Sun H, Trivedi KS. A bdd-based algorithm for analysis of multistate systems with multistate components. IEEE Transactions on Computers 2003; 52(12): 1608–1618.
[43] Bouissou M, Bruyere F, Rauzy A. BDD based fault-tree processing: a comparison of variable ordering heuristics. Proceedings of ESREL Conference 1997.

[44] Coudert O, Madre JC. Metaprime, an interactive fault-tree analyzer. IEEE Transactions on Reliability 1994; 43(1): 121–127.
[45] Xing L. Dependability modeling and analysis of hierarchical computer-based systems. Ph.D. Dissertation, Electrical and Computer Engineering, University of Virginia, 2002; May.
[46] Xing L, Dugan JB. Generalized imperfect coverage phased-mission analysis. Proceedings of the Annual Reliability and Maintainability Symposium, Seattle, WA, 2002; 112–119,
[47] Zang X., Sun H., and Trivedi KS. Dependability analysis of distributed computer systems with imperfect coverage. Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing 1999; 330–337.
[48] Caldarola L. Coherent systems with multistate components. Nuclear Engineering and Design 1980; 58: 127–139.
[49] Miller DM, Drechsler R. Implementing a multiple-valued decision diagram package. Proceedings of the 28th International Symposium on Multiple-valued Logic 1998.
[50] Xing L. Dugan JB. Dependability analysis using multiple-valued decision diagrams. Proceedings of the 6th International Probabilistic Safety Assessment and Management, Puerto Rico 2002.
[51] Xing L, Dugan JB. A separable TDD-based analysis of generalized phased-mission reliability. IEEE Transactions on Reliability 2004; 53(2): 174–184.
[52] Xing L. Efficient analysis of systems with multiple states. Proceedings of the IEEE 21st International Conference on Advanced Information Networking and Applications, Niagara Falls, Canada 2007; 666–672.
[53] Gulati R. A modular approach to static and dynamic fault tree analysis. M. S. Thesis, Electrical Engineering, University of Virginia, August 1996.
[54] Sune V, Carrasco JA. A method for the computation of reliability bounds for non-repairable fault-tolerant systems. Proceedings of the 5th IEEE International Symposium on Modeling, Analysis, and Simulation of Computers and Telecommunication System 1997; 221–228.
[55] Sune V, Carrasco JA. A failure-distance based method to bound the reliability of non-repairable fault-tolerant systems without the knowledge of minimal cutsets. IEEE Transactions on Reliability 2001; 50(1): 60–74.
[56] Dutuit Y, Rauzy A. A linear time algorithm to find modules of fault trees. IEEE Transactions on Reliability 1996; 45(3): 422–425.
[57] Manian R, Dugan JB, Coppit D, Sullivan KJ. Combining various solution techniques for dynamic fault tree analysis of computer systems. Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium 1998; 21–28.
[58] Inagaki T, Henley EJ. Probabilistic evaluation of prime implicants and top-events for non-coherent systems. IEEE Transactions on Reliability 1980; 29(5): 361–367.
[59] Amari SV. Computing failure frequency of noncoherent systems. International Journal of Performability Engineering 2006; 2(2): 123–133.
[60] Dutuit Y, Rauzy A. Efficient algorithm to assess component and gate importance in fault tree analysis. Reliability Engineering and System Safety 2001; 72: 213–222.
[61] Xing L. Maintenance-oriented fault tree analysis of component importance. Proceedings of the 50th Annual Reliability and Maintainability Symposium, Los Angeles, CA, USA. 2004; 534–539,
[62] Andrews JD, Beeson S. Birnbaum's measure of component importance for noncoherent systems. IEEE Transactions on Reliability 2003; 52(2): 213–219.
[63] Beeson S, Andrews JD. Importance measures for non-coherent-system analysis. IEEE Transactions on Reliability 2003; 52(3): 301–310.
[64] Birnbaum ZW. On the importance of different components in a multicomponent system. In: Krishnaiah P, Editor. Multivariate analysis. Academic Press, New York, 1969.
[65] Fussell J. How to hand calculate system reliability characteristics. IEEE Transactions on Reliability 1975; R-24: 169–174.
[66] Barlow RE, Proschan F. Importance of system components and fault tree events. Stochastic Processes and Their Applications 1975; 3: 153–173.
[67] Vesely WE. A time dependent methodology for fault tree evaluation. Nuclear Engineering and Design 1970; 13: 337–360.
[68] Andrews JD, Moss TR. Reliability and risk assessment. Longman Scientific and Technical, Essex, 1993.
[69] Anne A. Implementation of sensitivity measures for static and dynamic subtrees in DIFtree. M.S. Thesis, University of Virginia, 1997.
[70] Chang Y, Amari SV, Kuo S. Computing system failure frequencies and reliability importance measures using OBDD. IEEE Transactions on Computers 2004; 53(1): 54–68.

