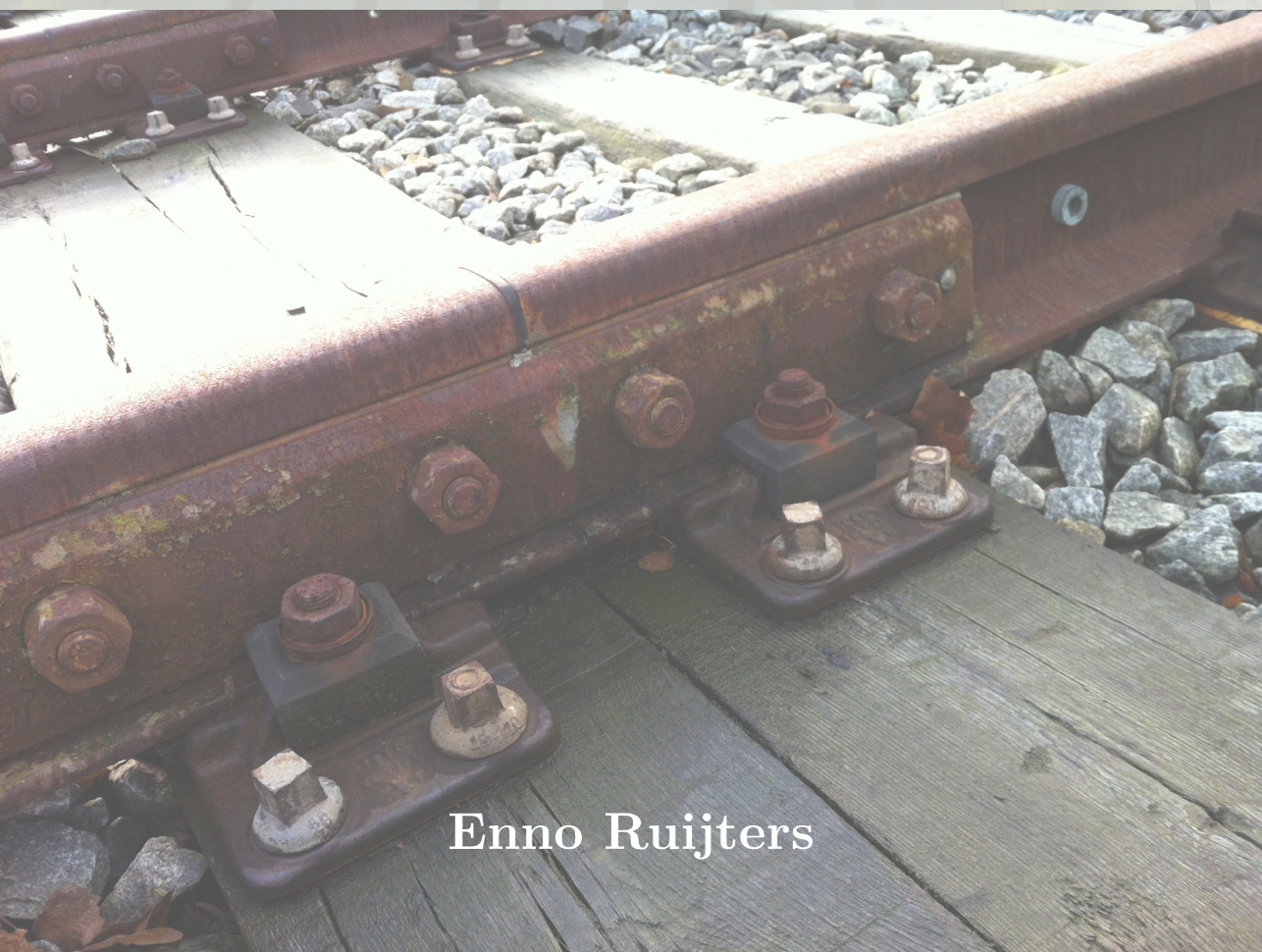
ZEN

and the art of

Railway Maintenance

Analysis and Optimization of Maintenance via Fault Trees and Statistical Model Checking



Enno Ruijters

Zen and the art of railway maintenance

Analysis and optimization of maintenance via fault trees and statistical model checking

Enno Ruijters

Graduation Committee:

Chairman:	prof. dr. J. N. Kok
Promotors:	prof. dr. M. I. A. Stoelinga
	prof. dr. ir. J.-P. Katoen

Members:

prof. dr. ir. T. Tinga	University of Twente
dr. ir. P.-T. de Boer	University of Twente
prof. dr. K. G. Larsen	Aalborg University, Denmark
prof. dr. ir. P. H. A. J. M. van Gelder	Delft University of Technology
prof. dr. J. Křetínský	Technical University Munich, Germany
dr. A. Cimatti	Fondazione Bruno Kessler, Italy

Referee:

ing. M. van Noort	ProRail

DIGITAL SOCIETY INSTITUTE

IDS Ph.D. Thesis Series No. 18-460
Institute on Digital Society
P.O. Box 217,
7500 AE Enschede, The Netherlands

IPA Dissertation Series no. 2018-10
Work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

Stichting voor de Technische Wetenschappen
The work in this thesis was supported by the ArRangeer project (smArt RAilroad maintenance eNGinEERing), funded by the STW-ProRail partnership program ExploRail under the project grant 122238.

ISBN: 978-90-365-4522-8
ISSN: 2589-4730 (IDS Ph.D. Thesis Series)
DOI: 10.3990/1.9789036545228
Available online at `https://doi.org/10.3990/1.9789036545228`

Typeset with LaTeX
Printed by Ipskamp Drukkers Enschede

Zen and the art of railway maintenance

Analysis and optimization of maintenance via fault trees and statistical model checking

Dissertation

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus
Prof. dr. T. T. M. Palstra
on account of the decision of the gratuation committee,
to be publicly defended
on Friday 25rd of May 2018 at 16:45

by

Enno Jozef Johannes Ruijters

Born on 2nd of February 1990
in Brunssum, The Netherlands

This dissertation has been approved by:

Prof. dr. ir. J.-P. Katoen (promotor)
Prof. dr. M. I. A. Stoelinga (promotor)

Abstract

Maintenance is crucial for the operation of modern systems. Timely inspections, repairs, and replacements help to prevent costly failures and downtime, and ensure that systems continue to function properly and safely.

At the same time, this maintenance is costly. It requires staff, spare parts, and often downtime while inspections or repairs are being performed. Too much maintenance means wasting money, reducing the overall usefulness of the system, and even risking accidents due to improper maintenance. It is therefore important to find a good *maintenance policy* that balances cost and dependability.

To achieve this balance, one must understand how a system wears out over time, and what the effects are of various actions to remove or prevent this wear. This thesis presents *fault maintenance trees* (FMTs), a novel formalism to allow the quantitative analysis of the effects of maintenance on costs and system dependability, to support the analysis and improvement of maintenance policies.

FMTs are based on the industry-standard formalism of *fault trees* (FTs), which have long been used to study the reliability of safety-critical systems such as nuclear power plants and airplanes. FTs have been used since the 1960s, and a wide range of extensions and variants have been developed. These support the analysis of systems with time-dependent failures, uncertainty of failure probabilities, and various other properties. The first part of this thesis provides an overview of the jungle of fault tree extensions, surveying over 150 papers on the topic.

The second part of this thesis introduces FMTs, which augment fault trees by including maintenance actions such as inspections and component replacements. With this information, we can calculate the probability of a system failure given a specific maintenance plan. FMTs also include information about the costs of different maintenance actions and failures, allowing one to calculate the expected total costs for a given policy. Thus, FMTs allow the comparison of different maintenance policies with respect to their effects on system reliability and cost, supporting the choice of the policy that best balances the two.

Technically, FMTs are analysed using *statistical model checking* (SMC), a state-of-the-art technique to analyse complex systems without the excessive memory requirements of many other analysis techniques for extended FTs. SMC allows us to compute statistically justified confidence intervals on quantitative metrics such as cost, system reliability, and expected number of failures over time.

SMC works well for many systems, but has a drawback that is particularly noticeable in our setting: Accurate estimates of low probabilities can take a long time to compute. We therefore provide a second analysis technique based on the recently developed Path-ZVA algorithm for *rare event simulation*. While this technique is currently limited to computing the average system availability, it requires much less computation time than SMC does for high-availability systems, without losing the statistical guarantees that SMC provides.

Finally, we want FMTs to be applicable in a practical setting. To this end, the third part of this thesis presents two case studies from the railway industry: an electrically insulated railway joint, and a pneumatic compressor. These case studies were performed in close collaboration with our industrial partners, and demonstrate that FMTs can accurately model real-life systems and maintenance policies, and provide insights to help improve maintenance plans.

Acknowledgements

This thesis is the culmination of my four-year PhD journey, and I would like to extend my gratitude to some of the people who have made this work possible. I particularly thank my supervisors Mariëlle Stoelinga and Joost-Pieter Katoen, and my research coach at ProRail, Martijn van Noort.

Mariëlle, as my daily supervisor, you have kept me on track and guided me through this period. Thank you for the thoughtful discussions and advice along the way. Your suggestions for improving my presentations and papers have been very helpful, and are reflected throughout this thesis.

Joost-Pieter, you were the one who originally pointed me to the open PhD position in Twente, and as my promotor we have had several fruitful discussions about my progress and future direction. The visits to your chair in Aachen were always a helpful source of inspiration and research ideas.

Martijn, you took on the job of research coach some time after the project had started, and you were instrumental in leading us to practical application of the theory we were developing. You provided information on case studies and organised meetings with subject experts, which shaped our research and always showed points of improvement.

I greatly enjoyed my time working at the Formal Methods & Tools group in Twente. The diversity of people and subjects here broadened my horizons, and our weekly lunch colloquia taught me about matters I would not have explored otherwise. I thank the people I shared an office with, Dennis, Waheed, Marcus, Rajesh, Buğra, Carlos, and Arnaud, for always being available for discussions. Dennis, since you worked on the same project as me, particular thanks to you for our discussions on fault trees, maintenance, and Markov automata. Carlos and Arnaud, you have started working where I left off on the successor of my project, thanks for helping me clarify some aspects I hadn't considered. Sebastian, who visited us at FMT for a few months, thank you for the interesting discussions on the semantics of DFTs, these helped me avoid some of their issues in fault maintenance trees. Finally, thanks to all the FMT members for the pleasant atmosphere and enjoyable discussions over lunch, tea breaks, and paper-cakes.

In the last year of my PhD, I spent several months at the Fondazione Bruno Kessler in Trento, Italy as a research internship. Thanks to Alessandro for making this possible, and for the fruitful discussions on my research while there. Thanks also to Marco and both Chiara's for the meetings on the case study, which helped put everything into perspective and provided inspiration for a similar project in Twente. Thanks to Gianni for the help in understanding the extensive toolchain. Furthermore, thanks to all the members of Alessandro's group at FBK for the enjoyable discussions over lunch and in the breaks.

One of the great benefits of being a PhD student is the opportunity to attend conferences and summer schools. Particularly memorable were the EATCS summer

school in Telç, the Marktoberdorf summer school, and the RAMS conference. The discussions and presentations at these and other events were a lot of fun, and a boundless source of ideas and inspiration for collaboration.

I was fortunate in my PhD to be able to collaborate with various partners in the railway sector, which has made my research more practically applicable and often demonstrated places where theoretical assumptions conflict with realistic practices. Thanks to all the people who worked with me on the case studies and other collaborations. Particular thanks to Judi Romijn and Gea Kolk at Movares, who provided help throughout the project and particularly on the EI-Joint case study. Thanks, also, to Peter Drolenga, Margot Peters, and Bob Huisman at NedTrain for their collaboration on the compressor case study.

Thanks to all the members of my committee for approving my thesis, and thanks for the helpful comments about further improvements.

Ten slotte wil ik mijn familie en vrienden bedanken voor hun steun en aanmoediging. Pap, mijn eerste ervaringen met computers waren met jou op je werk, en hiermee is het pad begonnen dat naar dit proefschrift heeft geleid. Suus, bedankt voor al je goede zorgen in de weekenden die ik in Oirsbeek heb doorgebracht. Niek en Gijs-Jan, bedankt voor de samenwerking in CodePoKE tijdens mijn studie in Maastricht, en voor de game-dagen daarna.

Contents

Chapter 1

Introduction

Maintenance is crucial for the cost-effective operation of modern systems. In the automotive industry, for example, a recent estimate concluded that *one minute* of downtime costs $22,000 on average [VG06]. The annual cost of unplanned downtime in the manufacturing industry is as high as $50 billion, with 42% of this being caused by equipment failure [Eme16].

Furthermore, maintenance can be safety-critical: Nobody would board an airplane without confidence that it has been properly maintained. In fact, the U.S. National Transportation Safety Board has identified at least 1,503 aviation accidents between 1988 and 1997 caused by insufficient or improper maintenance [GFK02], resulting in 504 deaths. Proper maintenance is thus clearly essential for safety-critical systems.

While crucial, all this maintenance is also very costly. Staff has to be paid, replacement parts bought, and systems shut down for maintenance. In Finland, maintenance costs make up about 5.5% of manufacturing companies' turnover [Kom02], with some companies spending as much as 25% of their turnover on maintenance. The goal, then, is to balance the cost of maintenance against the effects of failures.

In some cases, this balance is externally imposed: In the aviation industry, for example, the U.S. Federal Aviation Administration sets rules for the periodic inspections of aircraft (annual and 100-hour inspections, in additional to the manufacturer's maintenance manual) [FAA18]. In many cases, however, asset managers can decide for themselves how much maintenance is worth.

Making a well-informed decision about when to apply what maintenance requires a thorough understanding of the effects of such maintenance. This is the topic of this thesis: We present methods to analyse systems subject to maintenance, in terms of:

1. performance, by computing various metrics of the system dependability, such as availability, reliability, and expected number of failures over time, and

2. cost, by estimating the cost of both maintenance and downtime, each of which can be broken down into the costs of different maintenance actions and per-component failure costs.

This allows one to optimise the maintenance policy, focusing effort and cost on those parts of the system where they are most effective, thereby saving costs and/or improving reliability.

1.1 Reliability analysis

Reliability, as a general term, is defined as the state of being *reliable*, i.e., that something can be relied upon. Making this term more formal, we find that "reliability is the ability of a product of system to perform its intended for a specified time, in its life cycle conditions" [KP14]. The goal is reliability engineering, then, is to design and operate systems in such a way that they meet their requirements for reliability.

We can see the field of reliability engineering somewhat broader than only looking at reliability, and include other key performance indicators of system dependability. The most important ones are the so-called RAMS metrics [KP14]: *reliability*, *availability*, *maintainability*, and *safety*. Following [ALRL04], these are defined as:

- **Reliability:** Continuity of correct service.
- **Availability:** Readiness of correct service.
- **Maintainability:** Ability to be modified and repaired.
- **Safety:** Absence of events that are catastrophic for the user and environment.

Depending on the system and environment, other key performance indicators can also be important for dependability. Systems subject to malicious attackers, for example, should also meet requirements for *integrity* (the absence of improper system alterations [ALRL04]). Other extensions include the RAMSSHEEP aspects, extending RAMS with security, health, environment, economics, and politics [Rij12, WvG14].

Having established the dependability requirements for a system, it is necessary to plan how to meet these requirements. This begins at the design stage: appropriate use of high-quality components and design patterns such as redundancy help ensure reliability. The design should also already consider the operational requirements of the system, facilitating ease of maintenance, selecting components to reduce logistics requirements, etc. [RG12].

For short-lived products, reliability engineering typically ends once the product has been developed. For longer-lives assets, more work is required: Throughout the operational life of the system, one can continue to ensure its reliability, e.g. by monitoring its performance, making slight changes to the design, and scheduling maintenance as required.

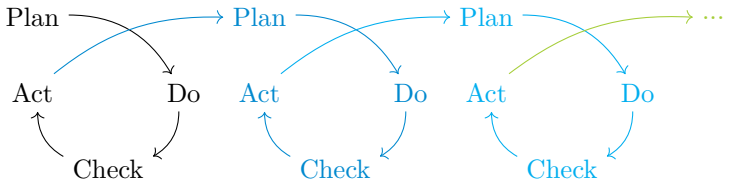


Figure 1.1: Three iterations of the plan–do–check–act cycle

PDCA cycle. After a system is designed and produced, it typically needs to be maintained. The topic of this thesis is the planning of such maintenance, to avoid unnecessary maintenance but still ensuring high reliability. In practice, the maintenance policy of long-lived systems often needs to evolve over time, as components start wearing out and new insights are gained into how the system behaves.

A popular framework for continuous improvement in product development and risk management, also used to keep the maintenance policy up to date, is the *plan–do–check–act* (PDCA) cycle, also called the Deming cycle [Dem86], illustrated in Figure 1.1. Looking specifically at maintenance, the cycle is a guideline for how to achieve continuous improvements to the maintenance planning. It consists of four steps:

- **Plan:** Develop a maintenance policy that will ensure that the system meets its dependability requirements.
- **Do:** Carry out the maintenance policy as planned.
- **Check:** Gather data about the maintenance performed and its effects. Identify any unexpected problems or situations.
- **Act:** If the gathered data confirms that the newly planned policy is an improvement, it now becomes the new standard policy. Otherwise, the old policy remains the standard. Either way, any unexpected information learned in the *Check* phase should be included in the *Plan* phase of the next cycle.

This cycle is repeated every time new information suggests a change to the maintenance policy. Examples include components wearing out faster than expected, design changes, or changes in costs shifting the optimal balance between preventive and corrective maintenance.

Continuous application of the PDCA cycle ensure that new maintenance policies are only implemented when their effectiveness is supported by data, while allowing new insights to be incorporated into the maintenance plan.

Reliability analysis. In order to decide what, if any, improvements should be made to a system to increase its dependability, one needs to analyse the system to assess its RAMS characteristics. Apart from highly system-dependent analyses (e.g., [Bor12] on the reliability of energy distribution grids), a number of widely-applicable techniques have been developed.

The method used in this thesis is fault tree analysis, which will be described in detail later. A complementary method is the *failure modes and effects analysis* (FMEA) [RH04, IEC06a]. This is a spreadsheet-based method, which works by listing all the various failure modes of the system's components, and identifying the effects of each failure mode in isolation. It is a relatively simple method to quickly identify potential dependability problems. It is also one of the oldest methods for reliability analysis, with the U.S. military standard dating back to 1949 [U.S49].

A brief overview of popular alternative reliability estimation methods is provided in Section 2.1. On the one hand, these include more structured spreadsheet-based methods such as the *HAZard an OPerability study* (HAZOP) [Kle99], popular in industrial fields such as the chemistry sector. On the other hand, there are highly detailed techniques for specifying the system behaviour, such as the *Architecture Analysis and Design Language* (AADL) [Soc17, FG12] from which failure modes can be automatically derived and, with sufficient quantitative information, the system dependability can be computed [BCK+11].

1.2 Maintenance

As mentioned earlier, maintenance is crucial to the continued functioning of most systems. From simple tasks like regularly replacing the batteries in your smoke detectors to long and complex overhauls of entire power plants, systems that go unmaintained tend to break down over time. It is therefore important to understand what maintenance is necessary to keep the system running smoothly.

At the same time, too much maintenance is expensive and can actually reduce the functionality of the system. If you have your car inspected every day, it will probably run for decades without problems. It will also rarely be used to actually fulfil its purpose, rather than constantly being in a garage. In extreme cases, maintenance can even cause safety issues, as demonstrated when an airplane crashed due to adhesive tape left blocking its sensors after maintenance [WS00]. Thus, the key to a good maintenance strategy is to find a plan that balances these downsides against the improved reliability.

In this thesis, we consider maintenance to be the actions taken in order to keep a system in working condition, or restore a failed system to its working condition. This include actions (such as inspections) that have no direct effect on the system condition, but are performed as part of an overall strategy to improve or maintain the condition.

One of the key questions when planning maintenance is to determine the optimal time to do it [Ebe97]. Broadly speaking, the timing can be divided into failure-based, use-based, and condition-based schedules [Git92]. A fourth strategy, opportunity-based maintenance, can be combined with the other schedules [Van91].

- A **Failure-based** strategy is simply to wait for a component to fail and then replace it. Such a *run-to-failure* strategy works best on components that either cannot be helped be earlier maintenance (e.g., replacing an intact window will not keep it from getting broken later) or for which failures are not expensive compared to maintenance (e.g., replacing a close-to-failing light bulb is generally no cheaper than replacing it once it has failed).

  If waiting for failures is not an option, due to cost or other requirements, some form of preventive maintenance is required. Such maintenance will typically follow one of the schedules below.

- **Use-based** schedules apply preventive maintenance after some measure of use of the system has been reached. An example of this is the common advice to replace the batteries in your smoke detector once per year, so they never get too close to empty. Slightly more advanced policies base their timing on the actual use of the system, such as oil changes for cars being performed after a certain number of miles driven.

- **Condition-based** schedules are the most advanced maintenance schedules. Here, the current condition of the components is determined in some way (e.g., by inspections or sensors), and maintenance plans are decided taking this into account. A component in very good condition may simply be left untouched for some time, while a component is poor condition is preventively replaced.

- **Opportunity-based** maintenance is sometimes combined with the other maintenance plans [Van91]. Here, one takes advantage of downtime due to other causes (e.g., other planned or unplanned maintenance) to perform preventive maintenance. For example, when a tire on your car has worn out, it can be more efficient to replace any other worn tires rather than use them a few more weeks.

Recently, so-called *predictive maintenance* has become a popular approach to maintenance planning. This is a particular case of condition-based maintenance, where the current (and sometimes historical) state of the system is used to predict the future behaviour, and maintenance is scheduled to prevent any predicted failures. This kind of planning requires a detailed insight into how the components degrade over time, but allows one to achieve very high reliability with minimal unnecessary maintenance.

The key to deciding between these strategies, and to determine an optimal plan within a strategy, is a good understanding of how your system wears out over time, what the effects are of the possible maintenance actions, and what information is available to base decisions on. With this information, one can determine what actions give the greatest benefit (e.g., replacement or partial repair), what metrics best indicates when to take these actions (e.g., time, use, or sensor data), and what values of these metrics indicates the best time to take action.

The approach we provide to gain this understanding is to integrate the effects of maintenance into the well-established reliability engineering formalism of fault tree analysis [VGRH81, RS15] (see Section 1.3).

Maintenance optimisation. The problem of deciding of what maintenance planning is optimal has been studied since the early 1950s and '60s [May60, Dek96]. At this time, most of the work focused on the estimation of the probability distributions of failure times of various components, and the derivation of optimal replacement schedules based on these distributions [BP75].

A drawback of the early work is that it mostly treats components in isolation, and finding an optimal policy for maintaining systems of heterogeneous components with different failure time distributions is considerably more complex. Dekker et al. [DWvdDS97] identify three types of dependencies in multi-component maintenance:

- **Economic** dependence, where maintenance costs can be reduced by simultaneously maintaining multiple components.
- **Structural** dependence, where the system structure dictates that multiple components are maintained at once. For example, many electronic systems have circuit boards that are usually replaced in their entirety, rather than replacing individual components on the board.
- **Stochastic** dependence, where the failure of one component provides information about the remaining lifetime of other components.

Surveys of maintenance models for multi-component systems can be found in [DWvdDS97] (focusing on economic dependence) and [CP91]. We note that the fault maintenance trees discussed in this thesis model all three types of dependencies, as its fault tree includes stochastic dependencies, and its inspection and repair models can affect multiple components at the same time.

More recently, approaches have been developed that can treat larger systems (i.e., with more component or complex policies), at the expense of not producing exact optima [SyD11]. An example is the application of genetic algorithms to optimal policies for opportunity-based maintenance [SWK95a, SWK95b]. This thesis does not present an optimisation method per se, but rather an analysis method that can be used within such an optimisation. In particular, many optimisation methods

contain a model of the degradation and failure behaviour of the system under study, and a parameterized model of the maintenance policy, and then use methods such as genetic algorithms [MZ00] or integer-linear programming [BHD06] to optimise the parameters of the maintenance policy. In this context, FMTs can be used as the model of both the system and the maintenance, provided the optimisation method can tolerate the statistical nature of FMT analysis.

One of the benefits of FMTs is the generality of the framework: Much existing work demonstrates models to optimise maintenance policies for specific settings, e.g.[PA13] for railways, with no clear way to apply the models to other settings. In contrast, FMTs provide a general approach by which models for particular settings can be constructed. Just as standard fault trees have been applied to many different industries, we hypothesise that FMTs can also be applied in different fields.

It has been shown that such external effects, such as usage profile and external temperature, are the dominant cause of variation in the degradation rate of many components [Tin10]. Thus, monitoring of such influences can provide better maintenance policies than simple time-based maintenance, and simulation of these effects with otherwise-deterministic degradation models has been shown effective in maintenance optimisation [TJ13]. FMTs provide some support for such external factors through the RDEP gate, but mostly rely on their inclusion in the probability distributions of the degradation rates. This is most applicable when, as in our case studies, the environment and usage are relatively static over the system's lifetime, and uncertainty in the actual degradation behaviour causes more variation than external factors.

Apart from the maintenance policy itself, various other factors need to be considered in ensuring effective maintenance. These include the management of a (spare) parts inventory [CP91], personnel [PG92], and documentation [Eas84]. These factors are beyond the scope of this thesis, although dynamic fault trees (Chapter 3) can provide some insight into spare parts management [DBB90].

A very important aspect of maintenance optimisation is that it must be applicable in practice. Several reviews [ND08, Dek95] concluded that case studies are not well-represented in the literature. One comment [Sca97] is that maintenance modellers should collaborate with maintenance engineers to ensure that the models are applicable to real-world systems. To that end, the case studies described in Chapters 7 and 8 were developed in close collaboration with partners from the railway industry, with the aim to ensure that FMTs can provide an accurate model of realistic systems and maintenance policies.

1.3 Fault Tree Analysis

Fault trees (FTs) are an industry-standard [ISO11], graphical modelling approach to describe how failures propagate through the system, i.e., how failures of components

interact to cause failures of the overall system [RS15]. By connecting subsystems using boolean connectors (e.g., OR), common patterns such as redundancy of components and subsystems can be expressed. The resulting models can be analysed to obtain various qualitative and quantitative dependability metrics.

They were developed in the 1960s to evaluate the reliability of a missile launch system [Eri99], and were quickly picked up by Boeing as a tool for reliability engineering of their safety-critical systems [Hix68]. Since then, they have been adopted by many other companies, and have become standardised by, e.g., the International Electrotechnical Commission [IEC06b] and ISO [ISO11]. In some fields, the use of fault tree analysis is specified by regulators, such as the U. S. Nuclear Regulatory Commission [VGRH81] and the Federal Aviation Administration [FAA00].

Fault trees are constructed by starting with an undesired event (called the *top (level) event*), and identifying the immediate requirements for this event to occur. Each of these requirements is further refined into its own causes, and so on, until the identified causes are sufficiently fine-grained that they do not need to be further refined. These final causes are the leaves of the tree, also called the *basic events*. The *intermediate events* use boolean connectors, or *gates*, to describe the interactions between failures of subsystems.

Example 1 *An example of a fault tree is shown in Figure 1.2. The top event here is 'Loss of cooling', which is refined into two possible causes: 'No coolant flow' and 'Loss of coolant'. Since either cause leads to a loss of cooling, these are connected by the OR-gate at the top. The event 'Loss of coolant' is refined into basic events 5 and 6, namely 'Coolant leak' and 'Valve stuck closed'. Again, there are connected by an OR-gate. The event 'No coolant flow' is modelled by and AND-gate requiring the loss of both the main and emergency pumps. Both of these pumps can fail independently, either by failure of the motor or loss of power.*

Once a system has been modelled using a fault tree, this tree can be analysed for various qualitative and quantitative metrics. Qualitatively, the most common analysis is to determine *cut sets*: combinations of component failures leading to system failure. For example, from Figure 1.2 one can see that the event 'Coolant leak' is sufficient to cause a loss of cooling. Such a single point of failure often points to weak points in the design that need to be addressed. Quantitatively, one can decorate the basic events with their probabilities of occurring, and compute the probability of the undesired event. In this way, one can demonstrate that the system meets dependability requirements. Alternatively, if the system does not meet requirements, various *importance measures* can be computed that identify which parts of the system have the largest impact on the dependability, which helps to determine the best way to improve it.

Overall, fault tree analysis can be applied to achieve a variety of goals:

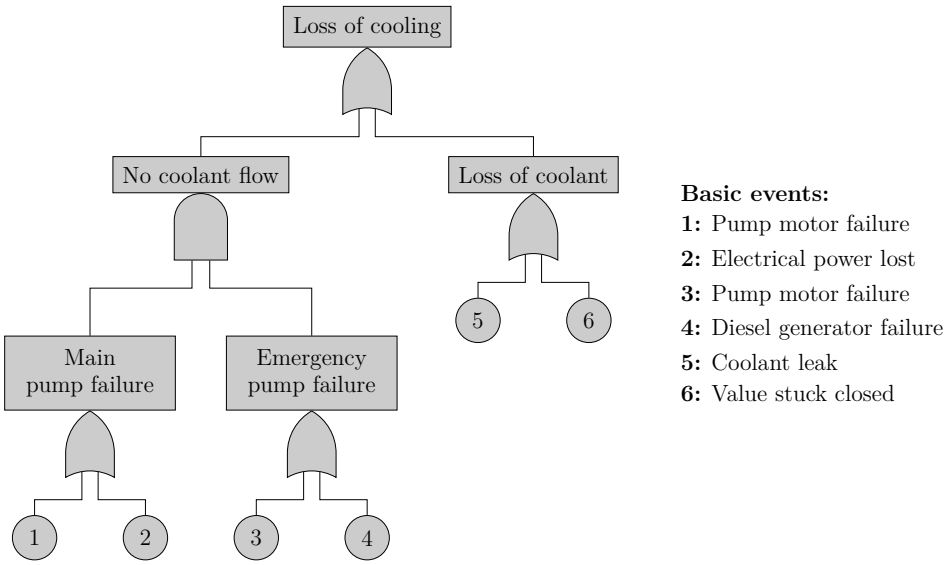


Figure 1.2: Fault tree of a hypothetical coolant system.

- Explain the structure of a system with respect to its dependability, helping to understand the overall failure behaviour of the system (e.g., using cut sets).
- Demonstrate compliance with regulations on the dependability of safety-critical systems (e.g., using quantitative analysis).
- Identify parts of a system where improvements in reliability have the greatest impact on overall system dependability (e.g., using importance measures).
- If a failure has occurred, and information is available about which parts of the system are definitely (not) functioning, identify the most likely causes of the failure (e.g., [HBA08], not discussed in this thesis).

Over the years, a wide range of extensions and variants of fault trees have been developed, which can better handle aspects such as uncertainties, dependencies between components, and repairs. An overview of these extensions is provided in Part I of this thesis.

1.4 Fault Maintenance Trees

While fault trees are widely to analyse system designs, and have been extended to cover some simple policies for repairs [FMIM05, BCRFH08], the impact of

maintenance on system dependability has traditionally not been included in fault tree analysis. The main topic of this thesis is the development of *Fault maintenance trees*, augmenting fault trees with powerful models for maintenance policies. This allows quantitative analysis of the effects of maintenance on costs and system performance, supporting the development of better maintenance plans.

Fault maintenance trees extend classic fault trees in three main ways: First, basic events are more detailed, containing models of how components degrade over time. Second, relationships between the degradation of different components are explicitly modelled using a new gate (the *RDEP*, or *rate-dependency*). This gate models the situation where a failure of one component or subsystem places increased stress on another component, accelerating that component's wear. Finally, *inspection* and *repair modules* are used to model detailed repair policies specifying what inspections are performed when, and what actions are taken depending on the result of the inspection. Repair modules can repair multiple components at once, thereby modelling potential cost reductions by clustering maintenance actions [dJKTT16].

Considering the maintenance policies described in Section 1.2, FMTs allow the specification of both preventive and corrective maintenance actions. We support failure-based, time-based, and condition-based maintenance, with the caveat that all maintenance actions must be specified in terms of time. If a use-based policy is needed, this must be converted to a time-based one, e.g. using information about the system's average use over time. Opportunity-based maintenance is partially supported, as repair modules can specify that multiple components are replaced at the same time, but such opportunistic replacements cannot be condition-based (although we expect that extending FMTs to include condition-dependent replacements would not be difficult).

Example 2 *Figure 1.3 shows part of a fault maintenance tree of a compressor. Just like in a normal fault tree, the top event is a gate (an OR-gate, in this case) describing that any of the child events is sufficient to cause a failure. A new addition is the RDEP gate describing the effects of oil pollution, namely that this causes accelerated wear of the bearings and screws (by a factor of three and two, respectively). Also new are the inspection module $\mathcal{I}$ and repair modules $\mathcal{R}_1$ and $\mathcal{R}_2$, where $\mathcal{R}_2$ specifies that the bearings, screws, and oil are repaired when the compressor fails, while $\mathcal{I}$ and $\mathcal{R}_1$ specify that the air filter is periodically inspected and, if necessary, replaced.*

The key benefit of fault maintenance trees is their ability to model the effects of different maintenance policies on system performance and cost. Quantitative analysis can compute the probability of system failure, the expected number of failures over time, and the expected numbers of each maintenance action performed. By assigning costs to failures, downtime, inspections, and repairs, one can compute the expected total cost of the system under a given policy. By varying this policy,

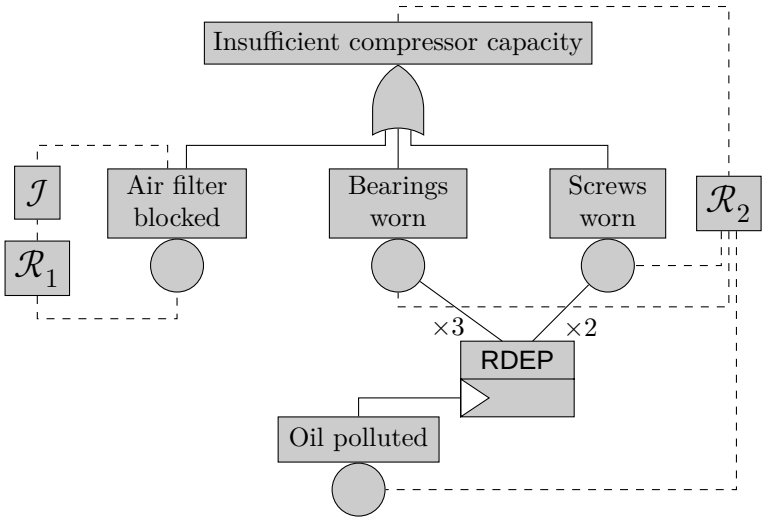


Figure 1.3: Example of a fault maintenance tree

FMTs allow the optimisation of the maintenance plan to find, e.g., the cheapest strategy that meets reliability requirements, or the strategy that offers the best performance within a given budget.

An important element of accurate modelling of the system, is the decision of which probability distributions to use for the degradation rates of the components. This has been shown to have a major impact on the optimal maintenance policy and the total maintenance cost [dJKTT15]. FMTs support arbitrary probability distributions, allowing modellers to choose the most appropriate one, or experiment with different distributions to examine how much the choice impacts the results.

Chapters 7 and 8 show how FMTs can be applied to practical systems, in this case from the railway industry, to calculate the cost-optimal number of inspections to perform per year, and to find maintenance actions whose costs outweigh their benefits.

Analysis. This thesis provides two methods for analysing FMTs: First, Chapter 5 describes an analysis via statistical model checking [BDL+12]. This is a state-of-the-art approach using Monte Carlo simulation to achieve statistically sound conclusions about a wide range of dependability metrics, such as the expected cost or system reliability.

One drawback of statistical model checking is particularly noticeable when analysing highly reliable systems: The computation time needed to obtain an

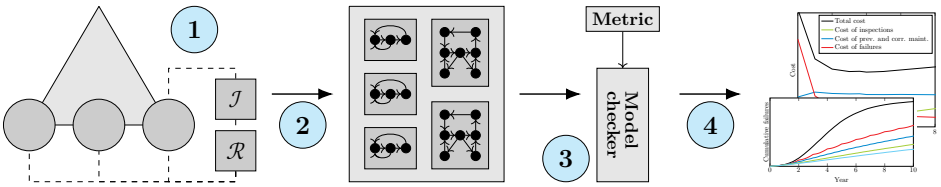


Figure 1.4: **Overview of the steps of fault maintenance tree analysis.** The steps are: (1) construction of the FMT describing the system and its maintenance policy, (2) translation of the FMT to a state-space model (stochastic timed automata or Markov chain), (3) analysing the state-space model using a stochastic model checker to compute the desired metric, and (4) interpreting the results of the model checker to validate the model and optimise the maintenance policy.

accurate estimate increases as the probability being estimated decreases. In reliability engineering, where failure probabilities are typically very small, this computation time can grow impractically large.

Chapter 6 describes how FMTs can be analysed using *rare event simulation* [KH51]. This technique modifies the system being analysed to make failure less rare, estimates the failures probability of this modified system, then applies a correction to the estimate to obtain a statistically sound estimate of the original probability. Our approach currently only supports calculation of the system *availability*, but can significantly reduce the computation time compared to the normal analysis by statistical model checking.

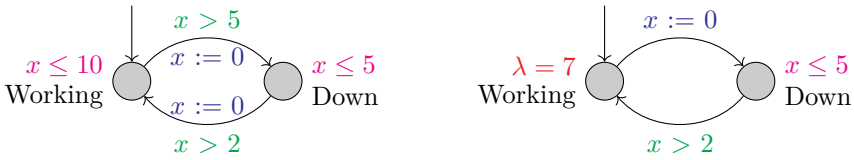
Figure 1.4 illustrates the overall process of a fault maintenance tree analysis.

1.5 Statistical Model Checking

The basic concept of *model checking* is to verify whether some model of a system satisfies a certain property. The term was introduced by Clarke and Emerson [CE82] to describe the process by which a concurrent computer program was verified to meet a property specified using temporal logic, and a similar process was independently developed by Queille and Sifakis [QS82]. Clarke, Emerson, and Sifakis were awarded the Turing award in 2007 for their work [CES09].

Originally, model checking was used to analyse systems with nondeterministic choices (i.e., in which it was unspecified how such choices are made). The outcome was a yes/no verdict whether the property is satisfied regardless of the choices made, and if it is not, a counterexample showing how the property is violated.

Later work [CY88, HJ94] introduced *probabilistic model checking*, in which choices are resolved using probabilities, forming a discrete-time Markov chain. Model checking of continuous-time Markov chains, in which also the time taken in



(a) Timed automaton

(b) Stochastic Timed automaton

Figure 1.5: Examples of (stochastic) timed automata.

each step is governed by an (exponential) probability distribution, followed several years afterwards [BHHK03].

For such probabilistic or stochastic systems, we are no longer restricted to checking qualitative properties, but we can also ask quantitative questions. For example, we can ask "*Is the probability of reaching a failed state within 10 years less than 1%?*" or "*How often, on average, does the model enter a failed state per year?*". Such questions are answered by *stochastic model checking*, and a variety of stochastic model checking tools have been developed such as STORM [DJKV17], PRISM [KNP11], and IscasMC [HLS+14].

Model checking for real-time systems was introduced in [AD94] with the formalism of timed automata. Timed automata consist of *locations*, i.e., discrete control states, and *transitions*, by which the model can move from one location to another. *Clocks* are used to track the passage of time, with *invariants* on locations and *guards* on transitions restricting when transitions may/must be taken.

Example 3 *Figure 1.5a shows an example of a timed automaton. The automaton begins in the location labelled 'Working' with clock x equal to 0. The invariant on this location specifies that some outgoing transition must be taken before 10 units of time have passed, while the guard on the top transition prevents it from begin taken before time 5. Thus, at some time between 5 and 10 (the exact time is nondeterministically chosen), the model moves to the location labelled 'Down'. The top transition also specifies that clock x is reset when the transition is taken. From this new location, between 2 and 5 time units elapse before the transition back to 'Working' is taken, the system is back in its original state.*

For the analysis of fault maintenance trees, we use the extended formalism of *stochastic timed automata* (STAs). These extend timed automata by allowing transition times to be governed by probability distributions rather than only constraints on clocks. An example is shown in Figure 1.5b, where the top transition is governed by an exponential distribution with mean time 7, rather than by a nondeterministic transition time as in Figure 1.5a.

Analysis by statistical model checking. A common problem in the analysis of complex systems using state space-based formalism such as STAs, is that the number of locations grows too large to fit in computer memory. Various reduction techniques (e.g., for fault trees specifically, [VJK18]) can help by reducing the number of locations, but still run out of memory for larger systems. The analysis of FMTs avoids this problem by using *statistical model checking* [BDL+12], which requires very little memory, at the cost of providing only confidence intervals rather than exact results.

Statistical model checking uses Monte Carlo simulation to estimate the probability that a run of the model satisfies the property of interest. To do so, we randomly sample runs of the model, and count the number of runs that satisfy the property and the number that don't. We then apply a statistical hypothesis test to compute a confidence interval for the probability of the property being satisfied, or to give a qualitative result that (with a given confidence) the probability is above or below a given threshold. An overview of various hypothesis tests for statistical model checking can be found in [RdBSH15].

1.6 Problem Description

The research described in this thesis was carried out as part of the ArRangeer project [UT12], itself part of the ExploRail program [SPN18].

This thesis presents part of the results of the ArRangeer project. The PhD thesis of Dennis Guck [Guc17] describes the rest, covering more theoretical advances in stochastic model checking and its use in the analysis of dynamic fault trees.

The goal of the ExploRail program is to reduce the vulnerability of the Dutch railway system to disruptions. This program is made up of nine research project. One of these projects is the ArRangeer project, which stands for *Smart railroad maintenance engineering with stochastic model checking.* The aim of the ArRangeer project was to extend fault trees with concepts from maintenance engineering, and analyse the resulting model using stochastic model checking.

We can state the overall goal of our research project as:

Research goal: Develop an approach to quantitatively analyse the dependability behaviour of a system under different maintenance policies, allowing the comparison of these policies with respect to dependability and cost.

To achieve this goal, we formulate several research questions. First, we would like to base our approach on existing methods for reliability engineering, To this end, we need to examine what methods already exist and how useful they are to our goal of including maintenance. This gives rise to our first research question:

Research question 1: What is the state-of-the-art in the quantitative analysis of system dependability, and how extendable are current approaches to include maintenance?

A brief literature search led us to decide on fault trees as the basis for our approach. They are already an industry-standard tool for reliability analysis, and a wide range of extensions has been developed (see Chapters 2–4). We found that, while some of these extensions include repair strategies, none currently support the complexity of the maintenance policies we would like to analyse. This leads to our next question:

Research question 2: How can the formalism of fault trees be extended to include complex maintenance policies, including inspections and condition-based repairs?

This question led us to develop fault maintenance trees (Chapter 5), which extend fault trees with advanced concepts policies, and also with more detailed descriptions of the wear-out behaviour of components and their dependencies.

To meet our goal of allowing the comparison of different maintenance policies, we need to enable the quantitative analysis of fault maintenance trees to compute the system dependability and cost under a given strategy. Thus, we find our next question:

Research question 3: How can fault maintenance trees be analysed to compute quantitative metrics on the system dependability and costs under a given maintenance policy?

We found that statistical model checking is a useful technique for the quantitative analysis of FMTs, allowing us to obtain various metrics, such as system reliability, expected number of failures, and expected costs. This technique provides statistically justified confidence intervals, and allows the analysis of complex systems within practical amounts of time and memory.

We did find that, when the system being analysed has high reliability, the amount of time required for a tight confidence interval increases. For the analysis of safety-critical systems, which typically have such high reliability, statistical model checking could not deliver results with the desired accuracy without spending too much computation time. This leads us to our next question:

Research question 4: How can we reduce the analysis time for the dependability of highly dependable systems?

We found that the recently developed Path-ZVA algorithm [RdBSJ18] for importance sampling can be adapted to the setting of FMTs (Chapter 6). This

algorithm improves the analysis time for the estimation of very low probabilities (such as the failure probability of a highly dependable system), without losing the statistically justified confidence intervals of statistical model checking.

Finally, we want to examine how FMTs can be applied in practice. For this purpose, we collaborated with two prominent companies in the railway industry (the Dutch railway infrastructure asset manager Prorail, and the Dutch rolling stock maintenance company NS/NedTrain) on two challenging case studies, to investigate our last research question:

Research question 5: Can FMTs be applied to analyse practical systems in the railway industry, and what insights does such an analysis provide?

We found that FMTs are able to model the degradation and maintenance of two systems, an electrically insulated joint (Chapter 7) and a pneumatic compressor (Chapter 8). Our analysis gave insights both into the effects of the current maintenance policy, and into how the policy might be improved.

1.7 Main contributions

This aim of this thesis is to develop the formalism of fault maintenance trees and demonstrate their applicability in practical cases. Specifically, this thesis presents the following contributions:

- *Survey of fault tree literature:* A large body of published work exists on fault tree analysis, including many extensions and variants on classical fault trees. Chapters 2–4 present an survey of over 150 articles on the topic, providing an in-depth summary of the state of the art.
- *Integration of maintenance into fault trees:* Fault maintenance trees (FMTs) are presented (Chapter 5), extending fault trees with advanced models of component degradation and maintenance policies. They allow the modelling of a wide of maintenance actions, and can be analysed using statistical model checking to compute dependability metrics such as reliability and availability, as well as costs. As such, they can be used to compare the effects different maintenance policies, enabling maintenance engineers to optimise their maintenance plans.
- *Rare event simulation for repairable DFTs:* We propose an approach (in Chapter 6) exploiting the recently developed Path-ZVA algorithm [RdBSJ18] for importance sampling for the analysis of repairable (dynamic) fault trees. This approach allows the estimation of the *availability* of highly reliable systems using much less computation time than traditional simulation techniques.

- *Demonstration of FMTs in practice:* Two case studies are presented (Chapters 7 and 8) applying FMT analysis on real-world systems from the railway industry, namely an electrically insulated joint and a pneumatic compressor. We show that FMTs can accurately model the reliability of these systems, and can be used to find improvements of their maintenance policies to reduce costs and increase their dependability.

1.8 Thesis outline

Figure 1.4 illustrates the general process of performing a fault maintenance tree analysis. The structure of the thesis roughly follows this diagram from right to left. In particular:

- **Chapter 2** introduces fault trees, explaining their structure and semantics and describing various analysis techniques that have been developed over the years. This chapter mostly concerns step 1 in the diagram.
- **Chapter 3** explains dynamic fault trees, a prominent extension of fault trees that is able to model more advanced concepts, such as spare parts and time-dependent failure behaviour. This chapter also describes the analysis of dynamic fault trees. This chapter concerns steps 1 and 2 in the diagram.
- **Chapter 4** describes various extensions of fault trees. These extend fault trees to cover a wide range of features, including, e.g., uncertainty about failure rates, advanced temporal dependencies between events, and repair policies. This chapter mostly described step 1 in the diagram.
- **Chapter 5** introduces fault maintenance trees, a novel extension of fault trees that adds models of components wearing out over time, and sophisticated maintenance policies to prevent or undo such wear. We also explain how FMTs are analysed using statistical model checking. This chapter addresses steps 1, 2, and 3 of the diagram.
- **Chapter 6** describes an alternative method for analysing FMTs using rare event simulation. This technique allows more accurate estimations of quantitative metrics using less simulation time, at the expense of increased memory consumption compared to the statistical model checker used in Chapter 5. This chapter addresses step 3 of the diagram.
- **Chapter 7** uses the industrial case study of an electrically insulated railroad joint to demonstrate the practical applicability of FMTs in industry. We show how FMTs are used to model the degradation and maintenance of this joint, validate the model against historical failure data, and show that the

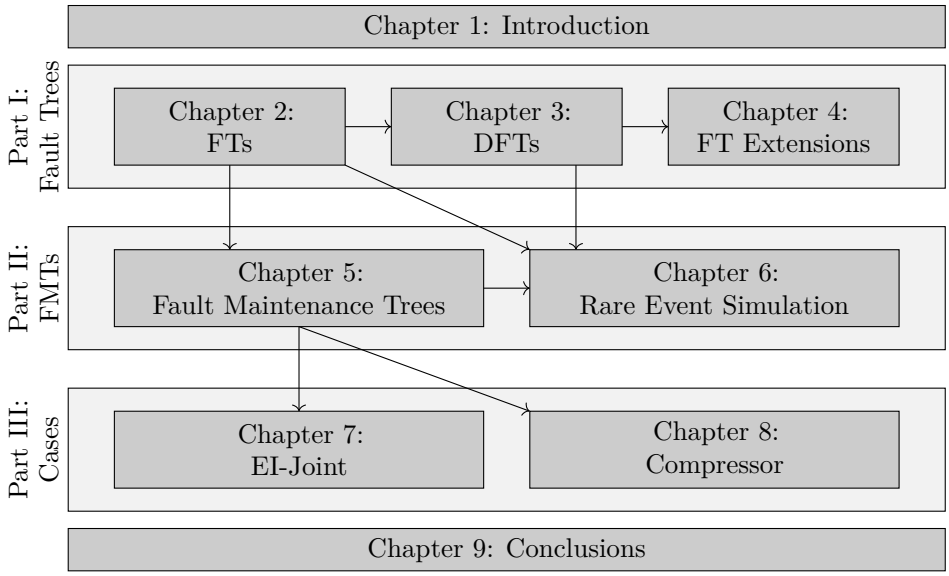


Figure 1.6: Dependencies between chapters

reference maintenance policy for such joints is approximately cost-optimal. This chapter concerns steps 1 and 4.

- **Chapter 8** applies FMTs to the case of a pneumatic compressor found on trains. We again show that FMTs can accurately model the wear and maintenance of this compressor, validate the model, and provides suggestions for attaining almost the same reliability at a reduced maintenance cost. This chapter discusses steps 1 and 4.
- **Chapter 9** concludes the thesis with a discussion of the advantages and disadvantages of fault maintenance trees and their analysis, as well as providing avenues for future research.

Reading guide. Although each chapter can be roughly understood individually, this thesis is intended to be read sequentially, and later chapters depend on concepts that are only described in detail in earlier chapters. Exceptions to this are Chapters 4 and 6, which are not needed for later chapters.

Figure 1.6 illustrates the dependencies between the chapters. Chapter 2 may be skipped by those already familiar with fault trees and their analysis, as may Chapter 3 for those familiar with dynamic fault trees and the Markov chain-based analysis thereof.

