Fault tree analysis
— an introduction —

Mariëlle Stoelinga, Enno Ruijters

Table of Contents

Chapter 1

Introduction

Fault tree analysis is one of the most often used tools in reliability engineering. It is a graphical technique whose main purpose is to identify potential causes of system failures by systematically breaking down high level system failures into their causal factors. Numerous analytical and statistical techniques are available to compute a wide variety of dependability metrics, such as the system availability and reliability. Hence fault tree analysis is a primary tool to assess if a system meets its safety or reliability requirements, to identify potential problem areas and actions to reduce risk, as well as to compare several design alternatives with respect to their dependability aspects.

Being put forward by Boeing and NASA in the '50s, fault trees are nowadays deployed by a large number of industries in the nuclear, aerospace, automotive, civil engineering, chemical industries. Fault tree analysis can either be qualitative or quantitative. *Qualitative* techniques point to critical paths and components. Typical techniques are the identification of (minimal) cut sets, (minimal) path sets and common cause factors. *Quantitative techniques* aim at computing various dependability metrics, indicating how dependable the system is. These assume that probabilistic information about the components' failure behavior is available, such as failure probabilities, rates or frequencies. Standard metrics are the system *reliability*, i.e., the probability that the system fails within a given mission time; the system *availability*, i.e., the average percentage of time that the system is up; the *mean time to failure (MTTF)*, i.e. the average time for a system to fail. Other quantitative techniques are the computation of importance factors, showing the impact of a particular component on the system dependability.

Fault trees indicate how individual component failures propagate through a system and lead to system failures — Safety-critical systems are often designed so that a single component failures does not lead to a system failure. The leaves of the tree, called *basic events* represent basic component failures, such as "Sensor 378 fails," "Conduit blockage obstructs paddle," or "Laser fails." Fault tree *gates* model how component or subsystem failures combine into higher level failures. The *top event*, also known as top level event, or top (level) undesired event, represents the system failure of interest. Fault tree analysis a top-down methodology in that one starts with the top event, which is refined into its constituting causes via gates, until one arrives at the basic events.

A very wide range of different fault tree variants exist — following [], one can truly speak of a fault tree jungle. Static fault trees are relatively simple in that they only consider boolean gates, like AND and OR. This also makes their analysis procedures relatively simple, even though an important challenge is that fault trees in practice can be huge. Apart from failure probabilities, also repair times can be added. More advanced features in static fault trees include, repair strategies in case of multiple components fail at the same time.

Chapter 2

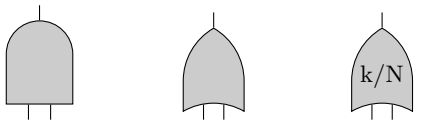
Fault tree models

2.1 Static fault trees

Static (also called standard) fault trees are the most basic kind of fault trees, where all gates are boolean. They appeal as a simple, intuitive and powerful modeling formalism.

2.2 Basic fault tree elements

A fault tree analysis starts by identifying the system failure of interest, represented in the top of the tree, such as "Train stranded due to compressor failure" or "No water flow in output pipe line." The top event, also called top level event, or top (level) undesired event, is then iteratively refined via *gates* that indicate how the event may occur. If no further refinement is possible or desirable, then one arrives at the leaves of the tree, called *basic events (BEs)*. These model the basic failures causes, often component failures, potentially in different failure modes (e.g. a valve stuck open versus stuck closed), or human errors.



(a) AND (b) OR (c) k/N

Fig. 2.1: Gates in standard fault trees

Static fault trees feature three type of gate types, depicted in Fig. 2.1.

- The AND-gate fails if all its children fail;
- The OR-gate fails if at least one of its children fails;
- The VOT(k)-gate fails if at least k of its children fail.

An VOT(k)-gate with N children is often called a VOT(k/N)-gate, or a k-out-of-N gate. Further, an OR-gate is equivalent a to VOT(1)-gate, and an AND-gate with N children is equivalent to a VOT(N)-gate. A fault tree *element* refers to either a gate or basic event. Some SFT variants consider additional gates, like the XOR (exclusive OR) and the NOT-gate [**?**,**?**]. These are discussed in Section **??**, and some additional modeling concepts in Section **??**. Since the failure of an SFT is determined by which basic events fail, and not by their order, SFTs are called *static*.

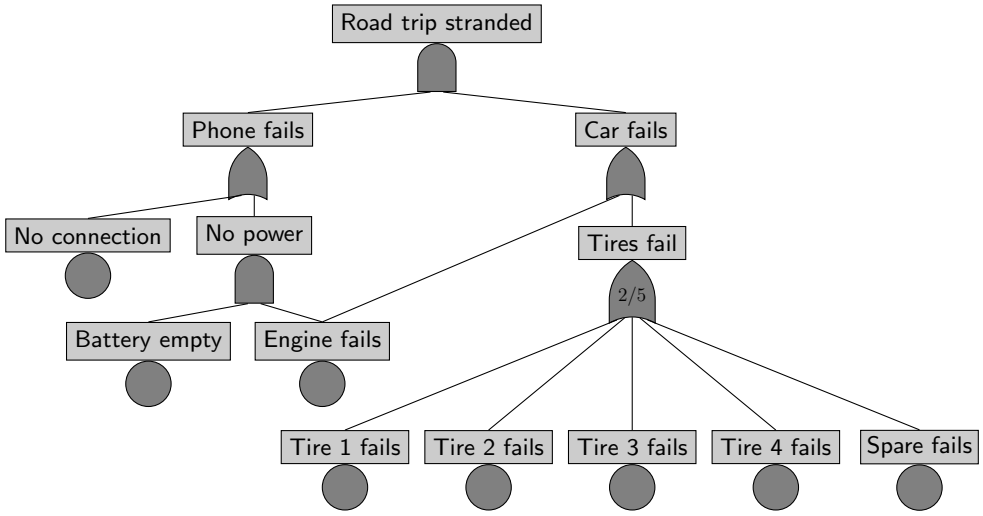


Fig. 2.2: Static fault tree modeling the stranding of a road trip.

Example 1. The fault tree in Figure 2.2 models the safety for a car trip, where a phone is carried to call road services in case of car problems. Hence, the top event is set to Road trip stranded and the road trip fails, if both the car fails and the phone fails, as modeled by the AND-gate to the events Phone fails and Car fails. For simplicity, we list only two causes to refine the Phone fails event: Either there is No connection or No power, where No connection is considered a basic event. Since the car engine can be used to power up an empty phone battery, the event No power occurs if both the battery died and engine fails, as modeled by the AND gate.

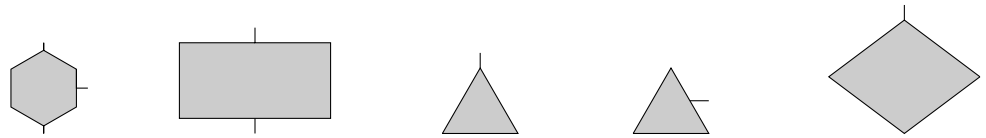
For the car to fail, we simplify to two causes again: the Engine fails, or the Tires fail. Car tires feature a typical mechanism in reliability engineering, namely *redundancy:* the spare tire can take over if one of the primary tires fail. If yet another tire fails, then the tire system fails; thus, the tire system fails if two out of the five tires fail.

This example shows that, technically speaking, fault trees are not trees, but rather directed acyclic graphs, since subtrees can be shared. That is, the output of one element can be input to several gates, as is the case for the BE Engine fails in Figure 2.2. Gates can also be shared, but this must be done is such a way that no cycles arise in the FT, i.e., the output of a gate can not be an input to one of its descendants lower in the tree.

2.3 More fault tree elements

Static fault trees support several other elements, depicted in Figure 2.3 and discussed below. These do not change the behavior of the fault tree, but are important for the purpose of modeling. Further, there are various other gates type that do offer additional functionality to static fault trees. These are discussed in Section **??**.

– *Inhibit gates.* The INHIBIT-gate models that an event can only occur under a certain condition, where the conditioning event is drawn to the right of the INHIBIT gate, see 2.3(a). An INHIBIT



(a) INHIBIT (b) Intermediate event (c) Transfer in (d) Transfer out (e) Undeveloped event

Fig. 2.3: Other fault tree elements

gate behaves the same as an AND gate, where the condition occurs as a basic event. Thus, the INHIBIT gate does provide any additional modeling capabilities, but is used to emphasize the fact that an additional condition must also occur.

– *Intermediate events.* An *intermediate event* represents the failure of a subsystem. As such, it connects two gates, or a BE and a gate. It is depicted by a rectangle, see 2.3(b). Indeed, the text boxes in Figure 2.2 are intermediate events. Intermediate events are crucial for constructing, documenting and understanding a fault tree.
– *Transfer events.* If an FT is too large to fit on one page or screen, then triangles are used to transfer events between multiple FTs to act as one large FT. The *transfer in* gate, shown in Figure 2.3(c), indicates an input that is given by a different tree. Similarly, the *transfer out* gate, shown in Figure 2.3(d), indicates the output of a tree serves as input to another tree.
– *Undevelopped events.* Finally, some leaves may have either insufficient information available or are believed to be of insufficient importance to further consider. These *undeveloped events* are denoted as diamonds, see Figure 2.3(d).

Chapter 3

The role of fault tree analysis in reliability engineering

3.1 Engineering reliable systems

Reliability engineering is a well-established branch of engineering that is concerned with methods and techniques to make systems function as intended, and prevent failures, both during design and operation. Since all components are subject to failure, key to a reliable design is that single component failures do not automatically lead to system failures. A wide variety of measures can be taken to increase the system reliability, and it is the task of the reliability engineer to select the most effective ones, within the given budget. Some typical measures are the following.