[71] Papoulis A. Probability, random variables, and stochastic processes (3rd Edition). McGraw-Hill Series in Electrical Engineering, McGraw-Hill, New York, 1991.

[72] Xing L. Reliability importance analysis of generalized phased-mission systems. International Journal of Performability Engineering 2007; 3(3): 303–318.

[73] Frank PM. Introduction to system sensitivity. Academic Press, New York, 1978.

[74] NUREG/CR-4780, Procedure for treating common-cause failures in safety and reliability studies. U.S. Nuclear Regulatory Commission, Washington DC, 1988; Vols. I and II.

[75] Tang Z, Dugan JB. An integrated method for incorporating common cause failures in system analysis. Proceedings of the 50th Annual Reliability and Maintainability Symposium, 610–614, Los Angeles, CA, 2004.

[76] Mitra S, Saxena NR, McCluskey EJ. Common-mode failures in redundant VLSI systems: a survey. IEEE Transactions on Reliability 2000; 49(3): 285–295.

[77] Vaurio JK. An implicit method for incorporating common-cause failures in system analysis. IEEE Transactions on Reliability 1998; 47(2): 173–180.

[78] Bai DS, Yun WY, Chung SW. Redundancy optimization of *k*-out-of-*n* systems with common-cause failures. IEEE Transactions on Reliability 1991; 40(1): 56–59.

[79] Pham H. Optimal cost-effective design of triple-modular-redundancy-with-spares systems. IEEE Transactions on Reliability 1993; 42(3): 369–374.

[80] Anderson PM, Agarwal SK. An improved model for protective-system reliability. IEEE Transactions on Reliability 1992; 41(3): 422–426.

[81] Chae KC, Clark GM. System reliability in the presence of common-cause failures. IEEE Transactions on Reliability 1986; R-35: 32–35.

[82] Fleming KN, Mosleh N, Deremer RK. A systematic procedure for incorporation of common cause events into risk and reliability models. Nuclear Engineering and Design 1986; 93: 245–273.

[83] Dai YS, Xie M, Poh KL, Ng SH. A model for correlated failures in *n*-version programming. IIE Transactions 2004; 36(12): 1183–1192.

[84] Fleming KN, Mosleh A. Common-cause data analysis and implications in system modeling. Proceedings of the International Topical Meeting on Probabilistic Safety Methods and Applications 1985; 1: 3/1–3/12, EPRI NP-3912-SR.

[85] Amari SV, Dugan JB, Misra RB. Optimal reliability of systems subject to imperfect fault-coverage. IEEE Transactions on Reliability 1999; 48 (3): 275–284.

[86] Vaurio JK. Common cause failure probabilities in standby safety system fault tree analysis with testing – scheme and timing dependencies. Reliability Engineering and System Safety 2003; 79(1): 43–57.

[87] Xing L. Reliability modeling and analysis of complex hierarchical systems. International Journal of Reliability, Quality and Safety Engineering 2005; 12(6): 477–492.

[88] Dobson I., Carreras BA, Newman DE. A loading-dependent model of probabilistic cascading failure. Probability in the Engineering and Informational Sciences 2005; 19(1): 15–32.

[89] Huang J, Zuo M. Dominant multi-state systems. IEEE Transactions on Reliability 2004; 53(3): 362–368.

[90] Li W, Pham H. Reliability modeling of multi-state degraded systems with multi-competing failures and random shocks. IEEE Transactions on Reliability 2005; 54(2): 297–303.

[91] Levitin G, Dai YS, Xie M, Poh KL. Optimizing survivability of multi-state systems with multi-level protection by multi-processor genetic algorithm. Reliability Engineering and System Safety 2003; 82(1): 93–104.

[92] Tang Z, Dugan JB. BDD-based reliability analysis of phased-mission systems with multimode failures. IEEE Transactions on Reliability 2006; 55(2): 350–360.

[93] Galileo Dynamic Fault Tree Analysis Tool, http://www.cs.virginia.edu/~ftree/.

[94] Fault Tree Analysis Software, http://www.fault-tree.net/software.html.

[95] Sullivan KJ, Coppit D, Dugan JB. The Galileo fault tree analysis tool. Proceedings of the 29th International Conference on Fault-Tolerant Computing, Madison, Wisconsin, June 15–18, 1999: 232–235.