Part I

Fault trees

Chapter 2

Introduction to fault trees

Risk analysis is an important activity to ensure that critical assets, like medical devices and nuclear power plants, operate in a safe and reliable way. Fault tree analysis (FTA) is one of the most prominent techniques here, used by a wide range of industries such as the aerospace [SVD+02], automotive [ISO11], and nuclear [VGRH81] industries. Various industrial standard have been developed for FTA, e.g. by the IEC [IEC06b] and by ISO for automotive applications [ISO11].

Fault trees (FTs) are a graphical method that model how failures propagate through the system, i.e., how component failures lead to system failures. Due to redundancy and spare management, not all component failures lead to a system failure.

As a model of this failure propagation, FTs are trees, or more generally directed acyclic graphs, whose leaves model component failures and whose gates describe which combinations of failures lead to (sub)system failures. Figure 2.2 shows a representative example, which is elaborated in Example 4.

Fault trees are used for various purposes within risk analysis:

- *Exploring design alternatives:* System designers often have several options for ensuring the dependability of their system, such as using more reliable (and expensive) components, using more components in a redundant fashion, etc. FTA can be used to assess the dependability of different designs to help select the best option [GJK+17a].

- *Demonstrating compliance:* Many industries are subject to legal requirements for dependability. For example, the US Department of Labor sets standards for the safety of equipment in workplaces, and specify fault trees as a tool to help demonstrate that equipment meets this standard [OSH94]. Similarly, the Federal Aviation Administration lists FTA as one of the tools for hazard analysis in high-consequence decisions [FAA98, FAA00].

- *Fault diagnosis:* Even if a system is highly reliable, there is always a possibility that failures still occur. Fault trees can be used in such situations to help identify the most likely causes of the failure, which helps speed up repairs [LY77]. This thesis does not discuss how to perform such diagnosis.

To perform a fault tree analysis, we distinguish between *qualitative* FTA, which considers the structure of the FT, and *quantitative* FTA, which computes values such as failure probabilities for FTs. In the qualitative realm, *cut sets* are an important measure, indicating which combinations of component failures lead to system failures. If a cut set contains too few elements, this may indicate a system vulnerability. Other qualitative measures we discuss are path sets and common cause failures.

Quantitative system measures mostly concern the computation of failure probabilities. If we assume that the failures of the system components are governed by a probability distribution, then quantitative FTA computes the failure probability for the system. Here, we distinguish between discrete and continuous probabilities. For both variants, the following FT measures are discussed:

- *System reliability* is the probability that the system fails with a given time horizon t.
- *System availability* is the percentage of time that the system is operational.
- *Mean time to failure* is the average time before the first failure.
- *Mean time between failures* is the average time between two subsequent failures.

Such measures are vital to determine if a system meets its dependability requirements, or whether additional measures are needed. Furthermore, we discuss sensitivity analysis techniques, which determine how sensitive an analysis is with respect to the values (i.e., failure probabilities) in the leaves; we also discuss importance measures, which give means to determine how much different leaves contribute to the overall system dependability.

In terms of analysis, we explain basic algorithms such as boolean algebra for cut sets, as well as more efficient algorithms such as binary decision diagram-based methods for computing reliability. Overviews of the various methods can be found in Tables 2.1 (Page 32 on methods for minimal cut sets), 2.3 (Page 43 on quantitative methods), and 2.4 (Page 59 on importance measures).

While SFTs (standard, or static, fault trees) provide a simple and informative formalism, they lack the expressivity needed to model certain often occurring dependability patterns. Therefore, several extensions to fault trees have been proposed, which are capable of expressing features that are not expressible in SFTs. Examples include spare management, different operational modes, and dependent events. *Dynamic Fault Trees* are the best known, and discussed in the next chapter. Other extensions, such as extended fault trees, repairable fault trees, fuzzy fault trees, and state-event fault trees, are popular as well. These extensions and their analysis techniques will be explored in Chapter 4. A graphical overview of the structure of these chapters can be seen in Figure 2.1.

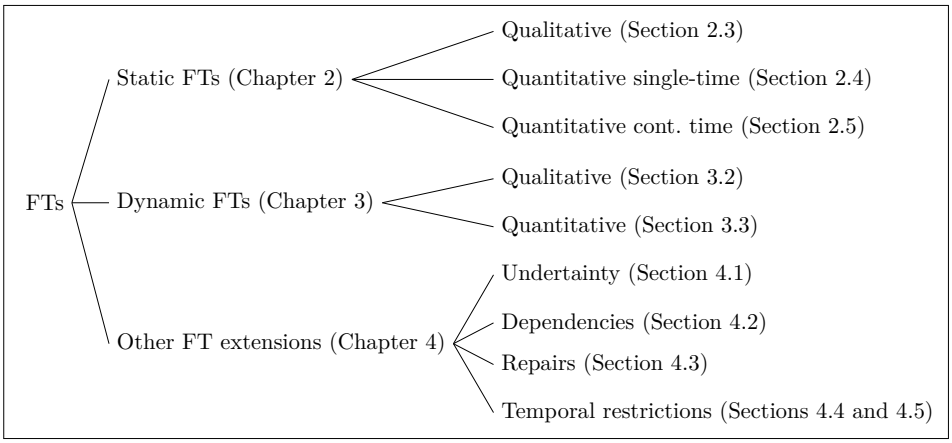


Figure 2.1: Broad overview of the structure of the next three chapters.

In researching fault trees and their extensions, we have reviewed over 150 papers on fault tree analysis, providing an extensive overview of the state-of-the-art in fault tree analysis.

Research Methodology Most literature for this chapter was found during a survey in 2014. This survey was intended to be as comprehensive as reasonable, but we cannot guarantee that we have found every relevant paper.

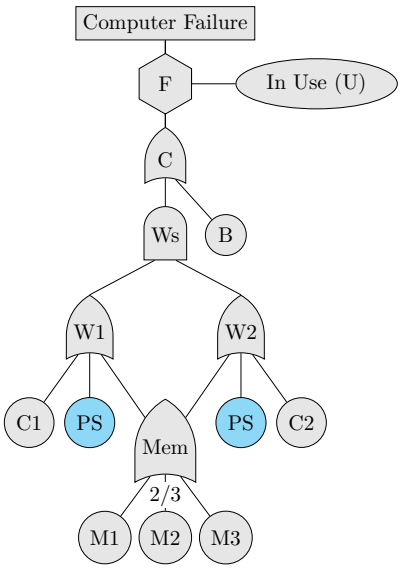
To obtain relevant papers, we searched for the keywords 'Fault tree' in the online databases
Google Scholar (`http://scholar.google.com`),
IEEExplore (`http://ieeexplore.ieee.org`),
ACM Digital Library (`http://dl.acm.org`),
Citeseer (`http://citeseerx.ist.psu.edu`),
ScienceDirect (`http://www.sciencedirect.com`),
SpringerLink (`http://link.springer.com`),
and SCOPUS (`http://www.scopus.com`). Further articles were obtained by following references from the papers found.

Articles were excluded that are not in English, or deemed of poor quality. Furthermore, to limit the scope of this survey, articles were excluded that present only applications of FTA, only methods for constructing FTs, or only describe techniques for fault diagnosis based on FTs, unless the article also presents novel analysis or modeling techniques. Articles presenting implementations of existing algorithms were only included if they describe a specific, working tool.

Origin of this chapter This chapter is extended from Chapters 1 and 2 of:

- Enno Ruijters and Mariëlle Stoelinga. "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools". *Computer Science Review*, 15–16:29–62, 2015. DOI: 10.1016/j.cosrev.2015.03.001, ISSN: 1574-0137.

Organization of this chapter After a brief overview of dependability formalisms other than fault trees in Section 2.1, Section 2.2 provides a definition of fault trees and their semantics, followed by their analysis methods. Section 2.3 discusses qualitative analysis, while Sections 2.4 and 2.5 discuss quantitative analysis in single-time and continuous-time FTs, respectively. Section 2.6 describes various qualitative and quantitative importance measures. Finally, Section 2.7 describes several available tools, and Section 2.8 presents some conclusions.



Legend:

F: Computer failure while in use
C: Computer failure
Ws: Failure of both workstations
B: Bus failure
W1: Failure of workstation 1
W2: Failure of workstation 2
C1: Failure of CPU 1
C2: Failure of CPU 2
PS: Failure of power supply
Mem: Failure of memory system
M1: Failure of memory module 1
M2: Failure of memory module 2
M3: Failure of memory module 3

Figure 2.2: **Example FT of a computer system** with a non-redundant system bus and power supply, two redundant CPUs of which one can fail with causing problems, and three redundant memory units of which one is allowed to fail. PS is coloured differently to indicate that both leaves correspond to the same event.

2.1 Related work

Apart from fault trees, there are a number of other formalisms for dependability analysis [BV10]. We list the most common ones below.

Failure Mode and Effects Analysis One of the first systematic techniques for dependability analysis was the Failure Mode and Effects Analysis (FMEA) [RH04, BCK+11]. FMEA, and in particular its extension with criticality FMECA (Failure Mode, Effects and Criticality Analysis), is still very popular today; users can be found throughout the safety-critical industry, including the nuclear, defence [U.S90], avionics [FAA05], automotive [Aut08], and railroad domains. These analyses offer a structured way to list possible failures and the consequences of these failures. Possible countermeasures to the failures can also be included in the list.

If probabilities of the failures are known, quantitative analysis can also be performed to estimate system reliability and to assign numeric criticalities to potential failure modes and to system components [U.S90].

Constructing an FME(C)A is often one of the first steps in constructing a fault tree, as it helps in determining the possible component failures, and thus the basic events [SVD+02].

HAZOP analysis A hazard and operability study (HAZOP) [Kle99] systematically combines a number of guide-words (like *insufficient*, *no*, or *incorrect*) with parameters (like *coolant* or *reactant*), and evaluates the applicability of each combination to components of the system. This results in a list of possible hazards that the system is subject to. The approach is still used today, especially in industrial fields like the chemistry sector.

A HAZOP is similar to an FMEA in that both list possible causes of a failure. A major difference is that an FMEA considers failure modes of components of a system, while a HAZOP analysis considers abnormalities in a process.

Reliability block diagrams Similar to fault trees, reliability block diagrams (RBDs) [MKK09] decompose systems into subsystems to show the effects of (combinations of) faults. Similar to FTs, RBDs are attractive to users because the blocks can often map directly to physical components, and because they allow quantitative analysis (computation of reliability and availability) and qualitative analysis (determination of cut sets).

To model more complex dependencies between components, Dynamic RBDs [DX06] include standby states where components fail at a lower rate, and triggers that allow the modeling of shared spare components and functional dependencies. This may improve the accuracy of the computed reliability and availability.

OpenSESAME The OpenSESAME modeling environment [WSB08] extends RBDs by allowing more types of inter-component dependencies, common cause failures, and limited repair resources. This is mostly an academic approach and sees little use in industry.

SAVE The system availability estimator (SAVE) [GCdSeS+95] modeling language is developed by IBM, and allows the user to declare components and dependencies between them using predefined constructs. The resulting model is then analyzed to determine availability.

AADL The Architecture Analysis and Design Language (AADL) [Soc17] is an industry standard for modeling safety-critical systems architectures. A complete AADL specification consists of a description of *nominal* behaviour, a description of *error* behaviour and a *fault injection* specification that describes how the error behaviour influences the nominal behaviour.

Such an AADL specification can be used to derive an FMEA table [ENN13] in a systematic way. If failure rates are known, quantitative analysis can also determine the system reliability and availability [BCK+11].

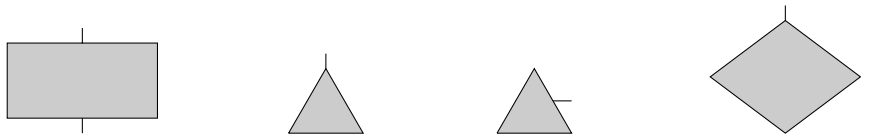
UML Another industry standard for modeling computer programs, but also physical systems and processes, is the Unified Modeling Language (UML) [RJB04]. UML provides various graphical models such as Statechart diagrams and Sequence diagrams to assist developers and analysts in describing the behaviours of a system.

It is possible to convert UML Statechart diagrams into Petri Nets, from which system reliability can be computed [BMM99, BDM02]. Another approach combines several UML diagrams to model error propagation and obtain a more accurate reliability estimate [PDAC05].

Möbius The Möbius framework was developed by Sanders et al. [DCC+02, SCD+03] as a multi-formalism approach to modeling. The tool allows components of a system to be specified using different techniques and combined into one model. The combined model can then be analyzed for reliability, availability, and expected cost using various techniques depending on the underlying models.

2.1.1 Legal background

FTA plays an important role in product certification, and to show conformance to legal requirements. In the European Union, legislature mandates that employers assess and mitigate the risks that workers face [EEC89]. FTA can be applied in this context, e.g. to determine the conditions under which a particular machine is



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

Figure 2.3: Graphical representations of non-basic events in fault trees

dangerous to workers [IEC06b]. The U.S. Department of Labor has also accepted the use of FTA for risk assessment in workplace environments [OSH94].

Similarly, the EU Machine Directive [EC06] requires manufacturers to determine and document the risks posed by the machines they produce. FTA is one of the techniques that can be used for this documentation [Hoo10].

The transportation industry has also adopted risk analysis requirements, and FTA as a technique for performing such analysis. The Federal Aviation Administration adopted a policy in 1998 [FAA98] requiring a formalized risk management policy for high-consequence decisions. Their System Safety Handbook [FAA00] lists FTA as one of the tools for hazard analysis.

2.2 Static fault trees

As discussed in the previous section, it is often necessary to analyze system dependability properties. A fault tree is a graphical model to do so: It describes the relevant failures that might occur in the system, and how these failures interact to possibly cause a failure of the system as a whole.

Standard, or static, fault trees (SFTs) are the most basic fault trees. They have been introduced in the 1960s at Bell Labs for the analysis of a ballistic missile [Eri99]. The classical *Fault Tree Handbook* by Vesely et al. [VGRH81] provides a comprehensive introduction to SFTs. Below, we describe the most prominent modelling and analysis techniques for SFTs.

2.2.1 Fault Tree Structure

A fault tree is a directed acyclic graph (DAG) consisting of two types of nodes: *events* and *gates*. An event is an occurrence within the system, typically the failure of a subsystem or component. Events can be divided into *basic events (BEs)*, which occur on their own, and *intermediate events*, which are caused by one or more other events. The event at the top of the tree, called the *top event (TE, a.k.a. 'top level event' or '(top) undesired event')*, is the event being analyzed, modeling the failure of the (sub)system under consideration.

Fault tree gates

Gates represent how failures propagate through the system, i.e., how failures in subsystems can combine to cause a system failure. Each gate has one output and one or more inputs. The following gates are commonly used in fault trees. Images of the gates are shown in Figure 2.4.

AND Output event occurs if all of the input events occur, e.g. gate *Ws* in the example.

OR Output event occurs if any of the input events occur, e.g. gate *C* in the example.

k/N a.k.a. VOTING, has N inputs. Output event occurs if at least k input events occur. This gate can be replaced by the OR of all sets of k inputs, but using one k/N gate is much clearer. Gate *Mem* in the example is a 2/3 gate.

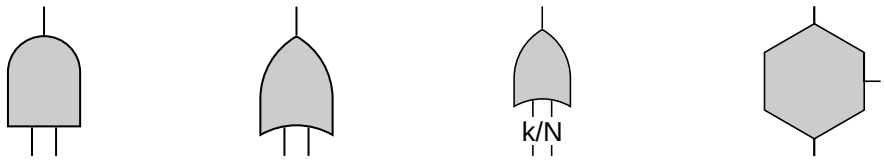
INHIBIT Output event occurs if the input event occurs while the conditioning event drawn to the right of the gate also occurs. This gate behaves identically to an AND-gate with two inputs, and is therefore not treated in the rest of this chapter. It is sometimes used to clarify the system behaviour to readers. Gate *F* in the example is an INHIBIT gate.

Example 4 *Figure 2.2 (modified from [MT95, BHMT96]) shows a fault tree for a partially redundant computer system. The system consists of a bus, two CPUs, 3 memory units, and a power supply. These components are represented as basic events in the leaves of the tree. The top of the tree (labeled Computer Failure here) represents the event of interest, namely a failure of the computer system.*

As stated, gates represent how failures propagate through the system: Gate F is an INHIBIT-gate indicating that a system failure is only considered when the system is in use, so that faults during intentional downtime do not affect dependability metrics.

The OR gate C, just below F, indicates that the failure of either the bus (basic event B) or the workstation subsystem causes a system failure. The workstation subsystem consists of two redundant units combined using an AND gate Ws so that both need to fail to cause an overall failure. Each workstation can fail because either the CPU (C1 or C2) fails or the power supply (PS) fails. Note that the event PS is duplicated for each subtree, but still represents a single event.

A failure of the memory subsystem can also cause a unit to fail, but this requires a failure of two memory units. This is represented by the 2/3 gate Mem. This gate is an input of both compute subsystems, making this a DAG, but the subtree could also have been duplicated like PS if the method used required a tree but allowed repeated events.



(a) AND gate (b) OR gate (c) k/N gate (d) INHIBIT gate

Figure 2.4: Graphical representations of the gates types in a standard fault tree

Non-basic events

In addition to basic events depicted by circles, Figure 2.3 shows other symbols for events. An intermediate event is depicted by a rectangle. Intermediate events can be useful for documentation, but do not affect the analysis of the FT, and may therefore be omitted. If an FT is too large to fit on one page, triangles are used to *transfer* events between multiple FTs to act as one large FT. Finally, sometimes subsystems are not really BEs, but insufficient information is available or the event is not believed to be of sufficient importance to develop the subsystem into a subtree. Such an *undeveloped event* is denoted by a diamond.

Extensions

Several extensions of FTs introduce additional gates that allow the modelling of systems that can return to a functional state after failure. These 'Repairable Fault Trees' will be described in Section 4.3. Static FTs can include simple repairs (as discussed in Section 2.5) by decorating BEs with repair rates, rather that using additional gates.

Other extensions include a NOT-gate or equivalent (e.g., exclusive-or), so that a component failure can cause the system to go from failed to working again [LGTL85], or a functioning component can contribute to a system failure. Such a system is called noncoherent. It may indicate an error in modeling [VGRH81], however some systems naturally exhibit noncoherent behaviour: For example, the combination of a failed safety valve and a functioning pump can lead to an explosion, while a failed pump always prevents this.

2.2.2 Formal definition

To formalize an FT, we use $\mathit{GateTypes} = \{\mathit{AND}, \mathit{OR}\} \cup \{\mathit{VOT(k/N)} \mid k, N \in \mathbb{N}^{>1}, k \leq N\}$. Following Codetta-Raiteri et al. [CRFIV04], we formalize an FT as follows.

Defintion 1 *An FT is a 4-tuple* $F = \langle BE, G, T, I \rangle$*, consisting of the following components.*

- *BE is the finite set of basic events.*
- *G is the finite set of gates, with* $BE \cap G = \emptyset$*. We write* $E = BE \cup G$ *for the set of elements.*
- $T : G \to GateTypes$ *is a function that describes the type of each gate.*
- $I : G \to \mathcal{P}(E)$ *describes the inputs of each gate. We require that* $I(g) \neq \emptyset$ *and that* $|I(g)| = N$ *if* $T(g) = VOT(k/N)$*.*

Importantly, the graph formed by $\langle E, I \rangle$ *should be a directed acyclic graph with a unique root TE which is reachable from all other nodes.*

This description does not include the INHIBIT gate, since this gate can be replaced by an AND. The INHIBIT gate may, however, be useful for documentation purposes. Also, intermediate events are not explicitly represented, again because they do not affect analysis.

Some analysis methods described in Sections 2.3 through 2.6 require the undirected graph $\langle E, I \rangle$ to be a tree, i.e., forbid shared subtrees. In this chapter, an FT will be considered a DAG. An element that is the input of multiple gates can be graphically depicted in two ways: The element (and its descendants) can be drawn multiple times, in which case the FT still looks like a tree, or the element can be drawn once with multiple lines connecting it to its parents. Since these depictions have the same semantics, we refer to these elements as *shared* subtrees or *shared* BEs regardless of graphical depiction.

2.2.3 Semantics

The semantics of an FT F describes, given a set S of BEs that have failed, for each element e, whether or not that element fails. We assume that all BEs not in S have not failed.

Typically, the semantics of an FT F are described as its *structure function*: the function $\pi_F : \{0,1\}^{|BE|} \to \{0,1\}$ that takes the status of all basic events as inputs (0 for a functioning event and 1 for a failed event) and yields whether the FT F is functional (0) or failed (1).

We use a slight variation on this function, which also provides information about the intermediate events.

Defintion 2 *The semantics of FT F is given by the function* $\pi_F : \mathcal{P}(BE) \times E \to \{0,1\}$ *where* $\pi_F(S, e)$ *indicates whether e fails given the set S of failed BEs. It is defined as follows.*

- *For* $e \in BE$, $\pi_F(S, e) = e \in S$*.*

- *For* $g \in G$ *and* $T(g) = AND$, *let* $\pi_F(S, g) = \bigwedge_{x \in I(g)} \pi_F(S, x)$.
- *For* $g \in G$ *and* $T(g) = OR$, *let* $\pi_F(S, g) = \bigvee_{x \in I(g)} \pi_F(S, x)$.
- *For* $g \in G$ *and* $T(g) = VOT(k, N)$, *let* $\pi_F(S, g) = \left(\sum_{x \in I(g)} \pi_F(S, x) \right) \geq k$.

Note that the AND gate with N inputs is semantically equivalent to an *VOT(N/N)* gate, and the OR gate with N inputs is semantically equivalent to a *VOT(1/N)* gate.

In the remainder of this chapter, we abbreviate the interpretation of the top event t by stating $\pi_F(S, t) = \pi_F(S)$. This corresponds to the structure function of the fault tree expressed using sets instead of boolean vectors.

It follows easily that standard FTs are *coherent*, i.e., if event set S leads to a failure, then every superset S' also leads to failure. Formally, $S \subseteq S' \wedge \pi_F(S, x) = 1 \Rightarrow \pi_F(S', x) = 1$.

2.3 Qualitative analysis

Fault tree analysis techniques can be divided into quantitative and qualitative techniques. *Qualitative techniques* provide insight into the structure of the FT, and are used to detect system vulnerabilities. We discuss the most prominent qualitative techniques, being (minimal) cut sets, (minimal) path sets, and common cause failures. We recall the classic methods for quantitative and qualitative fault tree analysis presented in [LGTL85] as well as many newer techniques.

In Tables 2.1, 2.2, 2.3, and 2.4, we have summarized the qualitative analysis techniques that we discuss in the current section.

Quantitative techniques are discussed in later sections. These compute numerical values over the FT. Quantitative techniques can be divided into *importance measures* (discussion in Section 2.6, indicating how critical a certain component is, and *stochastic measures*, most notably failure probabilities. The stochastic measures are again divided into those handling single-time failure probabilities and continuous-time ones, described in Sections 2.4 and 2.5, respectively.

Sensitivity analysis

Quantitative techniques produce values for a given FT, but it is often useful to know how sensitive these values are to the input data. For example, if small changes in BE probabilities result in a large variation in system reliability, the calculated reliability may not be useful if the probabilities are based on rough estimates. On

Author	Method	Remarks	Tool
Vesely et al. [VGRH81]	Top-down	Classic boolean method	MOCUS [FHM74]
Vesely et al. [VGRH81]	Bottom-up	Produces MSC for gates	MICSUP [PSC75]
Coudert and Madre [CM93]	BDD	Usually faster than classic methods	MetaPrime [CM94]
Rauzy [Rau93]	BDD	Only for coherent FTs but faster than [CM93]	Aralia [RD97]
Dutuit and Rauzy [DR96]	Modular BDD	Faster for FTs with independent submodules	DIFTree [DVG97]
Remenyte et al. [RA06, RPA08]	BDD	Comparison of BDD construction methods	-
Codetta-Raiteri [CR06]	BDD	Faster when FT has shared subtrees	-
Xiang et al. [XYM$^+$11]	Minimal Cut Vote	Reduced complexity with large voting gates	CASSI [XYM$^+$11]
Carrasco et al. [Cn99]	CS-Monte Carlo	Less complex for FTs with few MCS	-
Vesely and Narum [VN70]	Monte Carlo	Low memory use, accuracy not guaranteed	PREP [VN70]

Table 2.1: Summary of methods to determine Minimal Cut Sets of SFTs

the other hand, if the reliability is very sensitive to one particular component's failure rate, this component may be a good candidate for improvement.

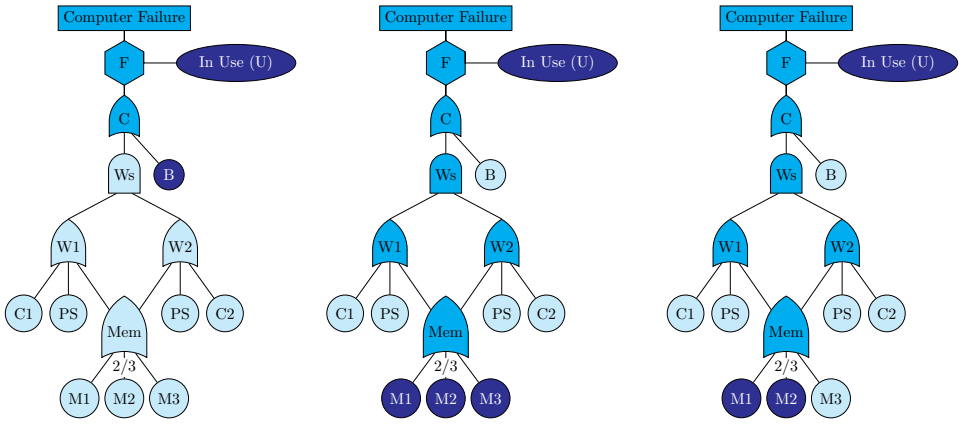
If the quantitative analysis method used gives an algebraic expression for the failure probability, it may be possible to analyze this expression to determine the sensitivity to a particular variable. One method of doing so is provided by [Rus85].

In many cases, however, sensitivity analysis is performed by running multiple analysis with slightly different values for the variables of interest.

If the uncertainty of the BE probabilities is bounded, an extension to FT called a *Fuzzy Fault Tree* can be used to analyze system sensitivity. This method is explained in Section 4.1.

2.3.1 Minimal cut sets

Cut sets and minimal cut sets provide important information about the vulnerabilities of a system. A *cut set* is a set of components that can together cause the system to fail. Thus, if an SFT contains cut sets with just a few elements, or elements whose failure is too likely, this could result in an unreliable system. Reducing the failure probabilities of these cut sets is usually a good way to improve overall reliability. Minimal cut sets are also used by some quantitative analysis



(a) Minimal cut set $\{U, B\}$

(b) Cut set $\{U, M1, M2, M3\}$

(c) Minimal cut set $\{U, M1, M2\}$

Figure 2.5: **Examples of cut sets of the FT in Figure 2.2.** Basic events in the cut set are indicated in dark blue, affected (failed) gates in lighter blue.

techniques described in Sections 2.4 through 2.6.

This section describes three important classes of cut set analysis: Classical methods which are based on manipulation of the boolean expression of the FT, methods based on Binary Decision Diagrams, and others. Table 2.1 summarizes these techniques.

Defintion 3 *A set $C \subseteq BE$ is a cut set of FT F if $\pi_F(C) = 1$. A minimal cut set (MCS) is a cut set of which no subset is a cut set, i.e., formally $C \subseteq BE$ is an MCS if $\pi_F(C) = 1 \wedge \forall_{C' \subset C} : \pi_F(C') = 0$.*

Example 5 *Figure 2.5 shows three examples of cut sets. (a) shows that $\{U, B\}$ is an MCS. The cut set in (b) is $\{U, M1, M2, M3\}$, but this is not an MCS since it contains the cut set $\{U, M1, M2\}$ shown in (c).*

Deciding whether a given set C is a cut set is trivial, as one can simple evaluate the semantics of the tree given C and examine whether the TE has failed or not.

Denoting the set of all MCS of an FT F as $MC(F)$, we can write an expression for the top event as $\bigvee_{C \in MC(F)} \bigwedge_{x \in C} x$. This property is useful for the analysis of the tree, as described below.

Boolean manipulation

The classical methods of determining minimal cut sets are the bottom-up and the top-down (a.k.a. Mocus) algorithms [VGRH81]. These represent each gate as a Boolean expression of BEs and/or other gates. These expressions are combined, expanded, and simplified into an expression that relates the top event to the BEs without any gates (thus, it is an expression of the structure function). At every step, the expressions are converted into disjunctive normal form (DNF), so that each conjunction is an MCS.

Example 6 *The first few steps of the top-down algorithm for the FT in Figure 2.2 are:*

- $F = U \wedge C$
- $F = U \wedge (B \vee Ws)$ *since* $C = B \vee Ws$.
- $F = (U \wedge B) \vee (U \wedge Ws)$ *converting to disjunctive normal form*
- *...*

Continuing in this fashion until all gates have been eliminated results in the minimal cut sets.

The *bottom-up* method begins with the expressions for the gates at the bottom of the tree. This method usually produces larger intermediate results since fewer opportunities for simplification arise. As a result, it is often more computationally intense. However, it has the advantage of also providing the minimal cut sets for every gate, as shown in the following example:

Example 7 *The first few steps of the bottom-up algorithm for the FT in Figure 2.2 are:*

- $Mem = (M1 \wedge M2) \vee (M2 \wedge M3) \vee (M1 \wedge M2)$ *since Mem is a 2/3 voting gate*
- $W1 = C1 \vee PS \vee Mem$
- $W1 = C1 \vee PS \vee (M1 \wedge M2) \vee (M2 \wedge M3) \vee (M1 \wedge M3)$
- *...*

This already shows the cut sets for Mem and W1. Continuing in this fashion eventually gives cut sets for all gates, including the top level event.

Binary Decision Diagrams

An efficient way to find MCS is by converting the fault tree into a Binary Decision Diagram (BDD) [Ake78, Bry92]. A BDD is a directed acyclic graph that represents a boolean function $f : \{0,1\}^n \to \{0,1\}$. The leaves of a BDD are labeled with either 0 or 1. The other nodes are labeled with a variable x_i and have two children. The left child represents the function in case $x_i = 0$; the right child represents the function $x_i = 1$. BDDs are heavily used in model checking, to efficiently represent the state space and transition relation [CM93, CGP99].

Example 8 *Figure 2.6d shows the BDD obtained by converting the FT in Figure 2.6a. Each circle represents a BE, and has two children: a 0-child containing the sub-BDD that determines the system status if the 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. For example, if components E_1 and E_2 have failed, we begin traversing the BDD at its root, observe that E_1 has failed, and follow the 1-edge. From here, since E_3 is operational we follow the 0-edge. We have now reached a leaf containing a 0, so this combination does not result in a system failure.*

To construct a BDD from a boolean formula $f(x_1, x_2, \cdots, x_n)$, one can use the Shannon expansion formula [Ake78] (slightly reformulated):

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} f(1, x_2, \cdots, x_n) & \text{if } x_1 = 1 \\ f(0, x_2, \cdots, x_n) & \text{if } x_1 = 0 \end{cases}$$

Based on this, the if-then-else method [CM93] translates a fault tree (or any boolean function) to a BDD as follows: we let x_1 be the top node, and $f(0, x_2, \cdots, x_n)$ and $f(1, x_2, \cdots, x_n)$ the functions for its children. We repeat this procedure for x_2, x_3, and so on until we reach the leaves of the tree. At the leaves, all variables have been replaced by constant values, and we evaluate the function to determine the value of that leaf.

Example 9 *Figure 2.6 shows how to obtain a BDD from a fault tree. We represent the fault tree as its boolean formula and begin the procedure: First, we introduce a BDD node for E_1 and write its 0-child as the formula with $\bot$ substituted for E_1, and its 1-child where a $\top$ is substituted. We simplify the resulting formulas by eliminating obviously irrelevant operators and variables (e.g., $\top \vee E_2 = \top$ regardless of E_2). Omission of this step would result in a larger, but still correct, BDD. We now repeat this procedure for the 0-child by splitting this node using the variable E_2. We replace the formula $\bot$ by a 0-leaf. Observing that we have two nodes with formula 'E_3', we construct the node for*

E_3, replace its children by leaves as they contain no more variables, and connect the 1-edges of E_1 and E_2 to this new node.

An alternative method for constructing BDDs was developed by [WH00]: the component-connection method. An example of this method is shown in Figure 2.7. This method first constructs small BDDs for every gate in the tree (e.g., for an AND-gate, this BDD consists of the children of the gates connected such that a 0 from any child leads to the 0-leaf, and 1s from every child leads to the 1-leaf). Next, any BDD nodes that correspond to other gates are replaced by the BDD of those gates. In this substitution, edges to the 0-leaf in the child BDD are replaced by the 0-edges of the parent BDD, and likewise for the 1-leaf. This substitution is performed until only one BDD remains in which all nodes correspond to basic events.

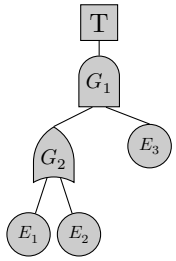
Obtaining cut sets using BDDs Cut sets can be determined from the BDD by starting at all 1-leaves of the tree, and traversing upwards toward the root. The set of all BEs reached by traversing a 1-edge from a particular leaf forms one CS.

The CS obtained by this method may not be minimal, depending on the algorithm used to construct the BDD. One approach to obtain MCSs is to perform a minimization on the BDD [Rau93] which ensures that only MCSs will be found. Alternatively, Rauzy and Dutuit [RD97] provide a method to construct BDDs encoding so-called prime implicants, from which MCSs can be directly computed.

The BDD method was first coined by [CM93] as well as by [Rau93]. It was improved by [SA96] by adding a minimization algorithm for the intermediate BDD. 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 [Sin96]. This is strongly influenced by the fact that BDDs very compactly represent boolean functions with a high degree of symmetry [RBM91], and fault trees exhibit this symmetry as the gates are symmetric in their inputs. A program that analyzes FTs using BDDs has been produced by [CM94].

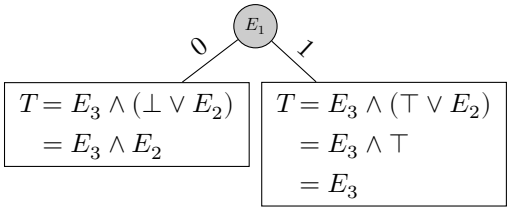
As for any boolean function, the conversion of an FT to a BDD is not unique: Depending on the ordering of the BEs, different BDDs can be generated. Good variable ordering is important to keep the BDD small. Unfortunately, even determining whether a given ordering of variables is optimal is an NP-complete problem [BW96]. Figure 2.8 shows that a different variable ordering can reduce the size of the resulting BDD.

Remenyte and Andrews [RA06, RPA08] have compared several different methods for constructing BDDs from FTs, and conclude that a hybrid of the if-then-else method [Rau93] and the advanced component-connection method by [WH00] is a good trade-off between processing time and size of the resulting BDD.

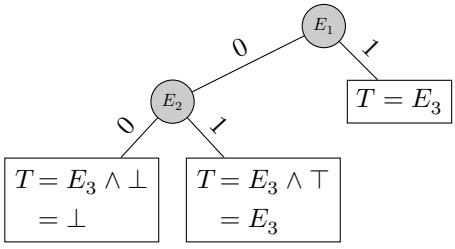


$T = E_3 \wedge (E_1 \vee E_2)$

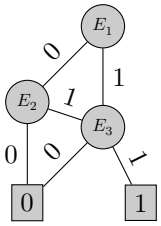
(a) Fault tree with boolean formula



(b) Introduction of BDD node for E_1

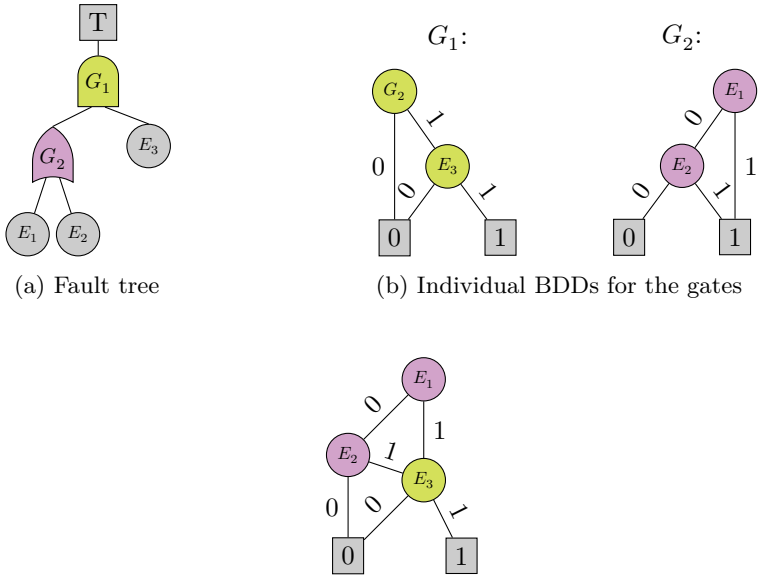


(c) Splitting of the leftmost child based on E_2



(d) Unification and splitting of the two children containing formula E_3

Figure 2.6: Example conversion of SFT to BDD using the if-then-else method (some simplification of the boolean formulas at the intermediate steps was performed for clarity).



(a) Fault tree

(b) Individual BDDs for the gates

(c) Substitution of the BDD for G_2 into the BDD for G_1

Figure 2.7: Example conversion of SFT to BDD using the component-connection method. Colors show where the elements of the combined model originated.

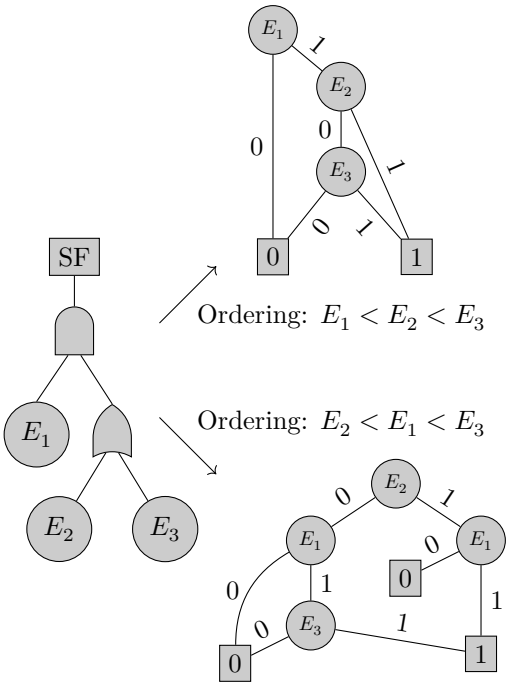


Figure 2.8: Example of how variable ordering affects BDD size. The upper BDD requires 3 nodes, while placing E_2 ahead of E_1 require the E_1 node to be present twice, as shown in the lower BDD.

Improvements to BDDs Tang and Dugan [TD04] propose the use of zero-suppressed BDDs (ZBDDs) to compute minimal cut sets. ZBDDs are similar to BDDs, except that nodes are not omitted if both their children are identical, but rather if their 1-child is a 0-leaf. This provides a more compact encoding if most cases lead to 0-leaves. Combined with reduction rules to ensure minimality of the cut sets, [TD04] shows that this approach is often more efficient for FTs than those based on classic BDDs in both time and memory use.

Dutuit and Rauzy [DR96] provide an algorithm for finding independent sub-modules (i.e., disjoint subtrees) of FTs, which can be converted separately to BDDs and analyzed, reducing the time and memory required to analyze the entire tree.

If subtrees of an FT are shared, then the approach by [CR06] called 'Parametric Fault Trees' can be used. This method performs qualitative and quantitative analysis on such a tree without repeating the analysis for each repetition of a subtree.

Miao et al. [MNTL13] have developed an algorithm to determine minimal cut sets using a modified BDD, and claim its time complexity is linear in the number of BEs, although their paper does not seem to support this claim. Moreover, this result seems incorrect to us, since the number of MCSs is already exponential in the number of BEs.

Other methods for qualitative analysis

FTs with voting gates with many inputs induce a combinatorial explosion in the number of mminimal cut sets, since a k/N voting gate means each combination of k failed components results in a separate cut set. The concept of a *minimal cut vote* was proposed by [XYM+11] as a term in an MCS to represent an arbitrary combination of k elements. This method is of linear complexity in the number of inputs to a voting gate, while the BDD approach has exponential complexity.

For relatively large trees with few cut sets, the algorithm by Carrasco and Suñé [Cn99] may be useful. Its space complexity is based on the MCSs, rather than the complexity of the tree like for BDDs. However, according to the article this method does seem to be slower than the BDD approach.

In practice, it is often not necessary to determine all MCSs: Cut sets with many components are usually unlikely to have all these components fail. It is often sufficient to only find MCSs with at most a small (depending on the system and desired accuracy) number of components. This may allow a substantial reduction in computation time by reducing the size of intermediate expressions [LGTL85].

Due to the potentially very large intermediate expressions, the earlier methods for finding MCSs can have large memory requirements. A Monte Carlo method can be used as an alternative. In the method by [VN70], random subsets of components are taken to be failed, according to the failure probabilities. If a subset causes a top event failure, it is a cut set. Additional simulations reduce these cut sets

into MCSs. While the memory requirements of the Monte Carlo method are much smaller, the large number of simulations can greatly increase computation time. In addition, there is a chance that not all MCSs are found.

2.3.2 Minimal path sets

A *minimal path set* (MPS) is essentially the opposite of an MCS: It is a minimal set of components such that, if they do not fail, the system remains operational.

Defintion 4 *$P \subseteq BE$ is a path set of FT F if $\pi_F(BE \backslash P) = 0$.*

Example 10 *In Figure 2.2, an MPS is $\{B, C1, M1, M2, PS\}$.*

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

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

then we can write a boolean expression for the TE as

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

Minimal Path Sets can, like MCSs, be used as a starting point for improving system reliability. Especially if the system has an MPS with few elements, improving the reliability such an MPS may improve the reliability of many MCSs.

Analysis

Any algorithm to compute MCSs can also be used to compute MPSs. In simple terms, we negate the fault tree (i.e., the top level event now represents system non-failure) and the basic events (now representing non-failed components) and find the minimal cut sets of this negated tree. I.e., we find the smallest sets of functional components that do not cause system failure. Boolean logic such as De Morgan's laws allow us to negate the tree without introducing a NOT-gate as follows: AND gates are replaced by OR gates, OR gates by AND gates, k/N voting gates by $(N - k + 1)/N$ 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 [BP75].

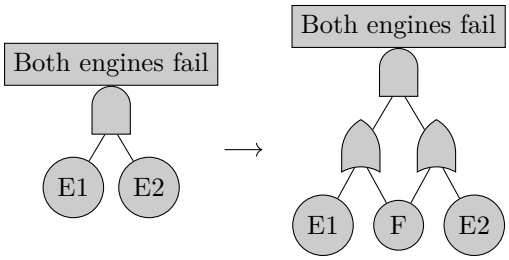


Figure 2.9: Example partial FT of a two-engine airplane, showing the addition of common cause 'fuel' (F) of two otherwise-independent engines (E1 and E2).

2.3.3 Common cause failures

Another qualitative aspect is the analysis of probable *common cause failures* (CCF). These are separate failures that can occur due to a common cause that is not yet listed in the tree. For example, if a component can be replaced by a spare to avoid failure, both this component and its spare are in one cut set. If the spare is produced by the same manufacturer as the component, a shared manufacturing defect could cause both to fail at the same time. If such common causes are found to be too likely, they should be modeled explicitly to avoid overestimating the system reliability.

Analysis

CCF analysis is generally not possible using automated methods from the FT alone, since CCF depend on external factors not modeled in the tree. Instead, experts may try to determine whether any cut sets have multiple components that are susceptible to a common cause failure. Such an analysis relies on expert insight, and is often quite informal.

Common causes can be added to an FT by inserting them as BEs and replacing the BEs they affect by OR-gates combining the CCF and the separate failure modes. An example is shown in Figure 2.9, modeling a two-engine airplane. While both engines can fail independently, the common cause 'fuel' can can both engines to fail at the same time.

2.4 Quantitative analysis: Single-time

Quantitative analysis methods derive relevant numerical values for fault trees. *Stochastic measures* are wide spread, as they provide useful information such as failure probabilities. *Importance measures* indicate how important a set of

Model	Reliability	Availability	MTTFF	MTTF	MTBF	MTTR	ENF
Discrete-time	+						+
Continuous-time	+	+	+				+
Repairable cont.-time	+	+	+	+	+	+	+

Table 2.2: Applicability of stochastic measures to different FT types

Author	Measures	Remarks	Tool
[VGRH81]	Reliability	Valid for infrequent failures	-
[BP75]	Reliability	Exact calculation based on MCS	KTT [VN70]
[Rau93]	Reliability	Exact, Uses BDDs for efficiency	-
[Ste86]	Reliability	Efficient for shared subtrees	-
[BPMC01]	Reliability	Allows dependent events	DBNet [MPB+05b]
[DRGSR+09]	Reliability	Monte Carlo, allows arbitrary distributions	DRSIM [DRGSR+09]
[AZ13]	Reliability	Fast Monte Carlo, requires special hardware	-
[BP75]	Availability	Translation to reliability problem	-
[DRGSR+09]	Availability	Monte Carlo, allows arbitrary distributions	DRSIM [DRGSR+09]
[AA04]	MTTF	Assumes exponential failure distributions	-
[Sch98]	MTBF	Exact method based on boolean expression	SyRePa [Sch90]
[AA04]	MTBF	Assumes exponential failure distributions	-

Table 2.3: Summary of quantitative analysis methods for SFTs

components is to the reliability of the system. Moreover, the *sensitivities* of these measures to variations in BE probabilities are important.

Moreover, stochastic measures can be used to decide whether it is safe to continue operating a system with certain component failures, or whether the entire system should be shut down for repairs.

We consider two types of FTs for quantitative analysis: Single-time FTs decorate each basic event with a single probability of failure, abstracting away the time at which the failure occurs. This is useful for systems with fixed, well-known mission times (e.g., a rocket where the interesting measure is the probability that it successfully reaches its goal, rather than how exactly when it fails).

Continuous-time FTs attach a time-dependent failure probability to each basic event, allowing the computation of various additional measures, such as mean time to failure or average uptime. Such FTs are useful for systems whose lifespan is not fixed or known in advance (e.g., an airplane where the maximal lifespan can actually be determined by computing the time when the failure probability becomes unacceptably high).

The next section first describes some basic probability theory, and then provides definitions and analysis techniques for several measures applicable to single-time FTs. In particular, we treat the basic methods of bottom-up propagation and Monte Carlo simulation, and the more advanced methods of binary decision diagrams (BDDs) and Bayesian networks.

Preliminaries on probability theory

Given a finite or countable set $\mathbb{S}$, a discrete $\mathbb{S}$-valued random variable is a function $X : \Omega \to \mathbb{S}$ that assigns an outcome $s \in \mathbb{S}$ to each stochastic experiment. The function $\mathbb{P}[X = s]$ denotes the probability that X gets value s and is called the *probability mass function*. In this section, we consider Boolean random variables, i.e., $s \in \{0, 1\}$ where $s = 1$ denotes a failure, and $s = 0$ a working FT element. Note that if $X_1, X_2, ... X_n$ are random variables, and $f : \mathbb{S}^n \to \mathbb{S}$ is a function, then $f(X_1, X_2, ... X_n)$ is a random variable as well.

2.4.1 Modeling failure probabilities

We first consider the relative simple case of the single-time FT. Here, we do not consider the evolution of a system over time: rather, a fixed time horizon is considered, during which each component can fail only once. This is applicable to many systems that have a well-known timeframe of interest, such as a rocket where the designers know the probability of each component failing during launch, and it is not interesting at exactly what time during this launch the rocket may fail.

We assume that the failures of the BEs are stochastically independent. If the FT has shared subtrees, then the failures of the gates are not independent.

The basic process of our analysis is that we attach a failure probability to each BE, and then combine these probabilities at the gates to obtain failure probabilities for the gates and, eventually, for the entire tree. To explain how the probabilities are combined, we state that each BE is either failed or not (with specified probabilities), and that each gate is likewise failed or not depending on the states of its child elements.

Formally, the BEs are equipped with a *failure probability function* $P : BE \to [0, 1]$ that assigns a failure probability $P(e)$ to each $e \in BE$, see Figure 2.11. Then, each BE e can be associated with random variable $X_e \sim \text{Alt}(P(e))$; that is $\mathbb{P}(X_e = 1) = P(e)$ and $\mathbb{P}(X_e = 0) = 1 - P(e)$.

Given a fault tree F with BEs $\{e_1, e_2, \dots, e_n\}$, the semantics from Definition 2 yields a stochastic semantics for each gate $g \in G$, namely as the random variable $\pi_F(X_{e_1}, \dots, X_{e_n}, g)$. We abbreviate the random variable for the top event of FT F as X_F.

Note that under these stochastic semantics, it holds for all $g \in G$ that

- $X_g = \min_{i \in I(g)} X_i$, if $T(g) = AND$,
- $X_g = \max_{i \in I(g)} X_i$, if $T(g) = OR$,
- $X_g = \left(\sum_{i \in I(g)} X_i \right) \geq k$, if $T(g) = VOT(k/N)$.

Example 11 *Figure 2.10 shows how these semantics work in an example FT. We assume that some of the children have failed (denoted by a 1) and others have not. Now, we can compute whether each gate has failed following the rules above. For the lower AND-gate, since a single functioning child means the gate continues to function, we select the minimal value of the children (0 in this example). For the OR-gate the converse holds: a single failed child means the gate has failed. We thus select the maximum of its children, which is here a 1. We therefore conclude that this combination of failed BEs leads to a failure of the top gate.*

2.4.2 Reliability

The *reliability* of a single-time FT is the probability that the failure does not occur during the (modeled) life of the system [BP75].