- *Redundancy* means that one component or subsystem is replicated multiple times, so that if one of these fails, the system remains operational. Both motors of airplane are realized redundantly: if one of them fails, the other motor has enough power to keep the airplane flying.
- *Spare management* could be seen as a form of redundancy. When the primary component, which is in operations by default, fails, a spare part can take over. A distinction is made between *cold, warm* and *hot* spares. A cold spare is not in operation when the primary is in function, like a spare tire in a car. A hot spare run simultaneously with the primary and can take over immediately in case of a primary failure, like a hard drive that continuously mirrors the data of the primary.
  A warm spare sits between cold and hot, and are prepared to quickly take over the primary, but is not fully operational at all times, for example a back up server that receives updates only periodically.
- *Diversification.* Apart from redundancy, diversification means that one component or subsystem is replicated with different solutions, making them less susceptible to common cause failures. Redundant software modules in nuclear power plants are often written by different teams of programmers, so that if one module contains a bug, the other operates correctly. Cold and warm spares take more time to put into operation, when their primary fails. However, since they are not in use they fail less frequently
- *Fault isolation* ensures that failures do not propagate through the system. Examples here are fire doors, which prevent fires to spread from one location to the other.
- *Fail safe mechanisms* are mechanisms to ensure that, if a system fails, it fails in a safe state. For example, whenever a failure is detected in a railroad signalling system, signals are automatically set to red.

It is widely acknowledged that human errors are as important as technical failures. For instance, if a fire door is left open by users, then its functionality vanishes. Hence, social and organizational measures, such as the training of personnel, clear instructions and procedures to operate machinery, and awareness that these procedures must actually be followed, are as crucial for system reliability as technical measures.

3.2 The role of fault tree analysis in decision making

The key question in reliability engineering is whether a system is reliable enough to be put in operation, and if not, which measures to take to improve the dependability. Fault trees have gained their popularity due to their role in making these decisions, and their aid in the following activities.

- *Understand failure propagation and root causes.* Constructing fault tree models greatly help to understand how failures occur in the system, and what are their root causes: Fault trees provide a systematic and visual way to break down complex failure scenarios into smaller pieces, until the root causes are identified. This better understanding is very helpful to prevent failures to occur, even if no analysis is carried out.
- *Improve system design.* One of the most important role of fault tree analysis is to make systems more reliable, by preventing the top event from happening, or reduce its probability. In particular, fault tree analysis can be use to
  - *Prioritize contributors* leading to the top event. Via quantitative or qualitative system techniques one can investigate which basic events matter most, and find mitigation measures for these events. Mitigating the top risks, rather than controlling all of them is a cost-effective way of risk management.
  - *Compare design alternatives*, that is, to compare different system designs or the effect of different mitigation measures on an existing design. Here, one can use either qualitative of quantitative measure to compare the alternatives.
- *Diagnose a fault that has happened.* For a system failure has that occurred, one can understand how it happened. If an existing tree is available, then one can trace back trace back the failure's root causes by ruling out the branches in the fault tree, based on events that did not occur. Alternatively, one can construct a new tree that identifies the root cause, by developing only those events that have occurred. Or, as SpaceX Elon Musk once wrote on Twitter after the explosion of aSpaceX falcon rocket in 2015: "That's all we can say with confidence right now. Will have more to say following a thorough fault tree analysis."
- *Monitor risks during operations.* Information that is available during system operation can be used to re-assess the system vulnerabilities. For example, if it is known that certain components are broken or disfunctional, then new cut sets can be calculated, and the failure probabilities can be re-assessed. Also, if there is practical evidence that the component failure rates or repair times are estimated too pessimistically, or too optimistically, then the fault tree can be re-assessed as well. Such procedures are commonly deployed in the nuclear industry and is expected to get more important by using sensor technology in combination with big data analytics.
- *Compliance to requirements.* Many governments or regulating bodies ask for provable and traceable evidence that a safety-critical system meets its dependability requirements. For example, such institutions may require the probability on a nuclear meltdown to be below 10^{-9}. Fault tree analysis can help to provide this evidence. Importantly, compliance to the requirements should be a means in reliability engineering, never a goal in itself. If the only goal is to comply to the requirements, it is quite easy to invent a fault tree and failure probabilities that shows compliance, but has little relation to practice.

Chapter 4

Mathematical formulation

Below, we define all concepts above in a mathematically rigorous way.

4.1 Fault trees

Formally, a fault tree is given as a tuple of its constituting elements. To make the definition more generic, we assume a set of *Gates* to be given.

Definition 1. *An* FT *is a 4-tuple* $F = \langle BE, IE, Gate, Inp \rangle$*, consisting of the following components.*

- BE *is the set of* basic events*.*
- IE *is the set of* intermediate events*, with* $BE \cap IE = \emptyset$*. Let* $E = BE \cup IE$ *be the set of all* events*.*
- $Gate : IE \to Gates$ *is a function that maps each intermediate event to a gate.*
- $Inp : IE \to \mathcal{P}(E) \setminus \emptyset$ *is a function that maps each intermediate event to its* inputs (a.k.a. children)*.*

In order to be meaningful, FTs have to meet two well-formedness conditions: (1) $\mathsf{VOT}(k)$*-gates have to have at least* k *children: if* $Gate(g) = \mathsf{VOT}(k)$*, then* $Inp(v) \geq k$*. (2) The graph formed by* $\mathcal{G} = \langle E, Inp \rangle$ *must be acyclic with a unique root TE that is reachable from all other nodes.*

4.2 Structure functions

Mathematically, the behavior of a fault tree F is described by its *structure function.* Given the status of all BEs, where 0 means operational, and 1 means failed, the structure function indicates whether the top level event has failed.

Definition 2. *The structure function of a fault tree* F *is the function* $\Phi_F : \{0,1\}^n \to \{0,1\}$ *that takes as input a status vector* $(b_1, b_2, \ldots b_n)$ *of* n *booleans, where* $b_i = 1$ *if the BE* i *has failed, and* $b_i = 0$ *otherwise.*

We often use a slight reformulation of Φ_F as a function $\Phi_F : \mathcal{P}(BE) \to \{0,1\}$ which takes as input not a status vector, but the set of all failed basic events. Thus, $\Phi_F(E')$ stands for $\Phi_F(b_1, \ldots b_n)$ with $b_i = 1$ if $b_i \in E'$ and $b_i = 0$ if $b_i \notin E'$.

Example 1. The structure function for the fault tree $F_{\mathsf{Road\ trip\ stranded}}$ is given by Boolean formula below, using the abbreviations from Table 4.1 to keep the formulas readable.

$$
\begin{aligned}
(\mathsf{Con} \vee (\mathsf{Bat} \wedge \mathsf{Eng})) \wedge (\mathsf{Eng} &\vee (\mathsf{T1} \wedge \mathsf{T2}) \vee (\mathsf{T1} \wedge \mathsf{T3}) \vee (\mathsf{T1} \wedge \mathsf{T4}) \vee (\mathsf{T1} \wedge \mathsf{TSp}) \\
&\vee (\mathsf{T2} \wedge \mathsf{T3}) \vee (\mathsf{T2} \wedge \mathsf{T4}) \vee (\mathsf{T2} \wedge \mathsf{TSp}) \\
&\vee (\mathsf{T3} \wedge \mathsf{T4}) \vee (\mathsf{T3} \wedge \mathsf{TSp}) \\
&\vee (\mathsf{T4} \wedge \mathsf{TSp}))
\end{aligned}
$$

Leaves		Gates	
Con	No connection	Road trip	Road trip stranded
Bat	Battery empty	Phone	Phone fails
Eng	Engine fails	Power	No power
Ti	Tire i fails	Car	Car fails
Sp	Spare fails	TS	Tires fail

Table 4.1: Abbreviations for elements in the road trip

Deriving the structure function of a fault tree. The structure function of a fault tree can be derived in a top down fashion.

Writing $\Phi_v(\cdot)$ for $\Phi_{\mathsf{Road\ trip}}(\cdot, v)$ fault tree whose top is v, we obtain:

$$\Phi_{\mathsf{Road\ trip}}(\mathsf{Con}, \mathsf{Bat}, \mathsf{Eng}, \mathsf{T1}, \mathsf{T2}, \mathsf{T3}, \mathsf{T4}, \mathsf{Sp}) =$$

$$\Phi_{\mathsf{Phone}}(\mathsf{Con}, \mathsf{Bat}, \mathsf{Eng}, \mathsf{T1}, \mathsf{T2}, \mathsf{T3}, \mathsf{T4}, \mathsf{Sp}) \wedge$$
$$\Phi_{\mathsf{Car}}(\mathsf{Con}, \mathsf{Bat}, \mathsf{Eng}, \mathsf{T1}, \mathsf{T2}, \mathsf{T3}, \mathsf{T4}, \mathsf{Sp}) =$$

$$(\mathsf{Con} \vee \Phi_{\mathsf{Power}}(\mathsf{Con}, \mathsf{Bat}, \mathsf{Eng}, \mathsf{T1}, \mathsf{T2}, \mathsf{T3}, \mathsf{T4}, \mathsf{Sp})) \wedge$$
$$(\mathsf{Eng} \vee \Phi_{\mathsf{TS}}(\mathsf{Con}, \mathsf{Bat}, \mathsf{Eng}, \mathsf{T1}, \mathsf{T2}, \mathsf{T3}, \mathsf{T4}, \mathsf{Sp})) =$$

$$(\mathsf{Con} \vee (\mathsf{Bat} \wedge \mathsf{Eng})) \wedge$$
$$(\mathsf{Eng} \vee \Phi_{\mathsf{TS}}(\mathsf{Con}, \mathsf{Bat}, \mathsf{Eng}, \mathsf{T1}, \mathsf{T2}, \mathsf{T3}, \mathsf{T4}, \mathsf{Sp})) =$$

$$\begin{aligned}&(\mathsf{Con} \vee (\mathsf{Bat} \wedge \mathsf{Eng})) \wedge \\ &(\mathsf{Eng} \vee (\mathsf{T1} \wedge \mathsf{T2}) \vee (\mathsf{T1} \wedge \mathsf{T3}) \vee (\mathsf{T1} \wedge \mathsf{T4}) \vee (\mathsf{T1} \wedge \mathsf{TSp}) \\ &\qquad \vee (\mathsf{T2} \wedge \mathsf{T3}) \vee (\mathsf{T2} \wedge \mathsf{T4}) \vee (\mathsf{T2} \wedge \mathsf{TSp}) \\ &\qquad \vee (\mathsf{T3} \wedge \mathsf{T4}) \vee (\mathsf{T3} \wedge \mathsf{TSp}) \\ &\qquad \vee (\mathsf{T4} \wedge \mathsf{TSp}))\end{aligned}$$

Again, we abbreviated the names of the nodes according to Table 4.1. Phone stands for Phone fails, Car stands for Car fails, Power for No power, and TS for Tires fail.

Formally, the structure function Φ_F of F is defined recursively in terms of the structure functions of its children. To do, we extend the structure function with an extra parameter e that indicates the current event. Thus, $\Phi_F(E', e)$ indicates whether event e fails, given that all BEs in E' fail. Then $\Phi_F(E')$ is defined as $\Phi_F(E', TE)$.

Definition 3. *Let $F = \langle BE, IE, Gate, Inp\rangle$ be a fault tree. The structure function $\Phi_F : \{0,1\}^n \times E \to \{0,1\}$ of F is defined as follows. We write $\boldsymbol{b} = (b_1, b_2, \ldots b_n)$ for the status vector of the BEs, and $Inp(e) = \{e_1, e_2, \ldots e_m\}$ for the set of children of intermediate event e.*

- *If $e \in BE$, then $\Phi_F(\boldsymbol{b}, e) = b_i$*
- *If $e \in G$ and $T(e) = \mathsf{AND}$, then $\Phi_F(\boldsymbol{b}, e) = \Phi_F(\boldsymbol{b}, e_1) \wedge \Phi_F(\boldsymbol{b}, e_2) \wedge \cdots \Phi_F(\boldsymbol{b}, e_m)$.*
- *If $e \in G$ and $T(e) = \mathsf{OR}$, then $\Phi_F(\boldsymbol{b}, e) = \Phi_F(\boldsymbol{b}, e_1) \vee \Phi_F(\boldsymbol{b}, e_1) \vee \cdots \Phi_F(\boldsymbol{b}, e_n)$.*
- *If $e \in G$ and $T(e) = \mathsf{VOT}(k)$, then $\Phi_F(\boldsymbol{b}, e) = (\sum_{i=1}^{m} \Phi_F(\boldsymbol{b}, e_i)) \geq k$.*

As expected, this definition yields that the AND gate with N inputs is semantically equivalent to an $\mathsf{VOT}(N)$ gate, and the OR gate with N inputs is semantically equivalent to a $\mathsf{VOT}(1)$ gate.

4.3 Equivalence and Normal forms

There are many ways to represent a fault tree with the same behavior. For example, the trees F and G in Figure **??** are equivalent. Since the behavior of a fault tree is given by its structure function, two fault trees are equivalent if and only if their structure function is the same. Recall that two functions are the same if, for all values of the input parameters, they yield the same result for the output parameters; their function expressions need not be the same. Indeed, for F and G in Figure **??**, we have $\Phi_F(b_1, b_2, b_3) = \Phi_G(b_1, b_2, b_3)$ for all $b_1, b_2, b_3 \in \{0, 1\}$.

They fail on exactly the same set of BEs, and hence their structure function is exactly the same. Moreover, there are also several ways to write the same structure function, for example ... *Normal forms* provide a standard way to represent the structure function. The binary decision diagram represent a short way to represent a structure function.

Further, when modeling a fault tree it is key to obtain a fault tree that is understandable, rather than obtain the smallest possible fault tree. Normal forms can be long, and are useful for certain analyses.

4.3.1 Conjunctive and disjunctive normal form

It is a classical result that every function can be converted into a disjunctive normal form, i.e., as an OR-gate of AND-gates. This is not surprising, as the if we look at the cut sets. The structure function fails if at least one cut set fails, and for each cuts to fail, we need al of its elements to fail. Hence we obtain:

Similarly to MCSs, a fault tree has a finite number of MPSs. If we denote the set of all MPSs of a fault tree as

$$MP(F) = \left\{ P \subseteq BE \middle| \begin{array}{c} \Phi(F, BE \backslash P) = 0 \quad \wedge \\ \forall_{P' \subset P} : \Phi(F, BE \backslash P') = 1 \end{array} \right\}$$

then we can write a boolean expression for the TE as

$$\Phi_F(...) = \bigwedge_{P \in MP(F)} \bigvee_{x \in P} x$$

Chapter 5

Qualitative analysis

5.1 Cut sets

One of the most fundamental analysis for fault trees is via cut sets, providing both qualitative information on system vulnerabilities, as well as qualitative analysis via metrics.

Definition 1. *A* cut set *is a set of BEs that together cause the fault tree to fail. A* minimal cut set *is a cut set if no elements can be left out from it.*

Example 1. Figure 5.1 illustrates three cut sets for the roa trip fault tree from Figure 2.2.

- $C_1 = \{\text{Battery empty}, \text{Engine fails}, \text{Tire 1 fails}, \text{Spare fails}\}$
- $C_2 = \{\text{No connection}, \text{Tire 1 fails}, \text{Spare fails}\}$
- $C_3 = \{\text{No connection}, \text{Engine fails}\}$

The latter two are minimal; the former is not because $\{\text{Battery empty}, \text{Engine fails}\}$ is also a cut set. A list of all minimal cut sets can be found in Table 5.2.

It is not difficult to see that C_1 is a cut set, by starting from the bottom and working the way up the tree. This is illustrated in Figure 5.1, where black basic elements and gates indicate that they have failed. Since both Battery empty and Engine fails, we have that No power is true. Therefore Phone fails, because this is an or gate. Also, since Tire 1 fails, and Spare fails, so Tires fail, since two of its five inputs to Tires fail fail. Finally, both Phone fails and Car fails, so Road trip stranded. Further, note that $C_4 = \{\text{Battery empty}, \text{Tire 1 fails}, \text{Tire 2 fails}\}$ is not a cut set, because the top level event remains grey.

Computing the list of all (minimal) cut sets is discussed in Section **??**.

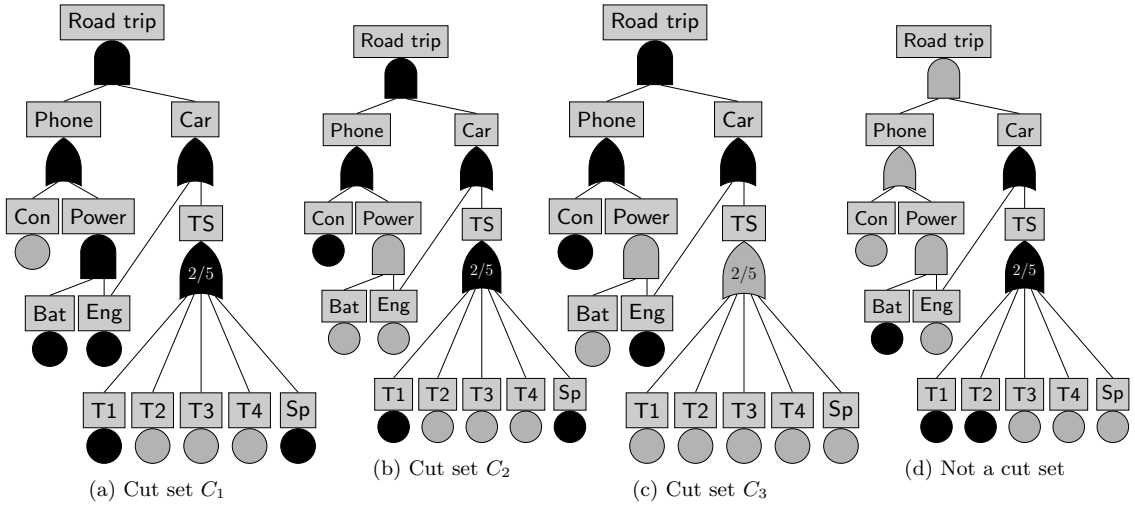
Coherency. An important property of static fault trees is their *coherency*, meaning that a cut set remains a cut set after additional elements to it. This property does no longer hold for fault trees with different gate types, such as the exclusive or (XOR) gate and NOT gate. Therefore, such fault tree extensions require different analysis methods.

5.2 Cut set metrics.

Metrics quantify the importance of a cut set. We define three metrics for cut sets: the order, frequency and probability. These metrics make most sense for minimal cut sets, but their definitions apply to any cut set.