Defintion 5 *The reliability of a single-time FT F is defined as $Re(F) = \mathbb{P}(X_F = 0)$.*

The reliability of a fault tree F with BEs $e_1, \dots, e_n$ can be derived from the non-stochastic semantics by using the stochastic independence of the BE failures:

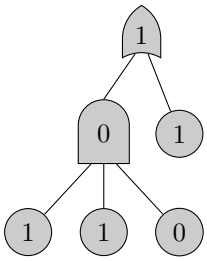


Figure 2.10: Example of the semantics of the FT gates. The children are decorated with 0 if they are functioning, or 1 otherwise. The value of the lower AND-gate is then the minimum of its children (i.e., 0 in this example) and of the upper OR-gate is the maximum (i.e., 1).

$$\begin{aligned}\mathbb{P}(X_F = 1) = &\sum_{b_1,\ldots,b_n \in \{0,1\}} \mathbb{P}(X_F = 1 | X_{e_1} = b_1 \wedge \ldots \wedge X_{e_n} = b_n) \\ &\cdot \mathbb{P}(X_{e_1} = b_1 \wedge X_{e_n} = b_n) \\ = &\sum_{b_1,\ldots,b_n \in \{0,1\}} \pi_F(b_1, \ldots, b_n) P_{b_1}(e_1) \cdot \ldots \cdot P_{b_n}(e_n)\end{aligned} \tag{2.1}$$

Here, $P_1(e) = P(e)$ and $P_0(e) = 1 - P(e)$. Computing (2.1) directly is complex. Below, we discuss several methods to speed up the reliability analysis.

Bottom up analysis

For systems without shared BEs, failure probabilities can be easily propagated from the bottom up, by using standard probability laws.

Example 12 *Figure 2.11 shows an example of how such probabilities propagate. Failure of the AND-gate requires all inputs to fail, which has a probability of* $0.3 \cdot 0.4 \cdot 0.1 = 0.012$. *The OR-gate fails if any input fails, i.e., remains operational only if all inputs do not fail. This has probability* $1 - (1 - 0.012)(1 - 0.1) = 0.1108$.

In more detail, if we have an AND-gate G and the input distributions $X_1, X_2, \ldots, X_n$ of its children are all stochastically independent (i.e., there are no shared subtrees), then we have

$$\begin{aligned}\mathbb{P}[X_{AND}(X_1, \ldots, X_n) = 1] &= \mathbb{P}[X_1 = 1 \wedge \ldots \wedge X_n = 1] \\ &= \mathbb{P}[X_1 = 1] \cdot \ldots \cdot \mathbb{P}[X_n = 1]\end{aligned}$$

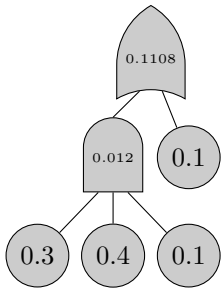


Figure 2.11: Example FT showing the propagation of failure probability in a single-time FT.

For the *OR*, we use

$$\begin{aligned}\mathbb{P}[X_{OR}(X_1,\dots,X_n)=1] &= 1-\mathbb{P}[X_{OR}(X_1,\dots,X_n)=0]\\ &= 1-\mathbb{P}[X_1=0\wedge\dots\wedge X_n=0]\\ &= 1-(1-\mathbb{P}[X_1=1])\cdot\dots\cdot(1-\mathbb{P}[X_n=1])\end{aligned}$$

The *VOT(k/N)* gate is slightly more involved. It is possible to rewrite the gate into a disjunctions of all possible sets of k inputs, obtaining

$$\begin{aligned}\mathbb{P}[X_{VOT(k/N)}(X_1,\dots,X_n)=1] = \mathbb{P}[&(X_1=1\wedge\dots\wedge X_k=1)\\ &\vee(X_1=1\wedge\dots\wedge X_{k-1}=1\wedge X_{k+1}=1)\\ &\dots\\ &\vee(X_{n-k}=1\wedge\dots\wedge X_n=1)]\end{aligned}$$

however, expanding this into an expression of simple probabilities requires the use of the inclusion-exclusion principle and results in very large expressions for gates with many inputs where k is neither very small nor close to N. It is more convenient to recursively define the voting gate:

$$\begin{aligned}\mathbb{P}[X_{VOT(0/N)}(X_1,\dots,X_n)=1] &= 1\\ \mathbb{P}[X_{VOT(N/N)}(X_1,\dots,X_n)=1] &= \mathbb{P}[X_{AND}(X_1,\dots,X_n)=1]\\ \mathbb{P}[X_{VOT(k/N)}(X_1,\dots,X_n)=1] &= \mathbb{P}\left[(X_1=1\wedge X_{VOT(k\text{-}1/N\text{-}1)}(X_2,\dots,X_n)=1)\right.\\ &\qquad\left.\vee(X_1=0\wedge X_{VOT(k/N\text{-}1)}(X_2,\dots,X_n)=1)\right]\\ &= \mathbb{P}[X_1=1]\cdot\mathbb{P}[X_{VOT(k\text{-}1/N\text{-}1)}(X_2,\dots,X_n)=1]\\ &\quad+\mathbb{P}[X_1=0]\cdot\mathbb{P}[X_{VOT(k/N\text{-}1)}(X_2,\dots,X_n)=1)]\end{aligned}$$

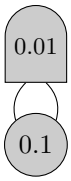


Figure 2.12: Example of an FT where bottom-up analysis fails due to non-independent subtrees. Here, the AND-gate syntactically has two children, which are actually the same BE. Since the rule for AND-gates is to multiply the failure probabilities of its children, we obtain a probability of 0.01 for the gate. In reality, of course, the failure probability of the gate should be the same as for the only child, namely 0.1.

This approach does not work when BEs are shared, since the dependence between subtrees is not taken into account. An extreme example of this is illustrated in Figure 2.12.

Binary Decision Diagrams

As discussed in Section 2.3.1, BDDs can be used to encode FTs very efficiently. In addition to the qualitative analysis already discussed, Efficient quantative analysis is also possible.

To construct a BDD for computing system reliability, one can use a method similar to Shannon decomposition [Rau93]:

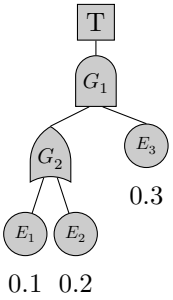
$$\begin{aligned}\mathbb{P}(f(x_1, x_2, \cdots, x_n)) = \mathbb{P}(x_1)\mathbb{P}(f(1, x_2, \cdots, x_n)) \\ + \mathbb{P}(\neg x_1)\mathbb{P}(f(0, x_2, \cdots, x_n))\end{aligned}$$

An example of this approach is shown in Figure 2.13.

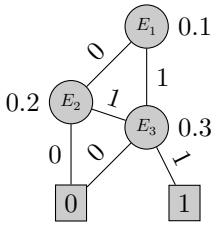
A caching mechanism is used to store intermediate results [Rau08], as intermediate formulas often occur is more than one subdiagram. This algorithm can be applied even to non-coherent FTs, and has a complexity that is linear in the size of the BDD.

Rare event approximation

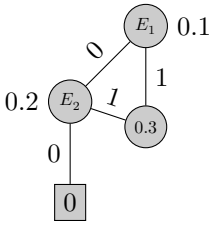
For systems with shared events, the total unavailability of the system can also be approximated by summing the unavailabilities of all the MCSs. This *rare event approximation* [SVD+02] is reasonably accurate when failures are improbable. However, as failures become more common and the probability of multiple cut



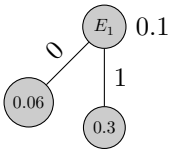
(a) Fault tree



(b) BDD with probabilities



(c) Substituting leaves for E_3



(d) Substituting children for E_2

Figure 2.13: Example of quantitative analysis of an FT via BDDs. First, we transform the FT into a BDD following one of the methods described in Section 2.3.1 and decorate the variables with their probabilities. Next, we take the lowest variable (E_3) and replace its probability by integrating its children (since the children here have probabilities 0 and 1, we get $(1-0.3)\cdot 0 + 0.3 \cdot 1 = 0.3$. We continue this process with the next variable (E_2), replacing its probability with $(1-0.2)\cdot 0 + 0.2 \cdot 0.3 = 0.06$), and finally with the last remaining variable E_1 obtaining $(1-0.1)\cdot 0.06 + 0.1 \cdot 0.3 = 0.084$ as the final probability.

sets failure increases, the approximation deviates more from the true value. For example, a system with 10 independent MCSs, each with a probability 0.1, has an unreliability of 0.65, whereas the rare event approximation suggests an unreliability of 1.

Example 13 *Considering Figure 2.2 and assuming all basic events have an unavailability of 0.1, the probability of a failure of gate Mem can be approximated as* $P_{fail}(Mem) \approx P_{fail}(\{M1, M2\}) + P_{fail}(\{M2, M3\}) + P_{fail}(\{M1, M3\}) = 0.03$. *As the actual probability is* 0.028*, the approximation has slightly overestimated the failure probability.*

If some cut sets have a relatively high probability, this rare event approximation is no longer accurate. If no component occurs in more than one cut set, the correct probability may be calculated as $P_{fail}(F) = 1 - \prod_{C \in MC(F)} (1 - P_{fail}(C))$.

If some components are present in many of the cut sets, more advanced analysis are needed. An exact solution may be obtained by using the inclusion-exclusion principle to avoid double-counting events. Alternative methods may be more efficient in special cases, such as the algorithm by [Ste86] which reduces repeated work if the FT contains shared subtrees.

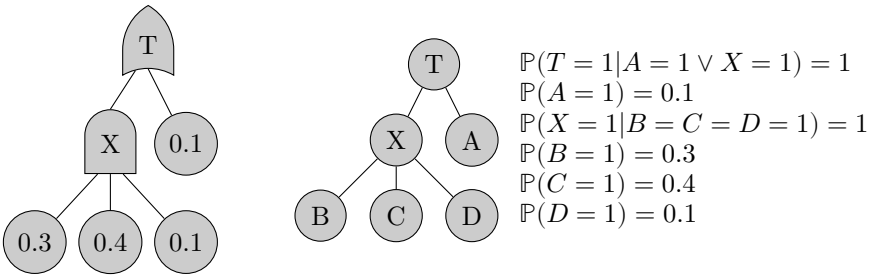
An algorithm using zero-suppressed BDDs [Rau08] closely resembles the calculation of MCSs, but instead computes system reliability using the rare event approximation. This method has a complexity linear in the size of the BDD, and is more efficient than first computing the MCSs and then the reliability.

Bayesian Network analysis

In order to accurately calculate the reliability of a fault tree in the presence of statistical dependencies between events, [BPMC01] present a conversion of SFT to Bayesian Networks. A *Bayesian Network* [BG08] is a sequence $X_1, X_2, \ldots, X_n$ of stochastically dependent random variables, where X_i can only depend on X_j if $j < i$. Indeed, the failure distribution of a gate in a FT only depends on the failure distributions of its children. Bayesian networks can be analyzed via conditional probability tables $\mathbb{P}[B|A_j]$ by using the law of total probability: for an event B, and a partition A_j of the event space, we have

$$\mathbb{P}[B] = \sum_j \mathbb{P}[B|A_j]\mathbb{P}[A_j]$$

For example, if X_4 depends on X_3 and X_2, then partitioning yields $\mathbb{P}[X_4 = 1] = \sum_{i,j \in \{0,1\}} \mathbb{P}[X_4 = 1 | X_3 = i \wedge X_2 = j] \mathbb{P}[X_3 = i \wedge X_2 = j]$. The values $\mathbb{P}[X_4 = 1 | X_3 = i \wedge X_2 = j]$ are given by conditional probability tables, and $\mathbb{P}[X_3 = i \wedge X_2 = j]$ are computed recursively.



(a) Fault tree (b) Resulting BN

Figure 2.14: The BN obtained by converting the FT in Figure 2.11 to a Bayesian Network

Example 14 *Figure 2.14 shows the conversion of a simple FT into a Bayesian Network. The BEs A, B, and C are connected to top event T and assigned reliabilities. Gates have conditional probabilities dependent on the states of their inputs. All nodes can have only states 0 or 1 corresponding to operational and failed, respectively. Classic inference techniques [BG08] can be used to compute* $P(T = 1)$*, which corresponds to system unreliability. Alternatively, if it is known that the system has failed, the inference can provide probabilities of each of the BEs having failed.*

In addition, Bobbio et al. [BPMC01] allow BEs with multiple states: Rather than being either up or failed, components can be in different failure modes, such as degraded operational modes, or a valve that is either stuck open or stuck closed. The Bayesian inference rules work the same for multiple-state fault trees, except that the random variables are no longer boolean but have multiple possible values, resulting in larger conditional probability tables. Also, Bobbio et al. [BPMC01] model common cause failures by adding a probability of a gate failing even when not enough of its inputs have failed, although this has the disadvantage of making the potential failure causes less explicit. Finally, gates can be 'noisy', meaning they have a chance of failure. For example, the failure of one element of a set of redundant components may have a small change of causing a system failure.

An important feature of Bayesian Network Analysis is that, not only can it compute the probability of the top event given the leaves, it can compute the probabilities of each of the leaves given the top event. This is very useful in fault diagnosis [LP07, Lam10], where one knows that a failure has occurred, and wants to find which leaves are the most like causes. Additional evidence can also be given, such as certain leaves that are known not to have failed.

Monte Carlo simulation

Monte Carlo methods can also be used to compute the system reliability. Most techniques are designed for continuous-time models [Cro71, DRGSR+09] or qualitative analysis [VN70], but adaptation to single-time models is straightforward. Each component is randomly assigned a failure state based on its failure probability. The FT is then evaluated to determine whether the TE has failed. Given enough simulations, the fraction of simulations that does not result in failure is approximately the reliability.

2.4.3 Expected Number of Failures

Definition

The *Expected Number of Failures* (ENF) describes the expected number of occurrences of the TE within a specified time limit. This measure is commonly used to evaluate systems where failures are particularly costly or dangerous, and where the system will operate for a known period of time.

A major advantage of the ENF is that the combined ENF of multiple independent systems over the same timespan can very easily be calculated, namely $ENF(S1, S2) = ENF(S1) + ENF(S2)$. For example, if a power company requests a number of 40-year licenses to operate nuclear power stations, it is easy to check that the combined ENF is sufficiently low.

Analysis

Since a single-time system can fail at most once, it is easy to show that the ENF of such a system is equal to its unreliability. Let N_F denote the number of failures system F experiences during its mission time, so that

$$\begin{aligned}\mathbb{E}[N_F] &= \sum_i i \cdot \mathbb{P}[N_F = i] \\ &= 0 \cdot \mathbb{P}[N_F = 0] + 1 \cdot \mathbb{P}[N_F = 1] \\ &= 0 + \mathbb{P}[X_F = 1] \\ &= Re(F)\end{aligned}$$

2.5 Quantitative analysis: Continuous-time

Where single-time systems treat the entire lifespan of a system as a single event, it is often more useful to consider dependability measures at different times. For example, an airplane manufacturer may be interested in knowing when the failure probability exceeds a certain threshold, so that the airplane can be maintained or

taken out of service before that time. Provided adequate information is available, continuous-time fault trees provide techniques to obtain such measures. This section provides, after a description of the basic theory, definitions and analysis techniques for these measures.

2.5.1 BE failure probabilities

Continuous-time FTs consider the evolution of the system failures over time. The component failure behaviour is usually given by a probability function $D_e : \mathbb{R}^+ \mapsto [0,1]$, which yields for each BE e and time point t, the probability that e has failed before time t (i.e., $D_e(t) = \mathbb{P}[\text{event } e \text{ fails before time } t]$).

In practise, the failure distributions can often be adequately approximated by exponential distributions, and BEs are specified with a failure rate $R : BE \mapsto \mathbb{R}^+$, such that $R(e) = \lambda \leftrightarrow D_e(t) = 1 - \mathrm{e}^{-\lambda t}$.

If components can be repaired without affecting the operations of other components, BEs have an additional repair distribution over time. Like failure distributions, repair distributions are often exponentially distributed and specified using a repair rate $RR : BE \mapsto \mathbb{R}^+$. More generally, BEs can be assigned repair distributions as $RD_e : \mathbb{R}^+ \mapsto [0,1]$. More complex and realistic models of repairs are discussed in Section 4.3, this section does not consider such models.

Operational semantics. Like for the single-time case, we can use random variables X_e to describe failures of basic events, and derive a stochastic semantics for the FT. However, due to the possibility of repair, it is helpful to introduce some additional variables. Consider a BE e with a failure distribution D_e and repair distribution RD_e. Now we take $F_{e,1}, F_{e,2}, ...$ as the relative failure times, and $Q_{e,1}, Q_{e,2}, ...$ as the relative repair times, with $Q_{e,1} = 0$ for convenience. It follows that $\mathbb{P}[F_{e,i} \leq t] = D_e(t)$ and $\mathbb{P}[Q_{e,i} \leq t] = RD_e(t)$ for $i > 1$. We can now define the random variables X_e and X_g.

For basic events, $X_e(t)$ is 1 if t is some time after a failure, and before the subsequent repair. We can rewrite this as follows:

$$
\begin{aligned}
& X_e(t) = 1 \text{ iff} \\
& \exists_i \left[\sum_{j<i} (Q_{e,j} + F_{e,j}) \leq t \wedge Q_{e,i} + \sum_{j<i} (Q_{e,j} + F_{e,j}) > t \right] \\
\Leftrightarrow\ & \exists_i \left[\sum_{j<i} (Q_{e,j} + F_{e,j}) \leq t \wedge t - Q_{e,i} < \sum_{j<i} (Q_{e,j} + F_{e,j}) \right] \\
\Leftrightarrow\ & \exists_i \left[t - Q_{e,i} \leq \sum_{j<i} (Q_{e,j} + F_{e,j}) \leq t \right]
\end{aligned}
$$

For gates, $X_g(t)$ is defined analogously to the single-time case. To summarize, we have the following definition:

Defintion 6

$$X_e(t) = \begin{cases} 1 & \textit{if } \exists_i : t - Q_{e,i} < \sum_{j<i}(Q_{e,j} + F_{e,j}) \leq t \\ 0 & \textit{otherwise} \end{cases}$$

$$X_g(t) = \begin{cases} \min_{i \in I(g)} X_i(t) & \textit{if } T(g) = And \\ \max_{i \in I(g)} X_i(t) & \textit{if } T(g) = Or \\ \left(\sum_{i \in I(g)} X_i(t) \right) \geq k & \textit{if } T(g) = Vote(k/N) \end{cases}$$

Depending on the failure distributions, the random variables of the BEs can have relatively easy distributions. For example, a BE with exponentially distributed failures with rate λ has probability $\mathbb{P}(X_e(t) = 0) = 1 - \mathrm{e}^{-\lambda t}$. The distributions of the gates typically do not follow convenient distributions, as e.g. the maximum of two exponentially distributed variables is not exponentially distributed.

Given the definition of X_i, classic statistical methods may be used to analyze the FT. For example, the *availability* of an FT F is described as $A(F) = \lim_{t \to \infty} \mathbb{E}(X_F(t))$, as explained in Section 2.5.3.

This method of analysis can be applied to FTs with arbitrary failure distributions, even if the BEs are statistically dependent on each other. Unfortunately, the algebraic expressions for the probability distributions often become too large and complex to calculate, so other techniques have to be used for larger FTs.

2.5.2 Reliability

Definition

The *reliability* of a continuous-time FT F is the probability that the system it represents operates for a certain amount of time without failing. Formally, we define a random variable $Y_F = \max\{t | \forall_{s<t} X_F(s) = 0\}$ to denote the time of the first failure of the tree. The reliability of the system up to time t is then defined as $Re_F(t) = \mathbb{P}(Y_F > t)$.

Analysis

In continuous-time systems, the reliability in a certain time period can be calculated by conversion into a single-time system, taking BE probabilities as the probability of failure within the specified timeframe.

Monte Carlo methods can also be used to compute system reliability. In the method by [DRGSR+09], random failure times and, if applicable, repair times are generated according to the BE distributions. The system is simulated with these failures, and the system reliability and availability recorded. Given enough simulations, reasonable approximations can be obtained. Modifying the method to record other failure measures is trivial.

For higher performance than conventional computer simulation, [AZ13] have developed a method for programming a model of an FT into a special hardware chip called a Field Programmable Gate Array, which can perform each MC simulation very quickly.

2.5.3 Availability

Definition

The *availability* of a system is the probability that the system is functioning at a given time. Availability can also be calculated over an interval, where it denotes the fraction of that interval in which the system is operational [BP75]. Availability is particularly relevant for repairable systems, as it includes the fact that the system can become functional again after failure. For non-repairable systems, the availability in a given duration may still be useful. The long-run availability always tends to 0 for nontrivial non-repairable systems, as eventually some cut set will fail and remain nonfunctional.

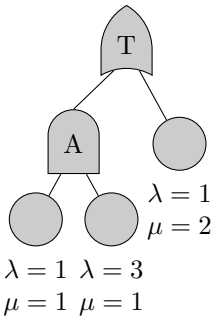
Defintion 7 *The availability of FT F at time t is defined as $A_F(t) = \mathbb{E}(X_F(t))$. The availability over the interval $[a, b]$ is defined as $A_F([a, b]) = \frac{1}{b-a}\int_a^b X_F(t)dt$. The long-run availability is $A_F = \lim_{t\to\infty} A_F([0, t])$ or equivalently, $A_F = \lim_{t\to\infty} A_F(t)$ when this limit exists.*

Analysis

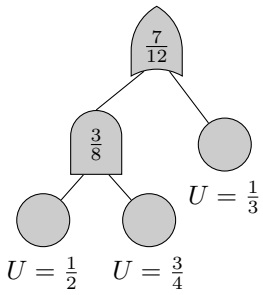
As the availability at a specific time is a probability, it is possible to treat the FT as a single-time FT, by replacing the BE failure distribution with the probability of being in a failed state at the desired time. The single-time reliability of the resulting FT is then the availability of the original. Failure probabilities of the BEs are usually easy to calculate depending on the failure time distribution, also for repairable FTs [BP75].

In the case of exponentially distributed failure times (with rate λ) and repair times (with rate μ), we obtain that the mean availability of the component is given by $A = \frac{\mu}{\lambda+\mu}$. An example of how this is used to compute the availability of an FT is shown in Figure 2.15.

Long-term availability of a system can be calculated the same way, provided the limiting availability of each BE exists. This is the case for most systems.



(a) Fault tree with failure rates λ and repair rates μ



(b) Corresponding FT with mean unavailabilities U

Figure 2.15: Example of computing the mean availability of an FT with failure and repair rates by first converting the rates to unavailabilities of BEs, and then applying the bottom-up method.

Availability over an interval cannot be calculated so easily. Since this availability is defined as an integral over an arbitrary expression, no closed-form expression exists in the general case. Numerical integration techniques can be used should this availability be needed.

2.5.4 Mean Time To Failure

Definition

The Mean Time To Failure (MTTF) describes the expected time from the moment the system becomes operational, to the moment the system subsequently fails.

Formally, we introduce an additional random variable $Z_F(t)$ denoting the number of times the system has failed up to time t.

Defintion 8 *To define $Z_F(t)$, we first define the failure and repair times of the gate:*

$$
\begin{aligned}
Q_{g,1} &= 0 \\
F_{g,i} &= \min\{t > Q_{g,i} | X_g(t) = 1\} - Q_{g,i} \\
Q_{g,i} &= \min\{t > F_{g,i-1} | X_g(t) = 0\} - F_{g,i-1}
\end{aligned}
$$

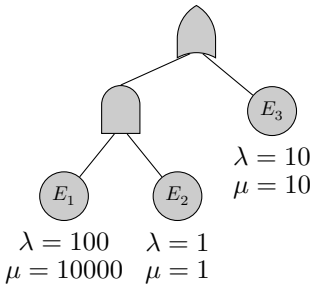


Figure 2.16: Example FT of a repairable system where MTTF and MTTFF differ significantly. Failure rates are denoted by λ, repair rates by μ.

We then define $Z_g(t)$ of a gate g as:

$$Z_g(t) = \max\left\{ i \in \mathbb{N} \middle| \sum_{j \leq i} (Q_{g,j} + F_{g,j}) \leq t \right\}$$

Now $Z_F(t) = Z_T(t)$ with T being the TE of FT F.

The MTTF up to time t is then $MTTF_F(t) = \frac{A_F(t) \cdot t}{Z_F(t)}$. The long-run MTTF is $MTTF_F = \lim_{t \to \infty} MTTF_F(t)$.

In repairable systems the time to failure depends on the system state when it becomes operational. The first time, all components are operational, but when the system becomes operational due to a repair, some components may still be non-functioning. This difference is made explicit by distinguishing between *Mean Time To First Failure* (MTTFF) and MTTF.

To illustrate this difference, consider the FT in Figure 2.16. Here, failures will initially be caused primarily by component 3, resulting in an MTTFF slightly less than $\frac{1}{10}$. In the long run, however, component 1 will mostly be in a failed state, and component 2 will cause most failures. This results in a long-run MTTF of approximately 1.

While MTTF and availability are often correlated in practise, only the MTTF can distinguish between frequent, short failures and rare, long failures.

Analysis

Many failure distributions have expressions to immediately calculate the MTTF of components. For example, a component with exponential failure distribution with rate λ has MTTF $\frac{1}{\lambda}$. For gates, however, the combination of multiple BE often

does not have a failure distribution of a standard type, and algebraic calculations produce very large equations as the FTs become more complex.

[AA04] have shown that the Vesely failure rate [Ves70] can be used to approximate the MTTF, and can do so efficiently even for larger trees.

2.5.5 Mean Time Between Failures

Definition

For repairable systems, the *Mean Time Between Failures* (MTBF) denotes the mean time between two successive failures. It consists of the MTTF and the *Mean Time To Repair* (MTTR). In general, it holds that MTBF = MTTR + MTTF.

The MTBF is defined similarly to the MTTF except ignoring the unavailable times. Formally, $MTBF_F(t) = \frac{t}{Z_F(t)}$, and in the long run $MTBF_F = \lim_{t\to\infty} MTBF_F(T)$ when this limit exists.

The MTBF is useful in systems where failures are particularly costly or dangerous, unlike availability which focuses more on total downtime. For example, if a railroad switch failure causes a train to derail, the fact that an accident occurs is much more important than the duration of the subsequent downtime.

The MTTR is often less useful, but may be of interest if the system is used in some time-critical process. For example, even frequent failures of a power supply may not be very important if a battery backup can take over long enough for the repair, while infrequent failures that outlast the battery backup are more important.

Analysis

An exact value for the MTBF may be obtained using the polynomial form of the FT's boolean expression, as described by [Sch98]. The Vesely failure rate approximation by [AA04] can also be used.

2.5.6 Expected Number of Failures

Definition

Like in a single-time FT, the ENF denotes the expected number of times the top event occurs within a given timespan. For repairable systems, it is possible for more than one failure to be expected.

Analysis

The ENF of a non-repairable system is equal to its unreliability. The ENF of a repairable system can be calculated from the MTBF using the equation $ENF(t) = \frac{t}{MTBF(t)}$, or using simulation.

Author	Measure	Remarks
Various	Cut set size	Very rough approximation
Various	Cut set failure measure	Specific to each failure measure
[Ves70]	Cut set failure rate	Applicable to exponential distributions
[Bir68]	Structural importance	Based only on FT structure
[Jac83]	Structural importance	Also for noncoherent systems
[AB03]	Structural importance	Also includes repairs
[CM11]	Init. & Enab. importance	For FTs with initiating and enabling events
[HL93]	Joint Reliability Importance	Interaction between pairs of events
[Arm95]	Joint Reliability Importance	Also for dependent events
[LJ07]	Joint Reliability Importance	Also for noncoherent systems
[Fus75]	Primary Event Importance	BE contribution to unavailability
[DR01]	Risk Reduction Factor	Maximal improvement of reliability by BE

Table 2.4: Summary of importance measures for cut sets and components

2.5.7 Sensitivity analysis

Quantitative techniques produce values for a given FT, but it is often useful to know how sensitive these values are to the input data. For example, if small changes in BE probabilities result in a large variation in system reliability, the calculated reliability may not be useful if the probabilities are based on rough estimates. On the other hand, if the reliability is very sensitive to one particular component's failure rate, this component may be a good candidate for improvement.

If the quantitative analysis method used gives an algebraic expression for the failure probability, it may be possible to analyze this expression to determine the sensitivity to a particular variable. One method of doing so is provided by [Rus85].

In many cases, however, sensitivity analysis is performed by running multiple analyses with slightly different values for the variables of interest.

If the uncertainty of the BE probabilities is bounded, an extension to FT called a *Fuzzy Fault Tree* can be used to analyze system sensitivity. This method is explained in Section 4.1.

2.6 Importance measures

In addition to computing reliability measures of a system, it is often useful to determine which parts of a system are the biggest contributors to the measure. These parts are often good candidates for improving system reliability.

In FTs, it is natural to compute the relative importances of the cut sets, and of the individual components. Several measures are described below, and the applicability of these measures is summarized in Table 2.4.

MCS size

An ordering of minimal cut sets can be made based on the number of components in the set. This ordering approximately corresponds to ordering by probability, since a cut set with many components is generally less likely to have all of its elements fail than one with fewer components. Small Cut sets are therefore good starting points for improving system reliability.

Stochastic measures

For a more exact ordering, the stochastic measures described above can also be calculated for each cut set, and used to order them.

For systems specified using exponential failure distributions, the probability $W(C,t)\Delta t$ of cut set C causing a system failure between time t and Δt is approximately the probability that all but one BE of C have failed at time t and that the final component fails within the interval Δt. If we write the failure rate of a component x as λ_x, and we write $Re_x(t)$ for the reliability of x up to time t, the probability of cut set C causing a failure in a small interval can be approximated as

$$W(C,t)\Delta t \approx \sum_{x \in C} \left(\lambda_x \Delta t \prod_{y \in (C \setminus \{x\})} Re_y(t) \right)$$

Cancelling the Δt on both sides gives

$$W(C,t) \approx \sum_{x \in C} \left(\lambda_x \prod_{y \in (C \setminus \{x\})} Re_y(t) \right)$$

This approximation assumes that the all components in C failing causes a system failure, which requires that no other cut set has failed before C. Thus, the approximation is only valid if the failure rates of all other cut sets is suitably low. If this is the case, it can be used to order cut sets by the rate with which they cause system failures. The full derivation of this approximation is provided by [VGRH81].

Structural importance

Other than ranking by failure probability, several other measures of component importance have been proposed. [Bir68] defines a system state as the combination of all the states (failed or not) of the components. A component is now defined 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 $\pi_F(\chi \cup \{c\}) \neq \pi_F(\chi \backslash \{c\})$.

[Jac83] 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 [AB03], 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 [Fus75]. That is, given that a set of components S has failed causing the FT to fail, what is the probability that e is one of the failed components. Formally, $VF_F(e) = P(e \in S | \pi_F(S) = 1)$. An algorithm to compute this measure is given by [DR01].

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 [DR01].

Initiating and enabling importance

In systems where some components have a failure rate and others have a failure probability, [CM11] 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, [HL93] 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.

[Arm95] 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, [LJ07] determines that the JFI can also be used for noncoherent systems.

2.7 Tool support

2.7.1 Commercial tools

In addition to the academic methods described in this section, commercial tools exist for FTA. The algorithms used in these tools are usually well documented. Several of these programs also allow the analysis of dynamic FTs, which will be explained in Chapter 3.

This subsection describes several commonly used commercial FTA tools. This list is not exhaustive, nor intended as a comparison between the tools, but rather to give an overview of the capabilities and limitations of such tools in general.

A.L.D. RAM Commander A.L.D. produces an FTA program as part of its RAM Commander toolkit [ALD]. This program can automatically generate FTs from FMECAs, FMEAs, or RBDs, and allows the user to generate a new FTA. It supports continuous and single-time FT, and can combine different failure distributions in one FT. Repairs are also supported.

The only supported qualitative analysis is the generation of minimal cut sets.

For qualitative analysis, the tool can compute reliability and expected number of failures up to a specified time bound, and availability at specific times as well as long-run mean availability. Failure frequency up to a given time is also supported. Moreover, the program can compute the importances and sensitivities of the BEs.

EPRI CAFTA CAFTA (Computer Aided Fault Tree Analysis) [EPR] is a tool developed by EPRI for FTA. It supports single- and continuous-time FTs, including non-coherent FTs and the PAND gate from dynamic FTs. Continuous-time BEs can have various probability distributions, including normal and uniform distributions. Several models of CCF are also included.

CAFTA can compute cut sets. For quantitative analysis, the program can compute reliability, and several importance and sensitivity measures.

Isograph FaultTree+ The Isograph FaultTree+ program [Iso] is one of the most popular FTA tools on the market. It performs quantitative and qualitative

fault tree analysis. It can analyze FTs with various failure distributions, and can replace BEs by Markov Chains to allow the user to arbitrarily closely approximate any distribution [JT88]. Dynamic FTs and Non-coherent FTs including NOT gates can also be analyzed.

Qualitatively, the program supports minimal cut set determination and the analysis of common cause failures. A static analysis is also supported for errors such as circular dependencies.

All the quantitative measures described in Section 2.5 can be calculated by FaultTree+. The program can also determine confidence intervals if uncertainties in the BE data are known. Without such information, sensitivity analysis can still be performed by automatic variation of the failure and repair rates. Importance measures that can be computed over the BE are the Fussell-Vesely, Birnbaum, Balow-Proschan, and Sequential importances.

ITEM ToolKit The ITEM ToolKit by ITEM software [ITEM] supports FTA, as well as other reliability and safety analyses, such as Reliability Block Diagrams [DP07].

This program uses Binary Decision Diagrams for its analysis, but can also perform an approximation method. The analysis supports non-coherent FTs, and several different failure models for BEs.

Qualitative analysis can determine minimal cut sets, and has four methods for common cause failure analysis.

Quantitative analysis supports reliability and availability computation. Uncertainty analysis of the results can be performed if input uncertainties are known, and sensitivity analysis even if they are not. The program can also compute importance measures, although for which measures is not specified.

OpenFTA The open-source tool OpenFTA [Ope] can perform basic FTA. It only supports non-repairable FTs, and allows only single-time BEs and BEs with exponentially distributed failure times.

OpenFTA supports minimal cut set generation, deterministic analysis of system reliability, and Monte Carlo simulation to determine reliability.

ReliaSoft BlockSim ReliaSoft's BlockSim program [Rel] can analyze Reliability Block Diagrams [DP07] and FTs.

Quantitative analysis can determine exact reliability of the system, including the changes in reliability over time. If information about possible reliability improvements is available, the program can compute the most cost-effective improvement strategy to obtain a given reliability.

Availability of repairable systems can be approximated using discrete event simulation. Given information about repair costs and spare part availability,

the analysis can determine the most effective maintenance strategy for a cost or availability requirement, as well as the optimal spare parts inventory.

BlockSim supports the determination of minimal cut sets, but does not appear to offer other quantitative analysis options.

PTC Windchill FTA The Windchill FTA program by PTC [PTC] allows the design and analysis of fault trees and event trees, including dynamic FTs. The program supports non-coherent FTs, as well as different failure distributions for the BEs.

Windchill FTA can compute minimal cut sets, as well as several methods for determining common cause failures.

Qualitative measures than can be computed include reliability, availability, and failure frequency. These can be determined using exact computations or by Monte Carlo simulation. The Birnbaum, Fussell-Vesely, and Criticaly importances of BEs can also be computed.

RiskSpectrum FTA The software suite RiskSpectrum [LRCRS] by Lloyd's Register Consulting includes an FTA tool. The overall suite is designed for probabilistic safety assessment, and includes tools for FMEA and human reliability analysis. RiskSpectrum FTA supports static fault trees with CCFs.

The analysis tool can perform qualitative analysis producing MCSs, and quantitative analysis including reliability and availability, as well as sensitivity and importance measures and time-dependent analysis.

2.8 Conclusion

This chapter has given an extensive overview of the formalism of fault trees and the various methods used to analyze them. We have shown how fault trees are used to model the failure behaviour of complex systems, and to analyze such systems for both qualitative and quantitative properties.

Qualitatively, fault tree analysis computes *cut sets* and *path sets*. These are used both to validate the tree, and to identify potential weaknesses in the system design where few or even single faults could lead to system failure.

Quantitatively, a wide range of metrics is available. Some, like *reliability*, are applicable to any system and are widely supported by analyses tools. Others, like *availability*, are only applicable or useful in more specific contexts (e.g., repairable systems). In any case, a large variety of analysis techniques and tools exist to compute these metrics, each with its own benefits and drawbacks.

We further provide information on several software tools, both academic and commercial, that can perform fault tree analysis.

This chapter treats the widely-used and well-known static (or, standard) fault trees. The next chapter will describe the less well-known but still relatively standard dynamic fault trees, and Chapter 4 surveys a wide range of other extensions and variants, which are not (yet) very standardized. Further chapters will explain the particular extension of the *Fault maintenance tree*, developed for this thesis.

Chapter 3

Dynamic Fault Trees

The previous chapter discussed static fault trees, which describe how combinations of failures of components (*basic events*) lead to failures of subsystems (*gates*) and eventually the entire system. Such static FTs can only model systems in which a combination of failed components results in a system failure, regardless of when each of those component failures occurred. In reality, many systems can survive certain failure sequences while failing if the same components fail in a different order. For example, if a system contains a switch to alternate between a component and its spare, then the failure of this switch after it has already activated the spare does not cause a failure.

Dynamic fault trees (DFTs) were developed by, among others, NASA [DBB90] to model such temporal dependencies in FTs. More specifically, DFTs introduce additional gates (most notably the priority-AND (PAND) and SPARE gates) whose behaviour depends on the order in which their children fail.

The increased expressive power of DFTs also necessitates more advanced analysis techniques. Qualitative analysis can be performed similarly to that of static FTs to obtain cut sets, but also to obtain *cut sequences* incorporating the temporal relationships between events. Quantitatively, a wide range of techniques has been developed to compute measures such as reliability and availability. Apart from some techniques using highly different approaches, many techniques fall into one of two broad categories:

- Exact techniques that translate the DFT into a more low-level formalism such as Markov chains [DBB90] or dynamic Bayesian networks [MPBCR08] from which the measure can be computed using standard algorithms.

- Simulation techniques that draw a large number of samples from the probability distribution described by the DFT, and estimate the desired measure with statistical confidence bounds.

Tables 3.1 and 3.2 list the qualitative and quantative analysis techniques described in this chapter.

Most techniques described in this chapter apply only to non-repairable DFTs. Some extensions to DFTs, such as the one by Boudali et al. [BCS07a] allow repairs,

Author	Method	Remarks
Tang et al. [TD04]	Cut sets	Postprocessing to convert cut sets to cut sequences
Liu et al. [LXZ+07, LZX+07]	Composition	Reduces work for shared components
Zhang et al. [ZZLL11]	Cut sequences	More compact representation of cut sequences
Chaux et al. [CRL+13]	Language theory	Allows repairs up to first TE occurrence
Merle et al. [Mer10]	Algebraic	Also allows quantitative analysis
Rauzy [Rau11]	ZBDD	Starting point for other analyses

Table 3.1: Overview of DFT qualitative analysis methods

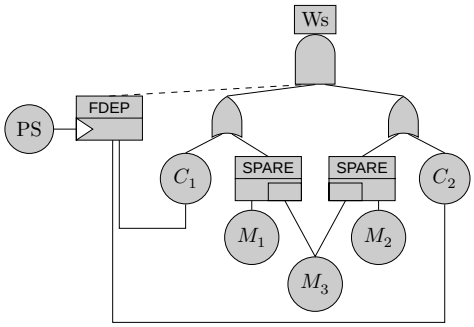


Figure 3.1: Example of a DFT, equivalent to subtree *Ws* in Figure 2.2 (page 24).

although the exact semantics of repairable DFTs are neither well-described, nor agreed on by the different extensions.

Origin of this chapter This chapter is expanded from Chapter 3 of:

- Enno Ruijters and Mariëlle Stoelinga. "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools". *Computer Science Review*, 15–16:29–62, 2015. DOI: 10.1016/j.cosrev.2015.03.001, ISSN: 1574-0137.

Organization of this chapter Since a DFT considers temporal behaviour, the methods used for the analysis of static FT cannot be directly used to analyze DFT. An overview of the various quantitative methods is shown in Table 3.2. The qualitative methods are listed in Table 3.1. Details of qualitative and quantitative analysis methods are given in Sections 3.2 and 3.3. We end with our conclusions in Section 3.4.

Author	Method	Repairs	Reliability	Availability	Other	Remarks	Tool support
Dugan et al. [DBB90]	Markov Chain		+	+	+	Suffers from state-space explosion	Galileo [SDC99]
Boudali et al. [BCS07a]	I/O IMC	+	+	+	+	Less state-space explosion for most models	CORAL [BCS07b], DFTCalc [ABvdB+13]
Volk et al. [VJK18]	Markov automata		+	+	+	Various optimizations to reduce state-space & Sound over- and underapproximations	STORM [DJKV17]
Codetta-Raiteri [CRFIV04]	Petri Nets		+	+	+	Intermediate model smaller than Markov Chain	DFT2GSPN [CR05a]
Pullum and Dugan [PD96]	Modularization		+			Fast when FT has small dynamic subtrees	SHADE Tree [PD96], DIFTree [DVG97]
Long et al. [LSH00]	SFL			+	+	No practical algorithm for realistic DFTs	
Han et al. [HGH11]	Approximation		+			Reasonable accuracy based on experiments	
Liu et al. [LXL+10]	Prob. distr.		+			For DFTs with large static subtrees, approx.	
Yevkin [Yev11]	Modularization		+			Reduces complexity of some specific subtrees	
Amari et al. [AGE03]	Approximation		+			Requires tree following certain rules	
Montani et al. [MPB05a]	DBN	+	+			Not exact, allows dependent BE	Radyban [MPB05a] [PBCRM07, MPBCR08]
Boudali and Dugan [BD05]	DBN	+	+			Not exact, allows multi-state, dependent BE	
Rongxing et al. [RGD10]	BDD & DBN		+			Efficient for DFTs with static subtrees	
Graves et al. [GHK+07]	DBN		+			Incorporates gate failure data	
Mo [Mo14]	MDD		+			Reduces state-space explosion	
Ni et al. [NTX13]	Algebraic		+			Finds MCS and performs quantitative analysis	
Durga Rao et al. [DRGSR+09]	Monte Carlo	+	+	+	+	Allows independently repairable components	
Boudali et al. [BNS09]	Monte Carlo	+	+	+	+	Allows non-Markovian systems	DFTSim [BNS09]
Liant et al. [LYZL09]	Monte Carlo	+	+	+	+	Requires cut sets, allows repairs	
Zhang et al. [ZMFW09]	Monte Carlo	+	+	+	+	Transforms to Petri Net	DRSIM [DRGSR+09]
Aliee et al. [AZ11]	Monte Carlo	+	+	+	+	Hardware method for fast simulations	
Ruijters et al. [RRdBS17]	Monte Carlo	+		+		Importance sampling to improve accuracy	FTRES [RRdBS17]
Rajabzadeh et al. [RJ10]	Hardware		+	+	+	Not exact, untested for large models	

Table 3.2: Overview of DFT quantitative analysis methods

3.1 Structure

The structure of a DFT is very similar to that of a static FT (described in Chapter 2), with the addition of several gate types shown in Figure 3.2. The new gates are:

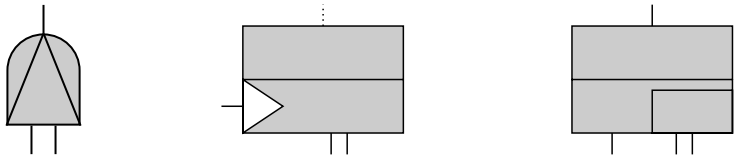
PAND (Priority AND) Output event occurs if all inputs occur from left to right.

FDEP (Function DEPendency) Output is a dummy and never occurs, but when the trigger event on the left occurs, all the other input events also occur.

SPARE Represents a component that can be replaced by one or more spares. When the primary unit fails, the first spare is activated. When this spare fails, the next is activated, and so on until no more spares are available. Each spare can be connected to multiple Spare gates, but once activated by one it cannot be used by another. By convention, spare components are ordered from left to right.

Example 15 *An example of a DFT is shown in Figure 3.1. This DFT has the same cut sets as the subtree rooted at G3 of Figure 2.2, but has a more intuitive informal description: M_3 is clearly shown as a shared spare for M_1 and M_2. Also, the system does not directly depend on the power supply PS. Instead, the failure of PS triggers a failure of both CPUs, which more accurately describes the system and eliminates the shared event at the expense of introducing a shared trigger.*

BEs can have an additional parameter α called the *dormancy factor*. This parameter is typically a value between 0 and 1, and reduces the failure rate of the BE to that fraction of its normal failure rate if the BE is an inactive input to a SPARE gate [BCS07c]. For example, a spare tire will not wear out as fast as one that is in operation. For BEs that are not inputs to a SPARE gate, α has no effect. Dormancy factors greater than 1 can be specified, but are rarely useful as they would indicate that a component fails faster when it is not in use than when it is being used.



(a) PAND gate (b) FDEP gate (c) SPARE gate

Figure 3.2: Images of the new gates types in a DFT

The introduction of the PAND and SPARE gates means that a DFT is not generally coherent. For example, an increase in the failure rate of the right input to a PAND can increase the reliability of the gate. In systems exhibiting this behaviour, overall reliability is often improved by forcing a component to fail immediately when the system is started, indicating that such non-coherence is often indicative of a modeling error or suboptimal system design.

In non-repairable DFTs, the FDEP gate can be removed by replacing each of its children by an OR gate of that child and the FDEP trigger. In repairable DFTs, the applicability of this approach depends on the definition of the FDEP gate: If failures triggered by the FDEP require separate repairs, the transformation is not correct. If repair of the FDEP trigger also restores the triggered components to operation, the transformation does preserve the behaviour.

Defintion 9 *A DFT is a tuple* $DF = \langle BE, G, T, I, DORM \rangle$*, where BE and G are the same as in a static FT (and we write* $E = BE \cup G$*). The function T denotes the gate type, but now* $T : G \mapsto DGT$*, with the set of dynamic gates* $DGT = GateTypes \cup \{FDEP, PAND, SPARE\}$*. I is replaced by an input function:* $I : G \mapsto E^*$ *yielding an ordered sequence of inputs to each gate.* $DORM : BE \mapsto \mathbb{R}^{\geq 0}$ *provides the dormancy factor for each the BEs.*

Since the output of the FDEP gate is a dummy output and not relevant to the behaviour of the FT, it is often useful to use a *pruned input function* which does not include FDEP inputs or outputs [BCS10].

Some types of DFT have additional gates, which are not included in the description above. These are:

Hot spare Special case of SPARE gate, where the dormancy factor of the spares is 1, i.e., the spare failure rate is the same as the normal failure rate [DBB90].

Cold Spare Special case of SPARE, with a dormancy factor of 0, i.e., spares cannot fail before activated [DBB90].

Priority OR Fails when the leftmost input fails before the others [WP09]. Can be replaced by a PAND and an FDEP.

Sequence enforcer Prohibits failures of inputs until all inputs to the left have failed [BCS07a]. Can be replaced by (cold) SPARE provided the inputs are not shared with other gates and failure times are exponentially distributed.

One-shot PDEP Special case of the FDEP gate, where the occurrence of the trigger event has some probability of causing a failure of the dependent events [PCRM10].

Persistent PDEP Special case of the FDEP gate, where the occurrence of the trigger event causes an increase in the failure rates of the dependent events [PCRM10].

We note that this chapter describes the general behaviour of DFTs, as agreed on by most tools. The exact details of some DFT constructs differ between implementations, as outlined by Junges et al. [JGKS16].

3.2 Qualitative analysis

A simple form of qualitative analysis of a DFT can be performed by employing the same techniques as used for SFTs; namely by replacing the PAND and SPARE gates by AND gates, and the FDEP gates by OR gates. This analysis will not capture the temporal requirements of the tree. Nonetheless, the cut sets can be used to improve system reliability, since at least one cut set must completely fail for a system failure to occur.

Example 16 *In Figure 3.3, this method replaces the PAND gate on the right by an AND gate. The resulting cut sets are* $\{P, B\}$ *and* $\{S, P\}$*. These cut sets can be useful, as preventing the failures of every cut set still prevents system failure. However, unlike in the SFT, the failure of* $\{S, P\}$ *does not necessarily cause a system failure, depending on the ordering of the failures.*

To capture these temporal requirements, Tang et al. [TD04] introduced the notion of 'cut sequences' as the dynamic counterpart to cut sets. A cut sequence is a sequence of failures which cause a system failure. Formally, a sequence $\langle e_1, e_2, \dots, e_n \rangle$ is a cut sequence of the DFT D if, given any failure times $F_{e_1} < F_{e_2} < \cdots < F_{e_n}$, the top level event occurs at or before F_{e_n}.

Tang et al. [TD04] also showed that these cut sequences can be determined by replacing the dynamic gates by static gates, determining the minimal cut sets, and then adding any sequencing requirements to the cut sets.

Example 17 *The DFT in Figure 3.3 has cut sequence set (CSS)* $\{\langle S, P \rangle, \langle P, B \rangle, \langle B, P \rangle\}$*. The sequence* $\langle P, S \rangle$ *is not a cut sequence since the failure of* S *after* P *does not trigger the PAND gate.*

It was pointed out by Junges et al. [JGKS16] that cut sequences can implicitly assume/require the non-failure of BE not included in the sequence before an included BE. For example, in Figure 3.4 the cut sequence $\langle A, B \rangle$ assumes that BE C does not failed before A.

Zhang et al. [ZZLL11] offer a more compact way of representing cut sequences, by adding temporal ordering requirements to cut sets. This allows one representation to cover multiple cut sequences at once, where some events are ordered independently of other events. This method would represent the CSS of Figure 3.3 as $\{\{S, P, S < P\}, \{P, B\}\}$.

Liu et al. [LXZ+07] provide an alternative method to determine cut sequences by composition of the cut sequences of the subtrees. This method reduces the

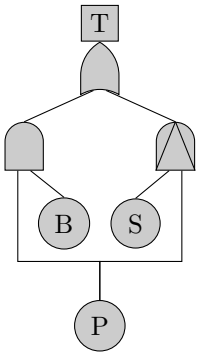


Figure 3.3: Example of a DFT with temporal sequence requirements. The system fails if both the primary (P) and backup (B) fail, or if the primary fails when the switch (S) to enable the backup has already failed.