Definition 2. *Let F be a fault tree, and C be cut set of F.*

- The order of C, *is the number of elements in C.*
- The frequency of BE e in a family of cut sets $\mathcal{C}$, *i.e. the number of cut sets containing e.*



(a) Cut set C_1 (b) Cut set C_2 (c) Cut set C_3 (d) Not a cut set

Fig. 5.1: Illustration of cut sets. Black BEs indicate the element of the cut sets. Black gates show failure propagation.

– The probabilities of cut set C. *If each BE e is assigned a probability p_e to fail, then the probability for a cut set $C = \{e_1, \dots e_n\}$ to fail equals $p_{e_1} \cdot \ldots \cdot p_{e_n}$.*

Example 2. Tabels 5.1 and 5.2 illustrate these cut set metrics for the road trip example.

– *Order.* Table 5.2 lists all minimal cut sets, together with their orders and probabilities. We see that there are two cut sets of order 2, the others have order 3.
– *Probabilities.* If we assume the BE failure probabilities as given in Table 5.1, then we obtain the cut set probabilities as given in the last column of Table 5.2. For the sake of clarity, we have assumed unrealistically high failure probabilities: computation steps become less tractable for very small numbers. Further, note that if we sum all probabilities, as listed in the last row of the table, then we overapproximate the exact failure probability of the fault tree, because BEs that appear in multiple cut sets are accounted for twice, see Sectionsec:fta-discrete for details.
– *Frequencies.* Further Table 5.1 also lists frequencies for each BE in the cuts sets from Table 5.2.

These metrics provide important information about the fault tree and the system represented.

Order. Cut sets of low order indicate an unreliable system, especially if one or more elements are likely to fail. It is an important task of a to assess if the (low order) cut sets give rise to design improvements. The cut set $C_3 = \{\mathsf{No\ connection}, \mathsf{Engine\ fails}\}$ in Example 3 is typical here. With only two elements, it has a very low order. Moreover, one of its elements, namely No connection, is likely to occur: connection failures happen regularly, especially in remote areas. Hence, this cut sets reveals a vulnerability; a design improvement can be to replace the regular phone by a satellite telephone, whose connection is much more reliable.

Probabilities. If historic failure data is available, then one can quantify how likely it is that a given cut set fails by computing the cut set probabilities. An advantage of this method is that we can see relative importance of cut sets: Table 5.2 clearly shows that the first two cut sets dominate. Hence,

Basic event	Full name	Probability	Frequency
Con	No connection	0.25	11
Bat	Battery empty	0.2	1
Eng	Engine fails	0.05	2
Ti	Tire i fails	0.01	4
Sp	Spare fails	0.01	4

Table 5.1: Failure probabilities

Minimal cut set	Order	Probability
Con, Eng	2	$0.25 \cdot 0.05 = 0.0125$
Bat, Eng	2	$0.2 \cdot 0.05 = 0.01$
Con, T1, T2	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.0000025$
Con, T1, T3	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.000025$
Con, T1, T4	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.000025$
Con, T1, Sp	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.000025$
Con, T2, T3	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.000025$
Con, T2, T4	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.000025$
Con, T2, Sp	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.000025$
Con, T3, T4	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.000025$
Con, T3, Sp	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.000025$
Con, T4, Sp	3	$0.25 \cdot 0.01 \cdot 0.01 = 0.000025$
TOTAL		$= 0.02275$

Table 5.2: Minimal cut sets ranked according to their probabilities

investing in decreasing the probabilities for those cut has the most impact on the system reliability. Another advantage of this quantitative approach is that we can assess the effect of measures: what happens to the cut set probabilities use a satellite telephone, whose connection fails with probability 0.08?

Frequency. Apart from the order of a cut set, the *frequency* of an an element in a cut is important. If the same element appears in many (low order) cut sets, this also indicates a vulnerability. We see that, again, No connection is a vulnerable element, appearing in 11 cut sets.

5.3 The role of cut sets in fault tree analysis

Cut sets, and especially minimal cut sets provide important information about the fault tree and the system it represents. They serve the important purposes of validation, vulnerability and common cause failures analysis, and quantitative analysis.

– *Validation of the fault tree.* Fault trees for realistic systems can be large, and modeling errors are easily made. Hence, validation of fault tree models is important. One way to do so, is to general all, or a a large number of (minimal) cut sets, and check their reasonableness: is it indeed the case that the failures of these BEs make the system fail?.

– *Pointing out system vulnerabilities.* As mentioned before, a cut set with just a few elements, or with elements whose failure is considered likely, indicates a system vulnerability. Hence, an important task is to assess if the (low order) cut sets give rise to design improvements. Apart from the order of a cut set, the *frequency* of an an element in a cut is important. If the same element appears in many (low order) cut sets, this also indicates a vulnerability, and may give rise to design improvements.
– *Common cause analysis.* Common cause failures, i.e., factors that cause failures in multiple components at the same time, often dominate the system reliability. Examples are production failures, harsh conditions, maintenance errors, etc. In railroad example, nails laying on the road is a typical case. Several methods exists for common cause analysis. One of these methods is via minimal cut sets, where experts assess whether the elements in a minimal cut set are subject to the same failure cause, and if so, this cause is likely to occur.
– *Quantitative analysis.* Minimal cut sets play an important role in quantitative analysis techniques, e.g, to approximate system failure probabilities, see Section **??**. Quantitative analysis itself is important to prioritizing the cut sets, so that effective measures are taken for the most important system risks. Moreover, quantitative analysis plays a key role in showing compliance with the dependability requirements.

Note that the analyses above are especially useful for minimal cut sets: to pinpoint to system vulnerabilities, it is more useful to consider a minimal number of components that cause the system to fail.

5.4 Path sets

A *path set* is the opposite of a cut set: It is a set of basic events such that, if they do not fail, then the system remains operational. A *minimal path set* is a path set from which no elements can be left out. Again, since fault trees are coherent, any superset of a path set is again a path set. Hence, a minimal path set is a path set from which no elements can be removed.

Example 3. The fault tree in Figure 2.2 has, among others, the following minimal path sets.

– $P_1 = \{\mathsf{Battery\ empty}, \mathsf{No\ connection}\}$: If the phone battery is not empty, and there is a connection, the road trip has not failed, because one can call road services.
– $P_2 = \{\mathsf{Tire\ 1\ fails}, \mathsf{Tire\ 3\ fails}, \mathsf{Tire\ 4\ failsSpare\ fails}, \mathsf{Engine\ fails}\}$: If 4 tires and the engine are operational, then one can drive the car, hence the road trip has not failed.
– $P_3 = \{\mathsf{No\ connection}, \mathsf{Engine\ fails}\}$: If there is a connection, and the engine is working, then we can use the engine to power up the phone, so we can call road services.

Just like minimal cut sets, can minimal path sets be used as a starting point for improving system reliability. Especially if there are minimal path sets with just a few elements, then improving the reliability of these BEs may increase the overall system reliability. In practice, cut sets are used more often than path sets. The computation of path sets is similar to the computation of cut sets, and is discussed in Section **??**.

5.5 Computing cut sets and path sets

Cut sets and path sets via structure functions.

The structure function enables us to define the notions of cut set and minimal cut set mathematically. Indeed, a cut set is a set of basic events for which the structure function yields 1.

Definition 3. *A* cut set *(CS) of a fault tree F is a set $C \subseteq BE$ such that $\Phi_F(C) = 1$. A* minimal cut set *(MCS) of F is a set $C \subseteq BE$ such that $\Phi_F(C) = 1 \wedge \forall_{C' \subset C} \,.\, \Phi_F(C') = 0$.*

Definition 4. *A* path set *of a fault tree F is a set $P \subseteq BE$ such that $\Phi_F(BE \backslash P) = 0$. The set P is a minimal path set if, in addition, $\forall_{P' \subset P} \,.\, \Phi_F(P') = 0$.*

Computing cut sets.

Several algorithms exist to compute the cut sets of a BDD: Boolean methods manipulate the structure function of the BDD. The fastest algorithms depend on Binary Decision Diagrams. Finally, there are approximation algorithms.

Boolean methods. One of the first algorithms for finding

The classical methods of determining minimal cut sets are the bottom-up and the top-down algorithms [1]. These algorithms represent each gate as a Boolean expression over BEs and gates. These expressions are combined, expanded, and simplified into an expression that relates the top event to the BEs without any gates.

This expression is called the *structure function*. At every step, the expressions are converted into disjunctive normal form (DNF), so that each conjunction is an MCS.

BDD-methods. Binary decision diagrams BDD are a compact way to encode a Boolean function as a directed acyclic graph [2]. BDDs have been successfully exploited to represent the FT's structure function, speeding up several analysis methods, including the computation of cuts sets, minimal cut sets and several dependability metrics.

To represent a boolean function $f : \{x_1, x_2, \ldots x_n\} \rightarrow \{0, 1\}$, one assumes an order on the variables, i.e. $x_1 < x_2 < ...x_n$. Variable x_1 appears at the top of the BDD, variable x_n appear just above the leaves; the leaves of the BDD are either 0 or 1. Nodes labeled with x_i have two successors, both labeled with x_{i-1}: the left child represents the function in case $x_i = 0$ and is usually depicted with a dashed line to x_{i-1}; the right child represents the function $x_i = 1$ is usually depicted with a solid line to x_{i-1}. We note that the chosen variable order has a large impact of the size of the BDD, and therefore on the efficiency of the analysis method; it does not impact the correctness of the results. Figure 5.2 shows the BDD encoding a fragment of the road trip FT with different variable orderings.

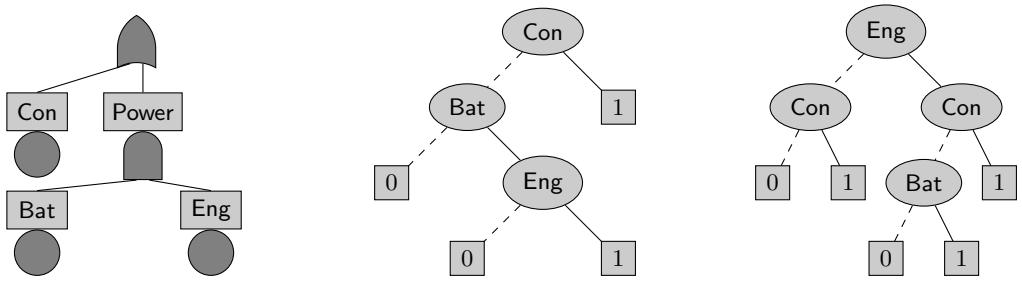
To construct a BDD from a boolean formula, one uses the Shannon expansion formula [2] to construct the top node x_1 of the BDD. [3,4].

$$f(x_1, x_2, \cdots, x_n) = (\neg x_1 \wedge f(0, x_2, \cdots, x_n)) \vee (x_1 \wedge f(1, x_2, \cdots, x_n))$$

and one applies the Shannon expansion to the children $f(0, x_2, \cdots, x_n)$ and $f(1, x_2, \cdots, x_n)$. Recursively applying this expansion until all variables have been converted into BDD nodes yields a complete BDD. Cut sets can be then determined from the BDD by starting at all 1-leaves of the tree, and traversing upwards toward the root.

Example 4. Figure 5.2 shows the conversion of an FT into a BDD. Each circle represents a BE, and has two children: a 0-child that determines the system status if this BE has not failed, and a 1-child for if it has. The leaves of the BDD are squares containing 1 or 0 if the system has resp. has not failed. By traversing the path leading to a 1 node, we obtain:

– The BDD in Figure 5.3c yields as cut sets: $\{\mathsf{Bat}, \mathsf{Eng}\}$ and $\{\mathsf{Con}\}$.



(a) Fault tree fragment (b) Ordering $\mathsf{Con} < \mathsf{Bat} < \mathsf{Eng}$ (c) Ordering $\mathsf{Eng} < \mathsf{Con} < \mathsf{Bat}$

Fig. 5.2: Example conversion of SFT to BDD, with two different variable orderings.

– The BDD in Figure 5.2c yields as cut sets: $\{\mathsf{Con}\}$, $\{\mathsf{Bat}, \mathsf{Eng}\}$ and $\{\mathsf{Con}, \mathsf{Eng}\}$.

All these are cut sets, but the last one is not minimal. The section below explains how to obtain a BDD representing the minimal cut sets.

Example 5. The derivation of the BDDs for the fault tree is as follows. The structure function for this fault tree is given by $\Phi = \mathsf{Con} \vee (\mathsf{Bat} \wedge \mathsf{Eng})$. For the variable ordering $\mathsf{Con} < \mathsf{Bat} < \mathsf{Eng}$, we obtain:

$$\Phi(\mathsf{Con}, \mathsf{Bat}, \mathsf{Eng}) = (\neg\mathsf{Con} \wedge \Phi(0, \mathsf{Bat}, \mathsf{Eng})) \vee (\mathsf{Con} \wedge \Phi(1, \mathsf{Bat}, \mathsf{Eng}))$$

Since $\Phi(1, \mathsf{Bat}, \mathsf{Eng}) = 1 \vee (\mathsf{Bat} \wedge \mathsf{Eng}) = 1$, the right branch of the x_1 immediately connects to the 1-leaf. To obtain the left branch of x_1, we have to consider $\Phi_L(\mathsf{Bat}, \mathsf{Eng}) = \Phi(0, \mathsf{Bat}, \mathsf{Eng})) = \mathsf{Bat} \wedge \mathsf{Eng}$. We obtain

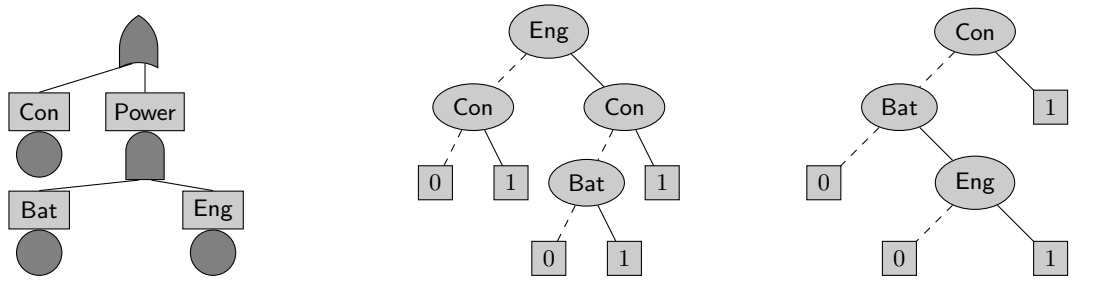
$$\Phi_L(\mathsf{Bat}, \mathsf{Eng}) = (\neg\mathsf{Bat} \wedge \Phi_L(0, \mathsf{Eng})) \vee (\mathsf{Bat} \wedge \Phi_L(1, \mathsf{Eng}))$$

Now we observe that $\Phi_L(0, \mathsf{Eng}) = 0 \wedge \mathsf{Eng} = 0$, so the left branch connect to the 0-leaf. We also see that $\Phi_L(1, \mathsf{Eng})) = 1 \wedge \mathsf{Eng} = \mathsf{Eng}$. The BDD encoding of Eng is also easy to obtain, since it takes the value 0 in case $\mathsf{Eng} = 0$ and 1 if $\mathsf{Eng} = 1$.

Computing minimal cut sets. The set of all BEs reached by traversing a 1-edge from a particular leaf forms one CS. To obtain a minimal cut set, one needs to adapt the BDD construction procedure. The basic insight is that the minimal cut sets are given by the formula:

$$\begin{aligned} f_{\min}(x_1, x_2, \cdots, x_n) = &(\neg x_1 \wedge f_{\min}(0, x_2, \cdots, x_n)) \\ &\vee (x_1 \wedge f_{\min}(1, x_2, \cdots, x_n) \wedge \neg f(0, x_2, \cdots, x_n)) \end{aligned}$$

To understand the rationale behind this formula, we observe that the status vector $x_1, x_2, \ldots x_n$ represents a minimal cut set in the following cases: (1) if $x_1 = 0$, then $x_2, \ldots x_n$ must represent a minimal cut set. (2) if $x_1 = 1$, then $x_2, \ldots x_n$ must represent a minimal cut set, and moreover, $0, x_2, \ldots x_n$ must not be a cut set. Indeed, if $0, x_2, \ldots x_n$ is a cut set — either minimal or not — then $1, x_2, \ldots x_n$ is not a minimal cut set. This is exactly what is expressed in the formula above.



(a) Fault tree fragment (b) Ordering Eng $<$ Con $<$ Bat (c) Ordering Con $<$ Bat $<$ Eng

Fig. 5.3: Example conversion of SFT to BDD representing minimal cut sets

One can compute the BDD for $f_{\min}$ via its Shannon expansion, or from the formula above by using the difference operator on BDDs. Alternatively, one can construct a BDD encoding the prime implicants. We refer the reader to [5] for more details.

While the conversion to a BDD has exponential worst-case complexity, it has linear complexity in the best case. In practice, BDD methods are usually faster than boolean manipulation. This is strongly influenced by the fact that BDDs very compactly represent boolean functions with a high degree of symmetry [6], and fault trees exhibit this symmetry as the gates are symmetric in their inputs.

Approximation of cut sets. Further, cut set analysis for large fault trees can be complex, both in terms of their computation and their interpretation. Therefore, minimal cut sets are often approximated. Either one simply disregards minimal cut sets that are too large. In Example **??**, one may consider only minimal cut sets of order 2. This means that the failure probabilities are underapproximated. Alternatively, one may truncate cut sets, i.e. stop adding elements after a certain order. Thus, one ends up with subsets of minimal cut sets, which are then no cut sets. In this case, the failure probabilities are overapproximated. Hence this method is safe: if the system is reliable enough for the overapproximated probabilities, then it is certainly safe for the real probabilities.

However, such approximations are less suitable to prioritize the cut sets, since these need a more precise insight how the various probabiliies compare.

Computing path sets

Any algorithm to compute MCSs can also be used to compute MPSs. To do so, the FT is replaced by its dual: AND gates are replaced by OR gates, OR gates by AND gates, k/N voting gates by (N-k)/N voting gates, and BEs by their complement (i.e. 'component failure' by 'no component failure'). The MCSs of this dual tree are the MPSs of the original FT [7].

Chapter 6

Quantitative analysis

Whereas qualitative analysis pinpoints to critical factors and root causes in a system design, quantitative analysis methods derive relevant numerical values for fault trees. Quantitative techniques require more effort, since quantitative data must be collected, assessed and maintained. However, quantitative techniques are important when provable or traceable evidence is needed that a system meets its dependability requirements. Further, quantitative analysis is useful when comparing design alternatives, as well as the effect that a design improvement has on the reliability.