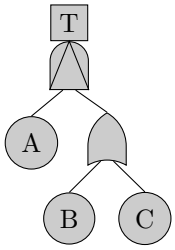


Figure 3.4: Example of a DFT with implied a non-failed BE in its cut sequences. The cut sequence $\langle A, B\rangle$ causes a failure of T, while the supersequence $\langle C, A, B\rangle$ does not.

amount of repeated work if the same components are present in multiple cut sets. Additionally, they show [LZX+07] that the cut sequences can be used to perform quantitative analysis.

A different definition of qualitative analysis for repairable DFTs is provided by Chaux et al. [CRL+13]. This method computes a language of failure and repair sequences in which the system is failed after the last element of the sequence. This language is described by a finite automaton that generates all such sequences (to keep the language finite, only the sequence up to the first system failure is considered). The complexity of this method is based on the length of the longest non-looped sequence of failures and repairs in the system.

Another algebraic method for determining and expressing cut sequences was developed by Merle et al., by extending the structure function used for static FTA (described in Section 2.3.1) to first include the Priority-AND gate [MR07] by allowing a 'before' relation as a boolean primitive. This method is subsequently developed to include the other DFT gates [MRLB10, Mer10, MRL11]. The structure function can subsequently be used to perform quantitative analysis [Mer10].

Example 18 *Considering again Figure 3.3, the FT has the boolean expression:*

$$(P \wedge B) \vee (S \wedge P \wedge (S < P))$$

This expression can be simplified using the law $A \wedge (A < B) = (A < B)$ *into*

$$(P \wedge B) \vee (P \wedge (S < P))$$

This is the minimal disjunctive normal form, showing that $P \wedge B$ *and* $P \wedge (S < P)$ *are the minimal sets of failures and sequence dependencies that yield a top event failure.*

More recently, Rauzy [Rau11] proposed a variant of Minato's Zero-Suppressed BDD [iM93] to include ordering information. This variant can be used to find the minimal cut sequences of DFTs, and the author believes that more efficient algorithms for other analyses can be based on this representation.

3.3 Quantitative analysis

This section describes analysis techniques for quantitative measures of DFTs. To recap Sections 2.4 and 2.5, the most important measures computed here are:

Reliability: Probability that no system-level failure occurs within a given time bound. I.e., if we let the random variable $X_F(t)$ be 0 if the system described by fault tree F is functional at time t, and 1 if it has failed, the reliability $R_F(t)$ is defined as $R_F(t) = \mathbb{P}[\forall_{s<t} : X_F(s) = 0]$. For non-repairable systems this is equivalent to $R_F(t) = \mathbb{P}[X_F(t) = 0]$.

Availability: Expected fraction of the time that no system-level failure is occurring. Using the above definition of $X_F(t)$, the time-bounded availability $A_F(t) = 1 - \frac{1}{t}\int_0^t X_F(s)ds$. The unbounded availability, mostly useful for repairable systems, is given by $A_F = \lim_{t\to\infty} A_F(t)$.

3.3.1 Algebraic analysis

Merle at al. [MRLB10] show that the cut sequences obtained by qualitative analysis can be used to obtain an expression for the reliability of the system. This approach begins by expressing the failure probability of the system in terms of the BEs, and then substitutes the probability distributions of the failure times of the BEs into this function.

Example 19 *Considering again the DFT in Figure 3.3, we compute that it has cut sequences* $\{\{P, B\}, \{P, S < P\}\}$. *We first apply the inclusion-exclusion principle to obtain the probability of a top level failure:*

$$\mathbb{P}(T) = \mathbb{P}(P \wedge B) + \mathbb{P}(P \wedge (S < P)) - \mathbb{P}(P \wedge B \wedge (S < P))$$

Now, expressions for the failure probabilities within a given time can be substituted. We denote the cumulative failure probability $F_e(t)$ *as the probability of* e *failing before time* t, *and the corresponding probability density as* $f_e(t)$. *Using the rule for time-dependent failures [MRLB10]:*

$$P([A \wedge (A < B)] < t) = \int_0^t f_A(u)F_B(u)du$$

We obtain the expression:

$$\mathbb{P}(T \leq t) = F_P(t) \cdot F_B(t) + \int_0^t f_P(u)F_S(u)du - F_B(t) \cdot \int_0^t f_P(u)F_S(u)du$$

While feasible for simple DFT, larger DFT with nested temporal dependencies (e.g., $(A < B) < C$) quickly result in deeply nested integrals making the approach computationally infeasible.

An alternative method is introduced by Long et al. [LSH00], which can compute availability at a specific time and the long-term expected number of failures per unit time. It uses a system of logic called 'Sequential Failure Logic' to describe the temporal restrictions within cut sets. Unfortunately, the equations produced are difficult to solve due to many multiple integrals, and only a special case where all failure and repair rates are identical is presented.

3.3.2 Analysis by Markov Chains

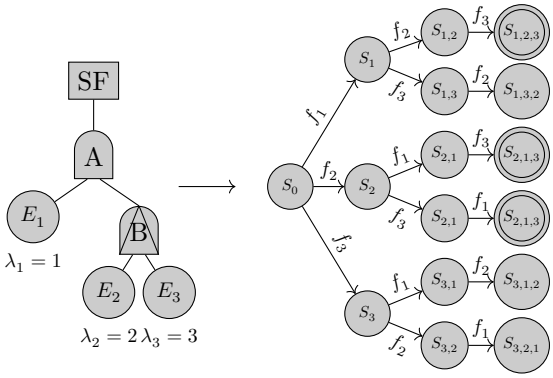


Figure 3.5: Example conversion of DFT to a Continuous-Time Markov Chain. States corresponding to system failures (goal states) are indicated by a double circle. Transition f_i denotes the failure of BE E_i, and occurs with rate λ_i.

The first method proposed to analyze DFTs was by Dugan et al. [DBB90, DBB92], and computes the unreliability of the system during a time window $[0, t]$. This method converts the DFT into a Markov Chain, in which the states represent the history of the DFT in terms of what components have failed and, where needed, in what order. Since the number of failed subsets grows exponentially in the number of BEs, this method is not practical for very complex systems.

Example 20 *Figure 3.5 shows a simple DFT converted into a Markov Chain. From the starting state S_0, in which all components are operational, three transitions are possible representing the failures of the three BEs. After the failure of the first BE, two more BEs can fail, and finally the last BE fails. If all three BEs have failed, and E_2 failed before E_3, system failure occurs, which corresponds to the circled (goal) states in the MC. In the other states the system is still operational. Existing tools such as PRISM [KNP11] and STORM [DJKV17] can be used to compute the probability of reaching a goal state within a certain time, corresponding to system unreliability.*

The MC in Figure 3.5 could be reduced without affecting the computed probabilities. For example, from S_3 no goal state can ever be reached. It is therefore acceptable to replace S_3 by an absorbing state to reduce the complexity of further analysis. A full discussion of minimization techniques is beyond the scope of this thesis, but several are listed in [BK08].

Codetta-Raiteri [CR05a] presents a transformation of DFTs to Stochastic Petri

Nets [CDFH93], which are in turn analyzed by conversion to Markov Chains. Although this method still suffers from a combinatorial explosion when constucting the Markov Chain, the Petri Nets are much smaller and easier to understand and extend.

Dehnert et al. [DJKV17] continue the direct translation of DFTs to Markov chains, but with more reduction techniques to reduce the state-space and analysis time of the chains.

Compositional analysis of DFTs

Boudali et al. [BCS07a, BCS07c] use a different method to calculate the reliability of a DFT, which reduces the combinatorial explosion in many common cases. They provide a *compositional* semantics for DFTs, i.e., each DFT element is interpreted as an Interactive Markov Chain [Her02] and the semantics of the DFT is the parallel composition of the elements. The papers provide several reduction techniques to minimize the resulting Markov Chain. In addition, it allows DFTs to be extended with repairable components and mutually exclusive events.

The analysis is performed by converting a DFT into an *Input/Output Interactive Markov Chain* for analysis. This model is constructed by computing the parallel composition of the I/O IMCs for parts of the tree, down to individual gates and events. Since intermediate models can be analyzed to remove unnecessary states, the total I/O IMC can be much smaller than the Markov Chain produced by earlier methods, and the combinatorial explosion is reduced. This overall process is shown in Figure 3.7.

The program DFTCalc was developed by Arnold et al. [ABvdB+13] to analyze reliability and availability of DFTs using the I/O IMC methodology.

Example 21 *Figure 3.6 shows the I/O IMC equivalents of the basic event E_1 and the gate A of the DFT in Figure 3.5. Below that, the parallel composition of the two elements are shown. This composition behaves as if the two separate elements are ran in parallel, with the output signal of the BE ($f_{E_1}!$) permitting the transition with input signal $f_{E_1}?$ in the gate's IMC.*

Observe that input signal $f_B?$ is still present in the composition, allowing this IMC to be composed with gate B later. Similarly, output action $F_{E_1}!$ allows the later composition with other gates in which E_1 is an input. If no such gates exist, the IMC can be minimized by removing these output transitions.

Unlike traditional Markov Chains, I/O IMC are capable of modeling nondeterminism between actions. Guck et al. [GTH+14] use this approach to model maintenance strategies where it is not specified which of multiple failed components to repair first.

More recently, Volk et al. [VJK18] have demonstrated a new state-space generation approach exploiting the structure of the DFT to apply various reduction

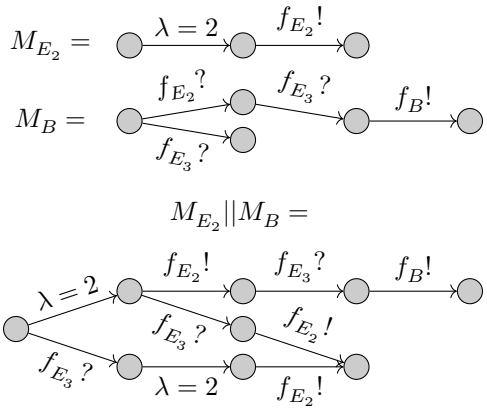
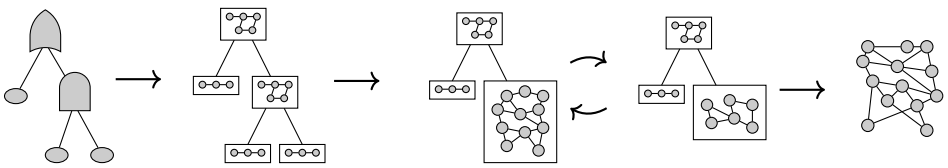


Figure 3.6: Example conversion of part elements E_1 and A of the DFT in Figure 3.5 to an I/O Interactive Markov Chain. Input signals are denoted by a question mark, output signals by an exclamation mark.

techniques. This approach is shown to be considerably faster than the compositional method, and can be combined with other reduction techniques [KS17].

Pullum and Dugan [PD96] developed a program to divide a DFT into independent submodules for computing reliability. Submodules containing only static gates can then be solved using a traditional BDD method, while submodules containing dynamic gates can be solved using Markov Chain analysis.

Example 22 *Suppose we are computing the availability at time t of the DFT in Figure 3.5. We can convert the entire DFT into a Markov Chain such as the figure shows, but only the subtree rooted at B is dynamic. We can therefore replace this subtree by a fictional node B^* and use a BDD to determine the minimal cut sets of the tree, which is only $\{E_1, B^*\}$. Following Section 2.5.3, the availability of the tree is given by $A_{SF}(t) = A_{E_1}(t) \cdot A_{B^*}(t)$. Markov chain*



(a) DFT (b) Transformation (c) Composition (d) Minimisation (e) IMC

Figure 3.7: Compositional aggregation with intermediate minimization.

analysis can now be used to compute the value $A_{B^*}(t)$, and $A_{E_1}(t)$ is the same as for a static fault tree.

Han et al. [HGH11] also modularize a DFT and use BDD for the static submodules, but use the approximation by Amari et al [AGE03] to solve the dynamic submodules. This avoids the state-space explosion problem of analysis by conversion to Markov Chain, while retaining a reasonable degree of accuracy.

Later, Liu et al. [LXL+10] proposed a method to modularize DFTs further, by also collapsing static subtrees of a dynamic gate, but keeping additional information about the probability distribution of these subtrees.

Yevkin [Yev11] provides additional modularization techniques, which can convert static subtrees and some dynamic subtrees into equivalent BEs, reducing the complexity of further analysis.

3.3.3 Analysis using Dynamic Bayesian Networks

The method by Bobbio et al. [BPMC01] of converting an SFT into a Bayesian Network (described in Section 2.4.2) was later improved by Montani et al. [MPB05a] by using a Dynamic Bayesian Network (DBN) to analyze DFTs. A DBN extends classic Bayesian Networks by introducing a notion of time, and allowing the conditional probabilities at the current time to depend on the values of the random variables at earlier times (in addition to their current value).

In this approach, the DBN is evaluated at many points in time, with the state probability distributions carried over from each timestep to the next. By also allowing nodes to have probabilities conditional on their own state in the previous timestep, dynamic behaviour can be included in the analysis. Due to the discretization, results from this method are not exact. Results can be made arbitrarily accurate by shrinking the discrete timesteps, though the computation time increases linearly in the number of timesteps. Only non-repairable FTs are analyzed by this method, however Portinale et al. [PCRM10] propose a similar method for repairable FTs. Other extensions from the earlier BN work such as noisy gates remain applicable.

The Bayesian Network method has been extended by Boudali and Dugan [BD05] to model DFT gates. This method can produce results equivalent to solving a discretized version of the Markov Chain corresponding to the DFT, but can also be extended with dependent component failures and multi-state components by changing the produced DBN. No comparison between this method and the method by Montani et al. [MPB05a] is presently available.

Example 23 *Figure 3.8 shows the dynamic Bayesian network of the DFT in Figure 3.5. Gates $\{A, B\}$ and basic events $\{E_1, E_2, E_3\}$ form the nodes of the network, while input relations in the DFT form one-way conditional probabilities. Basic events are not repairable, and thus remain failed if they were failed in*

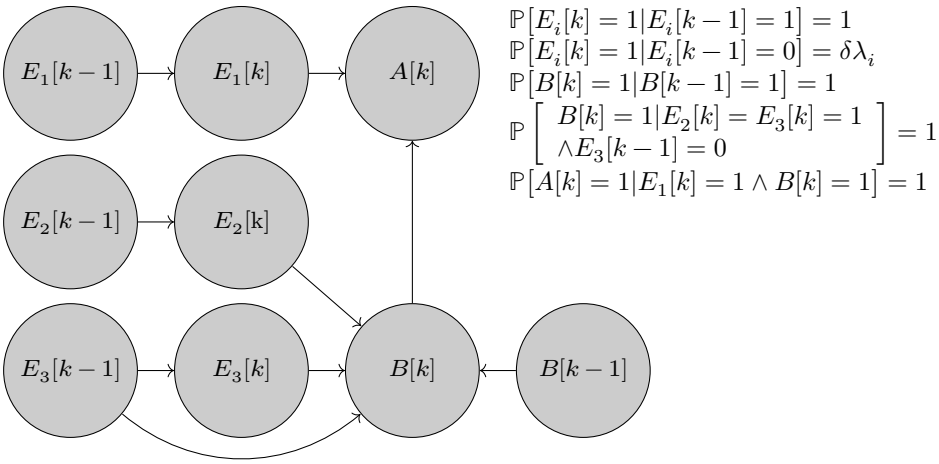


Figure 3.8: Dynamic Bayesian Network corresponding to the DFT in Figure 3.5 with timestep δ. Default rules with probability 0 have been omitted.

the previous timestep. Otherwise, the probability of their failure in the current timestep depends on their failure rate. This explains the first two conditional probability rules.

The next two rules give the behaviour of the PAND gate B. If it was failed in the previous timestep (i.e., $B[k-1]=1$, it remains failed (i.e., $B[k]=1$). Otherwise, it fails if both inputs are failed, and E_3 has not failed earlier. Note that behaviour on simultaneous failure is deterministic in this model (namely the PAND gate fails on simultaneous failure of its inputs).

Finally, the state of AND gate A is determined purely by its inputs.

As for other analysis methods, computational requirements can be reduced by modularizing the FT and using more efficient methods for the static subtrees. Such an approach combining BDD and DBN was proposed by Rongxing et al. [RGD10].

Since a BN allows arbitrary conditional probabilities to be specified, it is possible to include failure rates of gates in addition to that implied by the tree structure. This improves accuracy and reduces the effect of modeling errors. Such an approach was described by Graves et al. [GHK+07]. This is useful, since many real-life systems record component failures at an intermediate level, rather than diagnosing every fault to the level of the BE.

3.3.4 Other approaches

Mo [Mo14] described a method for converting a DFT into a multiple-valued decision diagram (MDD) to compute the reliability of non-repairable systems. In this approach, subtrees containing only static gates are directly converted into MDDs, while subtrees with dynamic gates are solved by conversion into a CTMC before the results are included in the MDD. This approach reduces the state-space explosion problem in many common cases, but in the worst case of a dynamic gate as the TE a full CTMC still needs to be solved.

A purely algebraic approach is suggested by Amari et al. [AGE03], which calculates the probability distribution at every gate by appropriately combining the distributions of the inputs. While this approach gives exact results and does not suffer from the state-space explosion effect common when using Markov Chains, only a subset of trees satisfying particular rules can be analyzed this way.

Ni et al. [NTX13] propose a different algebraic method for describing the DFT structure, which produces a boolean-like expression of the DFT. This method allows minimal cut sequence determination as well as quantitative analysis.

3.3.5 Simulation

Quantitative analysis can be performed by Monte Carlo simulation. Failures and/or failure times are sampled from their respective distributions, and the effect these failures have on the system are calculated.

Quantitative Monte Carlo analysis can be performed using the method by Durga Rao et al. [DRGSR+09], which can also be applied if the components are individually independently repairable.

Boudali et al. [BNS09] developed a program to analyze DFTs using Monte Carlo simulation. It allows BE failure distributions to change over time, and even based on different clocks for different BE, resulting in non-Markovian models. This is useful when, for example, a system takes time to warm up and this affects the failure rates.

If the minimal cut sets have already been determined, Liang et al. [LYZL09] propose a Monte Carlo method for computing the unreliability of an RFT. This approach allows the failure and repair rates to follow arbitrary distributions, but still does not allow repair policies other than independent component repair.

Zhang et al. [ZMFW09] showed that it is possible to convert a DFT to a Petri Net, on which quantitative analysis can be performed by simulation. Exact analysis on Petri Nets is normally done by conversion into Markov Chains, still resulting in a state-space explosion. Simulation, however, can be performed directly on the Petri Net, although the benefits compared to simulation of the untransformed DFT are not stated.

If very high performance is required, it is possible to construct a hardware

circuit to perform Monte Carlo Simulations much faster than normal computer simulation. Such an approach is described by Aliee and Zarandi [AZ11].

Fault trees with very low failure probabilities are generally difficult to analyze through simulation, as very many random samples are required to obtain a sufficiently large number of top level events to achieve tight confidence intervals. In work by Ruijters et al. [RRdBS17] (described in Chapter 6), a technique called *importance sampling* is used to reduce the number of simulations required to compute confidence intervals for repairable DFTs.

Rajabzadeh and Jahangiry [RJ10] propose a conversion of a DFT into an analogue electronic circuit, which outputs a voltage corresponding to the system failure probability. This approach does require an approximation for some of the gates, and the accuracy on larger models is not demonstrated.

A method for the analysis of the sensitivity of various model parameters is provided by Ou and Dugan [OD00].

3.4 Conclusions

This chapter has explained how the formalism of dynamic fault trees expands on static fault trees by introducing additional gates modelling common dependability patterns, notably those of order-dependent failures, warm and cold spare components, and functional dependencies. We have also provided an overview of the various qualitative and quantitative methods to analyze such dynamic fault trees.

Qualitatively, the cuts sets used in SFTs have been expanded to cut sequences to incorporate information about the order in which components fail, and various techniques have been proposed to find such cut sequences.

Quantitatively, the main methods involve translating a DFT to a Markov chain or dynamic Bayesian network, or simulation. Various other methods exist but are currently less developed.

Chapter 4

Fault tree extensions

While dynamic fault trees are the most popular extension to static fault trees, several other ways of extending FTs have been proposed. The extensions can be approximately divided into several categories.

1. Fault trees using *fuzzy numbers* can be used in cases where failure probabilities or behaviour are not known exactly.
2. Several extensions allow fault trees to model systems where basic events are *stochastically dependent,* such as when a failure of one component increases the failure rate of another component.
3. *Repairable Fault Trees* can represent more complex repairable systems than the simple repair rates in classic FT.
4. The temporal relations between events are important. Dynamic fault trees include certain temporal dependencies, but other extensions have been proposed as well.
5. State/Event Fault Trees were introduced to model systems and components with a state that varies over time, and where this state affects the consequences of component failures or the failure rates.
6. Miscellaneous extensions, e.g., integrating Attack Trees with FTs.

As for standard FTs, the extensions can be analysis qualitatively to obtain (some variation of) cut sequences, or quantitatively to obtain numerical measures such as reliability (probability of of failure before a given time bound) or availability (expected time that the system is not failed, mostly useful for repairable systems).

These extensions are discussed in sections 4.1 through 4.6, respectively. An overview of the relevant measures for each extension can be found in Table 4.1 (page 84), a summary of the extensions is provided in Table 4.2 (page 98).

Origin of this chapter This chapter is expanded from Chapter 4 of:

- Enno Ruijters and Mariëlle Stoelinga. "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools". *Computer Science Review*, 15–16:29–62, 2015. DOI: 10.1016/j.cosrev.2015.03.001, ISSN: 1574-0137.

	Measures				Methods
	Cut sets	**Reliability**	**Availability**	**Other**	
DFT with repair boxes [BCR04]		+	+	+	[BCR04]
Repairable FT [CRFIV04]		+	+	+	[CRFIV04]
Combined FT [CR05b, CR11]		+	+	+	[CR11]
Nondeterministic RFT [BCRFH08, BFCRH09, BFCRH14]		+	+	+	[BFCRH09, BFCRH14]
FT with Attack Tree [FMC09]	+	+	?		[FMC09]
Fuzzy FT [TFLT83]	=	+			[TFLT83, WX12, GeE99]
Fuzzy FT [Sin90]	=	+			[Sin90]
Fuzzy FT [LW97]	=	?		+	[LW97]
Fuzzy FT [KJG96]	=	+			[KJG96]
Fuzzy FT [SCC06, Li08]	=	+			[SCC06, Li08]
Fuzzy FT [RK11]	?	+			[RK11]
Fuzzy FT [EIO98]	?	?	+		[EIO98]
Fuzzy FT [HTZ04]	=	+			[HTZ04]
Fuzzy FT [PP94]	=	+			[PP94]
Extended FT [Buc99, Buc00b]	?	+	+	+	[Buc00b]
Multistate FT [ZWST03]	+	+			[ZWST03]
FT with mutual exclusion [TRSS00]	+	+			[TRSS00]
FT with dependent events [Vau02]	=	?	+		[Vau02]
BDMP [Bou02, Bou07, Bou08]	?	+	+	+	[Bou08, Bou07]
FT with AND-THEN [WM00]	+	?			[WM00]
DFT with simultaneity [WBP07, WP09]	+	?			[WBP07]
Formal FT Semantics [STR02]	+	?			[STR02]
FT with Duration Calculus [Glu07]				+	[Glu07]
Temporal FT [Pal02]	+	?			[Pal02]
Combined FT [CR11]	?	+	+		[CR11]
State-Event FT [KG04, Kai05]	?	+	+		[XHH+13, KG04]

Table 4.1: **Analysis and tool support for fault tree extensions.** Measures for which analysis techniques are described are marked with '+', measures for which existing analysis techniques apply unchanged are marked with '=', and measures for which analysis techniques have not yet been developed are marked with '?'. Blank spaces indicate the measure is not applicable to that extension.

4.1 FTA with fuzzy numbers

Fault trees using fuzzy numbers were introduced by Tanaka et al. [TFLT83] as a way to reduce the problem that failure probabilities of components are often not exactly known. Fuzzy numbers represent uncertainty by not specifying an exact number, but rather a range which contains the true value. Alternatively, they can be used as input to the FT, in which case they specify categories to which a probability belongs, to a greater or lesser degree.

Example 24 *Suppose we would like experts to specify a failure probability using the categories 'high', 'medium', and 'low'. It is possible to set exact endpoints and ask the experts to rate any value between 0 and 0.2 as low, this has two disadvantages: First, linguistic descriptions are commonly used so that the expert does not need to estimate an exact probability, and giving endpoints reintroduces that requirement. Second, if the expert estimates a probability to be approximately 0.2, the expert must decide whether this is low or medium, and the model does not capture the uncertainty that the expert may have.*

Alternatively, we can describe the categories as fuzzy subsets of the interval $[0, 1]$. *Figure 4.1 shows possible membership functions for the categories. Here, for example, the value 0.1 is said to be fully a member of 'low' and no member of either other category. Thus experts are assumed to always classify 0.1 as low. The value 0.3 is partly a member of 'low' with membership 0.5, signifying that half of the experts would classify 0.3 as low.*

Mahmood et al. [MAV+13] have conducted a literature review exploring different variations of Fuzzy Fault Trees, and various methods for their analysis. A brief overview is provided below.

FTs are often specified using fuzzy numbers for the probabilities or possibilities of basic events. A common method is to use fuzzy set theory: A fuzzy set has a membership function which gives, for any argument, the degree to which that

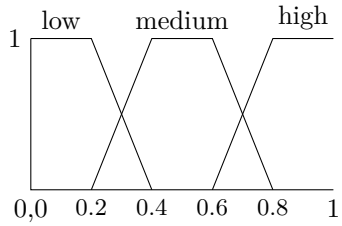


Figure 4.1: Example of fuzzy membership functions of the sets 'low', 'medium', and 'high'

argument is a member of the given fuzzy set. In this context, BE probabilities are given as a fuzzy subset of the interval [0, 1].

The membership function of a fuzzy subset of the real numbers is similar to the probability density function of a probability distribution. The difference is that where a PDF gives the probability of a variable having a value given the distribution this variable belongs to, the membership function gives the degree to which a value belongs to a fuzzy set, without making a claim regarding the likely values of variables given the fuzzy set.

If a fuzzy number contains only one possible value, it is the same as a conventional or *crisp* number.

Singer [Sin90] provides a method for computing the TE fuzzy probability if the membership function can be specified in a special form called an L-R type. This is a function that is symmetric about some point on the probability axis except for a scaling factor, and can be represented by a function of the form

$$m(p) = \begin{cases} L(p) = f(\frac{c-p}{l}) & \text{if } p < c \\ R(p) = f(\frac{p-c}{r}) & \text{if } p >= c \end{cases}$$

Where $f : \mathbb{R} \mapsto \mathbb{R}$ is some function, c is the point of symmetry, and l and r are scaling factors.

This method is frequently applicable since many common probability distributions (including the normal, uniform, and triangular distributions) can be described in this form.

An alternative method is described by Lin et al. [LW97], in which some of the BEs are described by multiple fuzzy numbers obtained from different experts. These fuzzy numbers could, for example, be derived from natural language expressions describing the events from 'very probable' to 'very improbable'. This method combines these multiple fuzzy probabilities into one crisp probability for the BE, and then analyses the FT as normal.

When multiple probability estimates are available, Kim et al. [KJG96] offer a method to use these to calculate 'optimistic' and 'pessimistic' fuzzy probabilities for the TE. This approach may be useful when each expert gives only small uncertainties due to natural variation in components, but different experts give these uncertainties over different ranges, for example due to different opinions of the likelihood of human error.

If the membership functions for the BE probabilities are themselves uncertain, this may be included in the model using 'intuitionistic fuzzy set theory', as described by Shu et al. [SCC06, Li08]. In this model, two membership functions describe an upper and lower bound on the membership. This can be used if, for example, a probability is believed to lie between 0.4 and 0.6, but it is not known what value in this range is the most likely.

Ren and Kong [RK11] provide a means for analyzing an FT when not only the

BE probabilities are uncertain, but also the effects of component failure on the rest of the system. In this framework, components can have multiple states rather than only operational and failed. Each gate can also have multiple states, and these states can be triggered by various combinations of input states. This can model a system which can continue operating after certain component failures, but only in a degraded way. Such a degradation can have other effects on the gates above it.

An alternative approach to uncertain network structure is the introduction of *noisy* gates [BPMC01]. These gates have some probability of failing when the standard gate would not, or vice versa. For example, a computer with redundant hard drives may fail to detect and correct certain errors, leading to a system failure even though the backup drive is perfectly functional.

In repairable FTs, uncertainty can exist not only in the BE failure rate but also in the repair rate. A system for accounting for this uncertainty in calculating the overall system availability is given by El-Iraki and Odoom [EIO98].

If the failure probabilities are very uncertain, Huang et al. [HTZ04] offer a method based on possibility measures that may offer better results than probability-based fuzzy number approaches. In this method, basic events are specified with possibilities representing estimated lower bounds on the failure probabilities. In this context, the possibility of the TE can be calculated quite efficiently.

It is also possible to model the probabilities as themselves being random variables with a normal distribution. As Page and Perry [PP94] showed, this allows a better quantification of the uncertainty in the result, although it may require more assumptions on the part of the FT designer.

More generally, Forster and Trapp [FT09] suggest that BE probabilities can be specified as intervals, within which the actual probabilities are sure to lie. Their method uses Monte Carlo simulation treating these intervals as bounds on a uniform distribution (although they mention that arbitrary distributions may be used) to compute the second-order probability mass function for the TE probability.

4.1.1 Importance measures for fault trees with fuzzy numbers

Aside from the TE probability, it can also be useful to determine which components have the greatest effect on this probability. Several methods for determining this have been developed.

Furuta et al. [FS84] suggested to extend the structural importance to be calculated using fuzzy probabilities, and named the resulting value the *fuzzy importance*.

Alternative measures were suggested by Suresh et al. [SBR96], which also include the amount of uncertainty contributed by each component. The *Fuzzy Importance Measure* of a component i is defined as $FIM(i) = ED\left[Q_{qi=1}, Q_{qi=0}\right]$, where ED denotes the Euclidean distance between two fuzzy numbers, $Q_{qi=1}$ is

the TE probability if event i has an occurrence probability of 1, and $Q_{qi=0}$ is the TE probability if event i has a probability of 0.

Similarly, the *Fuzzy Uncertainty Importance Measure* is defined as $FUIM(i) = ED\left[Q, Q_{qi=0}\right]$, where Q is the TE probability. This measure ranks a component as more important if its probability is less certain.

Finally, if the distributions of the BE probabilities can be bounded with certainty, for example based on manufacturer specifications, it is possible to use Interval Arithmetic to obtain exact bounds on the distribution of the TE probability, as shown by Carreras and Walker [CW01].

4.1.2 Analysis methods measures for fault trees with fuzzy numbers

Since the structure of most fuzzy FT is the same as that of classic FT, qualitative analysis can be performed without change. Some extensions, such as the multistate FT by Ren and Kong [RK11], require different methods.

One of the first methods proposed to analyze a Fuzzy Fault Tree is to determine the minimal cut sets, and perform a standard quantitative analysis using the Extension Principle developed by Zadeh [Zad75] to perform arithmetic on fuzzy numbers.

The use of the Extension Principle is computationally intensive for larger trees, and cannot be applied if repeated events are allowed in the tree. Soman and Misra [SM93] offer an alternative method to calculate the top event probability, called a 'resolution identity' using the 'α-cut' method, which does allow repeated events and has lower computational requirements. This method was extended to allow multi-state elements in [MS95].

Guimarẽes and Ebecken [GeE99] present a computer program named FuzzyFTA that can calculate the FIM and FUIM of any gate using either the fuzzy logic approach using α-cut or a Monte Carlo Simulation. The results of these methods are in agreement, although the fuzzy approach provides more information and is quicker.

Another approach described by Wang et al. [WXNM11, WX12] is the conversion of the FT into a Bayesian Network and performing analysis using fuzzy numbers on the resulting BN. It is shown that this approach can give the same results as traditional FT analysis, but it also has the additional flexibility provided by BN.

4.2 Fault Trees with dependent events

Classic FT assume that the BE are all statistically independent. This is often not true in practice, as events can have common causes, or the failure of one component can accelerate the failure of another.

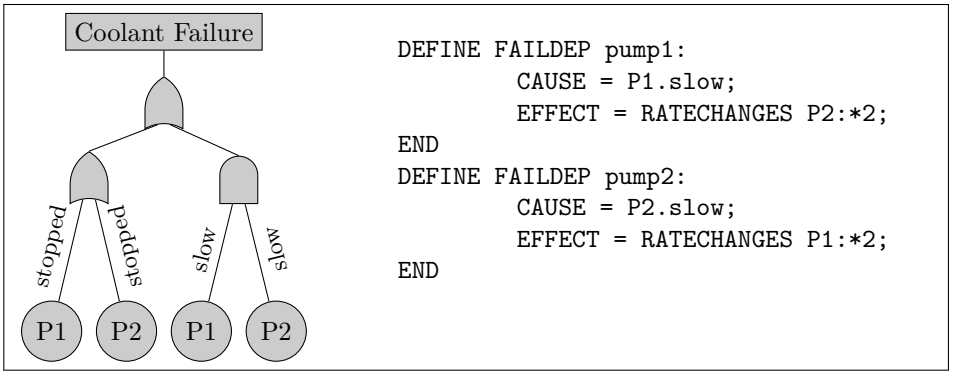


Figure 4.2: Example of an extended FT. Pumps P1 and P2 have failure modes 'stopped' and 'slow'. Either pump stopping or both pumps slowing leads to failure. Either pump slowing accelerates failure of the remaining pump.

Dynamic gates in DFTs can model some dependencies, as was explained in Chapter 3.

Buchacker [Buc99, Buc00b, Buc00a] suggests to modify Fault Trees into 'extended Fault Trees' that allow components to have states other than fully operational and fully failed. This allows the modeling of gradual degradation of a component over time, as well as components that can fail in multiple ways that have different interactions with other failures. In addition, this model adds dependencies between components affecting failure and repair rates. Figure 4.2 shows an example of an extended FT with multi-state components and dependent failure rates.

Another approach for systems with multistate components is provided by Zang et al. [ZWST03]. This approach represents the overall system by multiple fault trees, each of which is a fault tree for a particular failure state of the overall system. These trees are then combined into a single multistate decision diagram with dependent nodes, and analyzed to determine the overall probability of the system reaching each failure state.

Twigg et al. [TRSS00] suggest a method to specify mutually exclusive events. An example of a model where this is useful, is a valve that can fail open or closed. Since these failure modes cannot occur at the same time, a traditional FT cannot correctly model this component.

Yet another design is provided by Vaurio [Vau02], in which mutually dependent events are replaced by groups of independent events, such that a traditional analysis of the FT gives the correct results. A drawback of this approach is that each group of n dependent events is replaced by $2^n - 1$ independent events, which results in a combinatorial explosion if many events depend on each other.

For models with particularly complex interdependencies, Bouissou [Bou02,

Bou08] offers a formalism called *Boolean logic Driven Markov Processes* (BDMP) as an extension to fault trees. In this formalism, events are described by Markov Processes with designated failure states. Then, events in the FT can cause these events to switch to different processes, for example to increase the failure rate if another component fails.

In addition to analyzing the resulting Markov Chains to obtain reliability and availability, it is possible to extract cut sequences from a BDMP [CRL+12], and to construct a Finite State Automaton with equivalent behaviour to the BDMP [CRL+11].

Besides Markov Processes, Bouissou [Bou07] also describes the option to replace BEs with Petri Nets, although no method is described for switching these due to external events. This method can improve the modeling of DFT spare gates with shared spare components.

4.3 Repairable Fault Trees

To analyze the reliability of a system over a long period of time, it is often useful to include the possibility of repairing or replacing failed components during this time. These repairs may extend the time before a system failure occurs, such as when a failed redundant part is replaced, or they may return a failed system to normal operation.

Sometimes the simple repair rate model presented in Section 2.5 is not sufficient. Bobbio et al. [BCR04] introduced *Repair Boxes* which can be connected to a gate, and begins repairs on the BEs of the subtree of that gate only when the gate fails. Raiteri et al. [CRFIV04] extended these repair boxes to allow different repair policies to be used in the model. The resulting tree is called a *Repairable Fault Tree* (RFT). Figure 4.3 shows an example of an RFT.

In this formalism, each BE e has a failure rate $FR(e)$, which is the parameter of an exponential distribution that determines the time until the component fails.

Each RB is connected to one or more BE to repair, and one incoming BE or gate. When the incoming event occurs, the repair box is activated and begins repairs on the outgoing components according to the *repair policy*. Every component also has a *repair rate* that is the parameter of another exponential distribution modelling the time to repair the component.

Repair policies can be very simple, even equivalent to the simple repair rates model, or more complex, for example restricting the number of components that can be repaired simultaneously.

The major advantage of this approach is that it allows modelling of more realistic systems, and analysis of what repair strategies are best. A disadvantage is these trees cannot be quantitatively analyzed using combinatorial methods.

Flammini et al. [FMIM05] added the possibility of giving priority to the repair

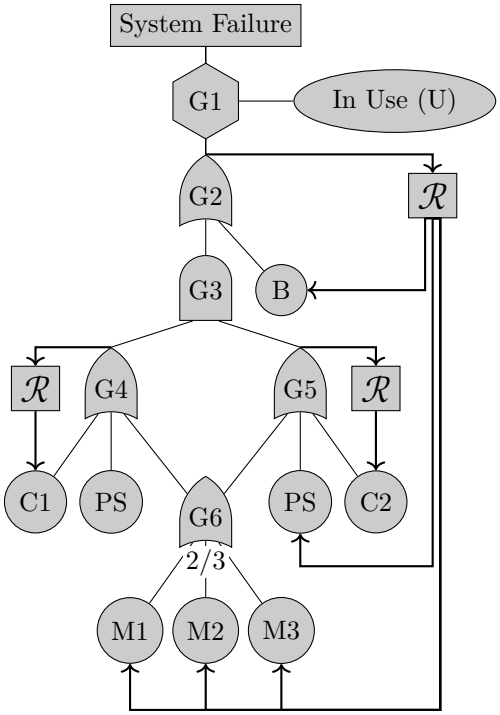


Figure 4.3: Example of an RFT, repairs on the shared components are only initiated when the entire system fails. CPUs 1 and 2 are repaired when their respective compute node fails.

of certain components, based on the repair rate, failure rate, or level of redundancy of the components. Other priority schemes can also be implemented within this system.

A different extension is provided by Beccuti et al. [BCRFH08, BFCRH14], which adds nondeterminism to the repair policies. This models cases where, for example, a mechanic individually decides which component to repair first. Conversion to Markov Decision Process allows optimal policies to be automatically derived from the FT when costs of unavailability, failures, and repairs are provided. A parametric version [BFCRH09] of the formalism allows for more efficient modeling and analysis if the FT contains subtrees that differ only in the parameters of the BEs.

Leaving repair policies nondeterministic also allows the computation of an *optimal* repair policy, by associating costs with unavailability, failures and repairs. Becutti et al. [BFCRH14] show that such an optimal policy can be computed by converting the FT into a Markov Decision Process.

4.3.1 Analysis

RFTs can be analyzed to obtain the same measures that apply to classic FTs with repair rates.

Traditional qualitative analysis of an RFT can be applied identifying cut sets etc. by ignoring the repair aspects. The purposes of such analysis are the same as for non-repairable FTs, such as identifying single points of failure or since such an analysis would ignore the repairability aspect.

Quantitative analysis is more useful, but also more difficult: Combinatorial methods are no longer sufficient, as the evolution of the system over time has to be considered.

For systems where each component can be individually and simultaneously repaired at a constant rate, Balakrishnan and Trivedi [BT95] proposed to convert the model into a Markov Chain, although this method uses an approximation to reduce computational requirements.

Another approximation is provided by Dutuit and Rauzy [DR05], although this approximation can also only be used in models with a constant repair rate. The approximation is shown to give results close to the exact solution and several other approximations.

The more general analysis method proposed by Raiteri et al. [CRFIV04] is to convert the RFT into a Generalized Stochastic Petri Net, and then translate this into a Markov Model. Existing analysis tools for Markov Models can then be applied. Flammini et al. [FMI+14] show that this method can be used on parts of a system while the non-repairable parts can be analyzed using traditional methods.

If the FT contains subtrees that can be effectively parameterized, the method by Bobbio et al. [BCR04] of converting the FT into a Stochastic Well-Formed Net (a coloured version of a Generalized Stochastic Petri Net) and then into a

Markov Chain may be more efficient, although this formalism only allows relatively simple repair boxes. A later extension by Codetta-Raiteri [CR05b, CR11] combines Parametric, Dynamic, and Repairable FTs, allowing complex repair policies, and also performs quantitative analysis by conversion to a Stochastic Well-Formed Net.

A later alternative is offered by Portinale et al. [PCRM10], which translates an RFT into a Dynamic Bayesian Net for analysis. This method also allows complex repair policies, as well as components with several different failure modes and statistically dependent failure probabilities.

For performing very fast Monte Carlo simulations, Kara-Zaitri and Ever [KZE09] developed a method for generating a hardware model of the system in a Field Programmable Gate Array, which can perform each Monte Carlo run many times faster than a conventional computer simulation.

4.4 Fault trees with temporal requirements

As discussed in Chapter 3, dynamic fault trees allow for the inclusion of certain types of temporal information. For some systems, more complex temporal dynamics are required for an accurate model. Several methods have been proposed to offer more flexibility.

One way that has been proposed by Wijayarathna et al. [WM00] is to add an AND-THEN gate. This gate's output event occurs if the second input occurs immediately after the first. For example, a fire safety system might have backup systems that take time to deploy, so a primary system fault before a fire is not a failure, nor is a fault after a fire has already been extinguished. Only a fault immediately after a fire starts (perhaps caused by the fire) causes a system failure.

Walker and Papadopoulos [WBP07, WP09] have suggested extending static FTs with Priority-AND, Priority-OR, and Simultaneous-AND gates. These allow the same temporal relations to be enforced as a dynamic fault tree, but also allow a requirement for simultaneous faults. Such a simultaneous fault is most likely caused by a shared dependency. This method can model any system that can be modeled using the AND-THEN gate. A reduction procedure also described by Walker and Papadopoulos [WP10] can be used to simplify the analysis.

An advantage of this system is that it can still be qualitatively analyzed using algebraic methods, rather than needing to be converted into a Markov Model or other state-space system.

Another construction is described by Schellhorn et al. [STR02], which extends classic FTs with cause-consequence OR- and Inhibit-gates, and synchronous and asynchronous cause-consequence AND-gates. In this model, the classic (called decomposition gates or D-gates) are true if their condition is true at all times. The cause-consequence gates (or C-gates) are true for some indeterminate period after their condition is met.

This construction cannot be used for quantitative analysis, as the C-gates do not have well-defined times at which they are true. Qualitative analysis is possible, as it is proven that the prevention of at least one event from every cut set prevents the TE in this model, just like in a static FT.

If timing information is needed beyond the sequence of events, several other extensions can be used. Gluchowski [Glu07] adds Duration Calculus [CHR91] to FT. This formalism allows reasoning about situations where delays are important. Unfortunately, it has not yet been proven that the gate formulas are decidable, and automated analysis tools cannot currently analyze the dynamic portion of these trees.

Another formalism is the Temporal Fault Tree (TFT) by Palshikar [Pal02]. This formalism adds several gates corresponding to operators in Propositional Linear Temporal Logic (PLTLP), such as PREV n, which is true if the input event has been true for the last n amount of time, and the SOMETIME-PAST, which is true if its input has ever been true.

TFT can impose many types of requirements on the event sequence, but have the disadvantage of requiring the user to understand the formalism of temporal logic.

Qualitative analysis of TFTs is performed by converting them into regular FTs with additional events for the PLTLP gates, and post-processing the resulting cut sets to recover the temporal requirements.

4.5 State-Event Fault Trees

Kaiser and Gramlich [KG04, KGF07] have proposed to extend Fault Trees by combining them with Finite State Machines. Such a State-Event Fault Tree (SEFT) allows for greater modularity, and keeps the diagram more readable than a traditional FT of a complex system. In addition, it can model systems and components that have different states with different failure modes. Computer programs are good examples of such systems.

SEFT have states and events. States describe conditions that last for some time, while events occur instantaneously. The two can be linked, as events can cause transitions between states, and a transition between states is an event. Like in an FT, gates can be used to require conditions before an event occurs. An SEFT distinguishes between a History-AND gate and a Sequential-AND or Priority-AND gate, in that the latter requires the input events to occur in a given order.

A later paper by Kaiser [Kai05] adds delay gates, to model events and state transitions that occur some time after an initiating event, conditional probability gates, that cause the output event to occur with some probability every time the input event occurs, and a set of adapter gates that allow certain translations between states and events.

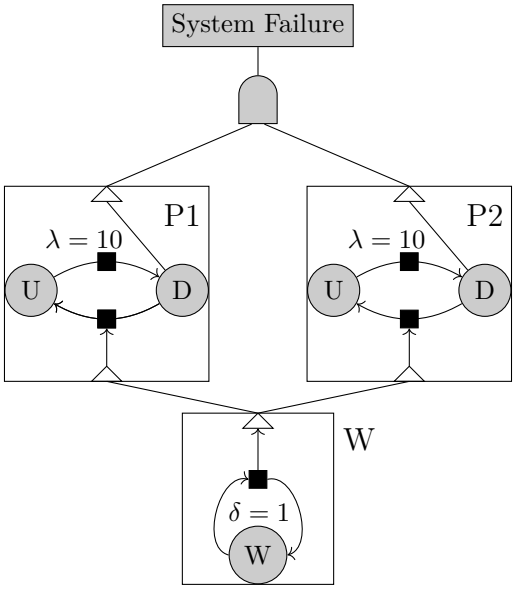


Figure 4.4: State-event fault tree example of two computer processes P1 and P2, which fail approximately once every 10 hours. The watchdog process W restarts any failed process once per hour. System failure occurs when both P1 and P2 are down.

Analysis of SEFT can be performed by translating them into Deterministic and Stochastic Petri Nets, and using existing tools to analyze the resulting DSPN.

Förster and Kaiser [FK06] provide a more efficient way of performing this analysis, by dividing the SEFT into modules, and converting any static modules found into Component Fault Trees (CFT). A hybrid analysis can then be performed combining BDD for the CFT and DSPN for the dynamic submodules, which is more efficient than using a DSPN for the entire tree.

Xu et al. [XHH+13] introduce formal semantics for SEFT, and provide a method based on these semantics to determine MCS. This method extends Interface Automata [dAH01] to Guarded Interface Automata, and translates an SEFT into a GIA Network. From this network the cut sequences can be determined and reduced into a minimal cut sequence set.

Another method for qualitative analysis is provided by Roth et al. [RL13], which converts the SEFT into an *extended Deterministic and Stochastic Petri-Net* (eDSPN), on which a reachability analysis can be performed to identify event sequences that result in failure.

4.6 Miscellaneous FT extensions

One particular extension that does not fit these categories was proposed by Fovino et al. [FMC09], and integrates Attack Trees with FT. Attack Trees describe vulnerabilities in a system that an attacker could exploit, and countermeasures that could remedy these vulnerabilities.

Since an outside attack could cause a system failure, the combination of AT with FT may provide a better estimate of the system failure probability, assuming probabilities for attack scenarios can be provided.

The integration is performed by designating certain BEs as attack nodes, and decorating these BEs with attack trees. The attack trees are then individually and separately analyzed to determine the probability of a successful attack. Once this analysis is complete, the FT is analyzed by substituting the computed probabilities into the BEs.

Attack trees are sufficiently different from fault trees that we consider them beyond the scope of this thesis. An overview of attack trees and related methodologies has been written by Kordy et al. [KPCS14].

4.7 Comparison

Tables 4.1 (page 84) and 4.2 (page 98) summarize the various extensions described above, with strong points denoted with a plus. The meaning of the headers in Table 4.2 is as follows:

Uncertainty How well the formalism can describe systems with uncertain probabilities and/or structure.

BE Dependence How well the method can model systems in which the basic events are not statistically independent.

Temporal Requirements How well the formalism can include requirements on the sequences or durations of events.

Repairable To what extent the method can include repairable components and descriptions of repair strategies.

Multi-state Whether the model can include components with more states than just failed or not.

BE Prob. distribution Whether the model can describe systems in which the basic events have failure distributions other than constant probability and inverse exponential failure rate.

4.8 Conclusion

This section has presented many extensions to fault trees, supporting aspects such as undertainty, stochastic dependencies between events, repairs, temporal requirements, and stateful systems.

The various extensions support different kinds of analysis. Most support quantitative analysis of certain measures, while some also provide qualitative measures. A few extensions provide concrete tool support, but most only describe the analysis technique and results for a small number of examples.

The large number of extensions, almost all incompatible with each other, can make it difficult to decide which formalism is best suited to a particular system. In [RSSR17], we have presented a metamodel that ameliorates a similar jungle of variants in the formalism of attack trees in which also (dynamic) fault trees are incorporated. In future work, this metamodel could be extended to cover many of the FT extensions as well, providing a unified formalism and allowing combining the best of all worlds.

	Modeling						
	Uncertainty	BE Dependence	Temporal requirements	Repairable	Multi-state	BE Prob. distribution	Remarks
DFT with repair boxes [BCR04]		+	+	+			Parametric, simple repair policy
Repairable FT [CRFIV04]				+			Complex repair policies
Combined FT [CR05b, CR11]		+	+	+			Dynamic, complex repair policies
Nondeterministic RFT [BCRFH08, BFCRH09, BFCRH14]				+			Allows nondeterministic repair choices
FT with Attack Tree [FMC09]						+	Models deliberate attacks
Fuzzy FT [TFLT83]	+						Models uncertain BE prob.
Fuzzy FT [Sin90]	+					+	Special membership functions
Fuzzy FT [LW97]	+						Linguistic description
Fuzzy FT [KJG96]	+						Multiple expert estimates
Fuzzy FT [SCC06, Li08]	+					+	Uncertain membership functions
Fuzzy FT [RK11]	+				+	+	Multi-state BEs
Fuzzy FT [EIO98]	+			+		+	Uncertain repair rates
Fuzzy FT [HTZ04]	+					+	For large uncertainties
Fuzzy FT [PP94]	+					+	Normally distributed rates
Extended FT [Buc99, Buc00b]		+		+	+		Multi-state BEs
Multistate FT [ZWST03]		+			+		Multi-state FT
FT with mutual exclusion [TRSS00]		+					Mutually exclusive events
FT with dependent events [Vau02]		+					Stat. dependent events
BDMP [Bou02, Bou07, Bou08][1]		+	+	+	+	+	Complex dependencies
FT with AND-THEN [WM00]			+				Requirement of immediacy
DFT with simultaneity [WBP07, WP09][2]			+				Requirement of simultaneity
Formal FT Semantics [STR02]	+		+	+			Includes delays
FT with Duration Calculus [Glu07]	+		+	+			Complex temp. requirements
Temporal FT [Pal02]			+	+			Includes temporal logic
State-Event FT [KG04, Kai05]		+	+	+	+		Combines FT and FSM

Table 4.2: Comparison of fault tree extensions

[1] Analysis tool: BDMP [Bou12]
[2] Analysis tool: Pandora [WP09]

Part II

Integrating maintenance into fault trees

Chapter 5

Fault maintenance trees

Reliability-centered maintenance (RCM) [Mou97] is a major trend in infrastructural asset management. Its goal is to improve maintenance planning by determining the assets where maintenance is most needed, and maintaining these assets more intensively than less critical objects. By focussing maintenance expenditure where it is most effective, RCM aims to balance maintenance costs against system dependability. This requires a good insight into the behaviour of the system w.r.t. reliability, and into the effects of maintenance.