Stochastic measures are wide spread, as they provide useful information such as failure probabilities. Typical measures are the system reliability, availability, mean time to failure, etc. Reliability requirements are often phrased in terms of these measures. For instance, the probability for nuclear power plant to fail within 10 year must be lower than 10^{-12}. During operation, stochastic measures can be used to decide whether it is safe to continue operating a system given certain component failures, or whether repairs or renewals should be performed. *Importance measures* indicate how important a set of components is to the reliability of the system. Moreover, *sensitivities* are important to identify how much these measures change with variations in BE probabilities.

Dependability metrics

Fault trees are an important tool for *probabilistic safety assessment*, which investigates if the failure probabilities of a system are acceptable. Probabilistic fault tree analysis assumes that the basic events are equipped with probabilistic information, from which a wide number of stochastic measures are computed.

One of the most common measures is the *system reliability*, or *reliability* for short, defined as the probability that the system does not fail within its mission time T. Computations often use the *(system) unreliability*, being the probability that the system does fail within its mission time T.

$$\mathsf{Reliability} = \mathbb{P}[\text{does not fail before time } T]$$
$$\mathsf{Unreliability} = \mathbb{P}[\text{fails before time } T] = 1 - \mathsf{Reliability}$$

From the reliability, one can compute the *mean time to first failure (MTFF)*, i.e., the average time that it takes of for a system to fail.

$$\mathsf{MTTF} = \mathbb{E}[\text{next system failure appears at time } T]$$

When failed components can be repaired, a system can, after a failure, become operational again. The *(system) availability* is the average percentage of time that a system is operational.

$$\mathsf{Availability} = \mathbb{E}[\text{system failure appears at time } T]$$

Computation of the system availability requires the BEs to be equipped To compute the availability, each BE needs to be equipped with a repair time — indeed, instantaneous repairs make the availability 100%/ For the system availability to make sense,

Probabilistic fault tree analysis can be divided into two classes: approaches in *discrete time*, and approaches in *continuous time*. The discrete time approach considers a fixed time horizon T and studies the probability that the system fails within time T. Hence, each basic event e gets assigned a probability p_e that the system fails within T from which the probability that the fault tree fails is computed. The continuous time approach studies the evolution of the failure probabilities over time. Hence, each basic event e gets assigned a continuous probability distribution f_e. Examples here are the exponential, Weibull and normal probability distributions.

Failures of BEs are assumed to be stochastically independent. If the FT has shared subtrees, then the failures of the gates need however not be independent. Thus, stochastic dependencies between BEs can be modeled by including additional gates, see Section **??**. This is a common way to include common cause failures.

6.1 Probabilistic fault tree analysis in discrete time

As stated, the discrete time case considers a fixed time horizon T, and the probability that the system fails before time T. Thus, we equip each BE e with a probability p_e for the BE to fail within time T, from which we compute the probability p_v for a gate v to fail within time T. Hence, we define for all $v \in V$

$$p_v = \mathbb{P}[v \text{ fails before time } T].$$

The time horizon T is often implicit, i.e., for the sake of brevity we write $\mathbb{P}[v \text{ fails}]$ for $\mathbb{P}[v \text{ fails before time } T]$. This notation is slightly informal, since all systems fail eventually, so the probability that a system fails is usually 1. A rigorous formulation of these notions in terms of random variables can be found in Section **??**.

The system reliability is given by p_{Top} of the top node Top. Just like the values of the structure function can be derived by propagation the function values from the leaves to the top, so can probabilities p_v be obtained. That is, for a gate $v \in G$ with children $v_1, \dots v_n$, we have

- $p_v = \mathbb{P}[v_1 \text{ fails} \wedge \dots \wedge v_n \text{ fails}]$ if $\mathit{Gate}(v) = \mathsf{AND}$.
- $p_v = \mathbb{P}[v_1 \text{ fails} \vee \dots \vee v_n \text{ fails}]$ if $\mathit{Gate}(v) = \mathsf{OR}$.
- $p_v = \mathbb{P}[\text{There are } i_1, i_2, \dots i_k \text{ with: } v_{i_1} \text{ fails} \wedge \dots \wedge v_{i_k} \text{ fails}]$ if $\mathit{Gate}(v) = \mathsf{VOT}(k)$.

These can be computed via the standard probability laws.

Example 1. Consider the fault tree modeling the stranding of a road trip again. Failure probabilities are indicated in Figure 6.1.

We denote by p_v the probability whether node v fails. We can derive the failure probabilities of the gates by applying the probability laws, working from the leaves to the top.

$$p_{\mathsf{Power}} = p_{\mathsf{Bat}} \cdot p_{\mathsf{Eng}} \tag{6.1}$$

$$p_{\mathsf{Phone}} = p_{\mathsf{Con}} + p_{\mathsf{Power}} - p_{\mathsf{Con}} \cdot p_{\mathsf{Power}} \tag{6.2}$$

$$= p_{\mathsf{Con}} + p_{\mathsf{Bat}} \cdot p_{\mathsf{Eng}} - p_{\mathsf{Con}} \cdot p_{\mathsf{Bat}} \cdot p_{\mathsf{Eng}} \tag{6.3}$$

To obtain p_{TS} we consider the complement probability that the tire system does not fail within the given time horizon. This happens if either 5 or 4 do not fail. The probability that all tires are up,

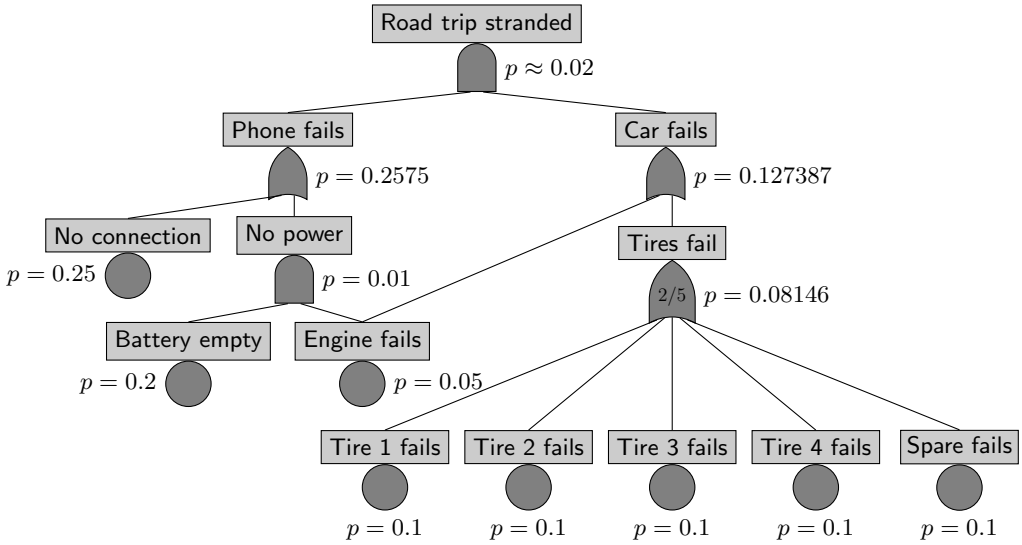


Fig. 6.1: Probabilistic analysis in discrete time.

i.e. none of them fails is given by $(1-p)^5$. Further, the probability that one specific tire i has failed, while all other four are up is given by $p(1-p)^4$. Since there are 5 ways of choosing the specific tire, we obtain $5{\cdot}p(1-p)^4$. Now the probability that the tire system fails is given by

$$p_{\mathsf{TS}} = 1 - (1-p_{\mathsf{T}})^5 - 5{\cdot}p_{\mathsf{T}}(1-p_{\mathsf{T}})^4$$

Since the Car is an OR-gate, we obtain

$$p_{\mathsf{Car}} = 1 - (1-p_{\mathsf{Eng}}){\cdot}\left((1-p_{\mathsf{T}})^5 - 5{\cdot}p_{\mathsf{T}}(1-p_{\mathsf{T}})^4\right) \tag{6.4}$$

Finally, to compute $p_{\mathsf{Road\ trip}}$, we need to take into account the fact that the subtrees are not independent, since they share the Eng gate. To computate, we use Bayes law

$$\mathbb{P}[A] = \mathbb{P}[A|B]{\cdot}\mathbb{P}[B] + \mathbb{P}[A|B^c]{\cdot}\mathbb{P}[B^c]$$

stating that the probability for an event A can be computed by distinguishing two cases: either B happens or B does not happen. However, these cases must be weighted by their respective probabilities to happen. Thus, the probability for A to happen equals the probability for A to happen, given that event B happens and weighted by probability for B to happen, plus the probability for A to happen give that B does not happen and weighted by probability for B to not happen.

To obtain $p_{\mathsf{Road\ trip}}$, we consider two cases: either Eng has failed, or it has not. Thus, we obtain

$$\begin{aligned}\mathbb{P}[\mathsf{Road\ trip\ fails}] =& \mathbb{P}[\mathsf{Road\ trip\ fails}|\mathsf{Eng\ fails}]{\cdot}\mathbb{P}[\mathsf{Eng\ fails}]+\\ & \mathbb{P}[\mathsf{Road\ trip\ not\ fails}|\mathsf{Eng\ not\ fails}]{\cdot}\mathbb{P}[\mathsf{Eng\ not\ fails}]\end{aligned}$$

Further, we use

$$\begin{aligned}\mathbb{P}[\mathsf{Road\ trip\ fails}|\mathsf{Eng\ fails}] &= \mathbb{P}[\mathsf{Phone\ fails} \wedge \mathsf{Car\ fails}|\mathsf{Eng\ fails}]\\ &= \mathbb{P}[\mathsf{Phone\ fails}|\mathsf{Eng\ fails}]{\cdot}\mathbb{P}[\mathsf{Car\ fails}|\mathsf{Eng\ fails}]\end{aligned}$$

The latter equality holds because, under the assumption that the car has failed, the failure of the phone and car are independent. Hence, we can multiply their probabilities. Finally, to obtain $\mathbb{P}[\mathsf{Phone}\text{ fails}|\mathsf{Eng}\text{ fails}]$, use Equation 6.3, and plug in $p_{\mathsf{Eng}} = 1$, since the engine has failed, yielding

$$\begin{aligned}\mathbb{P}[\mathsf{Phone}\text{ fails}|\mathsf{Eng}\text{ fails}] &= p_{\mathsf{Con}} + p_{\mathsf{Bat}}{\cdot}p_{\mathsf{Eng}} - p_{\mathsf{Con}}{\cdot}p_{\mathsf{Bat}}{\cdot}p_{\mathsf{Eng}} \\ &= p_{\mathsf{Con}} + p_{\mathsf{Bat}} - p_{\mathsf{Con}}{\cdot}p_{\mathsf{Bat}}\end{aligned}$$

Similarly, to obtain $\mathbb{P}[\mathsf{Car}\text{ fails}|\mathsf{Eng}\text{ fails}]$, we use Equation 6.3, but with $p_{\mathsf{Eng}} = 0$, since the engine does not fail, yielding

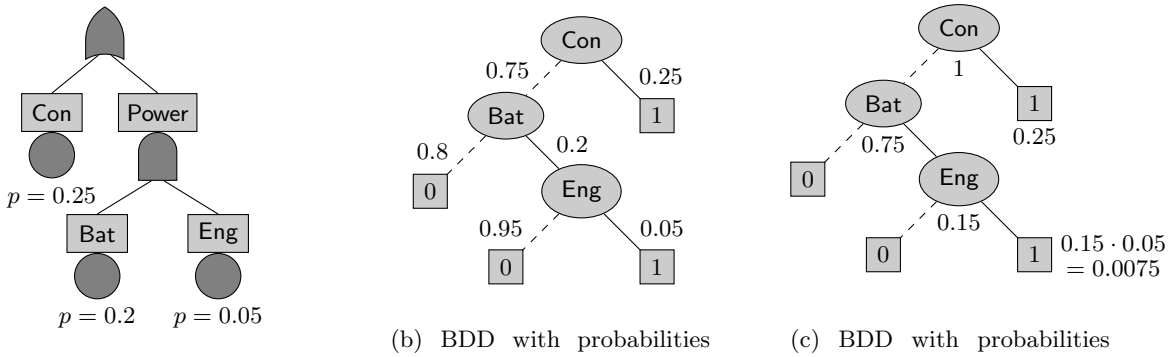
$$\begin{aligned}\mathbb{P}[\mathsf{Phone}\text{ fails}|\mathsf{Eng}\text{ not fails}] &= p_{\mathsf{Con}} + p_{\mathsf{Bat}}{\cdot}p_{\mathsf{Eng}} - p_{\mathsf{Con}}{\cdot}p_{\mathsf{Bat}}{\cdot}p_{\mathsf{Eng}} \\ &= p_{\mathsf{Con}}\end{aligned}$$

In a similar way, we obtain from Equation 6.4 that

$$\begin{aligned}\mathbb{P}[\mathsf{Car}\text{ fails}|\mathsf{Eng}\text{ fails}] = p_{\mathsf{Car}} &= 1 - (1{-}p_{\mathsf{Eng}}){\cdot}(1{-}p_{\mathsf{T}})^5 - 5{\cdot}p_{\mathsf{T}}(1{-}p_{\mathsf{T}})^4 \\ &1 \\ \mathbb{P}[\mathsf{Car}\text{ fails}|\mathsf{Eng}\text{ not fails}] &1 - (1{-}p_{\mathsf{Eng}}){\cdot}(1{-}p_{\mathsf{T}})^5 - 5{\cdot}p_{\mathsf{T}}(1{-}p_{\mathsf{T}})^4 \\ &1 - (1{-}p_{\mathsf{T}})^5 - 5{\cdot}p_{\mathsf{T}}(1{-}p_{\mathsf{T}})^4\end{aligned}$$

We observe that the computation is easy when there are no shared leaves or subtrees, since we can just propagate the probabilities using the probability laws for independent events. Fault trees with shared subtrees require more complex computations via this method. The computation is significantly simpeler when using the BDD representation, as explained in the next section.

Probabilistic fault tree analysis through BDDs



(a) Fault tree fragment

(b) BDD with probabilities decorating the edges

(c) BDD with probabilities propagated downwards

Fig. 6.2: Computation of the system reliability via BDDs

Given the BDD representation, one can easily compute the failure probabilities by multiplying the probabilities over each path leading to the 1-leaf. Assuming that BDD label x_j corresponds to

BE e_i, which is assigned probability p_i. If we walk from x_i to x_j, via a solid arrow, we multiply by p_i, and if we walk via a dashed arrow, we multiply by $1 - p_i$.

This is exemplified in Figure 6.2. To see that this method is correct, we observe the following:

- Every path in the BDD leading to a 1-leaf corresponds to a set of status vectors that make the FT fail.
- The probability for that set of status vectors to occur is obtained by multiplying their probabilities.
- Every path in the BDD corresponds to a disjoint set of status vectors. Hence, their probabilities can be added.

Note that this computation is correct, irrespective of the choosen BDD ordering.

Probabilistic fault tree analysis through cut sets

A common method to compute the failure probabilities in a fault tree is adding the probabilities of all minimal cut sets. This method is illustrated in Table 5.2:

This method is however an overapproximation, rather than an exact computation, of the system unreliability: one cut set can appear in multiple cut sets, and may therefore be counted twice. Further, the cut set method yields an overapproximation of the unreliability , so in that sense it is a safe method: if the cut set method yields a probability p that is lower than the maximally tolerated unreliability T, then the unreliability

6.2 Probabilistic fault tree analysis in continuous time

As stated previously, continuous-time analysis describes the failure behavior of each BE not with a single probability p_e, but rather with a probability distribution over failure times given by the probability density function f_e. That is to say, the probability that BE e fails before time T is given by

$$\mathbb{P}[\text{BE } e \text{ fails before time } T] = F_e(T) = \int_{-\infty}^{T} f(x)\mathrm{d}x$$

Since working with the integral is often rather cumbersome, we instead use the cumulative density function (CDF) F_e directly.

Commonly-used failure time distributions include:

The exponential distribution described by a parameter λ called the *failure rate*. The exponential distribution has a contant per-unit-time probability of failing (i.e., if BE e has not yet failed by time T, the probability that it fails before time $T+\delta$ is determined by λ and δ irrespective of the value of T). Its probability density function is given by $f_\lambda^{\exp}(t) = \lambda e^{-\lambda t}$ and its cumulative density function by $F_\lambda^{\exp}(t) = 1 - e^{-\lambda t}$.

The Weibull distribution described by parameters λ and k, called the *scale* and *shape* parameters, respectively. The Weibull distribution models components whose failure rates vary over time, e.g., due to aging gradually increasing the failure rate or due to manufacturing defects causing high failure rates for new components. The shape parameter k determines how quickly the failure rate changes, with $k < 1$ describing decreasing failure rates and $k > 1$ describing increasing failure rates. The Weibull distribution with $k = 1$ is equal to an exponential distribution with the same parameter λ.

The Weibull distribution has probability density function

$$f^{\text{Weib}}_{(k,\lambda)}(t) = k\lambda\,(t\lambda)^{k-1}\,e^{(-t\lambda)^{k-1}}$$

and cumulative density function

$$F^{\text{Weib}}_{(k,\lambda)}(t) = 1 - e^{-(t\lambda)^k}$$

The normal distribution (also called the *Gaussian* distribution) is described by parameters μ and σ, called the *mean* and *variance*, respectively. The normal distribution is commonly used if a more exact distribution is not known, for example if a component is itself constructed of many subcomponents that are not individually modelled. The mean of the distribution specifies the average failure time, while the variance described how much failure times vary over different components.

The normal distribution has probability density function

$$f^{\text{Norm}}_{(\mu,\sigma)}(t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(t-\mu)^2}{2\sigma^2}}$$

and its cumulative density function does not have a closed-form expression.

As continuous-time analysis considers the evolution of the components over time, it can also consider the possibility that a component fails and is subsequently repaired (and can later fail again, etc.). The repair times are often more complex to describe than failure times due to, e.g., resource constraints (like a mechanic that can only repair one component at a time), external dependencies (such as delivery times for replacement parts) and/or scheduling policies (like repairs preferably being performed during planned downtime). While various formalisms have been developed to model these complexities (e.g., complexRepairs), they are commonly approximated by attaching a probability distribution g_e for the repair time to each BE and assuming that repairs are performed independently.

Analysis of reliability. In the absense of repairs, the computation of system reliability reduces to discrete-time analysis, and the techniques of Section 6 can be applied. Specifically, since we consider coherent fault trees, a failed system never returns to a functioning state, and we only need to consider which components are failed at time T. Thus, we can compute the failure probabilities p_e from the probability distributions F_e (or f_e) using $p_e = F_e(t)$, and apply the aforementioned analysis techniques for the discrete-time case.