As explained in the preceding chapters, fault tree analysis (FTA) is an industry-standard, widely-used technique to gain insight into system dependability. By studying the failure behaviour of system components, and the interactions of these components in leading to system failures, we can obtain various measures relevant to the overall system dependability. These measures can be qualitative (e.g., *cut sets*) or quantitative (e.g., *reliability* or *availability*).

Also discussed in previous chapters is the inclusion of repairs in FTA. Standard fault trees support a simple model in which failed components are all simultaneously repaired with repair times following simple probability distributions such as the exponential distribution. Such models are relatively easy to analyze and obtain the system *availability* (expected fraction of time where the system is not failed), but their expressive power is limited as most realistic situations do not have the resources to repair all components at the same time.

Section 4.3 describes various extensions to FTA supporting more advanced repair models, mostly focussing on more elaborate descriptions of which components can be repaired at the same time.

While such extensions more accurately model repairs in resource-constrained situations, they still cannot model most practical maintenance situations. Real-life maintenance is often much more complex than only repairs. Apart from repairing failed components, practical policies often involve periodic inspections to determine which components need repairs, preventive repairs before components have failed, and interactions between components in both degradation and repair. For example, the tires on a car are not simply replaced when they are completely worn our. Rather, owners periodically examine the tire to check that they have enough tread left, replace the tires when the tread is about to become too shallow, and replace two or four tires at the same time even if some could be used for a few more weeks.

This chapter introduces fault maintenance trees (FMTs). These extend fault trees with advanced maintenance policies, including wear-out behaviour of components, and periodic inspections and repairs with complex policies to prevent and/or undo such wear. Figure 5.2 illustrates how wear-out, inspections, and repairs are included in an FMT.

An FMT consists of two or three components:

- A fault tree describing the failure behaviour of the system, extended with more advanced descriptions of the wear behaviour of components. This FT deviates from classic fault trees in two major ways: First, basic events are not described by a single probability or distribution of failure times, but rather can have multiple phases describing the degradation of the component over time. Second, a new gate is introduced, called the *rate-dependency* or *RDEP* gate. This new gate describes how failures of one component or subsystem accelerate the failures of other components. The details of this FT are described in Section 5.2.

- A maintenance policy describing inspections and repairs, specifying when each occurs and under what conditions they trigger further maintenance actions. This policy can include periodic inspections that examine the degradation levels of one or more BEs and can trigger repair actions, and repair actions that can occur periodically or in response to failed inspections. The maintenance policy is detailed in Section 5.3.

- Optionally, a cost model describing the costs of failures of the various (sub)systems, and of the maintenance actions. These costs can depend on the length of time a failure or maintenance action occurs, or can be fixed per occurrance. If costs are provided, they can be used to compute the total costs for a given maintenance policy, broken down into the different aspects. More details about the cost model can be found in Section 5.4.

FMTs support the calculation of a range of important dependability metrics, such as system reliability, availability, etc. under a given maintenance policy. They also support metrics important to maintenance planning and optimization, including expected cost over time. These measures permit the optimization of a maintenance policy w.r.t. certain goals, such as minimal cost or maximal availability within a given budget.

Technically, the analysis of FMTs is realized via statistical model checking [LDB10], a state-of-the-art Monte Carlo simulation technique that allows the various metrics to be computed efficiently and with statistical confidence intervals.

This chapter provides the methodological underpinnings of the industrial case studies in Chapters 7 and 8.

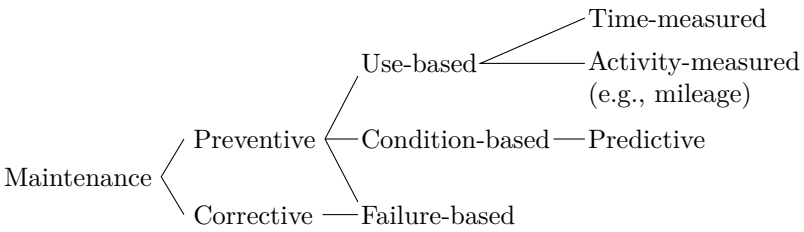


Figure 5.1: Overview of maintenance types and schedules.

Origin of the chapter. This chapter describes FMTs as introduced in:

- Enno Ruijters, Dennis Guck, Peter Drolenga, and Mariëlle Stoelinga. "Fault maintenance trees: reliability contered maintenance via statistical model checking". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2016. DOI: 10.1109/RAMS.2016.7447986, ISBN: 978-1-5090-0248-1.

Organization of the chapter. Section 5.1 discusses maintenance concepts and how they relate to fault trees. The modeling of degradation in FMTs is discussed in Section 5.2, and the modeling of FMT maintenance policies in Section 5.3. The cost model used in an FMT is briefly described in Section 5.4. Finally, the analysis of FMTs is described in Section 5.5 before ending with the conclusion in Section 5.6.

5.1 Maintenance concepts

Most long-lived systems require some level of maintenance to keep up performance and avoid premature failure. This maintenance ranges from very simple, such as changing the battery in a smoke detector every year, to very complex overhauls of entire power plants. This section provides an explanation of maintenance of physical systems, particularly as it is applied in the railway industry. We first discuss what kinds of maintenance are applied, and then how this maintenance is scheduled. An overview of the policies and schedules can be seen in Figure 5.1.

Types of maintenance. Maintenance actions can be broadly divided into two categories: *preventive* and *corrective* maintenance [Ebe97]. Preventive maintenance is performed before a failure (of a component) occurs, with the aim of reducing the likelihood of a failure in the future. Corrective maintenance is performed after a failure has already occurred, with the goal to restore the component or system to a functioning state.

Failure, here, can be defined in various ways, such as physical breakage, or degradation below some specified minimal performance. In this chapter, we define failure as the inability of a component or (sub)system to perform one or more of its required functions within the overall system under consideration.

Note that component failures can be distinct from system failures. Many systems are designed with some level of redundancy, such that not all components are required for the system to operate. For example, a datacenter typically has redundant power supplies, such that one supply can fail while the datacenter remains fully operational. How such component failures interact to cause system failures is described by a fault tree, as described in Chapter 2 and Section 5.2.

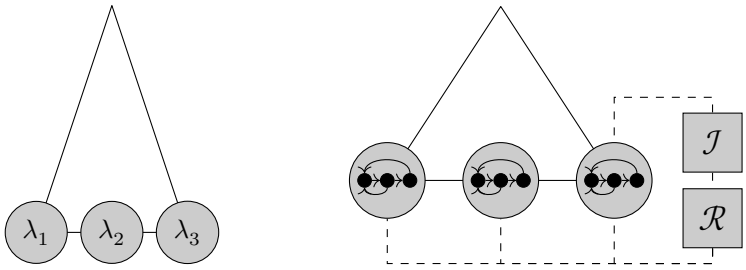
The choice of which type of maintenance to apply can depend on various factors, including the different costs of failures and maintenance actions, and whether the components are even able to be preventively maintained. In cases where failures are much more expensive than maintenance, such as the critical components of airplanes, preventive maintenance is almost always cost-effective. Conversely, when failures are not very expensive compared to maintenance, such as the lightbulbs in your home, corrective maintenance is often a better option. Some failures, such as lightning strikes, are difficult or impossible to prevent using maintenance and must be solved correctively, when they happen.

Maintenance planning. Once it has been established that some preventive maintenance needs to be performed, one needs to decide when to perform it. Broadly speaking, we can identify three methods of planning: *use-based*, *condition-based*, and *failure-based* maintenance [Git92].

Use-based maintenance is the most basic type of maintenance planning: It performs the specified task after some relevant unit of use, such as elapsed time or miles driven. We note here that 'use' is a fairly abstract concept, and does not necessarily refer to the system being active. For example, a rubber hose may need to be replaced after a specific time has elapsed, whether the hose has been actively used or lying on a shelf.

Condition-based maintenance is more elaborate, and specifies that certain maintenance must be performed based on the condition of the system. In some cases, the relevant aspects of this condition are self-evident, such as when a smoke detector starts beeping to indicate that its battery is low. Less obvious condition may require specific actions to determine, such as measuring the pressure in a car's tire. In the latter case, a combination of use-based and condition-based planning is often used, where inspections are performed on a use-based schedule and the outcome of these inspections is used for a condition-based decision to take maintenance actions.

A recent trend in maintenance planning is so-called *predictive* maintenance, where the current (and sometimes historical) state of the system is used to decide when in the future to perform the next inspection or maintenance action. This is a



(a) Traditional fault tree (b) Fault maintenance tree

Figure 5.2: **Illustration of how fault maintenance trees extend traditional (static) fault trees** by adding more basic events with degradation and inspection and repair modules.

particular kind of condition-based planning, where the plan is updated based on current observations.

Finally, failure-based maintenance is typically used for corrective maintenance. Here, action is only taken once a failure occurs. Such a maintenance policy is also called 'run-to-failure' [Blo05]. Note that while failure-based plans almost always specify some corrective action after a failure, preventive actions can be scheduled for the same time. For example, if the battery of one smoke detector is empty, it is likely that other batteries (installed at the same time) are also nearly empty. It is then more efficient to replace all the batteries at once, rather than to wait for each to fail over the course of a few weeks.

5.2 Fault tree modeling

Aside from a maintenance plan, FMTs contain a model of the degradation of the components of the system, and how failures of these components interact to cause (sub)system failures. Chapter 2 describes fault trees in general, this section explains how they are extended in FMTs to cover more complex degradation patterns.

As described in Chapter 2, fault trees are graphical descriptions of the propagation of component failure to failure at system level. In classic fault trees, the failures rates of components are assumed to be constant over time. In practice, however, several factors can change these failure rates: 1) physical components wear out over time, leading to increased failure rates as the component nears the end of its lifespan; 2) components can be subjected to increased stress when other subsystems fail, e.g., the failure of a cooling pump in a redundant system can require the remaining pump to perform more work, leading to an increased failure rate; and 3) periodic maintenance can prevent or even remove wear of component,

leading to decreased failure rates.

5.2.1 Basic events

Basic events in FMTs support arbitrary distributions of failure times. Concretely, we require that the degradation of the component can be described in a finite number of states with probability distributions over the time spent in each state. For simple components that are not subject to preventive inspections or repairs, the states can simply be 'operational' and 'failed'. If inspections are needed, the outcomes of an inspection can only depend on the state of the component, thus generally requiring more than two states (e.g., 'good', 'noticably worn', and 'failed'). The BEs move between these states at times governed by probability distributions. Most commonly these are exponentially distributed, making the BE model a Markov chain (usually describing a hypoexponential distribution, i.e., the Markov chain consists of a sequence of states) but FMTs allow arbitrary distributions of transition times including Weibull and normal distributions.

Repairs are events triggered externally to the BE, causing it to return to a less degraded state. In many cases, repairs return the component to a state as good as new, e.g., by replacing the battery in a sensor. Such a BE is shown in Figure 5.3a. More complex components can be affected by different repair actions, such as a complete replacement returning to the as-good-as-new state and a partial repair that leaves to component working but does not fully remove the wear. A more complex BE is shown in Figure 5.3b.

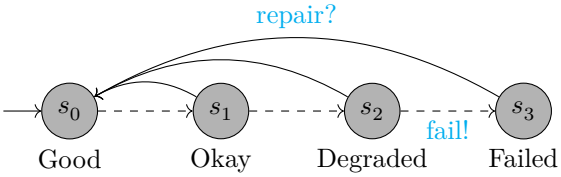
5.2.2 Gates

The gates of an FMT describe how events (basic events or intermediate events of other gates) propagate through the system, i.e., how combinations of events lead to subsystem failures and eventually the failure of the entire system.

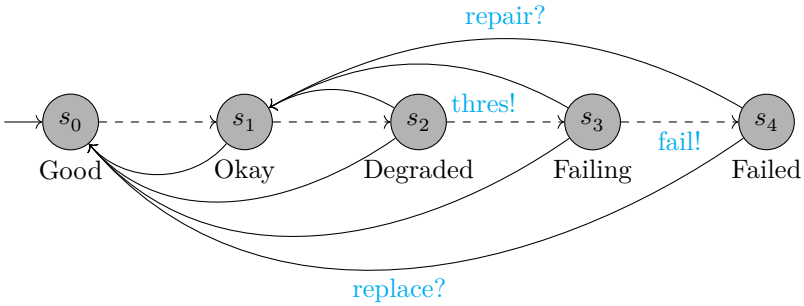
Apart from the gates of standard fault trees, FMTs introduce a new gate: the rate-dependency (RDEP) gate. This gate is described in detail in Section 5.2.3.

The traditional gates are the *AND*, *OR*, and *VOT(k/N)* gates failing, respectively, when all, any, or at least k of their children fail. Since the basic events of an FMT support repairs, the gates can also become operational again when the number of currently-failed inputs becomes sufficiently low. The current state of these static gates is thus determined entirely by the current states of their children.

Figure 5.4 shows the behaviour of an *AND*-gate with two children. The gate listens for failure and repair signals of both children, keeping track of how many children are currently failed. When both children are failed, the gate emits its own failure signal. If, after the gate has failed, any of the children are repaired, the gate also emits its repaired signal. The other gates are defined analogously to the *AND*-gate (following the semantics described in [BCS07c]).



(a) BE with four degradation phases and perfect repair.



(b) BE with five degradation phases, detectable degradation after state s_2, imperfect repair and perfect replacement.

Figure 5.3: **Basic diagrams of the behaviour of repairable basic events with degradation.** The dashed arrows indicate transitions occurring at stochastic times, the solid arrows are immediately taken when a repair action is performed. The cyan text indicate labels that are used to communicate with other elements of the FT.

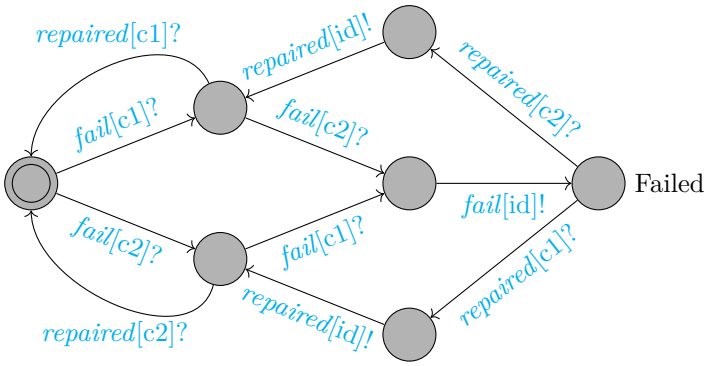


Figure 5.4: **Automaton of an AND-gate with two children** (with IDs c1 and c2) Only when *fail* signals have been received from both children, does the gate emit its own signal *fail*[id]. If either child emits a *repaired* signal, the gate responds by no longer treating that child as failed. If this occurs after the gate has failed, it emits its own *repaired*[id] signal.

The introduction of degradation into FMTs does not affect the traditional gates. One could imagine cases where multiple degraded components have similar effects as a broken one. For example, if multiple pumps are used to provide cooling water, multiple pumps producing a reduced flow has a similar effect to one pump failing entirely. While current FMTs do not support such cases, it would be a straightforward extension to introduce gates that can depend on the exact degradation level of their children, and even gates that can have multiple degradation levels themselves.

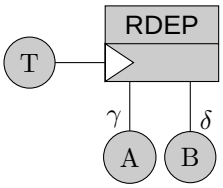


Figure 5.5: **RDEP gate** where the failure of the trigger event T causes basic event A to wear γ times faster, and basic event B δ times.

5.2.3 Rate dependencies

Rate dependency (RDEP) gates are a new gate type, modeling situations where the failure or degradation of one component affects the degradation rate of another. Such situations commonly occur in redundant systems, where the failure of one redundant element leads to increased use and thus wear of the other element, and systems affected by common cause failures where the presence of some harmful condition, such as excess vibration, leads to accelerated wear of many components. In contrast, RDEPs can also be used to model reduced failure rates, e.g., if a power supply fails, the components powered by that supply stop working and also stop incurring wear.

The depiction of the RDEP gate is shown in Figure 5.5. Each RDEP gate has a single input called its trigger and one or more BEs called the *dependent children*, each with its own acceleration factor. In the formal definition, BEs that are not dependent children are assigned an acceleration factor of 1.

When the trigger input is active, the failures of the dependent children proceed at an altered rate. If the children follow exponential distributions, this can be viewed as multiplying the rate of this distribution by the acceleration factor. For other distributions, we multiply the speed at which time passes for that child by the acceleration factor. When the trigger input becomes inactive, the children resume wearing at their normal speeds.

Formally, the acceleration due to an RDEP alters the failure time distribution in the following manner: Suppose we have a BE with a failure time given by T_A governed by a cumulative probability distribution D (i.e., $\mathbb{P}(T_A \leq t) = D(t)$), and this BE is affected by an RDEP gate with factor γ and a trigger occurring at time T_T governed by some probability distribution. We then have, assuming that the trigger is not repaired, that the new failure time becomes:

$$T_{A'} = \begin{cases} T_A & \text{if } T_A < T_T \\ T_T + \frac{1}{\gamma}(T_A - T_T) & \text{otherwise} \end{cases}$$

In other words, the failure distribution proceeds unaffected until time T_T, and if BE A is not failed at that time, the remaining time is divided by the factor γ. If trigger T is repaired at time T_R, the remaining time after that is multiplied by γ to return to the normal rate, i.e.:

$$T_{A''} = \begin{cases} T'_A & \text{if } T'_A < T_R \\ T_R + \gamma(T'_A - T_R) & \text{otherwise} \end{cases}$$

This procedure of failure and repair of the trigger can be repeated multiple times if needed.

There is no restriction on the number of RDEPs that can simultaneously affect one BE, and all acceleration factors are multiplied. Thus, if a BE is accelerated by

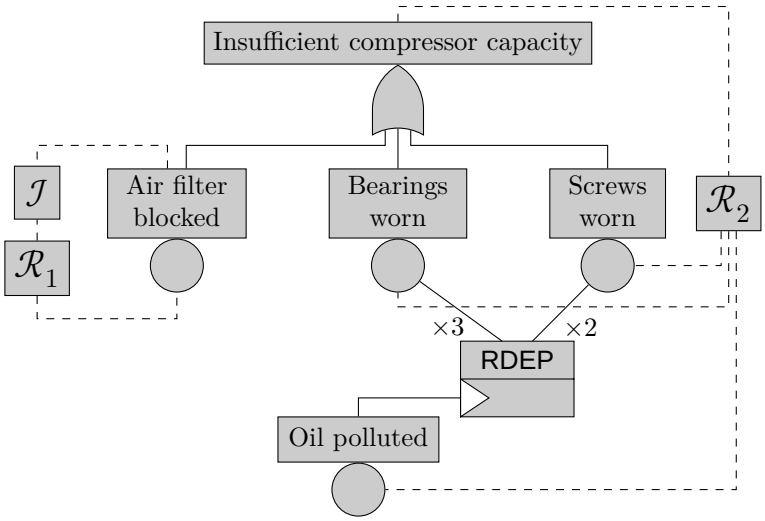


Figure 5.6: Example of a fault maintenance tree describing part of a pneumatic compressor. The basic event 'Oil polluted' accelerates the events 'Bearings worn' and 'Screws worn'. The repair boxes indicate that 'Oil pollution', 'Bearings worn', and 'Screws worn' are repaired when the top event occurs, while 'Air filter blocked' is repaired based on an inspection of the filter itself.

one RDEP with a factor 3 and another RDEP with a factor $\frac{1}{2}$, the BE will fail $\frac{3}{2}$ times faster than normal when both RDEPs are triggered.

Figure 5.6 shows an FMT with one RDEP gate, where 'Oil polluted' is the trigger, and 'Bearings worn' and 'Screws worn' are the dependent children.

5.2.4 Formal definition

The formal definition of a fault maintenance trees extends the formal definition of an FT (described in Section 2.2.2) with multi-state basic events, RDEP gates, and maintenance modules.

First, we extend the set of gate types to include RDEP gates. We define an RDEP gate to have a single acceleration factor γ, as RDEPs with children with different rates can be trivially split up into multiple RDEP gates, each with a single child and its rate. We thus define $\mathit{GateTypes} = \{\mathit{AND}, \mathit{OR}\} \cup \{\, \mathit{VOT(k/N)} \mid k, N \in \mathbb{N}^{>1} \wedge k \leq N\} \cup \{\mathit{RDEP}(\gamma) \mid \gamma \in \mathbb{R}^{+}\}$.

We now define a fault maintenance tree:

Defintion 10 *A fault maintenance tree is a tuple* $\mathcal{F} = \langle \mathit{BE}, \mathit{TE}, S, D, G, T, I, P,$

C⟩ consisting of the following components:

- *BE is a set of basic events.*
- *TE ∈ BE is the top level event.*
- *$S : BE \to \mathbb{N}$ specifies, for each BE, how many degradation states it has. For a traditional BE with exponentially distributed failure times, this is 2 (failed and non-failed), BEs with degradation over time are described with more states. The initial degradation state is state 0, the failed state of BE b is $S(b)$.*
- *$D : BE \to \mathbb{N} \to \mathbb{N} \to (\mathbb{R}^+ \to \mathbb{R}^+)$ specifies the distributions of transition times between the states. Specifically, $D(b, i, j)$ is the cumulative distribution function of the transition time from state i to state j of BE b. E.g., if b is an exponentially distributed BE with parameter λ, then $S(b) = 1$ and $D(b, 0, 1)(t) = 1 - e^{-\lambda t}$.*

  We require that the continuous-time Markov chain specified by every $D(b)$ is acyclic, i.e., if $D(b, i, j) \neq 0$, there is no sequence $\langle i = i_0, i_1, ..., i_k - 1, i_k = j \rangle$ such that $\forall_{l<k} D(b, i_l, i_{l+1}) \neq 0$.
- *G is a set of gates, with $BE \cap G = \emptyset$. We write $E = BE \cup G$ for the set of elements.*
- *$T : G \to GateTypes$ is a function describing the type of each gate.*
- *$I : G \to E^*$ describes the (ordered) inputs of each gate. The ordering is relevant only for the RDEP gate, where the first input denotes the trigger, and the second input denotes the dependent child.*
- *P is a maintenance policy, as will be explained in Section 5.3.*
- *C is a cost model, as will be explained in Section 5.4.*

If we denote the static gates as $SG = \{g \in G \mid \nexists \gamma : T(g) = RDEP(\gamma)\}$, we require that the graph formed by $\langle BE \cup SG, I \rangle$ is a directed acyclic graph with a root TE.

We note that, if an FMT does not contain any RDEP gates, the tuple $\langle BE, G, T, I \rangle$ is a static fault tree.

5.3 Maintenance modeling

Maintenance policies in FMT consist of two major elements: *inspection modules*, which periodically examine the state of one or more BEs and which initiate repair actions when any of those BEs is degraded beyond some threshold, and *repair modules*, which return one or more BEs to less degraded states upon activation by an inspection module, the failure of a BE, or after a certain amount of time. The timed automata describing the IM and RM are shown in Figures 5.7 and 5.8, respectively.

In the graphical form, we denote inspection and repair modules using square boxes containing $\mathcal{I}$ or $\mathcal{R}$ for inspections and repairs , respectively. Inspection modules are connected to the events they inspect, and to any repair modules they can activate. Repair modules are connected to the elements whose failures initiates a repair, and the elements repaired by the module. To prevent ambiguity, we connect triggering elements to the top of the module, and repaired events to the bottom.

Inspection modules are usually decorated with a frequency denoting how often the inspection is performed, and threshold phases for each of the inspected elements indicating when the inspection triggers further action. If, at the time of an inspection, at least one of the inspected events is at or beyond its threshold phase, all attached modules are actived.

Repair modules may be decorated with a frequency as well, if repairs are carried out on a regular schedule. Repair modules are further decorated with the states to which the repaired BEs return after the repair is performed. When the repair module is activated, either by its own timer or by an inspection, the BEs are placed back in these states, possibly also changing the states of their parent gates.

The example in Figure 5.6 shows a maintenance policy with three modules: A repair module $\mathcal{R}_2$ that repairs the BEs 'Oil pollution', 'Bearings worn', and 'Screws worn' whenever the top event occurs; an inspection module $\mathcal{I}$ periodically checking the air filter; and a repair module $\mathcal{R}_1$ replacing the air filter when the inspection fails.

More complex maintenance policies (e.g., where inspections trigger corrective actions only when multiple BEs have failed) can be described by constructing subtrees in the FT describing the exact conditions of the inspection or repair.

Formal definition. We will now define the maintenance policy P:

Defintion 11 *A maintenance policy is a tuple* $P = \langle M, T_P, I, T_R, R, A \rangle$ *with the elements:*

- M *is a set of maintenance modules.*
- $T_P : M \to \mathbb{R}^+ \cup \{\perp\}$ *is either a time describing the interval at which the module is activated, or* $\perp$ *for modules that are only activated externally.*

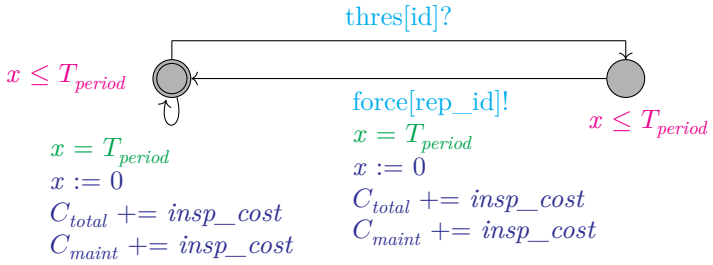


Figure 5.7: **Timed automaton for an inspection module** with ID 'id': Every T_{period} time units, an inspection is carried out. If the BEs being inspected have not reached the threshold for this inspection (i.e., sent a thres[id]! signal), the module takes the self-loop on the left and incurs a cost without taking further action. If the threshold has been reached by any of the inspected BEs, the module will be in the location on the right when performing the inspection, and will also incur its cost, but will now send a force[rep_id]! signal to the repair module to immediately start its repair.

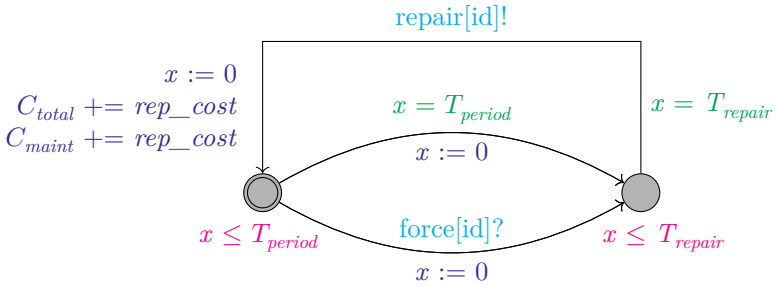


Figure 5.8: **Timed automaton for a repair module** with ID 'id': A repair is started every T_{period}, or whenever a force[id] signal is received. The repair takes T_{repair} time, after which a signal repair[id] is emitted to indicate that this repair is complete, the repair costs are added, and the automaton resets. All BEs affected by the repair listed to this repair[id] signal and transition to the repaired state upon receiving it.

- $I : M \to \mathcal{P}(E \times \mathbb{N})$ *describes the set of elements to inspect when the module is activated, and the threshold state for each of these events. Gates are considered to have two states: 0 for not failed and 1 for failed. If any of the elements is at or beyond its threshold, the inspection is said to be failed.*
- $T_R : M \to \mathbb{R}^+$ *is the repair time. That is, the time taken after a failed inspection or external activation, before the effects of* R *occur.*
- $R : M \to BE \to \mathbb{N} \to (\mathbb{N} \cup \bot)$ *is a function describing the repair effects: It gives, for each module, BE, and current state of the BE, the new state of that BE after repair. If* $R(m, b, i) = \bot$*, this means that the BE is not affected by the repair in its current state. Note that this is different from* $R(m, b, i) = i$*, as the latter also resets the probability distribution of the next transition time.*

  The repairs are performed after the abovementioned repair time T_R *after a failed inspection or external activation.*
- $A : M \to \mathcal{P}(M)$ *describes, for each module, a set of other modules to immediately activate if the current module is activated and at least one inspection fails.*

This definition allows inspection modules and repair modules to be described in the same structure, and allows hybrid inspection/repair modules. If a separation is desired, one can define an inspection module as a module i in which $\forall b, x : R(i)(b, x) = x$ (i.e., all BEs are left in their current state and not repaired) and a repair module as a module r in which $I(r) = \emptyset$, (i.e., a module that does not perform any inspection of its own).

5.4 Costs

FMTs can describe two kinds of costs incurred by maintained systems: cost of failure and cost of maintenance. These can be further broken down into failure costs of (sub)systems down to component level, and costs of the various maintenance actions.

Failure costs are incurred when an element of the fault tree is in its failed state. The cost can be instantaneous (i.e., incurred at the moment the failure occurs) or time-based (i.e., incurred per unit time until the failure is corrected). Multiple costs can be incurred at the same time, e.g., when an expensive component breaks leading to a system failure, the component can incur its own cost, as well as the cost of the system failure.

We note that we generally use failure costs only for the cost of the consequences of that failure, not the cost of the failed component (which is part of the repair cost).

Maintenance actions can similarly be decorated with individual costs, for each inspection and repair action. Should it be desirable to have a different cost for, e.g., a failed inspection than for a successful inspection, this can be modeled by introducing a repair action that has no consequences but has a non-zero cost. Similar modeling patterns can be used for, e.g., different replacement costs for failed and non-failed component.

Like failure costs, maintenance costs can be instantaneous per execution of the action, and/or time-based depending on how long the maintenance action takes. This is most useful when the action take stochastic amounts of time rather than the fixed time described in this chapter.

Finally, costs are tracked in the model at different levels of granularity: A total cost is tracked, as well as separate costs for failures, inspections, and repairs.

Formal definition. Recall from Section 5.2 that we defined an FMT as a tuple $\mathcal{F} = \langle BE, TE, S, D, G, T, I, P, C \rangle$ where C is a cost model. We now formally define this cost model (recalling that $E = BE \cup G$ denotes the set of all basic events and gates) as follows:

Defintion 12 *A cost model is a tuple $C = \langle C_{IF}, C_{TF}, C_M \rangle$ where:*

- $C_{IF} : E \to \mathbb{R}^+$ *describes the instantaneous cost of an element failing.*
- $C_{TF} : E \to \mathbb{R}^+$ *describes the per-unit-time cost of an element being failed.*
- $C_{IM} : M \to \mathbb{R}^+$ *describes the instantaneous cost incurred when the given maintenance module is activated.*
- $C_{RM} : M \to \mathbb{R}^+$ *describes the per-unit-time cost incurred while the given maintenance module is active (i.e., repairing).*

Note that this cost model does not provide for different costs depending on whether an inspection causes a repair or note. If such a difference is desired, it can be achieved by splitting the maintenance modules into an inspection module and a repair module.

5.5 FMT analysis via statistical model checking

Quantitative analysis of FMTs is performed using statistical model checking of stochastic timed automata (STAs) [BLR05]. That is, we first convert the FMT into a network of STAs, and then use the statistical model checker Uppaal-SMC [BDL+12] to compute the requested metrics.

A TA is a model consisting of locations and transitions between these locations. The locations represent control states of the system, and transitions describe

situations when the system may move from one location to another. Constraints on the edges and invariants on locations may be used to block or force certain transitions at certain times. These constraints and invariants are specified in terms of clocks, which increase linearly over time but may be reset when a transition is taken. Stochastic timed automata extend TAs by allowing transition times to be governed by probability distributions, not only by nondeterminism. Multiple (S)TAs can be combined using synchronisation on transitions, where some edges waiting for a signal *sig?* can only be taken simultaneous with a transition in another STA emitting the corresponding signal *sig!*.

An example of a TA can be seen in Figure 5.7, describing an inspection module. The initial location is the one on the left. Here, the clock x denotes the time since the previous inspection, and increases until it is reset when an inspection is performed. The invariant on the initial location prevents the TA from remaining in this location when the time to perform an inspection has been reached. Before this time, the guard on the self-looping transition prevents a premature inspection. When the clock x is equal to the time T_{period}, the self-loop is taken and the clock is reset. The edge to the location on the right is a synchronization transition on the channel *thres*, and is taken when a component has degraded enough to take the corresponding transition in its STA. After this, the IM still waits for the inspection time, but the transition back to the initial location now also synchronizes with the repair module to begin a repair. Finally, both transitions corresponding to performing an inspection add a fixed amount *insp_cost* to a global counter.

Each element of the FMT (i.e., each BE, IM, RM, and gate) is assigned a unique ID, and a template of the appropriate STA is instantiated with the specific parameters for the element. The STAs for the basic event, repair module, inspection

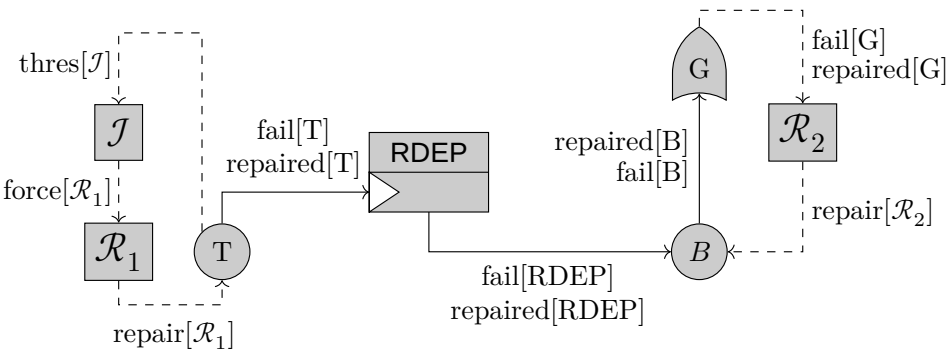


Figure 5.9: **Illustration of the signals used to communicate between the different FMT elements.** The arrows indicate the sources and destinations of the signals.

module, and AND-gate are shown in Figures 5.10, 5.8, 5.7, and 5.4 respectively. The other gates are modeled analogously to the AND-gate. The IDs are used to instantiate the synchronization signals. The various signals are illustrated in Figure 5.9.

The STA is then analyzed using the UPPAAL model checker. This approach has the advantage of allowing both quantitive analysis of the metrics described in Section 5.5.1 using statistical model checking, and qualitative analysis and validation of the structural correctness of the model using traditional model checking. The latter enables us to check properties of the model such as that every BE can be repaired, that every gate can fail, etc.

Qualitative checks require a state-space exploration of the model, which leads to exponential time-complexity as the number of FMT elements increases. Fortunately, the statistical model checker does not need to generate the full state space, and thus its computation time is relatively independent of the number of elements, but rather grows with the desired accuracy of the result. The downside of this approach is that the results are not exact values, but rather statistical confidence intervals. This is a particular problem when tight confidence bounds are desired, as the computation time required can grow very large (Chapter 6 explains this problem in more detail and presents an approach to ameliorate it).

5.5.1 Metrics

After converting an FMT into an STA, various metrics of the system can be computed. We list the most important ones below:

Reliability is defined as the probability of the system failing within a given time window. Formally, if we describe the behaviour of the system described by a fault tree F using $X_F(t) = 1$ when this system has failed at time t, and $X_F(t) = 0$ if it has not, the reliability is defined as $Re_F(t) = \mathbb{P}[X_F(t) = 0]$. Conversely, we use the term *unreliability* for the probability that the system has failed.

If we denote the failed state of the top event as *T.Failed*, the unreliability corresponds to the formula $\mathbb{P}[x \leq t]\{\diamond T.Failed\}$ (in UPPAAL's query language, "`Pr[x <= t] (<> T.failed)`").

Availability is the expected fraction of time in a given time window that the system is functioning (equivalently, the steady-state probability that the system is functioning). Formally, we say $A_F(t) = \mathbb{E}\left[\frac{1}{t}\int_0^t X_F(x)\mathrm{d}x\right]$.

To compute the expected fraction of time the system is up, we introduce an auxiliary clock a that is stopped, but not reset, while the top event is in the failed location (and resumed when a repair returns the top event to the functioning location). The availability within time t can then be expressed as

the UPPAAL query "`E[x <= t, N] (max: a / t)`" where `N` is the number of simulation runs desired.

Expected number of failures denotes the expected number of times the top event occurs within a given time bound. Formally, we define $I_F(t)$ to be the indicator function that is a Dirac delta every moment the system fails, i.e.:

$$I_F(t) = \begin{cases} \delta & \text{if } X_F(t) = 1 \wedge \lim_{\epsilon \downarrow 0} X_F(t-\epsilon) = 0 \\ 0 & \text{otherwise} \end{cases}$$

The expected number of failures before time t is then $ENF_F(t) = \int_0^t I_F(x)dx$.

To measure this value, we introduce a variable n that is incremented every time the top event enters its failed state, and use the query "`E[x <= t, N] (max: n)`".

Expected cost denotes the expected cost incurred within a given time frame. While not very useful for FTs without maintenance, costs are very useful when comparing different maintenance strategies. Typically costs are incurred either on a per-event basis, e.g. a fixed cost to replace a broken component, or per unit time, e.g. lost productivity while a system is down. Formally, we write $C(t)$ for the cumulative cost incurred up to time t, hence the expected cost is either $\mathbb{E}[C(t)]$ for a fixed time window, or $\frac{1}{t}\mathbb{E}[C(t)]$ for the average cost per unit time.

The STA for an FMT tracks costs in several variables: One for total cost, others for the total costs of inspections, total costs of failures, etc. To find the expected total cost of the system, we use the query "`E[x <= t, N] (max: C_total)`". The other costs can be found by replacing `C_total` by `C_insp`, `C_failure`, etc.

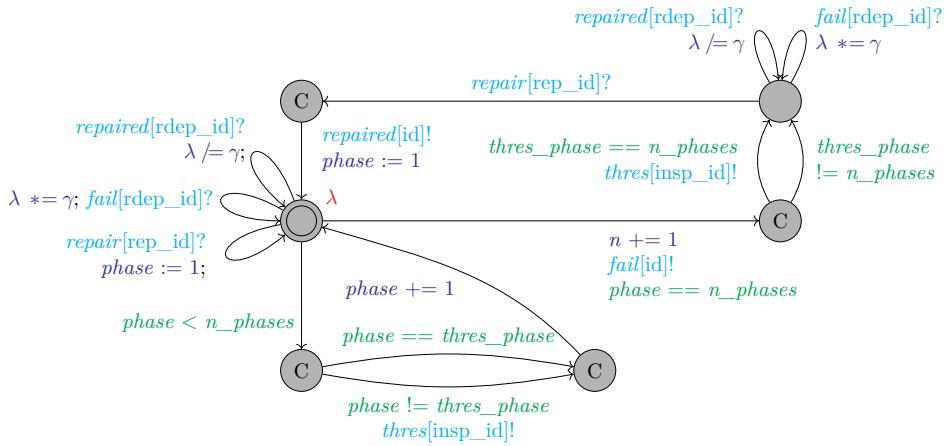


Figure 5.10: **STA of a basic event with one rate-dependency** and a failure time governed by a (*n_phases*, λ)-Erlang distribution, with a threshold for the inspection at phase *thres_phase*. The counter *phase* denotes the current phase, and is incremented according to exit rate of the initial location. If the current phase is equal to the threshold phase, a signal *thres*[insp_id] is sent to the listening IM. When the current phase equals the number of phases *n_phases* in the distribution, the STA emits a signal *fail*[id] to all listening gates, possibly emits the threshold signal, and waits for a signal *repair*[rep_id] from the RM. When this repair signal is received, the STA emits a signal *repaired*[id] to any listening gates, reset the current phase to 1, and returns to the initial location. The signal *fail*[fdep_id] triggers an acceleration of the degradation due the the failure of an FDEP trigger, and *repaired*[fdep_id] return the rate to normal.

5.5.2 Unified analysis via model-driven engineering

To automate the analysis of FMTs as well as other formalisms of fault trees, we apply techniques from model-driven engineering (MDE). MDE is a software engineering approach that uses models as first-class citizens, providing structured ways to describe such models (using *metamodels*) and special transformation languages to perform *model transformations*, converting and combining models into other models.

Using MDE, we allow fault trees to be converted between several tools, we support combinations of features from different fault tree formalisms as well as from attack trees (a similar formalism in security), and we provide model transformations allowing FTs and FMTs to be analyzed in UPPAAL.

Our approach uses the metamodels and model transformations for (attack-) fault trees described in [RSSR17], extended with a new metamodel describing the maintenance policies of FMTs. We also extend the model transformations to transform an FT with a maintenance policy (together forming an FMT) into a model that can be analyzed in UPPAAL.

Fault tree metamodel. To define models of FTs — or of any other domain — we need to specify the language of such models. In MDE, this language is specified using a *metamodel*, which captures the concepts and behaviour of fault trees and defines the permitted structure to which its models must adhere. Following standard terminology, we say that a model *is an instance of* its metamodel.

Figure 5.11 graphically shows the metamodel of an (attack-)fault tree in a UML-like diagram. The metamodel is divided into a structure part on the left, describing the basic events and gates that make up a fault tree, and a values part on the right, describing how attributes such as failure rates are attached to the basic events.

The root of a model of an FT is the `Tree` class, shown at the top of the figure. The containment relation to `Node` shows that a tree has one or more `Node`s (describing elements of the tree), while the reference relation shows that one of these `Node`s is designated the root of the tree.

The `Node` class is at the heart of the structure metamodel, modeling a basic event or gate. Each `Node` has several attributes: • *id* providing a unique identifier for the node, • *label* providing a human-readable description, • *nature* describing whether the element is a deliberate attack (from attack trees) or a randomly-occurring fault (from fault trees), and • *role* specifying whether the event contributes to the system failure or counteracts it (a *defense* node in attack trees).

The `Connector` describes what type of gate the element is. A basic event has no connector, while a gate has a connector of its type (AND, OR, etc.).

On the right of the diagram is the attributes metamodel, attaching concrete values to elements of the fault tree. Each `Attribute` is attached to a `Node`, and

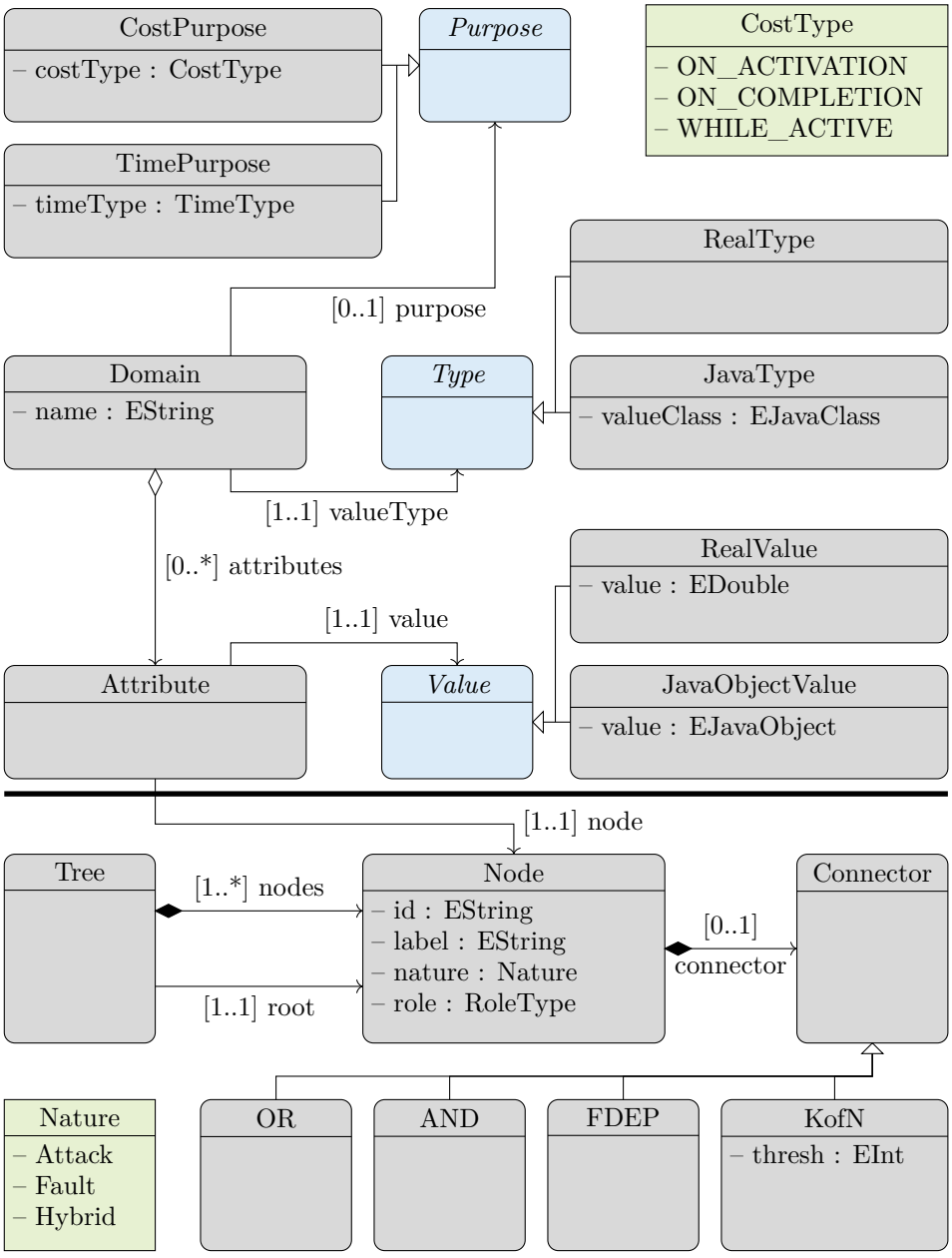


Figure 5.11: Partial metamodel of an (attack)-fault tree

decorates that node with a `Value`. The type of this value is given by a `Domain`, e.g., a domain of type `RealType` always has attributes with values of the class `RealValue`. Domains also describe what the attributes represent, using the `Purpose` class. For example, a domain describing the rate of the exponential distribution of the time to failure of a basic event will have a `TimePurpose` with type `EXPONENTIAL`.

Maintenance metamodel. To add maintenance to a fault tree, and thus create an FMT, we developed the metamodel shown in Figure 5.12. At the top right, the `MaintenancePolicy` class is shown. Such a policy consists of a set of `Modules`, which perform inspections and/or repairs. Each module has a `Condition` it needs to activate, referring to a time delay (`DelayCondition`), degradation state of an FT node (`NodeCondition`), or activation by another module (`ActivationCondition`). Boolean combinations of conditions can be made using the `CombiningCondition`.

Once a module is activated, it has an `Effect`. These are either triggers of other modules (via `TriggerEffect`s) or repairs of FT nodes (via `RepairEffect`s). Repairs and inspections refer to particular states of a node using the `NodeState` class, which can be a `LinearNodeState` for nodes with linear degradation (e.g., nodes with Erlang-distributed failure times) or model-dependent descriptions such as `PhaseTypeState` for nodes described by phase-type models with discrete states.

We note that all FMT-specific elements are contained in this maintenance metamodel. A major benefit of this approach is that it is possible to take fault trees developed for existing tools (e.g., DFTCalc [ABvdB+13]) that can already be converted into the fault tree metamodel [RSSR17], and add maintenance as a separate model.

Translation to UPPAAL. Once an F(M)T has been constructed following the metamodels above, *model transformations* can be applied to convert it to other types of models. Translations have already been developed to convert attack trees between various analysis tools [KSR+18], as well as from (attack-)fault trees to UPPAAL [SYR+17]. We extend the latter translation to include maintenance.

A model transformation follows the general structure shown in Figure 5.13. We take a model that is an instance of a particular metamodel (here, a combination of the attack-fault tree and maintenance metamodels), use a transformation engine to apply transformation in a transformation language (we use the Epsilon Transformation Language [KPP08]), and obtain an instance of the target metamodel (here, the UPPAAL metamodel [SYR+17]).

A snippet of the model transformation is shown in Figure 5.14. Here, we see that we first define a rule to transform a `Tree` from the attack-fault tree metamodel (`AFT`) to a model of a network of timed automata (`NTA`) of the UPPAAL metamodel. This rule creates various UPPAAL-related objects, and then iterates over the `Nodes` of the tree. Each node is converted using the built-in `equivalent()` function, which automatically selects the correct transformation rule for each node type.

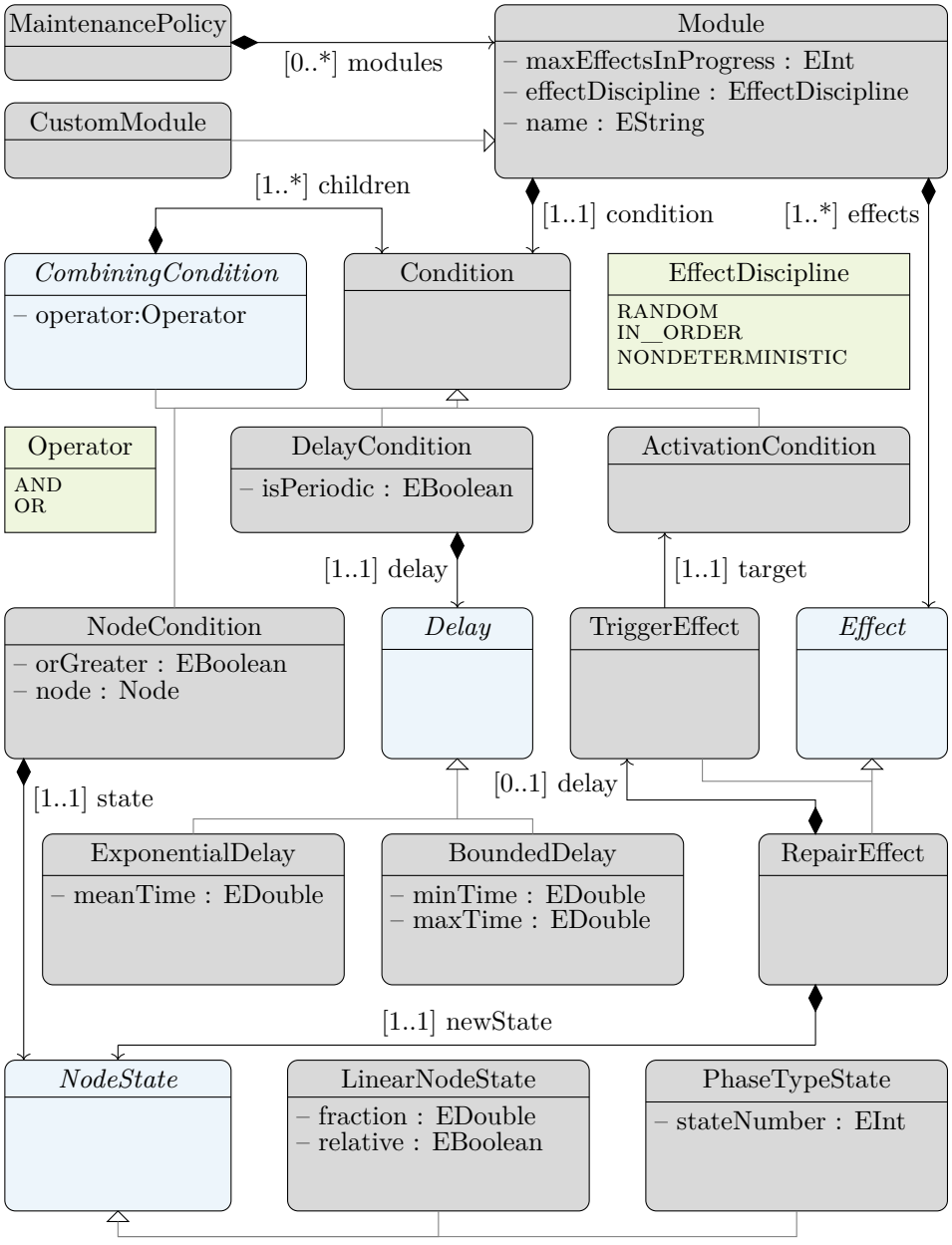


Figure 5.12: Metamodel of the FMT maintenance model.

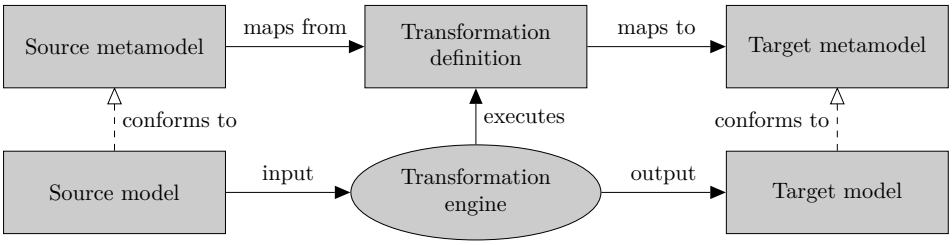


Figure 5.13: The concept of a *model transformation.*

The converted nodes are then added to the output model. At the bottom of the code is the start of the definition of the rule converting AND-gates.

Conclusion. Using the MDE approach, we have developed the ATTop tool translating a fault maintenance tree to an UPPAAL model for analysis [1]. Thanks to the reusability of metamodels, we were able to adapt an existing metamodel for attack-fault trees and extend it to cover maintenance. By keeping the maintenance aspects separate, we can reuse large parts of the existing transformations from various tools to the fault tree metamodel, allowing the user to add maintenance to an existing fault tree without changing the existing FT. We also reuse large parts of the transformation to UPPAAL, adding support for maintenance where applicable.

Our newly developed maintenance metamodel is extensible, just like the original metamodel for fault trees, allowing easy adaptation to other types of maintenance modules that may be developed.

5.6 Conclusion

In this chapter we have presented fault maintenance trees (FMTs). FMTs extend traditional fault trees by including advanced maintenance concepts, including gradual degradation and wear of components, dependencies of degradation rates on the failures of other components, and periodic inspections and repair to reverse degradation and failures.

Quantitative analysis of FMTs is performed by statistical model checking, a state-of-the art Monte Carlo simulation technique. FMTs are translated into networks of priced stochastic timed automata and analysed using the Uppaal-SMC tool to obtain various metrics such as system reliability, expected number of failures, and expected cost.

[1] ATTop, including the metamodels and model transformations described here, can be found at `https://github.com/utwente-fmt/attop/tree/maintenance`

```
rule Base transform t : AFT!Tree to out : Uppaal!NTA {
    out.systemDeclarations = new Uppaal!SystemDeclarations();
    out.systemDeclarations.system = new Uppaal!System();
    var iList = new Uppaal!InstantiationList();
    out.systemDeclarations.system.instantiationList.add(iList);
    for (node : AFT!Node in t.Nodes) {
        var converted = node.equivalent();
        if (converted <> null) {
            out.template.add(converted.get(0));
            out.systemDeclarations.declaration.add(converted.get(1));
            iList.template.add(converted.get(1).declaredTemplate);
        }
    }
    out.addTopLevel(t.Root);
}
rule andGate transform node : AFT!Node to ret : List {
    guard : node.nodeType.isKindOf(AFT!AND)
...
```

Figure 5.14: Snippet of the translation from the attack-fault tree metamodel (AFT) to the UPPAAL metamodel.

By applying concepts from model-driven engineering, the analysis tool provides automated translation from FMTs to Uppaal-SMC. This tool supports fault trees developed for existing tools, and the use of metamodelling provides extensibility to more advances maintenance concepts that may yet be developed.

Discussion. The models and analysis presented here can provide important insights into the dependability of a maintained system, including the reliability and cost associated with different maintenance policies. This analysis paves the way for reliability-centered maintenance, allowing practitioners to optimize the maintenance plan to maximize system reliability and minimize total costs.

FMTs can provide an important stepping stone in the development of models for predictive maintenance [Mob02]. As FMTs already contain models for the degradation of components, these can be used to predict the future behaviour of the system if the current degradation state can be monitored. Slightly modified maintenance modules can decide when to perform future inspections and repairs based on such states.

FMTs can already be practically applied in the railway industry, as will be demonstrated by the two case studies in Part III.

This chapter has presented FMTs with the gates of static fault trees. They

could be extended to cover dynamic gates, with two (relatively minor) challenges:

- The failure of dynamic gates depends not only on which children are currently down, but also on the order in which they have failed. The semantics of dynamic gates thus need to be extended to cover situations where some or all children are repaired after failing. Some work has been done in this direction [MCC+14, GSS15], but there is currently no agreement on the semantics. Note that there is already disagreement on some details of the semantics of non-repairable dynamic fault trees [JGKS16], which will likely affect the development of repairable semantics.
- Dynamic fault trees can, in some formalisms [JGKS16], exhibit nondeterminism. The analysis tool used in this chapter (Uppaal-SMC) reduces such nondeterminism to a probabilistic choice [BDL+12], while some analysis tools for non-repairable DFTs (e.g., DFTCalc[ABvdB+13]) leave nondeterminism unresolved and compute upper and lower bounds for the computed metric. To achieve similar behaviour for repairable DFTs using statistical model checking, it may be necessary to use different analysis techniques (e.g., [HHH14]).

Chapter 6

Analysis via importance sampling

As described in the previous chapters, (dynamic) fault tree analysis is an industry-standard technique for reliability analysis. By describing the basic failure modes of the system, and how those failure modes interact to cause larger failures, various qualitative and quantitative metrics can be computed. Furthermore, Chapter 5 described fault maintenance trees, extending fault trees to perform quantitative analysis of systems with complex maintenance and repair policies.

Various quantitative analysis techniques for dynamic fault trees were described in Section 3.3. While these techniques are useful for many practical systems, they suffer from the problem of a state space explosion. In order to compute the metric of interest (e.g., *reliability* or *availability*), most techniques construct a model describing every possible sequence of failures (and repairs, if applicable), possibly with some minimisation techniques [KS17]. As the number of these sequences grows exponentially in the number of basic events, such models grow too large to fit in computer memory for larger practical systems. While various reduction techniques can be applied to ameliorate this problem [KS17], scalability to large systems remains an issue.

A standard approach to overcome this problem is to use Monte Carlo (MC) simulation techniques, which do not need to construct the entire state space. The memory consumption of such techniques is very low, allowing the analysis of very large systems. Chapter 5 describes how such a simulation approach is used in the analysis of fault maintenance trees.

MC simulation solves the problem of memory consumption, at a trade-off for computation time. This introduces a new potential problem: obtaining accurate estimates of the dependability of highly reliable system can require large numbers of simulation runs. The problem, here, is that failures of reliable systems are generally rare. In the automotive industry, for example, the probability of safety-critical failures (Automotive Safety Integrity Level D) may be limited to around 10^{-8} per hour [ISO11, GJK+17b]. For such low probabilities, a very large number of simulation runs is needed to obtain reasonably tight statistical confidence intervals.

To overcome this difficulty, rare event simulation techniques have been developed

since the 1950s [KH51]. Such techniques alter the model being simulated or the simulator to make simulation runs more likely to reach states of interest. Afterwards, the altered runs are used to estimate the probability of interest in the original model. In this way, statistically justified results can be obtained with far fewer simulations than would otherwise be required.

The most rare event simulation technique most applicable to DFTs, importance sampling, crucially relies on finding a good *change of measure* (CoM) to reduce the variance of the estimated probability. A CoM can reduce the number of simulations required for good estimate by several orders of magnitude. Conversely, a poor CoM can actually increase the number of simulations required, or even lead to biased results.

This chapter presents a novel approach to analyse DFTs with maintenance through rare event simulation. We adapt the recently-developed Path-ZVA algorithm [RdBSJ18, Rei13] to the setting of repairable DFTs. The algorithm provides an automated framework for importance sampling of Markovian systems with a provably good CoM, allowing the estimation of the probabilities of rare events with minimal user intervention.

To obtain a Markovian model of a DFT, we reuse the compositional semantics introduced in [BCS10], providing behaviour compatible with existing tools [ABvdB+13]. Repairs for basic events follow the inspection and repair policies developed for FMTs (as described in Chapter 5). As the Path-ZVA algorithm relies on the Markovian nature of the system being modelled, we treat only the fully stochastic subset of DFTs (i.e., where children of PAND and SPARE gates are fully independent subtrees).

We show that this approach, as implemented in our tool FTRES, is able to analyse systems too large for existing Markov-chain-based techniques, while obtaining confidence intervals much tighter than those obtained by traditional MC simulation. Three case studies, one from the railway industry and two from the DFT literature, show that this technique can be fruitfully applied in practice.

Approach. Our overall approach to rare event simulation for FMTs relies on an on-the-fly conversion of the FMT into an input/output interactive Markov chain (I/O-IMC). This I/O-IMC is a Markovian model describing the behaviour of the FMT. Given the I/O-IMC, we apply the Path-ZVA algorithm for importance sampling to alter its transition rates to make the top level event of the FMT more probable. We then sample simulation traces from this model, measuring the unavailability of the FMT, and apply a correction for the adjusted transition rates.

More concretely, we take the following steps:

1. Use the DFTCalc tool to compute I/O-IMCs for all elements of the FMT. Traditionally, one would compute the composition of these elements to obtain one I/O-IMC describing the behaviour of the FMT. Our approach computes

the necessary states of the composition on-the-fly in the following steps.

2. Perform a breadth-first search of the Markovianized composition (explained in Sections 6.2.2 and 6.2.3) of these elements to identify the most likely paths that reach a failed state.

3. Apply the Path-ZVA algorithm (explained in Section 6.1) to adjust the transition probabilities along these most likely paths, and keep track of how the probabilities were altered.

4. Sample traces of the adjusted model, storing the how much time of each trace was spend in unavailable (i.e., failed) states, and how much the total probability of each trace was altered by step 3.

5. Average the unavailabilities of the traces, correcting for the altered probability of each trace.

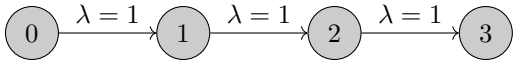
Origin of this chapter The analysis method discussed in this chapter was first presented in:

- Enno Ruijters, Daniël Reijsbergen, Pieter-Tjerk de Boer, and Mariëlle Stoelinga. "Rare event simulation for dynamic fault trees". In *Proceedings of the International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 10488 of *Lecture Notes on Computer Science*, pages 20–35. Springer, September 2017. DOI: 10.1007/978-3-319-66266-4_2, ISBN: 978-3-319-66265-7.

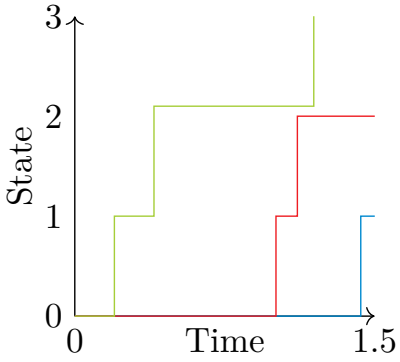
Organisation of the chapter We begin the chapter with an introduction to rare event simulation in Section 6.1, and describe the repairable DFTs that can be analysed using our approach in Section 6.2. Section 6.3 explains how we combine these elements to analyse repairable DFTs using importance sampling. Section 6.4 describes our case studies and their results. Finally, Section 6.5 gives our conclusions.

6.1 Rare Event Simulation

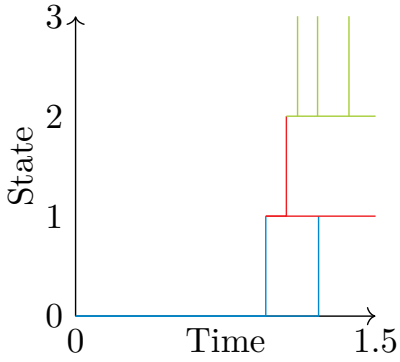
Monte Carlo simulation is a commonly applied technique to estimate quantitative metrics in cases where exact solutions are impractical to compute, or where no methods are known to compute them [Fis96]. A common disadvantage of such techniques is that many events of practical interest occur only very rarely. In such cases, accurately estimating the probability of the event is difficult: unless a very large number of simulation runs are performed, the event may not be observed in



(a) System being simulated



(b) Without importance splitting (only three sample runs shown)



(c) With importance splitting

Figure 6.1: Example of importance splitting.

any of the runs, or otherwise may not be seen frequently enough to draw statistically sound conclusions.

Reliability engineering is precisely a field where such rare events are of primary interest: A highly reliable system, by definition, only fails rarely. For example, the European Rail Traffic Management System specifies that the probability of a transmitted message being corrupted must be less than $6.8 \cdot 10^{-9}$ [Gro98]. Proving that a model meets this level of reliability with 95% confidence requires at least $4.4 \cdot 10^{8}$ simulations (in the ideal case where no failure is observed within those runs).

To allow simulation-based estimation of such low probabilities, rare-event simulation techniques have been developed. These techniques make the event of interest occur more frequently, either by modifying the system being studied or the way simulation runs are sampled, and afterwards compensate for the artificially increased probability.

The main approaches to rare event simulation can be divided into two categories: *importance splitting* and *importance sampling*. Both of these were developed in the early days of computing [KH51].

Importance splitting. Splitting modifies the simulation engine to select those sample runs that are likely to reach the event of interest. In particular, the engine

begins by simulating runs as usual, and tracks how 'close' each run gets to the interesting event. This 'closeness' is measured by the *importance* of the current simulation state at any given time. A good measure of importance is one where the more important a state is, the more likely it is to observe the interesting event after visiting the state.

To drive the simulation runs to the rare event, additional simulation runs are started from certain reached states of high importance, with the expectation that these new runs will reach yet more important states. Eventually, some of the simulation runs reach the event of interest. Afterwards, by combining the probabilities of reaching different levels of importance from lower levels, one can obtain the probability of reaching the rare event from the initial (least important) state.

Example 25 *Consider the system shown in Figure 6.1a, and the property of interest 'Probability of reaching state 3 within 1.5 time units'. The actual probability of this event is approximately* 0.19.

Figure 6.1b shows the first three sample runs out of 12, each run starting at time and state 0 and running until either reaching state 3 or time 1.5. Out of the full 12 runs, only one reached state 3, giving a point estimate of 0.083, and a 95% confidence interval of the probability as $[0.002, 0.384]$. *Clearly, this is not very useful.*

Figure 6.1c shows an example of the results of an importance splitting technique: We first draw four runs, stopping each run when it reaches time 1.5 or state 1 (the blue lines). Two of the runs reached state 1. Next, we start four new simulations, starting from state 1 and the times when the earlier simulations reached state 1. We now stop the simulations when they reach state 2 (or time 1.5). This time, one run reached state 2. Again, we start four independent simulations from state 2 with the starting time the earlier run reached it. We notice that three of these samples reached state 3 before the time limit.

We can now estimate the total probability as

$$\begin{aligned}\mathbb{P}(\textit{reach state } 3) =& \mathbb{P}(\textit{reach state } 1) && (1)\\ &\times \mathbb{P}(\textit{reach state } 2 \textit{ from state } 1) && (2)\\ &\times \mathbb{P}(\textit{reach state } 3 \textit{ from state } 2) && (3)\end{aligned}$$

In this manner, we obtain a point estimate of $\frac{2}{4} \times \frac{1}{4} \times \frac{3}{4} = 0.09375$. *Due to the tiny sample size of* $N = 4$ *to keep this example short, standard approximations for the confidence interval cannot be applied. For larger sample sizes, accurate confidence intervals can be obtained.*

Many different techniques for importance splitting exist with different procedures for determining importances, and deciding how many additional simulation runs to start at which states. For an overview, we refer the reader to [LLLT09].

Importance splitting is most useful for systems where the rare event is reached after a large number of transitions, each with a moderately low probability. Such systems provide many opportunities for restarting the simulation runs, getting incrementally closer to the target state. In the context of DFTs, however, the target (system failure) is usually reached after only a few transition of very low probability, namely the failures of a few highly reliable components.

Importance sampling. For the aforementioned reason, our approach does not use importance splitting, but rather importance *sampling*. A survey of this technique can be found in [Hei95]. The intuition behind importance sampling is that the event of interest is made more probable by altering the probability distributions of the system being simulated. When drawing a simulation run, the simulator also records the *likelihood ratio* of the sampled values, defined as the probability of the current run in the original system divided by its probability in the modified system.

In MC simulation without importance sampling, N simulation runs are performed, and the i'th simulation run is recorded as an outcome I_i which is 1 if the event of interest was reached, and 0 otherwise. The probability of reaching the event is then estimated as:

$$\hat{\gamma}_{orig} = \frac{1}{N} \sum_{i=1}^{N} I_i$$

In importance sampling, the simulator also tracks the likelihood ratio L_i of the run, defined as the probability of drawing that run in the original system divided by the probability of the run in the modified system.

Details of the computation of L_i depend on the system being simulated. For example, we consider the system in Figure 6.3. Suppose we observe the path $I \to B$. Now, in the original system, we have $\mathbb{P}_{orig}(I \to B) = 0.01$, while in the modified system we have $\mathbb{P}_{IS}(I \to B) = 0.1$. We thus have the likelihood ratio $L_{I \to B} = \frac{0.01}{0.1} = 0.1$. In the general case, if the i'th simulation run observes trace π, we have $L_i = \frac{\mathbb{P}_{orig}(\pi)}{\mathbb{P}_{IS}(\pi)}$.

Example 26 *Figure 6.2 shows how a likelihood ratio can be obtained by considering fragments of a path in isolation. Our total path consists of some path segment π_{pre} with likelihood ratio L_{prefix}, ending in state s_i, and taking the blue transition to a path segment $s_j \pi_{suf}$ with likelihood ratio L_{suffix}.*

We can write the probability of the entire path π as $\mathbb{P}(\pi = \pi_{pre} s_i s_j \pi_{suf})$

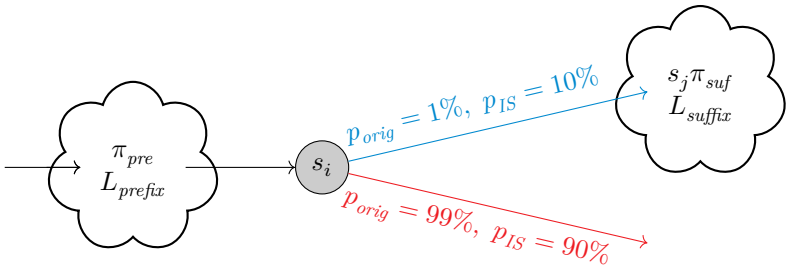


Figure 6.2: **Example of computing a likelihood ratio in a small part of a trace.** The total likelihood ratio when taking the blue transition is $L = L_{prefix} \times \frac{1}{10} \times L_{suffix}$.

Decomposing this into segments, we obtain (conditioning on prefixes of paths):

$$\begin{aligned}\mathbb{P}(\pi) =& \mathbb{P}(\pi_{pre}s_is_j\pi_{suf}) \\ & \mathbb{P}(\pi_{pre}s_is_j\pi_{suf} \mid \pi_{pre}s_is_j) \times \mathbb{P}(\pi_{pre}s_is_j) \\ & \mathbb{P}(\pi_{pre}s_is_j\pi_{suf} \mid \pi_{pre}s_is_j) \times \mathbb{P}(\pi_{pre}s_is_j \mid \pi_{pre}s_i) \times \mathbb{P}(\pi_{pre}s_i)\end{aligned}$$

We can now compute the total likelihood ratio:

$$\begin{aligned}L_{total} = \frac{\mathbb{P}_{orig}(\pi)}{\mathbb{P}_{IS}(\pi)} &= \frac{\mathbb{P}_{orig}(\pi_{pre}s_is_j\pi_{suf} \mid \pi_{pre}s_is_j)\mathbb{P}_{orig}(\pi_{pre}s_is_j \mid \pi_{pre}s_i)\mathbb{P}_{orig}(\pi_{pre}s_i)}{\mathbb{P}_{IS}(\pi_{pre}s_is_j\pi_{suf} \mid \pi_{pre}s_is_j)\mathbb{P}_{IS}(\pi_{pre}s_is_j \mid \pi_{pre}s_i)\mathbb{P}_{IS}(\pi_{pre}s_i)} \\ &= \frac{\mathbb{P}_{orig}(\pi_{pre}s_is_j\pi_{suf} \mid \pi_{pre}s_is_j)\mathbb{P}_{orig}(\pi_{pre}s_is_j \mid \pi_{pre}s_i)}{\mathbb{P}_{IS}(\pi_{pre}s_is_j\pi_{suf} \mid \pi_{pre}s_is_j)\mathbb{P}_{IS}(\pi_{pre}s_is_j \mid \pi_{pre}s_i)} L_{prefix} \\ &= \frac{\mathbb{P}_{orig}(\pi_{pre}s_is_j\pi_{suf} \mid \pi_{pre}s_is_j)}{\mathbb{P}_{IS}(\pi_{pre}s_is_j\pi_{suf} \mid \pi_{pre}s_is_j)} \frac{1}{10} L_{prefix} \\ &= L_{suffix}\frac{1}{10}L_{prefix}\end{aligned}$$

Thus, we can conclude that the likelihood ratio of a path is the product of the likelihood ratios of the individual transitions along that path. This allows us to easily determine the likelihood ratio of a given sample run, by tracking the product of the likelihood ratios of all transitions up to the current point.

Having obtained the likelihood ratios L_i, the estimator of the probability of interest is then:

$$\hat{\gamma}_{IS} = \frac{1}{N}\sum_{i=1}^{N} I_i L_i.$$

In this way, if the rare event is reached on a run that was originally much less likely (very low L_i), it counts very little towards the probability estimate. In contrast, if the rare event is reached on a run with an unchanged probability ($L_i = 1$), its contribution to the estimate is also unchanged compared to normal MC simulation.

Example 27 *Figure 6.3a shows a discrete-time Markov chain which, from initial state I, has a 1% probability of reaching a bad state B, and a 99% probability of reaching the good state G. If one were to estimate the probability of reaching B by standard MC simulation with 100 runs, there is a 36% probability of not observing G at all. In the most likely case (1 observed instance), the 95% confidence interval for the probability is* $[0.0002, 0.0545]$*, which is very wide.*

If one makes the rare event 10 times as likely, as shown in Figure 6.3b, the same 100 simulations will observe far more runs reaching G. In the most likely case of observing 10 runs reaching G, a 95% confidence interval of the probability in the modified system is $[0.049, 0.176]$*. Compensating for the increased likelihood, one obtains a 95% confidence in the original system of* $[0.0049, 0.0176]$*, over four times as precise as the original estimate.*

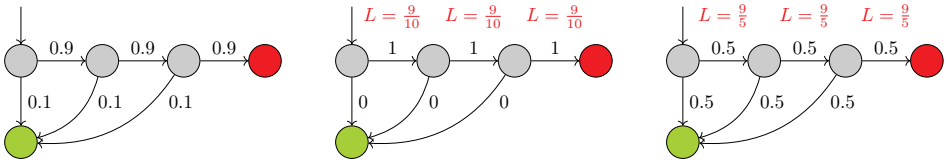
6.1.1 Change of Measure

While the general idea behind importance sampling is simple, making the interesting but rare event less rare (i.e., increasing its *probability measure*) and multiplying the observed probability by how much less rare it is, actually finding a good way of making this event more likely can be more involved. This process is called the *change of measure* (CoM).

In general, one wants to make transitions (in our setting, component failures)

(a) Original system (b) Modified system

Figure 6.3: **Example of a change of measure for importance sampling.** The event of interest is reaching state G. In the original system this event has a probability of 1%, a possible modification for importance sampling increases this probability to 10%, giving a likelihood ratio of $\frac{1\%}{10\%} = \frac{1}{10}$.



(a) Original system: Variance ≈ 0.198

(b) Zero-variance estimator: Variance $= 0$

(c) Worse estimator: Variance ≈ 3.72

Figure 6.4: **Examples of different changes of measure** and their effects variance of the estimator of the probability to reach the red state.

that bring the system closer to the goal (e.g., system failure) more likely, while transitions leading away from the goal (e.g., component repairs) less likely. However, choosing these transitions poorly can produce estimators with higher variance than standard MC simulation. For example, one could make the most reliable components more likely to fail. This could lead the simulator to find many runs in which these components fail, but such runs have low contributions (i.e., low likelihood ratios). The runs in which less reliable components fail, which are normally more probable, become even less likely, and thus poorly estimated. Particularly bad choices of CoM can even produce estimators that are biased or have infinite variance.

The 'holy grail' of importance sampling is the *zero-variance estimator* (ZVE) [KH51]. That is a system modified in such a way that the event of interest is always reached, and the likelihood ratio is in fact the probability of reaching the event in the original system P_o. When such an estimator is used, each simulation contributes $\frac{1}{N} I_i L_i = \frac{1}{N} 1 P_o$, and thus the estimated probability is a constant regardless of the number of simulations. Unfortunately, obtaining this zero-variance estimator requires knowledge of P_o which is the value being estimated to begin with. Therefore, any practical technique will, at best, approximate the ZVE [LT11].

Example 28 *Figure 6.4a shows a discrete-time Markov chain, in which we estimate the probability of reaching the red state. The actual probability is clearly* $0.9^3 = 0.729$.

Figure 6.4b shows a zero-variance estimator of this probability. Every sample run will reach the red state, and thus yield outcome $I_i = 1$. *Every sample run also has the same likelihood ratio, namely* $0.9^3 = 0.729$. *Thus, each simulation estimates the probability to be the true probability of 0.729. In this example, the ZVE is easy to construct, as there is only one path reaching the red state, so we simply force this to always be the path sampled.*

Figure 6.4c illustrates the downside of a poorly chosen change of measure: if we don't use our knowledge of the path to the target state, and simply make each transition equally likely, we end up reducing the likelihood of reaching the target.

We thus obtain a greater variance than in the original system.

6.1.2 The Path-ZVA Algorithm

Many different methods have been proposed to find a good change of measure. In our approach, we apply the Path-ZVA algorithm [RdBSJ18, Rei13]. This is an algorithm with provably good performance on a large class of Markovian models, making it particularly suited for simulation of DFTs. The algorithm also does not require the exploration of the entire state space, but only of those states on *dominant paths* (i.e., paths with the fewest low-probability transitions) to the target state(s).

Path-ZVA produces a CoM suitable for estimating the probabilities of events of the form "reaching set of states A (*goal states*), starting from state B (*initial state*), and before reaching a state in set C (*taboo states*)", where the system must frequently visit some states in C. In our setting, the goal states are those states in which the system has failed, while the initial state is the state in which the system is in perfect condition. The initial state is also the only taboo state. This means that we estimate the probability "System failure occurs before the system is repaired to a perfect state, starting from a perfect-condition system".

The CoM can also be used to estimate the fraction of time the system spends in the goal states, allowing us to compute the *unavailability* (average fraction of time that the system is down). Both for the time spent in A and the probability of reaching A, a point estimate and a confidence interval are returned.

Given these properties, Path-ZVA is very suitable for estimating the unavailability of a multi-component system, as is typically the case in DFTs, as long as

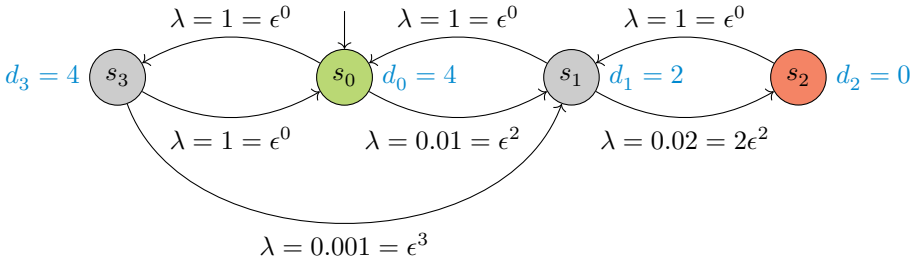


Figure 6.5: **Illustration of the Path-ZVA model.** We are interested in the probability of reaching the red state (the *goal state*), starting from the green state (the *initial state*), before returning to the green state (also the *taboo state*). We parameterize all transition rates with a rarity parameter $\epsilon = 0.1$. Distances computed by Path-ZVA importance sampling are written in blue.

the system is fully repairable (so the taboo/initial state C is frequently reached), and all failure and repair times can be described using a Markovian model.

The intuition of the Path-ZVA algorithm is that it first finds, for each state, the minimal distance from that state to the target. This distance is approximately the number of rare transitions that need to be taken before the target is reached. The algorithm then adjusts the transition rates according to the destination states' distances, so that states closer to the target become more like, and states further away from the target become less likely.

This model relies on the transition rates being specified using a *rarity parameter* ϵ. Each possible path to the event of interest consists of a number of transitions of the Markov chain, each of which has a rate of the form $r \cdot \epsilon^k$. The dominant paths are those paths in which the sum of the powers k of ϵ are smallest. In the limit of $\epsilon \to 0$, these paths dominate the total probability of reaching the target.

Example 29 *Figure 6.5 shows an example of how the distances are computed by Path-ZVA. The target is state s_2, which thus has a distance $d_2 = 0$. State s_1 can reach the target in one transition with rate $2\epsilon^2$, i.e., rarity 2, and thus has distance $d_1 = 2$. The most likely path from s_0 to the target is via s_1, and both transition in the path $s_0 s_1 s_2$ have rarity 2, so the distance is $d_0 = 2 + 2 = 4$. Finally, The most probable path from s_3 is the path $s_3 s_0 s_1 s_2$, giving distance $d_3 = 0 + 2 + 2 = 4$. Note that the path $s_3 s_1 s_2$ is shorter in number of transitions, but has a higher total rarity (5), and is therefore not the most likely path in the limit of $\epsilon \downarrow 0$.*

Note that existing work assumes that this parameterization is given, and we are unaware of any systematic approach to converting models with known rates to ϵ-parameterized versions. In this chapter, we fix a value of $\epsilon < 1$ (typically $\epsilon = 0.1$), and compute k and r for each transition such that $1 < r < \frac{1}{\epsilon}$.

Once the dominant paths have been found, the states on these paths have their outgoing transition probabilities weighted by the rarity their destination states. For example, if a state s_i has two transitions with probability $\frac{1}{2}$ to destinations with distances $d_k = 2$ and $d_l = 3$, we compute the transition weights $w_{ik} = \frac{1}{2}\epsilon^2$ and $w_{id} = \frac{1}{2}\epsilon^3$. We then normalize these weights to obtain probabilities, giving $p_{ik}^{IS} = \frac{\epsilon^2}{\epsilon^2+\epsilon^3}$ and $p_{il}^{IS} = \frac{\epsilon^3}{\epsilon^2+\epsilon^3}$ (for $\epsilon = 0.1$, this means $p_{ik}^{IS} \approx 0.9$ and $p_{il}^{IS} \approx 0.1$). For the continuous-time setting, these new probabilities are multiplied by the original exit rate of the state, so that the total exit rate is unchanged by the Path-ZVA algorithm.

As an optimization, once we know the distance d_0 from the initial state to the target, we know that all states further than d_0 from the target or the initial state will never be on a dominant path. We can leave the transition rates from these states unchanged as they will have only a very small contribution to the total probability (a vanishing contribution in the limit of $\epsilon \downarrow 0$). This means that

the distance-finding algorithm only needs to explore a subset of the state space (typically several orders of magnitude smaller than the full state space) containing the potentially dominant paths. More details can be found in [RdBSJ18].

Thus, Path-ZVA takes a Markov chain with initial state s_0 and target state s_T, with the transition probability from state s_i to state s_j given by $p_{ij}\epsilon^{k_{ij}}$. We now perform the following procedure:

1. Perform a breadth-first search, starting in s_0, to find a path $s_{t_0}s_{t_1}\cdots s_{t_{n-1}}s_{t_n}$ with $t_0 = 0$ and $t_n = T$, $\forall_i : p_{t_i t_j} > 0$, and with minimal distance $d_0 = \sum_{i=0}^{n-1} = k_{t_i,t_{i+1}}$.

2. Decorate every state s_i with its distance to the initial state d_i^I.

3. Store the states $\Lambda = \{s_i | d_i^I \leq d_0\}$.

4. Store the states $\Gamma = \{s_j \notin \Lambda | \exists s_i \in \Lambda : p_{ij} > 0\}$ that can be reached in one transition from Λ.

5. Using a backward search from s_T, decorate every state $s_i \in \Lambda \cup \Gamma$ with its minimal distance to the target d_i.

6. For every state $s_i \in \Lambda$, compute the new outgoing transition probabilities:

   (a) For every state s_j, compute $w_{ij} = p_{ij}\epsilon^{k_{ij}}\epsilon^{d_j}$.

   (b) Normalize the new transition probabilities, such that for every state s_j we let $p'_{ij} = w_{ij} / \sum_k w_{ik}$.

   (c) Compute the likelihood ratio of the transition $L_{ij} = \frac{p_{ij}\epsilon^{k_{ij}}}{p'_{ij}}$.

7. For every state $s_i \notin \Lambda$, leave the transition probabilities unchanged (i.e., $\forall_j p'_{ij} = p_{ij}\epsilon^k_{ij}$), giving likelihood ratio $L_{ij} = 1$.

Under mild conditions, it can be proven that the method leads to estimators having the desirable property of Bounded Relative Error [Rei13]. This means that as the event of interest gets rarer due to rates in the model being chosen smaller, the estimator's confidence interval width shrinks proportionally to the probability of interest, making its *relative error* bounded (cf. [LBTG10]). That is, if we have a model parameterized by a rarity factor ϵ, we denote by $\gamma(\epsilon)$ the probability of interest of the model, and by $\sigma_{IS}(\epsilon)$ the standard deviation of the estimated probability obtained using importance sampling (using Path-ZVA), then we have that $\lim_{\epsilon \downarrow 0} \frac{\sigma_{IS}(\epsilon)}{\gamma(\epsilon)} < \infty$. This is not the case for standard MC simulation without rare event simulation.

6.2 Fault Maintenance Trees

As explained in Chapter 2, fault tree analysis (FTA) is a widely-used technique for dependability analysis, and one of the industry standards for estimating the reliability of safety-critical systems [ISO11]. By decomposing the possible failures of the system into different kinds of (partial) failures and further into elementary failure causes, the failure probabilities of the system as a whole can be computed. Such quantitative analysis can compute measures such as the system *reliability* and *availability*.

Standard, also called *static*, fault trees combine different failure modes using boolean connectors, namely the AND-, OR-, and VOT(k)-gates, failing when all, any, or at least k of their children fail, respectively. The elementary failure causes (called *basic events*) are usually given as either probabilities describing the odds of failing within a fixed time window, or with exponential failure rates describing the probability of failure before any given time. For repairable systems, repair times in standard fault trees are usually also specified by exponential rates.

Figure 6.6 shows an example of such a fault tree. It models a case study from [GSS15], studying part of the interlocking system of a railway corridor. The system consists of relay and high-voltage cabinets, redundantly implemented such that a single cabinet of either type can fail without causing a system failure. In the figure, the event of interest (multiple cabinets failing) is described by the OR-gate at the top. Its children are two VOT(2)-gates and an AND-gate. The leaves of the tree

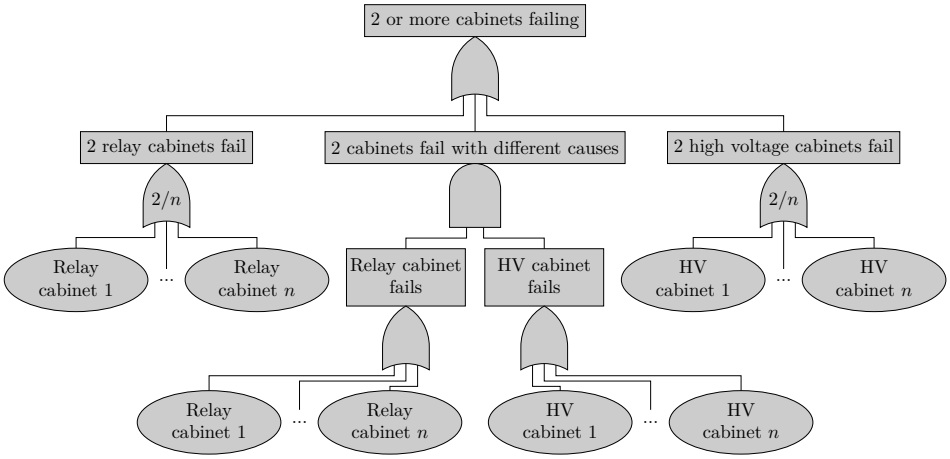


Figure 6.6: **Example fault tree of the relay cabinet case study.** Due to redundancy, the system can survive the failure of any single cabinet, however two failures cause system unavailability. The number of cabinets varies, and is indicated by n.

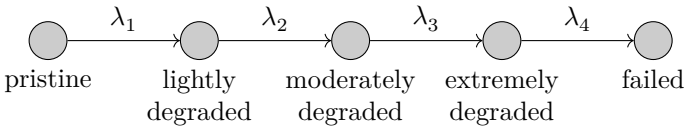


Figure 6.7: Basic event with multiple degradation phases.

are the BEs describing the failures of individual relay and high voltage cabinets.

6.2.1 Dynamic and Repairable Fault Trees

Chapter 3 explains *dynamic fault trees* (DFTs) [DBB90], which extend fault trees with additional gates to model common patterns in designs of highly reliable systems. In particular, DFTs add gates for modelling spare component with different failure rates when they are not being used, functional dependencies where the failures of a component or subsystem is the cause of other components failing (e.g., a failed power supply leading to inoperative pumps), and the priority-AND gate used when the ordering in which failures occur is relevant.

Chapter 5 describes how fault trees are further extended to *fault maintenance trees* (FMTs) with support for advanced maintenance and repair policies. Such maintenance is usually required to prevent failures when possible, and the repairs correct what failures still occur. Since this maintenance is essential for ensuring the dependability of the system, it is important to take the maintenance policy into account when performing reliability analysis. Similar maintenance policies have been developed for DFTs in [GKS+14].

A key component of the implementation of maintenance in (D)FTs is the non-exponential basic event. The traditionally used exponential distribution is memoryless (i.e., the remaining time to failure is independent of how long the component has already been in operation), which does not accurately describe the behaviour of components subject to gradual wear. To support wear and maintenance modelling, BEs in FMTs can progress through multiple phases of degradation, as depicted in Figure 6.7. Inspections can periodically check whether some BEs have degraded beyond some threshold phase, and repairs can return them to their undegraded phase if they have. periodic replacements simply return their BEs to their undegraded phase periodically.

6.2.2 Compositional Semantics

The analysis used in this chapter follows the compositional semantics in terms of input/output interactive Markov chains given in [BCS10], with subsequent extensions for maintainable systems [GKS+14]. This compositional approach

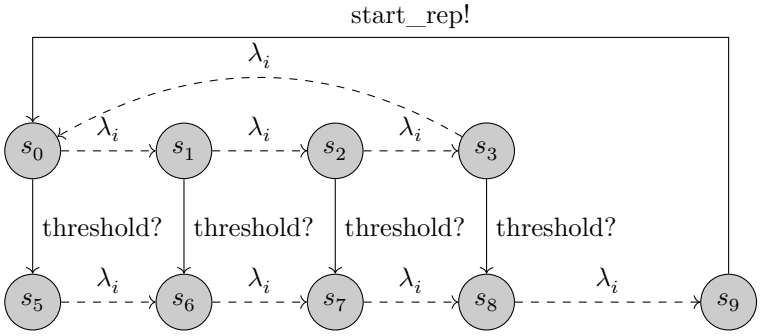


Figure 6.8: **Inspection module with Erlang-distributed time between inspections.** If no threshold signal is received, the model loops in the top row of states. When a threshold signal is received, the model moves to the bottom row, waits until the time an inspection is performed, signals that a repair is needed, and moves back to the initial state.

converts each element of the DFT (i.e., gate and basic event) to an I/O-IMC, and composes these models to obtain one large I/O-IMC for the entire DFT. Intermediate minimisation helps to keep the size of the state-space to a minimum, allowing the analysis of larger models.

The I/O-IMCs for the gates are the same as those introduced in Chapter 5 (e.g., Figure 5.4 on page 108 for the AND-gate). The I/O-IMCs for the inspection and repair modules also follow those in Chapter 5, except that the exact inspection and repair time are approximated using an Erlang distribution by a chain of exponential distributions. For example, Figure 6.8 shows a Erlang-approximated inspection module.

Input/Output Interactive Markov Chains.

I/O-IMCs are a modelling formalism combining continuous-time Markov chains with discrete actions (also called *signals*). They have the useful property of being composable, as the signals allow several I/O-IMCs to communicate [BCS10].

An example of this composition is shown in Figure 6.9. The input signals (denoted by a '?') can only be taken when the corresponding output signal (denoted by '!') is taken. Internal actions (denoted by ';') and Markovian transitions (denoted by Greek letters) are taken independently of the other modules. If multiple non-Markovian transitions can be taken from a state, which transition is taken is nondeterministically chosen.

In the example, all component models begin in their initial states. From t_0 the transition 'b?' cannot be taken unless the output transition 'b!' is also taken, so

both initial states can only perform their Markovian transitions. Assuming the leftmost model takes its transition with rate λ first, the composition enters state s_1, t_0. From here, two options are possible: (1) the internal action 'a;' from s_1 to s_2 can be taken, leaving the rightmost model in state t_0, or (2) the output transition 'b!' from s_1 to s_3 can be taken together with the input transition 'b?' from t_0 to t_1. In the latter case, the composed model takes a transition 'b!' allowing it to be composed with yet more models, and enters state s_3, t_1, from which neither component model can take further transitions. If the internal action was taken instead, the transition from t_0 to t_2 with rate μ remains possible, leading to the terminal state s_2, t_2.

6.2.3 Reducing I/O-IMCs to Markov Chains

Step 2 of our approach involves computing the parallel composition of the I/O-IMCs of the elements of the FMT. Our technique requires that the (composed) I/O-IMC be reduced to a Markov Chain, which means resolving all nondeterminism. In our setting, we assume that all nondeterminism is spurious (i.e., how the nondeterminism is resolved has no effect on the computed availability). Therefore, if we are in a state where we can choose a non-Markovian transition, we apply the maximal progress assumption[EHZ10] and take this transition. If multiple non-Markovian transitions can be taken from the same state, we do not specify which one is taken (as an implementation matter, we take whichever occurs first in the specification file). Thus we are only left with states with only Markovian transitions, which can be used as an input for the Monte Carlo simulation.

Formally, Algorithm 6.1 specifies how we take an IMC (reformulating the transition relations as transition functions), and compute the outgoing transitions from a state in the Markovianized CTMC. Note that we require the IMC to have no interactive cycles (i.e., no *Zeno runs*), or the algorithm may fail to terminate.

Example 30 *Figure 6.10 illustrates the Markovianization process. Upon encountering the transition* $s_0 \xrightarrow{\lambda} s_1$*, we notice that* s_1 *has non-Markovian transitions. We now arbitrarily select either the 'a;' or 'b;' transition. If we select 'a;', we*

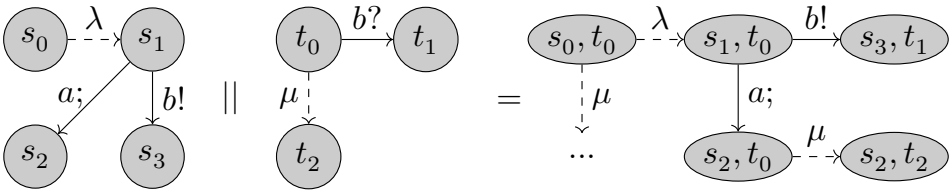


Figure 6.9: Example of the partial parallel composition of two I/O-IMCs.

Algorithm 6.1 Markovian transition function

Context: IMC with set of states S, set of actions A, interactive transition function $I : S \to \mathcal{P}(A \times S)$, Markovian transition function $M : S \to \mathcal{P}(\mathbb{R}_{>0} \times S)$.
Input: State $s \in S$.
Output: Set of Markovian transitions $R \subseteq \mathbb{R}_{>0} \times S$.

$R \leftarrow \emptyset$
for $(r, t) \in M(s)$
  while $I(t) \neq \emptyset$
    take $(a, t') \in I(t)$
    $t \leftarrow t'$
  end while
  $R \leftarrow R \cup \{(r, t)\}$
end for
return R

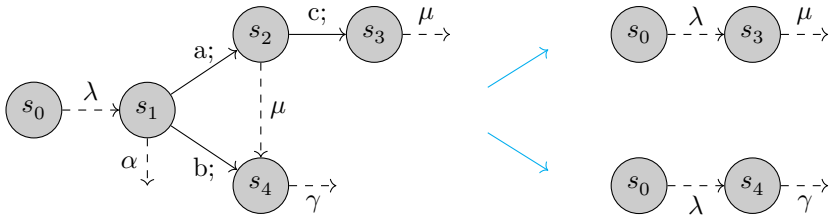


Figure 6.10: **Example of the possible conversions of an I/O-IMC to a Markovian model** following the procedure described in Section 6.2.3.

replace the initial transition by $s_0 \xrightarrow{\lambda} s_2$*, and note that* s_2 *again has a non-Markovian transition. We thus replace the initial transition again by* $s_0 \xrightarrow{\lambda} s_3$*, resulting in the upper right model. If we initially choose the 'b;' transition, we instead obtain the model on the bottom right.*

This process leaves undefined which transition is taken in nondeterministic states. In most practical DFT models, the only source of nondeterminism is the order in which gates fail when an element has multiple parents. Such nondeterminism is spurious, in that it has no effect on the outcome of the analysis. It is therefore acceptable to leave the exact resolution undefined for such models.

In models where nondeterminism can actually affect the results of the analysis, our determinisation is clearly not correct. We therefore apply our analysis only on DFTs in which a syntactic check rules out the possibility of non-spurious nondeterminism. In particular, we require the children of PAND or SPARE gates

to be entirely independent subtrees. We have found that in practice, most DFTs in the literature already satisfy this condition.

6.3 Methodology

Our overall approach to rare event simulation for FMTs relies on a conversion of the FMT into an input/output interactive Markov chain (I/O-IMC). This I/O-IMC is a Markovian model describing the behaviour of the FMT. Given the I/O-IMC, we apply the Path-ZVA algorithm for importance sampling to alter its transition rates to make the top level event of the FMT more probable. We then sample simulation traces from this model, measuring the unavailability of the FMT, and apply a correction for the adjusted transition rates.

In our approach, we follow the semantics of [BCS10], which describes the behaviour of dynamic fault trees as I/O-IMCs. These semantics were extended in [GKS+14] to include periodic maintenance actions. One of the major benefits of these semantics is that the I/O-IMC is specified as a parallel composition of many smaller I/O-IMCs, each of which models one element (i.e., gate, basic event, or maintenance module) of the FMT.

We note that the semantics of the FMT analysed using this approach differs somewhat from the semantics described in Chapter 5. The FMTs described there support arbitrary probability distributions for inspection and repair times, which are not supported in I/O-IMCs. We therefore approximate these distributions using Markovian models. In particular, we approximate exact times (e.g., 'inspect every two weeks') using an Erlang distribution with the specified time as its mean.

Overall, given an FMT, our analysis technique consists of the following steps:

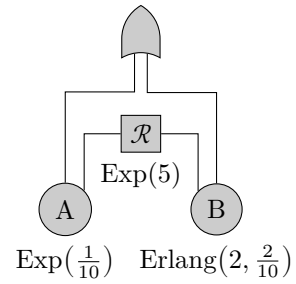
1. Use the DFTCalc tool to compute I/O-IMCs for all elements of the FMT.

2. Apply the steps Path-ZVA algorithm, as explained in Section 6.1.2, to adjust the transition probabilities and compute the corresponding likelihood ratios. Since only the most likely paths receive altered probabilities, the rest of the model can be computed on-the-fly.