If repairs are considered, this approach no longer applies, as it is now possible that a system failure occurred before time T and subsequently repaired. There is no easy general-case expression for the reliability in this case. Nonetheless, several approaches are available for reliability calculations:

Exact analysis for some probability distributions. For example, if the failure and repair time distributions are both exponential, it is possible to express the behavior of the FT as a continuous-time Markov chain (CTMC) An example of such a translation is shown in Figure X. Exact analysis techniques are available to calculate metrics from a CTMC such as the probability of reaching a certain set of states within a time bound, which corresponds to the FT reliability (probability of reaching a set of failed states before time T).

Monte Carlo Simulation. If exact analysis is not feasible due to the distributions used or due to the computational resources required, one can estimate the system reliability using Monte Carlo simulation instead [8]. In this approach, random failure and repair times are generated following the specified probability distributions. The FT is simulated using these times, and the simulator records whether a failure occurs before time T or not. Given enough simulations, each with independently samples failure and repair times, the system reliability can be approximated to arbitrary accuracy.

TODO: find out what approximation is used in FT+

Analysis of availability. Availability is only a useful metric if repair times are provided, as otherwise it is always 0 (at some time the system will fail, and then it will remain failed forever).

In general, three ways are used to calculcate availiability:

Exact analysis for independents basic events. One useful property of availability is that, since we are considering behavior after the system has reached a steady-state, we effectively do not need to consider time-evolution. In particular, if we can assume that the basic events are independent, and we can calculate the steady-state availability of each basic event, we can apply the techniques of discrete-time analysis (Sec. 6.1) to propagate these availability towards the top level event.

The calculation of the availability of a basic event depends on the distributions of failure and repair times. In most cases, these distributions are assumed to be independent of the total time (i.e., a component that fails after one minute takes just as long to repair as a component that fails after ten years), and the steady-state availability is given as $\mathbb{P}_S[\text{not failed}] = \frac{MTTF}{MTTF+MTTR}$ where $MTTF$ is the mean time to failure and $MTTR$ is the mean time to repair.

As an example, exponential distributions are frequently used for both failure and repair times. If a BE is modeled with an exponential failure time distribution with rate λ and an exponential repair time distribution with rate μ, we find that the $MTTF$ is $\frac{1}{\lambda}$ and the $MTTR$ is $\frac{1}{\mu}$. We thus have a mean availability of this BE of $\frac{\frac{1}{\lambda}}{\frac{1}{\lambda}+\frac{1}{\mu}} = \frac{1}{1+\frac{\lambda}{\mu}}$.

Exact analysis for some probability distributions. If basic events are not independent, it is no longer easy to combine availabilities for basic events into the top level availability. Nonetheless, it is still possible to calculate the exact availability for some probability distributions. For example, if all probability distributions are exponential, the FT can be translated into a continuous-time Markov chain, for which steady-state probabilities can be calculated by various tools.

Monte Carlo simulation Like for reliability, exact analysis is not always feasible, so one may estimate the system availability using Monte Carlo simulation. Two approaches may be applied here:

If the FT is known to always eventually return to the fully repaired state, the availability can be estimated by *renewal cycles* A sequence of failures and repairs that take the FT from the fully repaired state, through some intermediate states, back to the repaired state. In this approach,

the simulator samples failure and repair times from the appropriate distributions until the FT has returned to its fully repaired state, recording how much time is spent in total (T_{cycle}) and how much time in the cycle the system is failed (T_{failed}). Once enough such cycles have been samples, we can estimate the availability as $A = 1 - \frac{\mathbb{E}[T_{\text{failed}}]}{\mathbb{E}[T_{\text{cycle}}]}$.
If the FT does not always return to the fully repaired state, e.g. if some components cannot be repaired, the renewal cycle approach cannot be directly applied. In this case, the typical approach is to use some method of steady-state detection to estimate when the simulation has reached steady-state, then sample some further time, recording the availability of that time (A_{run}) and using these measures to estimate the total availability. This approach has the disadvantage of not providing even strict statistical guarantees (as there is no general way to verify that steady-state has been reached or that enough time has been sampled in the steady state), but it can still provide useful estimates.

Mean Time to Failure The mean time to failure ($MTTF$) is a useful metric for any FT that is guaranteed to eventually fail (which is true for most FTs). Unfortunately, unlike for reliability and availability, it is not generally possible to combine $MTTF$s for basic events into $MTTF$s for gates. Therefore, the analysis of $MTTF$ requires the entire FT to be analyzed together.

There is no general closed-form expression for the $MTTF$, as

6.3 Importance measures

Other than ranking by failure probability, several other measures of component importance have been proposed. Structural importance measures are defined by considering only the structure of the fault tree, while stochastic importance measures also take into account the probabilistic failure behavior of the basic events.

Birnbaum importance measure. Birnbaum [9] defines a BE as critical to a state if changing the component state also changes the TE state. The fraction of states in which a component is critical is now the Birnbaum importance of that component.

Formally, an FT with n components has 2^n possible states, corresponding to different sets χ of failed components. A component e is considered critical in a state χ of FT F if $\pi(F, \chi \cup \{c\}) \neq \pi(F, \chi \backslash \{c\})$.

[**?**]acksonJackson1983 extended this notion to noncoherent systems, in a way that does not lead to negative importances when component failure leads to system repair. An additional refinement was made by [**?**]ndrews and BeesonAndrews2003, to also consider the criticality of a component being repaired.

The *Vesely-Fussell importance factor* $VF_F(e)$ is defined as the fraction of system unavailability in which component e has failed. Formally, $VF_F(e) = P(e \in S | \pi_F(S) = 1)$. An algorithm to compute this measure is given by [10].

The *Risk Reduction Worth* $RRF_F(e)$ is the highest increase in system reliability that can be achieved by increasing the reliability of component e. It may be calculated using the algorithm by [10].

Initiating and enabling importance In systems where some components have a failure rate and others have a failure probability, [**?**]ontini and MatuzasContini2011 introduce a new importance measure that separately measures the importance of *initiating events* that actively cause for the TE, and *enabling events* that can only fail to prevent the TE.

To illustrate this distinction, consider an oil platform. If the event of interest is an oil spill, the event 'burst pipe' would be an initiating event, since this event leads to an oil spill unless something else prevents it. The event 'emergency valve stuck open' is an enabling event. It does not by itself cause an oil spill, it only fails to prevent the burst pipe causing one. The distinction is not usually explicit in the FT, since both these events would simply be connected by an AND gate.

Initiating events often occur only briefly, and either cause the TE or are quickly 'repaired'. Repair in this case can also include the shutdown of the system, since that would also prevent the catastrophic TE. In contrast, enabling events may remain in a failed state for along time.

Due to this difference, overall reliability of such a system can be improved by reducing the failure frequency of initiating events, or by reducing the frequency or increasing the repair rate of enabling events. This is one reason for the distinction between the two in the analysis.

Joint importance To quantify the interactions between components, [**?**]ong and LieHong1993 developed the *Joint Reliability Importance* and its dual, the *Joint Failure Importance*. These measures place greater weight on pairs of components that occur together in many cut sets, such as a component and its only spare, than on two relatively independent components. This may be useful to identify components for which common cause failures are particularly important.

[**?**]rmstrongArmstrong1995 extends this notion of the Joint Reliability Importance to include statistical dependence between the component failures, and proves that the JRI is always nonzero for certain classes of systems. Later, [11] determines that the JFI can also be used for noncoherent systems.

References

1. W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, *Fault Tree Handbook.* Office of Nuclear Regulatory Reasearch, U.S. Nuclear Regulatory Commision, 1981. *(page 16)*
2. S. B. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 509–516, Jun. 1978. *(page 16)*
3. O. Coudert and J. C. Madre, "Fault tree analysis: 10^{20} Prime implicants and beyond," in *Proceedings of the Reliability and Maintainability Symposium (RAMS).* IEEE, 1993, pp. 240–245. *(page 16)*
4. A. B. Rauzy, "New algorithms for fault tree analysis," *Reliability Engineering & System Safety*, vol. 40, no. 3, pp. 203–211, 1993. *(page 16)*
5. A. Rauzy and Y. Dutuit, "Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within Aralia," *Reliability Engineering & System Safety*, vol. 58, no. 2, pp. 127–144, Nov. 1997. *(page 18)*
6. D. E. Ross, K. M. Butler, and M. R. Mercer, "Exact ordered binary decision diagram size when representing classes of symmetric functions," *Journal of Electronic Testing*, vol. 2, no. 3, pp. 243–259, Aug. 1991. *(page 18)*
7. R. E. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing.* Holt, Rinehart, & Winstron, 1975. *(page 18)*
8. K. Durga Rao, V. Gopika, V. V. S. Sanyasi Rao, H. S. Kushwaha, A. K. Verma, and A. Srividya, "Dynamic fault tree analysis using monte carlo simulation in probabilistic safety assessment," *Reliability Engineering & System Safety*, vol. 94, no. 4, pp. 872–883, Apr. 2009. *(page 25)*
9. Z. W. Birnbaum, "On the importance of different components in a multicomponent system," Department of Mathematics, University of Washington, Tech. Rep., 1968. *(page 26)*
10. Y. Dutuit and A. B. Rauzy, "Efficient algorithms to assess component and gate importance in fault tree analysis," *Reliability Engineering & System Safety*, vol. 72, no. 2, pp. 213–222, 2001. *(page 26)*
11. L. Lu and J. Jiang, "Joint failure importance for noncoherent fault trees," *IEEE Transactions on Reliability*, pp. 435–443, Sep. 2007. *(page 27)*