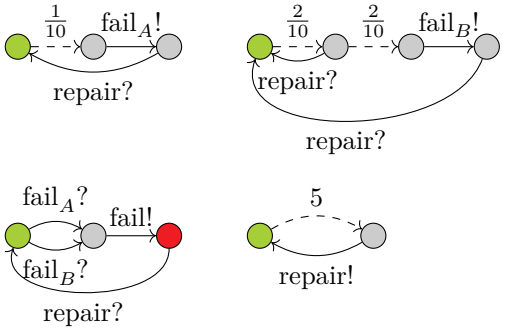
3. Sample traces of the adjusted model, ending each trace when it returns to the initial state, storing the likelihood ratio L_i, total time D_i, and time spent in unavailable (i.e., failed) states Z_i.

4. Average the total time $\overline{DL}$ and unavailable time $\overline{ZL}$ of the traces, multiplied by the likelihood ratios. Now $\overline{ZL}/\overline{DL}$ is the output estimated unavailability.

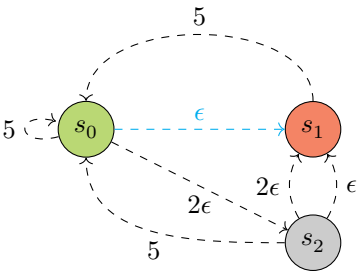
Example 31 *Figure 6.11 illustrates the steps of our approach on a simple FMT with two components, and periodic repair.*



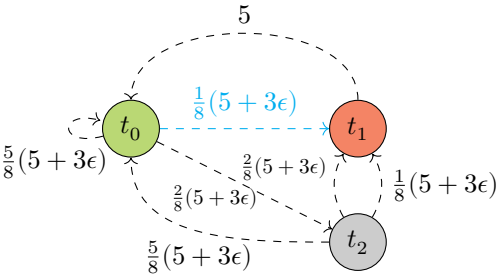
(a) Fault maintenance tree



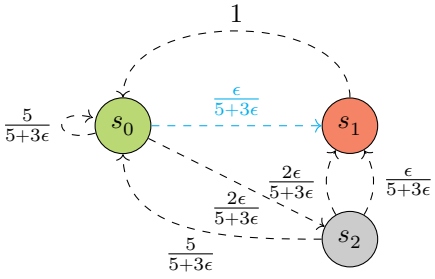
(b) I/O-IMCs



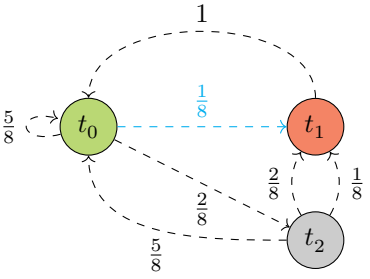
(c) Composed, Markovianised model ($\epsilon = \frac{1}{10}$)



(d) Adjusted model



(e) Embedded DTMC of (c)



(f) Embedded DTMC of (d)

Figure 6.11: Example of the steps to apply importance sampling to a DFT.

1. *We convert every element of the FMT in Figure 6.11a into an I/O-IMC shown in Figure 6.11b.*

2. *We compose these I/O-IMCs and remove the non-Markovian transitions, obtaining the model shown in Figure 6.11c. In this transformation we also rewrite the transition rates to include the rarity parameter ϵ. By searching this model, we identify that we can reach the failed state in one transition.*

3. *We identify all paths reaching the goal in one transition, which is only the blue transition in Figure 6.11c.*

4. *Applying Path-ZVA, we increase the likelihood of the transitions along the previously identified path, resulting in the model shown in Figure 6.11d.*

5. *We draw simulation traces from the adjusted model. For example, we can draw three traces (in practice one would draw many thousands):*

   (a) $t_0 t_0$ *(*$L_0 = \frac{8}{5+3\epsilon} \approx 1.5$*) with total time* $D_0 = 0.045$ *(sampled from an exponential distribution with mean* $\frac{1}{5.3}$*) and no time in unavailable states (*$Z_0 = 0$*).*

   (b) $t_0 t_2 t_0$ *(*$L_1 \approx 0.23$*) with total time* $D_1 = 0.196 + 0.055 = 0.251$ *and no unavailable time.*

   (c) $t_0 t_1 t_0$ *(*$L_2 \approx 0.15$*) with total time* $D_2 = 0.436 + 0.196 = 0.632$ *and unavailable time* $Z_2 = 0.196$*.*

6. *Finally, we combine the samples to obtain our average unavailability. For the samples drawn above, we would obtain* $\hat{U} = \frac{1}{3}\left(L_0 \frac{Z_0}{D_0} + L_1 \frac{Z_1}{D_1} + L_2 \frac{Z_2}{D_2}\right) \approx \frac{1}{3}\left(0 + 0 + 0.15 \frac{0.196}{0.632}\right) \approx 0.016$*. More detailed statistical measures, such as confidence intervals, can also be computed.*

6.4 Case Studies and Results

We evaluate the effectiveness of the importance sampling analysis method described in this chapter on three parameterized case studies. We compare our FTRES tool to the DFTCalc tool, which evaluates DFTs numerically through stochastic model checking [ABvdB+13], and to a standard Monte Carlo simulator (MC) built into FTRES without importance sampling.

The case studies we use are parameterized versions of one DFT taken from industry and two well-known benchmarks from the literature. The industrial case models a redundant system of relays and high-voltage cabinets used in railway signalling, and was taken from [GSS15]. The other two cases are the fault-tolerant parallel processor (FTPP) [DBB90] and the hypothetical example computer system (HECS) [SVD+02].

Experimental Setup.

For each of the cases, we compute the long-run unavailability (exact for DFTCalc, 95% confidence interval for FTRES and MC).

The failure times of the basic events are modelled as exponential distributions in the HECS case (following [SVD+02]), while those for the railway cabinets and FTPP cases are modelled as an Erlang distribution where the number of phases P is a parameter ranging from 1 to 3 phases; clearly, $P = 1$ corresponds to the exponential distribution.

We measure the time taken (with a time-out of 48 hours) and the memory consumption in number of states (which is negligible for MC). For DFTCalc we measure both peak and final memory consumption. Simulations by FTRES (after the CoM is computed) and MC were performed for 10 minutes.

All experiments were conducted on a dual 2.26 GHz Intel® Xeon® E5520 processor and 24 GB of RAM.

6.4.1 Railway Cabinets

This case, provided by the consulting company Movares in [GSS15], is a model of a redundant system of relay and high-voltage cabinets used in railway signalling.

The model, shown in Figure 6.6 comprises two types of trackside equipment used in the signalling system: Relay cabinets house electromechanical relays that respond to electronic controls signals by switching electrical power to e.g. switch motors and signals lights. Relays are also a safety-critical part of the interlocking system, as they are wired in such a configuration as to prevent safety violations such as moving switches in already-occupied sections of track. The high-voltage cabinets provide connections from the local power grid to operate the relays and other electrically-powered systems.

We consider several variants of the FT for given parameter values. We augment the FT with a periodic inspection restoring any degraded basic events to perfect conditions. The time between executions of this action is governed by an Erlang distribution with two phases, and a mean time of half a year. We vary the number of cabinets in the system from 2 to 4.

Table 6.1 shows the results of the FTRES, DFTCalc, and MC tools. We note that, whenever DFTCalc is able to compute a numerical result, this result lies within the confidence interval computed by FTRES. We further see that the 2-phase models with 4 cabinets, and the 3-phase models with 3 or 4 cabinets could not be computed by DFTCalc within the time-out (times shown in Figure 6.12), while FTRES still produces usable results. Finally, while the standard Monte Carlo simulation produces reasonable results for the smaller models, on the larger models it computes much wider confidence intervals. For the largest models, the MC simulator observed no failures at all, and thus computed an unavailability of 0.

	N	P	Unavailability DFTCalc	FTRES	MC
Railway cabinets	2	1	$4.25685 \cdot 10^{-4}$	$[4.256; 4.258] \cdot 10^{-4}$	$[4.239; 4.280] \cdot 10^{-4}$
	3	1	$7.71576 \cdot 10^{-4}$	$[7.713; 7.716] \cdot 10^{-4}$	$[7.694; 7.751] \cdot 10^{-4}$
	4	1	$1.99929 \cdot 10^{-3}$	$[1.998; 2.000] \cdot 10^{-3}$	$[1.999; 2.004] \cdot 10^{-4}$
	2	2	$4.55131 \cdot 10^{-8}$	$[4.548; 4.555] \cdot 10^{-8}$	$[1.632; 4.387] \cdot 10^{-8}$
	3	2	$6.86125 \cdot 10^{-8}$	$[6.846; 6.873] \cdot 10^{-8}$	$[0.673; 1.304] \cdot 10^{-7}$
	4	2	—	$[2.358; 2.394] \cdot 10^{-7}$	$[2.282; 3.484] \cdot 10^{-7}$
	2	3	$5.97575 \cdot 10^{-12}$	$[5.714; 6.252] \cdot 10^{-12}$	—
	3	3	—	$[5.724; 7.914] \cdot 10^{-12}$	—
	4	3	—	$[0.337; 1.871] \cdot 10^{-11}$	—
FTPP	1	1	$2.18303 \cdot 10^{-10}$	$[2.182; 2.184] \cdot 10^{-10}$	—
	2	1	$2.19861 \cdot 10^{-10}$	$[2.198; 2.199] \cdot 10^{-10}$	—
	3	1	$2.21420 \cdot 10^{-10}$	$[2.213; 2.215] \cdot 10^{-10}$	—
	4	1	$2.22979 \cdot 10^{-10}$	$[2.229; 2.230] \cdot 10^{-10}$	$[0; 2.140] \cdot 10^{-8}$
	1	2	$1.76174 \cdot 10^{-20}$	$[1.761; 1.763] \cdot 10^{-20}$	—
	2	2	$1.76178 \cdot 10^{-20}$	$[1.756; 1.770] \cdot 10^{-20}$	—
	3	2	—	$[1.673; 1.856] \cdot 10^{-20}$	—
	4	2	—	$[1.257; 2.553] \cdot 10^{-20}$	—
	N	**k**	**DFTCalc**	**FTRES**	**MC**
HECS	1	1	$4.12485 \cdot 10^{-5}$	$[4.118; 4.149] \cdot 10^{-5}$	$[2.615; 10.64] \cdot 10^{-5}$
	2	1	—	$[3.010; 3.061] \cdot 10^{-9}$	—
	2	2	—	$[8.230; 8.359] \cdot 10^{-5}$	$[0; 1.734] \cdot 10^{-4}$
	3	1	—	$[3.024; 3.213] \cdot 10^{-13}$	—
	3	2	—	$[8.853; 9.106] \cdot 10^{-9}$	—
	3	3	—	$[1.230; 1.261] \cdot 10^{-4}$	$[0; 1.267] \cdot 10^{-4}$
	4	1	—	$[1.328; 8.213] \cdot 10^{-17}$	—
	4	2	—	$[1.145; 1.270] \cdot 10^{-12}$	—
	4	3	—	$[1.744; 1.817] \cdot 10^{-8}$	—
	4	4	—	$[1.609; 1.667] \cdot 10^{-4}$	—

Table 6.1: Comparison of the unavailabilities computed by DFTCalc, FTRES, and MC simulation for the case studies with N cabinets/processor groups.

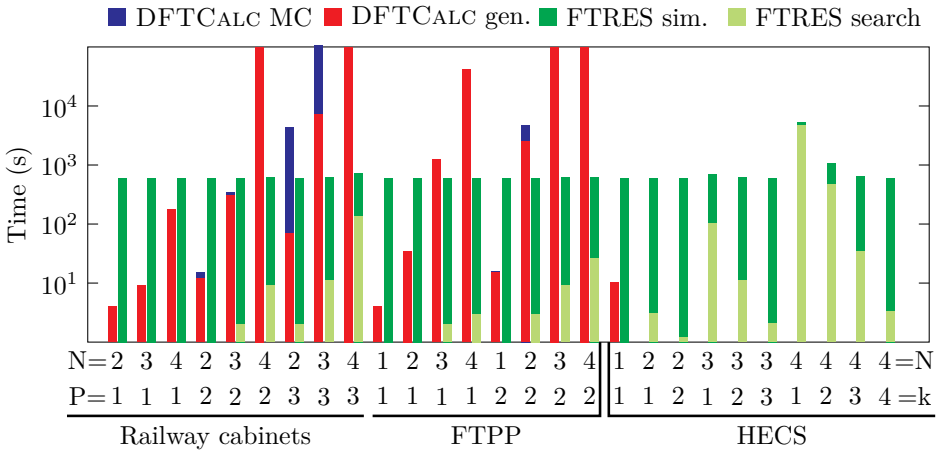


Figure 6.12: **Processing times for the different tools:** Times for model generation and model checking for DFTCalc, and for the graph search and simulation for FTRES. Bars reaching the top of the graph reached the time-out of 48 hours. Most bars for the HECS case study are omitted as they all timed out. Exact times can be found in Table B.1 on page 248.

Figure 6.15 shows the generated state spaces for both tools. Since FTRES only needs an explicit representation of the shortest paths to failure, it can operate in substantially less memory than DFTCalc. Although the final model computed by DFTCalc is smaller due to its bisimulation minimisation, the intermediate models are often much larger.

6.4.2 Fault-Tolerant Parallel Processor

The second case study is taken from the DFT literature [DBB90], and describes a fault-tolerant parallel computer system. This system consists of four groups of processors, labelled A, B, C, and S. The processors within a group are connected by a network element, independent for each group. A failure of this network element disables all connected processors.

The processors are also grouped into workstations, numbered 1 to n. Each workstation depends on one processor per group, where the processor of group S can act as a spare for any of the groups. Therefore, if more than one processor (or its connecting network element) in a workstation fails, the workstation fails.

Maintenance is performed through a periodic replacement restoring all degraded components to their perfect conditions. The time of this replacement follows a four-phase Erlang distribution with a mean time of 2 time units between repairs.

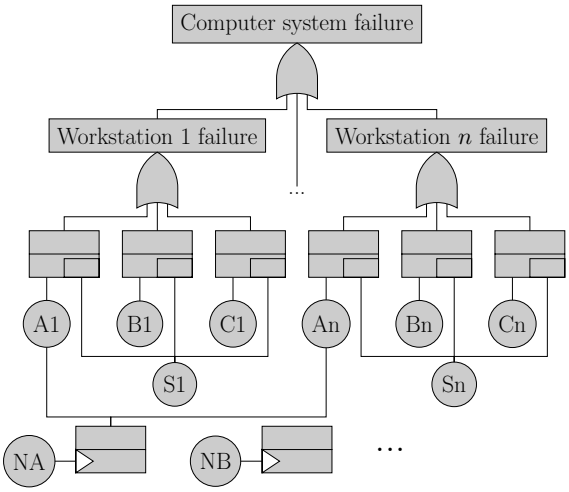


Figure 6.13: **DFT of the fault-tolerant parallel processor.** Connections between the FDEP for B omitted for clarity, as well as the FDEPs for groups C and S.

The numerical results and computation times for this case study can be found in Table 6.1 and Figure 6.12 respectively. We can see that the unavailability does not vary much with the number of computer groups, since the network elements are the dominant failure causes and are not affected by N. We again observe that DFTCalc runs out of time in the two largest cases while FTRES still performs well. Wider confidence intervals are produced, though still useful for most practical purposes. The standard MC simulation observed no failures for most of the models.

Figure 6.15 lists the generated state spaces for both tools. Again, FTRES requires less peak memory than DFTCalc.

6.4.3 Hypothetical Example Computer System

Our final example is the classic benchmark DFT of the Hypothetical Example Computer System (HECS), described in [SVD+02] as an example of how to model a system in a DFT. It consists of • a processing unit with three processors, of which one is a spare, of which only one is required to be functional. It further contains • five memory units of which three must be functional, • two busses of which one must be functional, and • hardware and • software components of an operator interface. The DFT of the HECS is shown in Figure 6.14.

We parameterize this example by replicating the HECS N times, and requiring

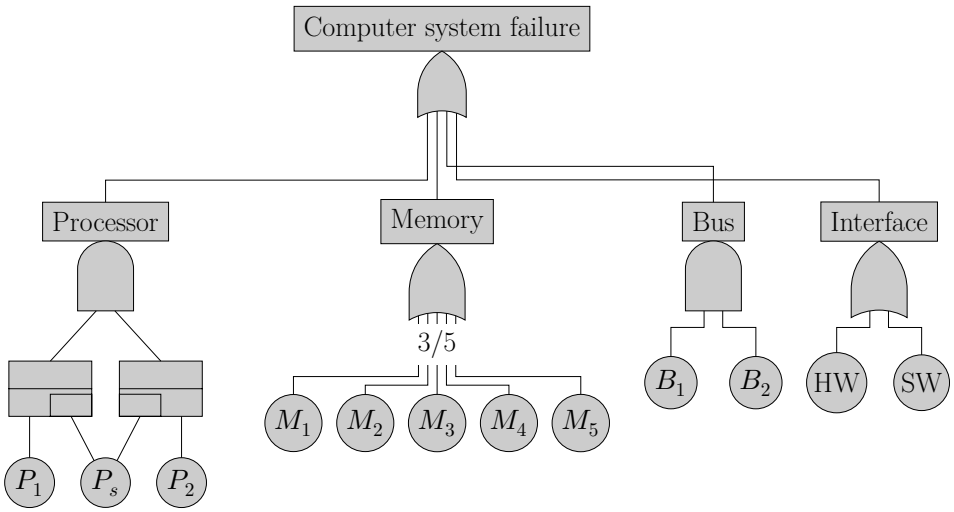


Figure 6.14: DFT of the hypothetical example computer system.

k of these replicas to be functional to avoid the top level event. The basic events in this case remain exponentially distributed, and we add maintenance as a periodic replacement of all failed components on average every 8 time units (on a 2-phase Erlang distribution).

As for the other cases, Table 6.1 lists the unavailabilities computed by the tools, while Figures 6.12 and 6.15 show the processing time and state spaces computed, respectively. We notice that except for the simplest case, DFTCalc is unable to compute the availability within 48 hours, and the MC simulator in many cases failed to observe any failures, and produced very wide confidence intervals in the cases where it did. FTRES, on the other hand, produced reasonable confidence intervals for all cases (although the interval for the $(4, 1)$ case is fairly wide, it also has the largest state space and a very small unavailability).

6.4.4 Analysis results

As the sections above show, FTRES outperforms DFTCalc for larger models, and traditional MC simulation for models with rare failures. In particular, FTRES:

- requires less peak memory than DFTCalc in every case, and requires less time for large models, while still achieving high accuracy.

- can analyse models larger than DFTCalc can handle.

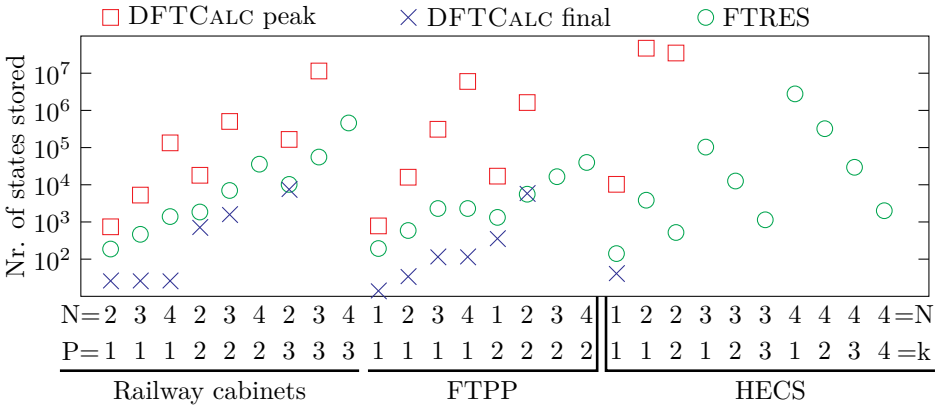


Figure 6.15: Numbers of states stored in memory for the different cases with N cabinets/processor groups. For DFTCalc, both the largest intermediate and the final (minimised) state spaces are given. Numerical results can be found in Table B.2 on page 249.

- gives confidence intervals up to an order of magnitude tighter than those estimated by MC in similar processing time.

6.5 Conclusion

Traditional analysis techniques for (repairable) dynamic fault trees suffer from a state-space explosion problem hampering their applicability to large systems. A common solution to this problem, Monte Carlo simulation, suffers from the rare event problem making it impractical for highly-reliable systems. This chapter has introduced a novel analysis technique for repairable DFTs based on importance sampling. We have shown that this technique can be used to obtain tight confidence intervals on the availability of highly reliable systems with large numbers of repairable components.

Our method uses the compositional semantics of [BCS10] and [GKS+14], providing flexibility and extensibility in the semantics of the models. By deploying the Path-ZVA algorithm [RdBSJ18], we only need to explore a small fraction of the entire state space, substantially reducing the state-space explosion problem. At the same time, the algorithm uses importance sampling to significantly reduce the number of simulations required for accurate estimation.

We have demonstrated using three case studies that our approach can handle considerably larger models than DFTCalc, and provide more accurate results than classical Monte Carlo simulations.

Future work. Relevant extensions of our approach could generalise the algorithm to compute metrics other than availability. Of particular interest would be the *reliability*. Furthermore, we currently restrict ourselves to purely Markovian models, which means we can only approximate the semantics described in Chapter 5. Another promising avenue to investigate is how to include non-Markovian transition times. This would allow fault maintenance trees to be analysed in their full expressive power. Finally, the DFT semantics of Boudali et al. [BCS10] produces nondeterministic transitions for many DFTs. Our current conversion to a Markovian model can only be applied if this nondeterminism is spurious, and it could be interesting to examine whether non-spurious nondeterminism could be meaningfully incorporated in our approach.

Tooling. For our analysis, we use the models of the DFT elements produced by DFTCalc, as well as its description of how to compose them. In this way, we ensure that our semantics are identical to those used in the existing analysis.

DFTCalc produces IMCs for the DFT elements, and a specification describing how the IMCs are composed. It then uses the CADP [GLMS13] tool to generate the composed IMC which can be analysed by a stochastic model checker.

FTRES instead uses the models and composition specification to generate the composition on the fly, and applies the importance sampling algorithm to compute the unavailability of the model.

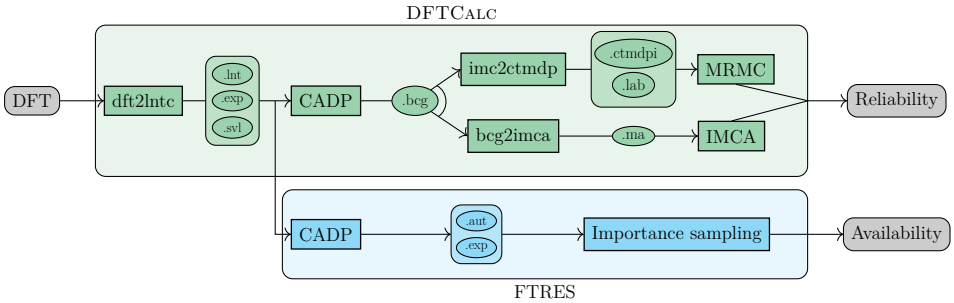


Figure 6.16: Diagram of the workflow of FTRES and DFTCalc.

Figure 6.16 shows how the various programs interact to obtain numerical metrics from a (repairable) DFT.

Part III

Case studies

Chapter 7

FMTs in practice: Analysis of the electrically insulated joint

Chapter 5 introduced the formalism of fault maintenance trees (FMTs) and described how FMTs can be analysed to obtain important dependability metrics such as reliability and expected costs.

In this chapter and the next, we investigate whether FMTs can be applied in the industrial setting of railway systems to improve maintenance policies. This chapter considers electrically insulated joints (EI-Joints, example shown in Figure 7.1), a railroad component that is both commonly used, and a notable source of disruptions (causing about 1000 disruptions annually in the Netherlands). The next chapter will discuss a pneumatic compressor.

Problem Statement. We like to use the EI-joint to find out if FMTs are a useful tool to investigate maintenance questions, and to obtain trustworthy results. In particular, we aim the answer the following research questions:

1. Do FMTs provide sufficient expressiveness to model the complex maintenance policies used in practice?

2. Can we construct an FMT of the EI-joint with sufficient accuracy to make recommendations about its maintenance policy?

3. Can we find improvements to the reference maintenance policy of the EI-joint to reduce cost and/or increase the reliability of the joint?

4. How does the newly developed NRG joint compare to the currently installed EI-joints in terms of reliability and cost?

Case description. EI-Joints are a part of the train detection system, and provide a physical connection between two sections of rail while keeping them electrically

separated. This allows the detection system to determine which of these separated sections is occupied by a train.

The degradation behaviour and maintenance actions for the EI-joint are typical for physical assets, with both random and wear-induced failures, repairs and renewals, and different options for maintenance strategies. Furthermore, both failures and maintenance actions have significant costs.

In close collaboration with the Dutch national railway network infrastructure manager ProRail, we have conducted a reliability analysis of EI-joints. We have developed an FMT of the joint, and translated this FMT into an Uppaal-SMC model from which we can obtain dependability metrics using statistical model checking.

We analyse the dependability of the EI-joints, computing their reliability, expected number of failures, and expected costs over time. In particular, we investigate a reference maintenance strategy provided by ProRail, as well as potentially better strategies. We study:

1. *Variations in inspection intervals*, as more frequent inspections lead to fewer failures but also higher inspection costs.

2. *Periodic preventive replacements*, replacing the joint after a given length of time regardless of its (observable) condition. This increases maintenance costs due to the replacement and planned downtime, but may reduce failures if some types of wear are not visible on inspection.

3. *Replacement of an entire joint* instead of repairing individual components.

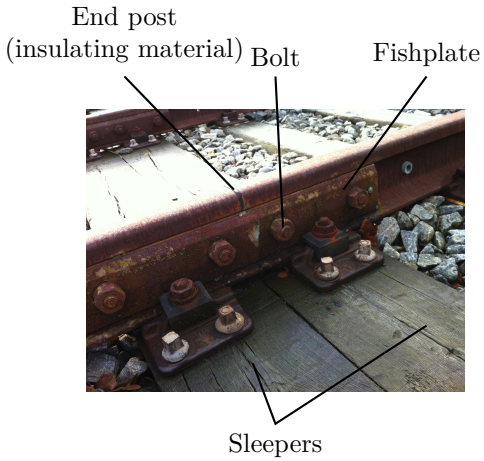


Figure 7.1: An electrically insulated joint with the visible components indicated.

This may be useful if several, otherwise separate, failure modes degrade at a similar rate. As Section 7.1 will describe, many failure modes in the EI-Joint have similar expected times to failure, and replacing the joint when the first failure occurs may prevent other failures that would occur shortly afterwards.

4. *Repair when observing higher or lower degradation level.* ProRail's policy specifies acceptable amounts of degradation, and if an inspection finds a greater amount, a repair is performed. By reducing the permitted amount, more failures can be prevented at the cost of more replacements. Conversely, a higher level means fewer replacements but a higher probability of failure before the next inspection.

Furthermore, we analyse a new type of EI-joint, the NRG joint, to examine whether replacing the existing joints with this new type will reduce disruptions and costs.

Our analysis finds that (1) the current inspection policy is nearly cost-optimal when combining cost of failure and cost of maintenance, (2) periodic preventive replacements improve reliability, but are more expensive than corrective replacements, (3) the optimal inspection policy does not vary much with the load level of the track, and (4) the new NRG joint is both more reliable and less expensive in operation. During the case study, we also found some inaccuracies in ProRail's documentation, which were corrected, and noted that this documentation could be improved by including information about dependencies between different failure modes.

An important contribution is the extensive validation of our model: To provide confidence in the results of our analysis, we have compared the results predicted from our analysis with actual data from a failure database. Our predicted results agree with actual results from the field strongly enough to make recommendations based on our model.

During our analysis, we were able to construct an FMT of the EI-joint that models the reference policy, demonstrating that FMTs are sufficiently expressive to model practical maintenance policies. We validate the FMT against ProRail's database of failures and replacements, finding that our model gives predictions in line with reality. This gives us confidence that our model can also make accurate predictions of alternative maintenance policies. These predictions show that the reference policy is already close to cost-optimal. We find several alternatives that increase the reliability of the joint, but at the expense of higher total cost.

Origin of this chapter. This chapter describes the case study published in:

- Enno Ruijters, Dennis Guck, Martijn van Noort, and Mariëlle Stoelinga. "Reliability-centered maintenance of the electrically insulated railway joint via fault tree analysis: A practical experience report". In *Proceedings of the*

46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 662–669, 2016. DOI: 10.1109/DSN.2016.67, ISBN: 978-1-4673-8891-7.

Organisation of this chapter. In Section 7.1 we describe the EI-Joint and its maintenance process. Section 7.2 explains the process used to construct the model of the EI-Joint. Section 7.3 explains the analysis performed and the results obtained, and Section 7.4 presents our conclusions.

7.1 Case description

Many railroad networks use electrical detection to determine the presence of trains on the tracks (e.g., in The Netherlands [Pro15]). This system works by detecting when the axles of a train electrically connect the two rails, illustrated in Figure 7.2. To determine the location of a train, tracks are divided into several, electrically isolated, sections. This separation is provided by *electrically insulated joints* (EI-joints, shown in Figure 7.1).

To detect the presence of a train, a small detection voltage is applied across the rails at one end of a section, and detected at the other end. If a train is present on the track, its axles will short circuit the detection current, so that the signal is not detected. The interlocking system then locks switches in their positions, sets signals appropriately, etc. This system is fail-safe, as loss of the detection current (e.g., due to a broken track or failure of the signal generator) results in the same lack of current at the detector as when a train is present, so that signals do not allow trains to enter the section, switches cannot be moved, etc.

The location of a train is determined by creating electrically separate sections of track, each of which has its own detection current and detectors. On straight stretches of rail, these sections are several hundred metres to several kilometres in length. In areas with switches or level crossings, the sections are often much shorter.

The sections are connected using EI-joints, which maintain electrical separation between sections, while mechanically connecting the rails. The joints do not compromise the fail-safe nature of the detection system: if failed joints allow the detection current from one section to travel to another section, a train on either section will short out the current. The result is then that both sections appear occupied, which means no other trains can enter them, but no dangerous situations can occur.

Due to the large number of these joints in the railroad network (approx. 50,000 in the Netherlands) and the susceptibility of the joints to environmental conditions, EI-joints are a relatively frequent cause of disruptions. Failures can occur for

various reasons, both internal to the joint such as broken bolts, and external to the joint such as metal shavings bypassing the insulation.

Visual inspections can be performed to determine whether some of these failures are likely to occur soon, and corrective action, such as sweeping away iron shavings, can prevent certain failures from occurring. More detailed inspections, such as measuring the electrical resistance, can detect more types of failure, but are more expensive as they require the track to be taken out of service so a worker can safely perform the measurement.

Some failures can be corrected in isolation, such as sweeping away metal particles that are lowering the electrical resistance. Other failures can only be prevented or corrected by replacing the entire joint, e.g., internal cracking in the end plate. Some failures, such as vandalism, cannot be prevented by normal maintenance and can only be corrected after they occur.

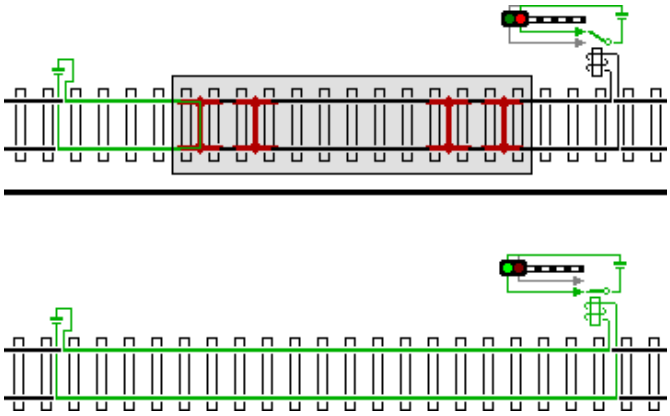


Figure 7.2: **Depiction of the track circuit for train detection:** the detection signal, depicted as the green line, is generated at the left of the images, and the detector is at the right. The bottom image depicts the situation where the track is clear, the red lines on the top picture indicate the axles of a train. [0]

7.1.1 Joint construction

The major components of the EI-joint are shown in Figure 7.1, they are:

- The *end post* is a thin piece of insulating material between the rails. It provides a smooth surface for the train wheels to roll and keeps the end of the rail from touching each other.

[0]Source: https://commons.wikimedia.org/wiki/File:Track_circuit.png

- The *fishplates* are long, steel plates on either side of the rail, providing mechanical strength to keep the joint from flexing too much when a train drives over it. An insulating layer between the rails and fishplates prevents it from making electrical contact.

- *Bolts* hold the fishplates in place. Insulating bushings prevent them from touching the rails.

- *Sleepers* provide mechanical support for the rails. As the joints are a weak spot in the rails, they should always be supported by two sleepers to prevent flexing under load.

Two types of joints are commonly used in the Netherlands: *glued* and *constructed*. In glued joints, the joint is produced in the factory fully assembled with several metres of rail on each side. Glue is used to provide more mechanical strength than would be delivered by the bolts alone. Glued joints are installed or replaced by cutting a piece of track out of the existing railway, and welding the joint assembly in its place. As most joints in the Netherlands are glued joints (45,000 out of 50,000), this chapter focuses on the analysis of that type.

Constructed joints are delivered as components, and installed by drilling holes and cutting a small gap in an existing rail, placing the components in place and holding them there using the bolts. Due to their lower strength, constructed joints are only used on low-speed, lower-traffic sections of rail in the Netherlands such as shunting tracks.

7.1.2 Failure modes

The possible failure modes of the EI-joint are well-documented internally by ProRail, in the form of an FMECA. This document lists the failure modes, the conditions under which they can occur, and the expected time to failure if no maintenance is performed (under the assumption that the required conditions are present).

ProRail's FMECA also provides a reference maintenance policy, which ProRail considers to adequately mitigate the risks of all the listed failure modes. The details of this policy are kept confidential, as different contractors tender their own policies when bidding for maintenance contracts. Each tendered policy is compared to the reference policy, to ensure that the contractor still provides sufficient mitigations for the listed risks.

EI-joints are subject to two general categories of failures: *Mechanical failures* where the joint no longer provides a physical connection of the rails, and *electrical failures* that lead to an unintended electrical connection between the rails. The former type are very rare, but have potentially catastrophic consequences (derailment of trains). The latter failures are more common and are generally not considered safety-critical due to the fail-safe nature of the detection system (i.e., they cause

BE nr.	Failure mode	ETTF	ETTF NRG	Phases	Prob. cnd.
1	Poor geometry	5	5	4	10%
2	Broken fishplate	8	12	4	33%
3	Broken bolts	15	20	4	33%
4	Rail head broken out	10	15	4	33%
5	Glue connection broken	10	15	4	33%
5a	Manufacturing defect	-	-	-	0.25%
5b	Installation error	-	-	-	0.25%
6	Battered head	20	22	4	5%
7	Arc damage	1	1	3	0.2%
8	End post broken out	7	8	3	33%
9	Joint bypassed: overhang	5	8	4	100%
10a	Joint shorted: shavings (normal)	1	1	4	12%
10b	Joint shorted: shavings (coated)	10	10	4	3%
11	Joint shorted: splinters	200	200	1	100%
12	Joint shorted: foreign object	250	250	1	100%
13	Joint shorted: shavings (grinding)	5000	5000	1	100%
14	Sleeper shifted	5000	5000	1	100%
15	Internal insulation failure	5000	5000	1	100%
16	End post jutting out	20	20	1	100%

Table 7.1: **Parameters of the basic events of the FMT for the EI-joint.** The column 'ETTF' lists the expected time to failure in years, assuming no maintenance is performed. The column 'ETTF NRG' lists the expected time to failure in the NRG joint. The column 'prob. cnd.' gives the probability that a given joint is subject to the condition that allows this failure mode to occur. Modes 5a and 5b have a fixed probability of occurring every time a joint is installed.

signals to stay red and switches to lock in their current position, which prevents trains from moving, but does not create unsafe situations), although they do cause disruptions of the train service.

Table 7.1 lists the most significant failure modes, together with important failure parameters: Each mode is characterised by the expected time to failure assuming no maintenance is performed, the number of degradation phases we consider in our modelling (as described in Section 5.2), and the probability that a given joint is subject to this failure mode. The latter is needed, since not all failure modes occur in all situations. For instance, Line 1 in the table shows that only 10% of the EI-joints are subject to poor geometry; 90% of the joints have a sufficiently stable surface so that this failure mode never occurs.

A description of these failure modes is:

- **Poor geometry:** Due to poor surface conditions or incorrect installation, the rails meet each other at an angle (usually vertically, though sometimes

horizontally). While this does not directly hamper the operation of the joint, the poor geometry leads to increased wear due to increased flexing or passing train wheels bumping against the rails.
Required conditions: Poor surface condition or incorrect installation.
Maintenance action: Correct surface conditions (if needed) and replace joint with correct geometry.

- **Broken fishplate:** The fishplate breaks, usually at one of the bolts. In many cases, cracking and tearing can be seen before a complete break occurs.
Required conditions: Located on soft surface, or occurrence of other failure mode causing increased stress.
Maintenance action: Replace joint.

- **Broken bolts:** One or more bolts breaks, often breaking off at the head. Once one bolt has broken, increased stress can quickly lead to the remaining bolts failing, particularly on the same side of the joint.
Required conditions: Located on soft surface, or occurrence of other failure mode causing increased stress.
Maintenance action: Replace joint.

- **Rail head broken out:** A piece of one of the rails breaks off of the rail surface. This is usually caused by a deformation/discontinuity in or near the end post, causing passing train wheels to knock against the surface and weakening it until it breaks.
Required conditions: Imperfections in nearby track.
Maintenance action: Replace joint.

- **Glue connection broken:** The glue holding some components of the joint together fails. Apart from reducing the mechanical strength of the joint, this can allow openings to form that water can seep into, causing further damage. Some cases of broken glue connections are caused by defects introduced in manufacturing or excessive stress placed on the joint during installation.
Required conditions: Located on soft surface, or occurrence of other failure mode causing increased stress.
Maintenance action: Replace joint.

- **Battered head:** Deformation of the surface(s) of the rail(s) near the joint. This can occur due to a deformation/discontinuity in the surface near the joint, as passing train wheels 'bump' or the surface and cause damage where they land. As battered heads cause yet more such 'bumping', it often accelerates other failure modes.
Required conditions: Damage to end post or nearby track.
Maintenance action: Grinding can correct small defects, larger deformation requires replacement of the joint.

- **Arc damage:** The rails carry large electrical currents as part of the power supply to trains. Normally, special filters are used to allow this current to flow between the rails, bypassing the joint, while filtering out the detection currents. If these filters fail or are too small for the current required, electrical arcing can occur which causes local overheating of the rail and joint.
  Required conditions: Damaged or insufficient filter transformer(s).
  Maintenance action: Replace failed filters and joint.

- **Joint bypassed: overhang:** As trains pass over the joint, the pressure from their wheels slightly 'rolls out' the surface of the track. Over time, this causes a thin layer of steel to be rolled over the end post. If this overhanging layer gets too long, it can create an electrical connection between the rails.
  Maintenance action: Saw off overhanging metal.

- **Joint shorted: shavings:** If the wheels of a train do not pass smoothly over the track, but rather scrape against it (e.g., if the flanges of the wheels scrape against the side of a curved track), small metal particles get shaved off of the track and wheels. Accumulation of such particles can form conductive paths across the joint, lowering the electrical resistance. Some joints have an insulating coating applied to the nearby rail, so that the shavings need to form a longer path to contact both rails.
  Required conditions: Installed in curved track.
  Maintenance action: Sweep away metal shavings.

- **Joint shorted: splinters:** Splinters are thin pieces of metal scraped from the rail when a wheel skids along it. This most often occurs when an emergency brake is applied, as this can lock the wheels and make then slip until the train stops. If a splinter lands across a joint, it can create an electrical connection between the rails.
  Required conditions: None, but more frequent if installed where emergency braking occurs (mostly near stations).
  Maintenance action: Sweep away metal splinters.

- **Joint shorted: foreign object:** A conductive object is placed connecting the two rails. One frequent cause of this is vandals placing coins over a joint.
  Maintenance action: Remove object.

- **Joint shorted: shavings:** One maintenance action taken to extend the life of the entire length of rail is 'grinding'. This shaves a thin layer of steel off of the surface of the track, removing surface defects that could grow into larger defects. If cleaning is not properly performed afterwards, the metal shaved off of the rail can create electrical paths between the rails across the joint.
  Maintenance action: Sweep away metal splinters.

- **Sleeper shifted:** Maintenance of the track sometimes requires the rail to be detached from its sleepers (e.g., when a sleeper needs to be replaced). Sometimes, the sleepers are not returned to their original position, which can leave a joint insufficiently supported, resulting in increased flexing under load.
  Maintenance action: Move or install additional sleeper(s).

- **Internal insulation failure:** Sometimes, an electrical connection is measurable between the rails with no externally visible cause. In these cases, the joint is replaced and no further investigation is conducted into the cause, as such failures are quite rare.
  Maintenance action: Replace joint.

- **End post jutting out:** Due to deformation of the end post or, less commonly, the rails, part of the end post extends above the surface of the rail. This causes the wheels of passing trains to knock against them, as well as bump down against the subsequent piece of tail.
  Maintenance action: Grind away piece extending above surface.

7.1.3 Inspections and repairs

A maintenance policy provided by ProRail to carry out this case study consists of several annual inspections, followed by maintenance to repair any faults found by the inspection. This policy is referred to as the *reference policy* in this chapter.

Inspections are performed by workers walking along the track and visually inspecting (among other things) the joints for signs of damage or wear. To keep the cost of inspections down, the track remains in operation during the inspection. Unfortunately, this means that some parts of the joints (e.g., the parts on the inside of the track) are difficult to see and defects here may be overlooked. If defects are observed, more detailed inspections, such as electrical resistance measurement, may be taken when the track is taken out of service.

If faults are found, either due to failure or inspection, a corrective action is taken to correct it. Exactly which action is taken depends on the type of fault. Some faults, such as metal shavings causing a short circuit, can be immediately repaired without affecting any other failure mode. Other failure modes require a more general corrective action, such as grinding the surface of the rails, which also repairs wear of other failure modes. Finally, some failures require a complete replacement of the joint, thus repairing degradation of all other failure modes.

7.1.4 NRG-Joint

Recently, a new model of EI-Joint, called the NRG-joint, has been developed by the company Voestalpine Railpro. ProRail is considering to use this new joint

Figure 7.3: The newly developed NRG-joint [BV16].

in future installations and replacements of joints. The NRG-joint is designed to have an improved reliability compared to the previous model, at the expense of increased cost to purchase. We use FMTs to compare the expected reliability of the NRG-joint to the existing joint types.

The NRG-joint (shown in Figure 7.3) is a redesigned version of a glued EI-joint, with the following changes:

- Elongated fishplate to spread the stress on the plate.
- Six bolts instead of four, to reduce flexing when a train drives over the joint.
- Repositioned bolts to distribute stress over the bolts more evenly.

We worked with experts at ProRail to adjust the FMT of the glued joint to reflect the redesign. We examined each failure mode, and asked the expert how this mode would change in the new design. Some modes, (e.g., 'Broken bolts') were adjusted to have considerably longer expected times to failure, while other modes (e.g., 'foreign object') are unaffected by the redesign and left unchanged. As only six joints were installed as a trial at the time of our analysis, we have to trust expert judgement and cannot compare to historical failure data.

In this analysis, we included the same experts' judgement to adjust the model for the constructed joint model. We compare the reliabilities for the three joints, to check that our adjustments match reality for the models for which failure data is available.

7.2 Approach

In conducting the case study, we worked together with experts at ProRail to construct and validate an FMT of the EI-joint, determine the appropriate metrics

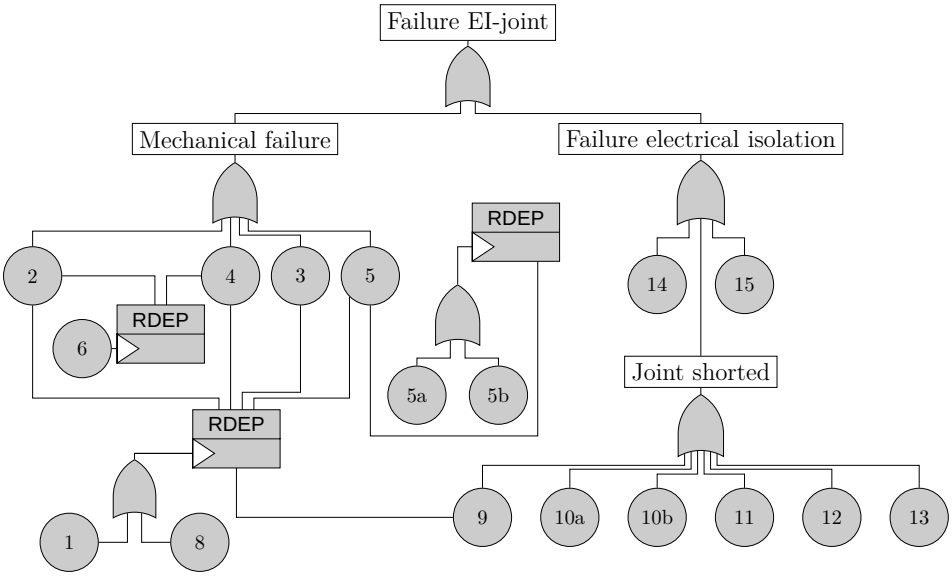


Figure 7.4: **Fault tree describing the major failure modes of the EI-joint.** The numbers in the basic events correspond to the section numbers of the failure modes. Failure modes 5a and 5b are specific causes of failure mode 5 (broken glue connection), due to manufacturing defects and installation errors, respectively. Failure modes 6, 7, and 16 have been merged into mode 6, as these are specific causes of the same fault. Failure mode 10 (short due to shavings) is separated into 10a for joints without additional protective coating, and 10b for joints with protective coating.

for evaluating the maintenance policies, and find an optimal maintenance strategy. This section describes our process in analysing the case.

7.2.1 Qualitative modelling

To construct the FMT of the EI-joint, we used existing data from ProRail's FMECA, together with interviews with experts and historical data about recorded failures. Using the FMECA to identify failure modes, and expert judgement about the relationships between these modes, we created the structure of the FMT shown in Figure 7.4. The maintenance policy was based on the reference policy described in the FMECA, as well as information obtained in meetings with experts from ProRail, from the consulting company Movares, and from the maintenance contractor Arcadis/Assetrail.

The FMECA lists the failure modes that can occur in the EI-joint, along with the worst-case consequences (i.e., derailment for mechanical failures, erroneous indications of occupied tracks for electrical failures) these modes have. The purpose of this document (in Dutch, the *instandhoudingsconcept* maintained by ProRail) is to demonstrate, together with the maintenance policy, that the risk of the failure is suitable low relative to the possible consequences. I.e., that failures that could cause derailments are entirely prevented by maintenance, while erroneous detections, as a less damaging outcome, are reduced to an acceptable rate.

We found that the FMECA is limited in its description of relationships between the failure modes, e.g., that poor geometry will accelerate other failure modes. After constructing a fault tree without these relationships, discussions with experts were conducted to introduce RDEP gates representing these relationships.

The maintenance policy follows what is described in the FMECA, and was confirmed by expert judgement. Discussions with ProRail and contractors indicated that actual maintenance varies greatly depending on location and choice of contractor, but the reference policy is a good indication of the national 'average' policy.

Upon obtaining the information needed to construct the structure of the FMT, we created an Uppaal model using the templates described in Chapter 5. Each failure modes is modelled as a basic event, and they are connected using OR and RDEP gates.

The maintenance policy is described using inspection and repair modules (described in Section 5.3). These modules proved versatile enough to model the complex maintenance policy described in Section 7.1 where periodic inspections observe the condition of all the basic events at the same time, and initiate appropriate repair modules (with many repairs modules triggerable by multiple inspections, e.g., the total joint replacement which is initiated for most mechanical failure modes). The repair modules, in turn, perform corrective actions, often on multiple BEs at once (again using joint replacement as an example, it resets the degradation of all

basic events).

7.2.2 Quantitative modelling

Apart from the structure of the fault tree and the maintenance actions, quantitative analysis requires the estimation of the degradation/failure rates of the components, as well as the timing and effects of the maintenance policy. In particular, we need the following information:

- The *mean time to failure* of each failure mode.
- The *number of phases* in the Erlang distributions governing the failure time of each failure mode.
- The *probability* of a given joint to be susceptible to each failure mode.
- The *time between inspections* for each inspection.
- The *threshold* of degradation at which action is taken following an inspection.
- The *repaired components* of each type of repair. That is, when a particular failure mode is repaired, which other components are repaired at the same time.
- The *acceleration factors* of the RDEP gates.

An expected time to failure (ETTF) is listed in the FMECA for each failure mode, however this is not sufficient for an FMT: because basic events progress through a number of degradation phases before failure, we need to include the number of these phases. In addition, the FMECA describes the conditions required for a failure mode to apply, but does not describe how often these conditions apply.

To elicit mode information, we sent a questionnaire to several experts in the railway industry. This questionnaire (included as Appendix A) asked several questions to determine the ETTF, variance of the ETTF (used to determine the number of degradation phases), frequency of the condition required for the failure, and the nature of the degradation process (linear wear, non-linear wear, or exponentially distributed).

Our conclusions from the returned answers were:

- The ETTFs from the FMECA were close to the expert estimates.
- The variances of the failures modes were estimated and included in the model using the number of phases in the Erlang distribution of the failure times.

- All the failure modes were due to either non-linear wear or exponentially distributed failures. Answers for some failure modes were contradictory (e.g., stating that the failure is exponentially distributed, and thus memoryless, but preventable by maintenance). In these cases, we used our own judgement.
- Many different failure modes require the same conditions, and the frequencies of most of these conditions are relatively easy to estimate (e.g., the number of joints installed in curved track was readily available).

Acceleration factors for the RDEPs were estimated by comparing the number of failures predicted by the FMECA to cause an acceleration to the number of failures predicted of the accelerated failure mode. Further refinements were made in discussion with experts at ProRail, in particular to resolve cases where a failure mode is accelerated by multiple causes.

Some further adjustments of the data was performed to align with historical failure data, by identifying those failure modes for which far more/fewer failures were predicted by the model (including maintenance) than were recorded. In collaboration with experts, we identified whether the deviations were likely due to incorrect estimates of the degradation parameters or due to inaccurate reporting, and adjusted our model where necessary.

Costs. Information about the costs for repairs and inspections was provided by ProRail, and corresponds to the amount ProRail pays their contractors for performing the respective action. The cost of a failure is dominated by the *social cost of unavailability*, i.e., the estimated cost to passengers and freight carriers due to delays. While the exact value is confidential, the values in our model are based on a four-hour disruption (the contractually specified time a contractor may take to correct the issue) of a moderately busy track.

7.2.3 Metrics

We analyse several aspects of the dependability of the EI-joint, which can be used to compare different maintenance policies and help in deciding which policy is better. We consider the reliability, expected number of failures, and costs:

- *Reliability.* The probability of experiencing no system failures within a given time period. We compute the probability that within a certain period, there is never a time where a set of BEs is in a failed state leading to the occurrence of the top level event of the FMT. While this is not the most interesting metric for decision-making, we use it to demonstrate the importance of the choice of inspection frequency.
- *Expected number of failures.* We compute the expected number of occurrences of the top event in a given time window. Since all failures of the EI-joint can

be repaired, there can be multiple failures over time. We can also compute the number of failures of individual components or subtrees of the FMT.

- *Costs.* We can measure several expected costs incurred by the system over time. Specifically, we consider the costs of maintenance and failures. We can further separate costs into the costs of inspections, specific maintenance actions, and failures.

7.2.4 Validation

To check whether our model accurately represents the behaviour of the EI-joint, we need to compare results predicted by the model to recorded data from the field.

As we have already used the failure data from individual failure modes to adjust the model parameters, it is to be expected that their failure rates are close to reality, and this is not a great metric for validation. Instead, we have three metrics we can use for comparison: total failure rate, replacement rate, and total maintenance cost.

The total failure rate is the number of times per year the top level event occurs. Due to the relationships between the different failure modes, this number is not the same as the sum of the rates of the individual failure modes. The accuracy of this figure thus provides support for our claim that the model reflects reality.

The replacement rate is the number of new joints installed to replaced failed/worn joints every year. As joint replacements are not included in the normal maintenance contracts, they are tracked separately. Thus, this value is available with high accuracy.

The total maintenance cost is more difficult to obtain from historical data: ProRail has different kinds of contracts with its contractors, with different pricing schemes and maintenance policies. Furthermore, exact financial information is confidential and therefore not included in this thesis. Nonetheless, we conferred with experts at ProRail who agreed that our predicted costs were close to the actual costs.

Section 7.3 shows the results of our validation, concluding that our model reflects reality well enough to make suggestions about improved maintenance policies following the model's predictions.

7.3 Analysis and results

In this section we describe the results of several experiments we conducted on the FMT of the EI-Joint. As described in Section 7.2.4, we first validated the FMT against observations from the field. Therefore, we used the model as constructed, i.e., we analysed the EI-joint under the reference policy. Since we concluded that the model is in line with the real world, we continued with finding possible

improvements of the reference policy. Therefore, the maintenance strategy within the FMT was modified by changing inspection frequencies and replacements. This led to a description of how an optimal maintenance strategy of the EI-Joint can be constructed.

The results in this section were obtained using the Uppaal-SMC analysis as description in Chapter 5. The results are averages of 40,000 simulation runs each. The variance between the simulation runs is low enough that a 95% confidence interval around the mean results has a width less than 1% of the indicated value.

7.3.1 Reference policy

To validate our model, we first estimate the total failure rate of the joint over time, shown in Figure 7.5, under the reference maintenance policy. We notice that the failure rate is mostly constant after the first few years, indicating that the per-year average is a good metric for comparison. We thus divide the total number of failures after 50 years by 50 to obtain the annual failure rate. This number is within the margin of error of ProRail's incident tracking.

Next, the expected number of occurrences of each failure mode per year was estimated. A graphical breakdown of the causes of failures is displayed in Figure 7.6. We find that approx. one quarter of the failures are due to mechanical failures, and the rest due to electrical failures. Furthermore, we can see which failures contribute most to the total failure rate (i.e., numbers 9 through 12), and which have almost no impact at all (e.g., 3 and 13, part of the 'other' block).

ProRail maintains a record of joint failures by cause, and we compare the predicted number of failures to the recorded number in Table 7.2. Since the predicted failure rate is almost constant, we assume we can multiply the expected failure rate by the number of joints to obtain the total number of failures, regardless of the age of the joints in operation.

While most of failure modes show agreement between the model and reality, some show significant differences. The difference between actual and predicted failure rates for BE 1 is likely explained by inaccurate reporting, as engineers often report only the immediate defect rather than the underlying poor geometry. BEs 2, 4, and 5 concern mechanical failures, which are typically often corrected during maintenance before the officially specified threshold is reached.

As an additional validation, we estimate how often a joint is replaced due to maintenance. Our model predicts approx. 3680 replacements per year, on a population of 50,000 joints. ProRail records indicate approx. 3000 replacement joints are installed each year. We expect that this difference is due to some failure modes where the maintenance action induces a replacement in the model, whereas in some cases in the real system the degradation may not has progressed so far, resulting in only a minor maintenance action.

Next, we consider the costs of the joint. Figure 7.7 shows the costs over the

BE	Failure cause	Predicted	Actual	Difference
1	Poor geometry	110	48	62
2	Broken fishplate	129	83	46
3	Broken bolts	2.3	2.1	0.2
4	Rail head broken out	68	30	38
5	Glue connection broken	70	37	33
6	Battered head	3.4	5.5	2.1
7	Arc damage	7	3.4	3.6
8	End post broken out	12	9.4	2.6
9	Joint bypassed: overhang	212	200	12
10	Joint shorted: shavings	156	150	6
11	Joint shorted: splinters	254	261	7
12	Joint shorted: foreign object	199	200	1
13	Joint shorted: shaving from grinding	10	10	0
14	Damage by maintenance	19	18	1

Table 7.2: **Comparison of predicted and actual failure rates of different failure modes.** Values are yearly occurrences in a population of 50,000 joints.

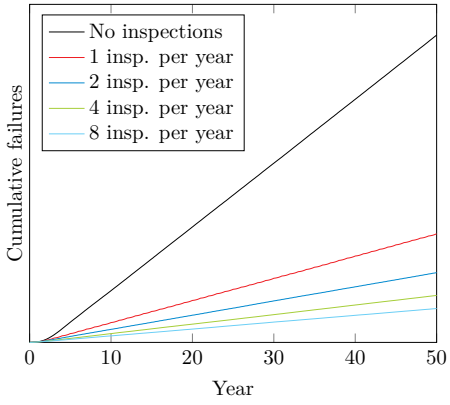


Figure 7.5: **Cumulative expected number of failures of one EI-joint over time,** for different inspection rates.

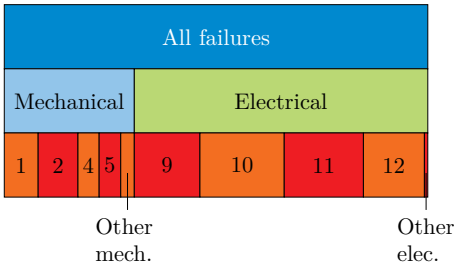


Figure 7.6: **Breakdown of failures of the EI-joint by cause.** The numbers in the bottom row indicate individual failure modes, and correspond to the numbers in Table 7.1.

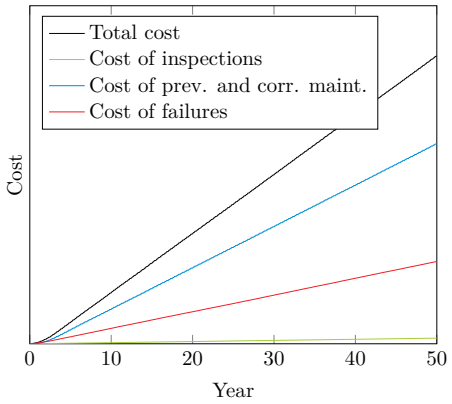


Figure 7.7: **Cumulative costs of one EI-joint over time,** split up by type of cost.

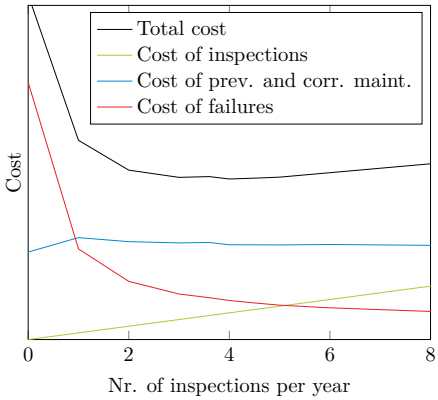


Figure 7.8: **Different types of total costs for one joint,** depending on the inspection frequency.

lifetime of the joint, separated into the cost of inspections, failures, and maintenance (including both preventive and corrective actions). As can be expected from the progression of the cumulative number of failures, also the costs progress very linearly over time. The computed values do not deviate much from ProRail's estimates.

7.3.2 Optimisation of maintenance policy

Having concluded that the model is a reasonably accurate description of the behaviour of the EI-joint, we present some options for improving the reliability and/or costs of the joint.

We consider varying the inspection frequency to reduce the total cost, and several policies for more frequent replacement of the entire joint to improve reliability. We find that it is possible to obtain considerable improvements in joint reliability, but these improvements are outweighed by the large costs needed to achieve them.

Inspection frequencies. First, we consider the possibility of performing more or fewer inspections. Figure 7.5 shows the cumulative expected number of failures over time for different numbers of inspections, and Figure 7.9 the unreliability. We see that the introduction of any inspections at all significantly reduces the number of failures, but subsequent increases of the number of inspections have a much smaller effect. This is due to failures either occurring gradually and being detected even with infrequent inspections, or occurring suddenly, and rarely being found by any inspection before failing.

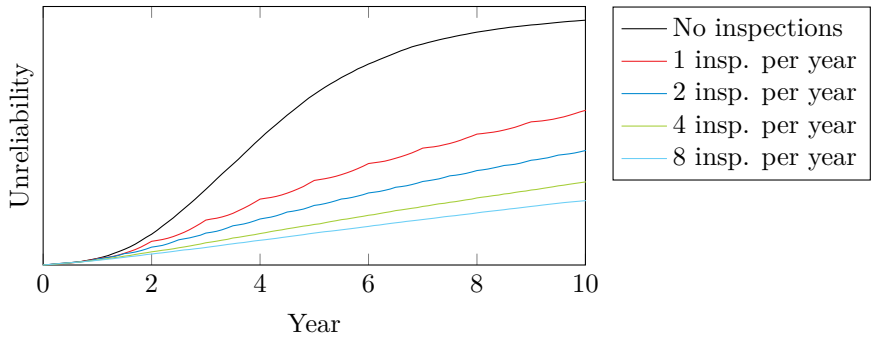


Figure 7.9: **Unreliability of the EI-joint** under different maintenance policies.

In terms of improving reliability, clearly more inspections are always better. Nonetheless, these results show diminishing returns when increasing the inspection frequency above approximately two per year.

To estimate the cost-optimal number of inspections, we plot the total cost per year for different inspection frequencies, shown in Figure 7.8. As expected, the costs of failures decrease with more inspections, while the costs of inspections increase. The maintenance costs are fairly constant over different inspection frequencies, as increased inspections do lead to more necessary repairs, only repairs performed sooner.

The optimal number of inspections in terms of total cost is found around four inspections per year. The difference in total cost between approx. 2 and 6 inspections per year falls within the margin or error of the simulation, so no more precise optimum can be determined. The reference policy lies within this optimal range, so we cannot improve it by changing the inspection frequency.

Replacements. Several other options for maintenance policies are listed in Table 7.3. We consider three options:

- Always replacing the entire joint when any maintenance is required. Since many of the failure modes have similar expected times to failure, it is reasonable to assume that the first failure indicates that more failure will follow soon. Thus, replacing after the first failure could prevent subsequent failures.
- Adjusting the inspections to take preventive action well before the reference threshold. The current policy specifies a level of degradation that, when observed in an inspection, leads to a maintenance action being performed. We consider reducing this threshold to take action at a lower level of degradation, reducing the likelihood that the degradation will cause a failure before the next inspection.

Policy	Maint. cost	Total cost	Failure frequency
Current	1	1	1
Replace instead of repair	2.20	1.65	0.76
Reduce threshold by 1/3	1.49	1.16	0.48
Replace every 5 yrs.	2.49	1.85	0.88
Replace every 10 yrs.	1.59	1.34	0.96
Replace every 20 yrs.	1.30	1.17	0.97

Table 7.3: Comparison of the effects of different maintenance policies, relative to the reference policy.

Optimal inspection frequency					**Relative cost**				
Cost	*Failure rate factor*				*Cost*	*Failure rate factor*			
factor	2	3/2	1	1/2	*factor*	2	3/2	1	1/2
1/2	8	8	5	2	1/2	0.94	0.99	0.98	0.91
1	8	8	4	2	1	0.92	0.99	1	0.92
3/2	8	6	4	2	3/2	0.92	0.96	1	0.89
2	6	6	3	2	2	0.94	0.98	0.98	0.88

Table 7.4: **Optimal inspection frequencies per year for different relative failure rates and costs, and total cost of this policy compared to the reference policy** (4 per year). All costs (i.e., inspection, repair, and failure) are increased by the same factor.

- Periodic replacements of the joint regardless of inspection result. Since the normal lifespan of a joint is known, it may be useful to replace the joint before this time to correct any (possibly hidden) degradation.

Our analysis finds that all these policies have higher total cost than the reference policy. The reduced threshold on inspections does significantly decrease failures for only a modest increase in total cost, but since total cost includes the social cost of failure, we do not consider this a net gain. It is also questionable whether all failure modes show signs of wear sufficiently early to allow this policy to be implemented.

It is likely that the failure rates of the joint vary depending on the intensity of their use. Additionally, costs of unavailability due to failure or repair increase as the number of passengers passing over the joint increases. We have not precisely determined the correlation of these effects, but we have analysed the optimal inspection frequency for several variations of costs and failure rates. The optimal inspection frequencies are listed in Table 7.4, as well as the relative cost of the optimal inspection policy compared to the previously computed optimum of 4 inspections per year.

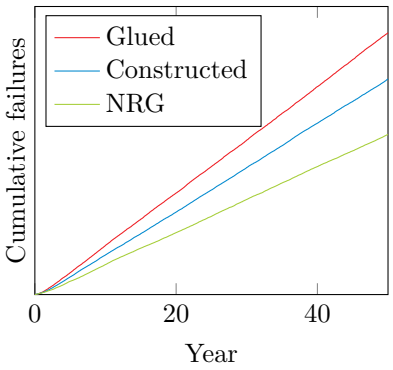


Figure 7.10: **Comparison of failure rates for the the constructed, glued, and NRG joints.**

We find that the optimal inspection frequency is determined primarily by the degeneration rate, rather than by the cost. Furthermore, the optimal inspection policy has at most a 12 percent cost saving compared to a general policy of four inspections per year.

7.3.3 Comparison to new joint model

Figure 7.10 shows a comparison of the number of failures over time of the three different joint types (glued, which is the type discussed in the rest of this section, constructed, and the new NRG-joint). As expected, the NRG-joint shows a moderate reduction in failure rate of around 25% compared to the glued joint. In comparison, the constructed joint shows an increase of around 20%. The latter is consistent with observed data, suggesting that our method for adjusting the model is valid.

In addition to the reliability analysis, we compared the expected costs of the joints under different maintenance policies. Although we cannot show the figures due to confidentiality, we observe a reduction in total cost under reference maintenance policy sufficiently large to justify the additional initial expense of the NRG joint. We further observe that the optimal maintenance policy for the new joint involves a reduced inspection frequency, yielding even greater cost savings.

7.3.4 Modelling power of FMTs

Our first research question of this chapter is about whether FMTs have sufficient expressivity to capture the degradation behaviour and maintenance policies of

real-world systems. During this case study, we were able to model most of the information available on the EI-joint using the FMT, and our validation shows that the model is sufficiently accurate to make realistic predictions.

During the case study, we found two aspects of the degradation behaviour that could not be directly included in the model: First, the reference policy provided by ProRail describes the maintenance thresholds (i.e., at what level of degradation does an inspection trigger a repair) in terms of physical observations. For example, overhang is removed when less than 3mm of separation is observed between the ends of the rails. FMTs model degradation as a more abstract 'degradation phase', which does not have an obvious correspondence to such a physical measurement.

For the EI-joint, we assumed, after discussion with experts, that the maintenance threshold is set at approximately 60% of degradation. Our validation confirms that this is a reasonable assumption. For other systems, with more varied maintenance threshold, individual threshold phases will need to be found.

A second difficulty is the acceleration factor of the failure mode 'poor geometry'. FMTs support a single acceleration factor, which applies whenever the triggering failure mode has occurred. Poor geometry does not fit exactly in this model, as the geometry can be more of less bad, with different accelerations depending on how bad the geometry is. For our FMT, we used the acceleration that, following expert judgement, corresponded to geometry that was noticably degraded, but not so bad as to warrant immediate corrective action.

7.4 Conclusion

7.4.1 Conclusions on EI-joints

We have modelled and analysed several maintenance policies for the EI-joint via fault maintenance trees. Our analysis concludes that a maintenance policy of four inspections per year is near the optimal policy, and that small variations in this policy have little effect on the total cost. We also find that qualitatively different maintenance policies may increase reliability, but this effect is outweighed by the increased maintenance costs. Finally, we find that the newly designed NRG-joint is a cost-saving replacement for the current glued joints.

One may wonder how surprising it is that the reference maintenance strategy is cost optimal under the existing circumstances. We argue that it might not be so, because the EI-joint is a well-understood railroad element. Nevertheless, our analysis has provided useful insights in the degradation behaviour of the joints, for instance in critical accelerating factors.

Answering the research questions in the introduction of this chapter, we conclude:

1. Section 7.2 describes the process of modelling the EI-joint and its maintenance

policy in an FMT. The maintenance policies applied by ProRail can be accurately modelled in the FMT, and the inspection and repair modules are a good match to the visual inspections and preventive/corrective maintenance actions prescribed.

2. Section 7.2.4 describes our process of validating our FMT of the EI-joint against its actual dependability. Section 7.3.1 shows that the predictions of our model are a good match to its actual behaviour, as recorded by ProRail. Based on this, we expect that the model's prediction with different maintenance policies will also be sufficiently accurate to justify recommendations about these policies.

3. Section 7.3.2 shows our FMT's predictions about the effects of different maintenance policies. We find that the reference policy is already within the cost-optimal range with respect to the inspection frequency, and we cannot recommend more cost-effective policies. We do find several variations on the policy that give better reliability, but at a higher total cost.

4. We show a comparison of the new NRG joint to the currently installed EI-joints in Section 7.3.3. We find that the NRG-joint offers better reliability under the current maintenance policy, and that its cost-optimal inspection frequency is lower than that of the current joints. We also find that the total cost of the NRG-joint is lower than that of the current joints, although we cannot show this result due to confidentiality.

Discussion and future work. In summary, we have demonstrated that FMTs can be applied to this industrial setting, and can provide valuable insights into the effects of different maintenance policies in terms of reliability and cost. Since EI-joints exemplify many aspects typically found in industrial systems in terms of degradation behaviour and maintenance policy, we expect that they may be fruitfully applied in many other applications.

Future work includes the extension of FMTs to include continuous degradation processes rather than the current discrete degradation phases. This extension could provide a more natural description of the degradation in terms of the underlying physics (e.g., describing overhanging rail surface in 'millimetres of overhang'), as well as a more natural and easier to understand maintenance description (e.g., removing overhang when exceeding '3 millimetres' rather than 'phase 2').

Another avenue of future work is to extend FMTs to take into account specific environmental and usage conditions that affect degradation. The current model is based on average degradation rates, where a more detailed model could suggest different maintenance policies depending on local traffic intensity, soil conditions, etc.

Chapter 8

FMTs in practice: Analysis of the pneumatic compressor

In this chapter, we demonstrate the applicability of FMTs to an industrial case study: The trainboard pneumatic compressor. This study was performed in close cooperation with NedTrain.

Compared to the EI-joint discussed in the previous chapter, the compressor has a more complex maintenance policy: Several types of inspections and overhauls are performed at different times, ranging from a simple visual inspection to a complete overhaul. The degradation behaviour is also more complex, with components experiencing non-linear acceleration of wear rates from the wear of other components.

Problem Statement. The goal of this case study is to explore whether FMTs can be usefully applied in NedTrain's reliability engineering efforts of the pneumatic compressor. In particular, we examine the following research questions:

1. Are FMTs sufficiently expressive to capture the degradation behaviour and maintenance policies of the compressor?

2. Can FMTs provide useful information with sufficient accuracy to make recommendations about its maintenance policy?

3. How do variations of the maintenance policy affect the reliability of the compressor?

Case description. Pneumatic compressors (shown in Figure 8.1, schematic in Figure 8.2) are devices that produce compressed air, used in trains to power systems such as doors and brakes. As they are critical to the operation of the train, they are also potential causes of disruptions. Various causes of failure that occur that must be prevented through periodic maintenance, lest the train become stranded resulting in high costs due to disruption of the train schedule and delayed passengers.

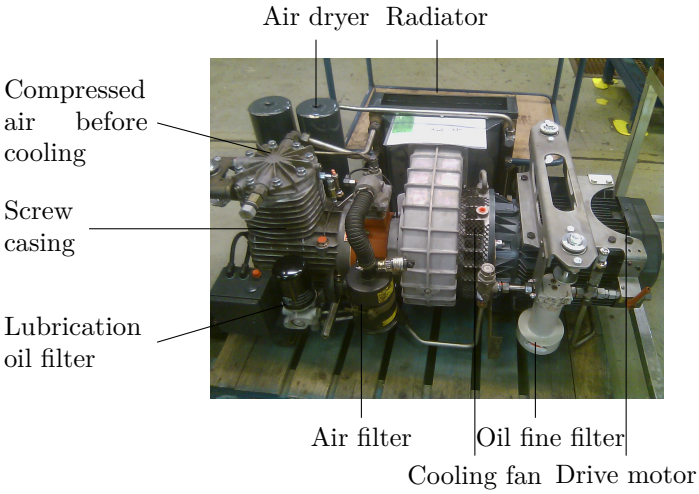


Figure 8.1: **A pneumatic compressor.** Air is drawn in from the environment via the air filter and compressed by a set of screws. The compressed air is then cooled, and moisture and oil particulates are removed before the air enters the pneumatic system of the train. [1]

The compressor is a relevant case study for three reasons: (1) The analysis is useful for NedTrain's internal operations for logistics and maintenance engineering purposes; (2) the failure behaviour of the compressor in well documented through internal documentation and historical failure data; and (3) maintenance of the compressor is performed relatively independently of the rest of the train, as the compressor can be replaced by a (recently maintained) one from stock, which gives more freedom to optimise the maintenance program.

In this case study, most of the modelling work was performed by NedTrain experts, while the translation to FMTs and their analysis was performed by us.

We compare the dependability and costs of compressors subject to different maintenance policies. This allows us both to validate the model against actual recorded failures, and to offer suggestions for improvements in the policy that lead to cost savings or increased dependability.

Our analysis find that FMTs are a useful tool for analysing and optimising the maintenance policy of the compressor. We obtain dependability estimates which agree with historical failure data, and identify possible avenues of improvement of the maintenance policy. In particular, we find that (1) one of the maintenance actions has a substantially greater effect on system reliability than the others, and

[1] Image © NS, 2016.

(2) a currently scheduled overhaul may not be cost-effective.

Origin of this chapter. This chapter describes the case study published in:

- Enno Ruijters, Dennis Guck, Peter Drolenga, Margot Peters, and Mariëlle Stoelinga. "Maintenance analysis and optimization via statistical model checking: Evaluation of a train's pneumatic compressor". In *Proceedings of the 13th International Conference on Quantitative Evaluation of SysTems (QEST)*, volume 9826 of *Lecture Notes on Computer Science*, pages 331–347. Springer, August 2016. DOI: 10.1007/978-3-319-43425-4_22, ISBN: 978-3-319-43424-7.

This case study was performed as part of an internship by an MSc student at NedTrain, published (with more details on the internal process at NedTrain) in:

- Peter Drolenga. "Fault maintenance tree analysis in train systems", 2015. In: M. Peters, B. Huisman, and S. Hoekstra (supervisors) Industrial internship report.

Acknowledgement. This case study was performed in collaboration with NedTrain; in particular, most of the modelling was performed by Peter Drolenga as part of his internship at NedTrain, and Margot Peters.

Organisation of this chapter. In Section 8.1 we explain how the compressor works and how it is maintained. Section 8.2 describes to process applied by NedTrain and us to model and analyse the compressor. Section 8.3 shows the results of the analysis, and Section 8.4 presents our conclusions.

8.1 Case description

Pneumatic systems have long been used as a control mechanism in trains. Braking systems operated by air pressure date back to 1869 [Wes69], and are still in use today. Although electronics are starting to replace or supplement pneumatic control, modern trains still use pneumatics for emergency brakes and other applications, such as opening and closing doors automatically and raising the pantograph to connect to the overhead electrical line.

Many systems operated pneumatically are safety-critical, and these are designed to be fail-safe: A loss of air pressure disrupts their functionality, but poses no danger. Brakes, for example, are loosed by high pressure and applied when the pressure drops. A failed compressor, therefore, does not constitute a safety risk. Nonetheless, since a failed compressor leaves the train stranded, such failures cause costly and lengthy disruptions.

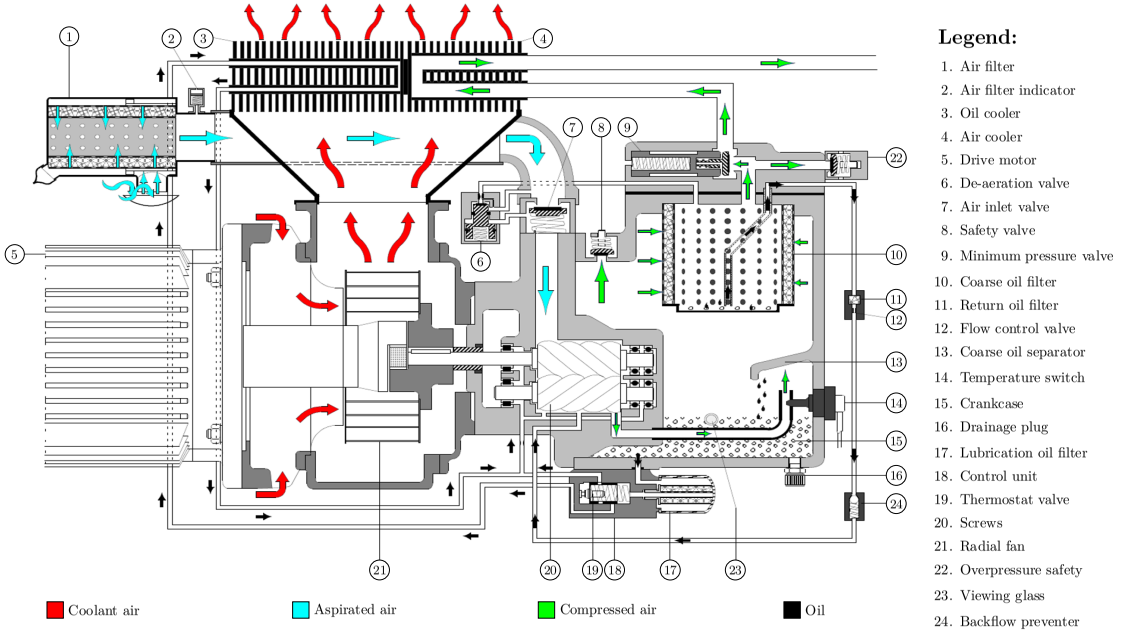


Figure 8.2: **Schematic view of the components of the compressor.** Modified from [Ned14].

As these compressors are critical to the operation of the train, they are also a potential cause of disruptions. Various types of failures can occur, such as oil leaks and clogged filters. Inspections are performed to determine whether failures are likely to occur soon, and preventive action, such as replacing a nearly-full filter, can be taken to prevent the failure occurring in the field. Some components such as filters are also periodically replaced, since replacing them all in one service is cheaper than spreading the replacements over multiple services when inspections find a problem.

8.1.1 Purpose and operation

To provide high-pressure air for the pneumatically operated systems, modern trains use electric compressors. In addition to generating a high pressure, the compressor also clears the air of dust and debris, and removes moisture which could cause corrosion or freezing in pipes and pneumatically-powered devices.

We examine the particular model of compressor used in Dutch VIRM (*Verlengd InterRegioMaterieel*) trains of series 1, 2, and 3. This compressor operates using rotating screws that take air from the outside and compress it into a pipe. Before reaching the screw, the air first passes through a filter to remove any dust or debris. The screw is lubricated using oil. Due to the relatively high temperatures and airflow, micro-particles of oil are carried in the airflow through the system. To remove this oil, the air passes through two additional filters. Finally, the air is cooled and passed to the pneumatic system.

A schematic of the compressor with its airflow can be seen in Figure 8.2. Air enters via a filter (1) due to pumping action of the screws (20). As the air passes through the screws, droplets are carried by it to the high-pressure side of the screws (15), these are removed by the oil separator (13) and oil filter (10) before the air is cooled (4) and passes to the rest of the pneumatic system.

Various pressure-activated valves ensure that the system is not damaged by over- or under-pressure and that the pressurised air does not flow back out through the compressor when it is switched off. A thermostatic valve (19) causes some oil to flow through a radiator to maintain an optimal temperature. Should this cooling mechanism fail (e.g., if the radiator is obstructed), the temperature switch (14) will disable the compressor when the oil/air temperature gets too high.

Failure modes

Based on documentation of failure characteristics and expert opinions of system engineers and mechanics, the structure of the FMT was constructed. The resulting FMT is displayed in Figure 8.3.

As shown in the FMT, compressor failures can be divided into two categories: *Complete failures* where the compressor does not operate at all, and *degraded*

operation where the compressor does not generate a sufficiently high pressure. This division helps validate the model, since these categories of failures are easy to distinguish in a practical fault condition.

For this chapter, we consider only failures that prevent the train from operating, meaning complete failure or so much degradation that immediate repair is necessary. Other forms of degraded operation can be analysed similarly.

More details on the particular failure modes are listed below:

- **Motor does not start at appropriate pressure:** The compressor should automatically activate when the pressure in the pneumatic system drops below a threshold value. If the activation sensor fails, pneumatic pressure may fall below required levels, resulting in a stranded train.

- **De-aeration valve defective:** When the compressor is started, pressure on the high-pressure side must be relatively low. A de-aeration valve therefore lets compressed air escape when the compressor is switched off. If this valve is defective and the pressure is too high at the next start, a safety relay detects that the motor is drawing too much power and switches it off.

- **Two starts in short time:** In most cases when the above-mentioned de-aeration valve has failed, air can still slowly leave the compressor and allow it to start later. Thus, a defective valve only causes a failure if the subsequent start is within several seconds of the compressor switching off.

- **Radiator obstructed:** An obstruction due to e.g., dust of the radiator will prevent adequate cooling, eventually leading to a shutdown to prevent over-temperature damage.

- **Oil thermostat defective:** A defective thermostat will lead to insufficient/excess cooling of the oil. Excess cooling will reduce the efficiency of the compressor, but not cause a failure. Insufficient cooling, on the other hand, will lead to an over-temperature safety shutdown.

- **Low oil level:** Insufficient oil has two main consequences: increased wear of the screws, and increased temperature (since the oil is also used as coolant). In practice, the compressor usually experiences an over-temperature shutdown before much additional wear is incurred.

- **Pressure valve leakage:** Several pressure valves ensure that the compressor produce such a high pressure that other pneumatic systems are damaged. If one or more of these valves do not close properly, air can leak out, eventually preventing the system from reaching an adequate pressure to operate the pneumatically-powered systems.

- **Air filter obstructed:** If an obstruction (e.g., dust), prevents airflow through the intake filter, output pressure will also be reduced. If the airflow is severely hampered, too little compressed air will be produced to keep the pneumatically-powered systems operational.
- **Degraded air filter:** Damage to the air filter can allow particles of dust and debris to enter the compressor. As the screws grind against these particles, the screws experience increased wear.
- **Particle-induced damage:** This is wear caused by particles in the intake air scraping against the screws (accelerated by the above-mentioned degradation of the air filter, although even properly filtered air will contain some small particles). If the screws become too worn, air can leak out past them, reducing the pressure that can be generated.
- **Oil pollution:** Damage to the oil filter can allow particles to pass into the screws via the oil. Similar to particles in the air, these eventually damage the screws.
- **Lubrication-induced wear:** The screws require an adequate amount of clean lubrication oil to function effectively. If too little oil is available, or the oil is polluted, the screws cannot turn smoothly and the moving parts wear out.
- **Motor/bearings degraded:** An electrical motor acts as the power source for the compressor, and is lubricated by the same oil that lubricates the screws. If this motor fails (often due to failure of the bearings inside), the compressor will produce no/insufficient compressed air.
- **Oil fine filter saturated:** The oil fine filter is located outside the compressor, and traps any remaining small droplets of oil present in the airflow. If this filter becomes saturated, the rest of the pneumatic system may eventually be damaged by escaping oil particles. If the filter is saturated when the system is inspected, the compressor is subjected to a preventive overhaul to check for damage.
- **Degraded capacity:** During some inspections, mechanics measure the time needed to bring the pneumatic system to operating pressure under controlled conditions. If this time is too long, the compressor is preventively subjected to an overhaul.

Table 8.1 lists the types of failure that can occur, together with their failure parameters: Each failure mode is characterised by the expected time to failure assuming no maintenance is performed, and the number of degradation phases we consider in our model.

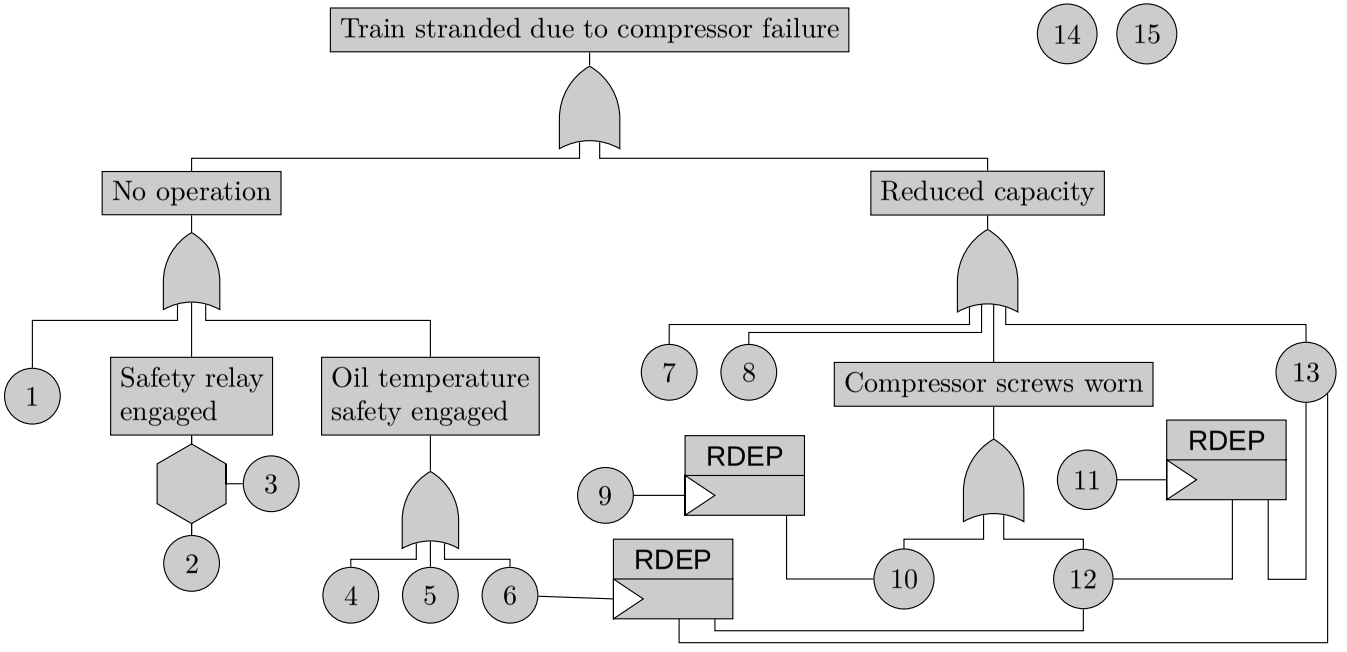


Figure 8.3: **Fault Tree describing the major failure modes of the compressor.** The numbers in the basic events correspond to the numbers of the failures modes in Table 8.1.

Nr.	Failure mode	Nr. of phases	ETTF
1	Motor does not start at appropriate pressure	3	16.6
2	De-aeration valve defective	3	200
3	Two starts in short time	2	0.001
4	Radiator obstructed	4	5.5
5	Oil thermostat defective	3	16.6
6	Low oil level	4	5.5
7	Pressure valve leakage	3	3.3
8	Air filter obstructed	2	500
9	Degraded air filter	4	5
10	Particle-induced damage	4	120
11	Oil pollution	4	5.5
12	Lubrication-induced wear	4	120
13	Motor/bearings degraded	4	120
14	Oil fine filter saturated	3	30
15	Degraded capacity	2	10

Table 8.1: **Parameters of the failure modes of the compressor.** The failure times of the components follow an Erlang distribution with the indicated number of phases and total expected time to failure (in years) assuming no maintenance is performed. The values have been scaled for anonymity. Failure mode 3 is not strictly a failure, but rather an event that is required for mode 2 to lead to failures. Also failure modes 14 and 15 are not failures, but rather indicators of degradation that are used to initiate maintenance actions, as described in Section 8.1.

	State BE 6			
State BE 11	0	1	2	3
0	1	2	4	6
1	2	4	6	10
2	4	6	10	15
3	6	10	15	30

Table 8.2: **Specification of the acceleration factor of BEs 12 and 13,** depending on the states of BEs 6 and 11. The non-degraded state is state 0, the failed state is state 3.

While modelling the compressor, it was noted that several failure modes are related to each other, such as degradation of the air filter leading to increased wear of the screws as particles pass through the filter and reach the screws. While it is possible to model these independently as 'particle-induced wear under normal condition' and 'particle-induced wear with ruptured filter' (since a degraded air filter is not by itself a cause of failure), this leads to difficulty when describing the maintenance policy. The RDEP gates offer a much more natural description of a single BE with degradation that is accelerated by another BE.

The wear of the compressor screws and the motor and bearings is complicated due to multiple causes. Particles can enter the compressor despite the filter, which causes degradation of the screws. The rate at which particles pass through the filter is significantly increased if the filter is already worn. A second mode of wear is caused by insufficient lubrication of the screws and of the motor. This can be caused by pollution of the oil, or by insufficient oil, or a combination of both. Special variants of the RDEP gate are used that capture the simultaneous but non-linear accelerating effects of filter and oil degradation. Table 8.2 specified how much the affected BEs are accelerated depending on the states of the triggering BEs.

One behaviour that was not included in the model is the low oil level, which can be accelerated by oil leaks in several components. Since it is unlikely that multiple such leaks occur at the same time, we instead chose to model the oil pressure as a single BE.

The parameters of the BEs are listed in Table 8.1. The failure rates were obtained by consultation with experts within NedTrain, specifically system engineers and mechanics, to include both theoretical estimates and practical information. The estimates were further informed by experiments conducted at the overhaul facility operating a compressor in a simulated environment.

8.1.2 Maintenance

The current maintenance policy followed by NedTrain consists of some specific inspections every two days, and scheduled services every three months with a larger service every nine months. A minor overhaul is performed every three years and a major overhauls every six years (for reasons of confidentiality, these times have been scaled with the same factor as the BE failure rates).

The bi-daily inspection is mostly performed at night, while the train is prepared for service. Mechanics check the on-board diagnostic system for recorded events such as overpressure, and perform an inspection to find oil leaks or excessive noise. If this inspection finds a defect, an unscheduled service is necessary to correct it.

During the scheduled services, consumable parts such as filters are replaced, and components of the compressor are inspected for signs of wear. Some functional tests of the overall performance of the compressor are also performed, such as measuring the time needed to pressurise the pneumatic system for the entire train starting from atmospheric pressure.

Every three years, the compressor is removed from the train and shipped to NedTrain's component workshop for an overhaul. Minor and major overhauls are alternated. During an overhaul, the compressor is disassembled and all components are examined and replaced if needed. During a minor overhaul components with a small amount of wear are reused. During a major overhaul, all worn components are replaced, and the compressor is considered as good as new afterwards.

Each maintenance action can also lead to more intensive services if problems are found that cannot be corrected during the scheduled service. For example, if a minor service inspection finds that the compressor is not producing sufficient pressure but cannot find the cause, the compressor can be sent in for an overhaul.

NedTrain has specified the current maintenance policy, which is based on a balance between performance, risks, and costs. The specification of this policy consists of the frequency with which each maintenance action must be performed, and for each BE and degradation level the effect of the action. The fault tree and maintenance plan described in Sections 8.1.1 and 8.1.2 were constructed by the research department of NedTrain.

Most maintenance actions return various components to the undegraded state if they are found in a certain degraded state. This is modelled using separate inspection and repair modules for the different BEs. For example, as shown in Table 8.3, an inspection module inspects BE 11 every month checking whether it has reached phase 3 and if so, repairs it. Some repair actions, in particular the major overhaul, are initiated when other maintenance actions find excess wear. In this case, the BE is modified to have multiple inspection thresholds for different inspection modules.

The current model makes a few assumptions: First, we assume that all maintenance is carried out exactly on schedule. In practice, maintenance actions with

scheduled intervals greater than one month are sometimes performed in the last 10 - 20% of the interval, to optimise allocation of resources. Since the fluctuations in inspection times are small compared to the inspection interval and do not occur often, we expect this assumption not to significantly distort the results.

We also assume that inspections are perfect, i.e., an inspection always leads to a repair if the degradation level is past the threshold. While this may seem questionable, we argue that the actual inspections are performed well enough that this is not a significant source of error in the model. Moreover, we assume that repairs occur instantly. Since the degradation rates already factor in that the compressor is not in use all the time, we consider it reasonable to also factor in the relatively short time spent in repair.

In the FMT, inspection modules describe the inspection rates and the threshold at which corrective action is performed. Different BEs have different thresholds, depending on the visibility of the degradation of a component and the importance of correction.

Quantitative parameters on degradation patterns and parameters were estimated based on interviews with maintenance engineers responsible for the maintenance plan and system engineers specialised in pneumatics, as well as experiment reports of a simulation environment where compressors can be tested.

While FMTs support arbitrary failure time distributions, determining the exact distribution of each BE was beyond the scope of this case study. Instead, we have modelled the BEs as exponential distributions or Erlang distributions with few phases, as these overestimate the number of failures in the relatively short times between maintenance actions. Due to the very high cost of failure compared to maintenance, relying on a conservative model and performing more maintenance than required is preferable to using an optimistic models and experiencing more failures in the field.

While describing the maintenance policy, we found two properties of the system that are used in maintenance scheduling (BEs 14 and 15), which are in fact complex properties influenced by the degradation of most basic events. Since the exact effect is too complex to include in the model, we instead treat these as basic events that do not contribute to the top level event but are included in the maintenance policy.

8.2 Approach

The modelling approach used for the pneumatic compressor is very similar to that of the EI-joint, see Chapter 7.

Most of the modelling process was performed by NedTrain, so we do not go into great detail here.

BE	Phase	Maintenance action	Result phase
1	2	M1/M2	1
1	2	O1	1
2	2	O1	1
3	2	Any	1
4	3	M1/M2	2
4	Any	O1	1
5	2	M1/M2	O2
5	2	O1	1
6	Any	M1/M2	1
6	Any	O1	1
7	2	I1	1
7	2	M1/M2	1
8	Any	M1/M2	1
8	Any	O1	1
9	Any	M1/M2	1
9	Any	O1	1
11	3 or 4	M1	1
11	Any	M2	1
11	Any	O1	1
13	2 or 3	M1/M2	1
13	2 or 3	O1	1
14	2	M1/M2	1
14	3	M1/M2	O2
14	Any	O1	1
15	2	M1/M2	O2
15	Any	O1	1
Any	Any	O2	1

Legend:

- **I1:** bi-daily inspection
- **M1:** three-monthly maintenance
- **M2:** nine-monthly maintenance
- **O1:** minor overhaul
- **O2:** major overhaul

Table 8.3: **Maintenance description for the compressor.** Given a BE, a phase of degradation, and a maintenance action, the table lists the effect of that action on the degradation of the BE. I.e. the last column lists the phase to which the BE moves when the given action is performed while the BE is in the listed phase. If the top event occurs, and after some maintenance actions denoted with result 'O2', a large overhaul is immediately performed resetting all components to their undegraded state.

8.2.1 Qualitative modelling

The failure modes of the compressor were established by NedTrain, following internal documentation, documentation from the manufacturer, and consultations with the system engineer and technicians responsible for the pneumatic system [Dro15].

A complication that arises in the compressor that was not present in the EI-joint, is that multiple partial failures could combine to cause a system failure. Where the failure modes of the EI-joint were mostly working or failed, many components of the compressor can operate at a reduced capacity without immediately causing a failure. For example, worn screws could result in lower, but still adequate, output capacity, which can combine with a slightly leaky valve to cause an unacceptable pressure drop.

The FMT does not model such combinations of partial failures for two reasons: First, most failures can be attributed to a single (dominant) failure mode, even if another provides some small contribution. A model with independent failure modes is thus considered to adequately reflect reality. Second, quantifying the combined effects of degraded components was considered too complex, and beyond the scope of this case study.

One combination of partial failures was included: The combined accelerating effects of low oil level and oil pollution on the wear of the screws and motor. This was included because a combination of these accelerations is relatively common, and the acceleration model used by NedTrain (specifying an acceleration factor based on the sum of the degradation levels of the input BEs) is easy enough to model.

The maintenance policy requires additional information that is not needed for the failure model of the compressor: One maintenance action is to perform a preventive overhaul in response to observations of an oil filter outside the compressor (which does not contribute to compressor failures). Another is a regular functional test of the entire compressor, the outcome of which is determined by the precise state of all the various components.

The oil filter saturation and functional test depend on complex interactions between component degradations in the compressor, which are considered beyond the scope of this case study. Since the additional maintenance actions may still affect compressor reliability, the FMT includes the filter and functional test as independent BEs, thus effectively making these maintenance actions (stochastically) time-based.

8.2.2 Quantitative modelling

A full modelling of the failure rates of the components was considered beyond the scope of this case study. Instead, the rates were estimated on order of magnitude

[Dro15]. As this step was performed entirely by NedTrain, we cannot provide details of how these estimates were obtained.

The failure rates were estimated for an 'average' usage profile of a compressor. Since this model of compressor is applied in a single type of train (the VIRM), most compressors will exhibit a similar usage, so this 'average' profile is justified. We also assume that a compressor that is out of service (while stored in a warehouse until it needs to be installed in a train) does not degrade, so our unit of time is 'years of operation', not necessarily calendar years.

8.2.3 Metrics

Our analysis of compressor focuses on the dependability, as information regarding costs was not available. Two metrics are the main drivers of NedTrain's maintenance policy: expected number of failures and unplanned maintenance events:

- *Expected number of failures.* We compute the expected annual number of occurrences of the top event 'Train stranded due to compressor failure'. Such failures are very costly, as they interrupt train service on the affected track for several minutes to hours, and may require evacuation of the passengers of the affected train. NedTrain's estimate of the costs of such a failure are over 100,000 euros [Dro15].

- *Unplanned maintenance events.* If one of the planned maintenance actions/inspections finds a defect that cannot be immediately repaired, the compressor is removed from the train and replaced by a functioning one, and the failed compressor is sent away for repairs. While such a repair does not normally affect train service, it does remove the affected train from service for some time and thus requires a reserve train to be used. The more often this occurs, the larger the reserve fleet must be. It is thus preferable to reduce unplanned maintenance events if possible.

8.2.4 Validation

As the main purpose of this case study was to examine the modelling capabilities of FMTs with regards to NedTrain's failure models and maintenance policies, little empirical validation was performed. The failure rates were estimated to around an order of magnitude, and the resulting system failure rate and number of unplanned maintenance events was compared to the actual rates (see Section 8.3). While the results were within the expected range, the high uncertainty of the inputs makes is impossible to perform a more precise validation.

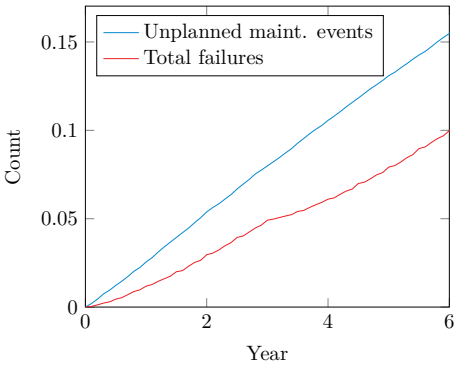


Figure 8.4: **Number of compressor failures and unplanned maintenance events over time.** Note that each failure also causes unplanned maintenance, but these are not included here.

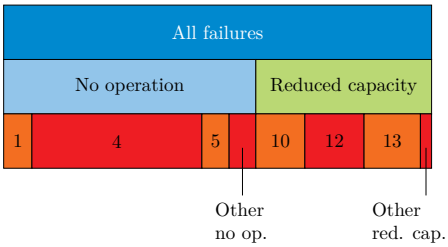


Figure 8.5: **Breakdown of the failures of the compressor by cause.** The numbers in the bottom row correspond to the failure causes in Table 8.1.

8.3 Analysis and results

In this section we describe the results of several experiments we conducted on the FMT of the compressor. As a first step, we have validated the FMT using the current maintenance policy against observations from the field. Therefore, we used the model as constructed, i.e., we analysed the compressor under the current policy. Since we concluded that the model is in line with our expectations based on failure data, we continued with finding possible improvements of the current policy. Therefore, the maintenance strategy within the FMT was modified by changing inspection frequencies and replacements. This led to a description of how an optimal maintenance strategy of the compressor can be constructed.

The results in this section are averages of 10,000 simulation runs each. The variance between the simulation runs is low enough that a 95% confidence interval around the mean results has a width of less than 5% of the indicated value, both with the original and the anonymized values. The analysis required approx. 12 CPU-hours per model on a dual 2.26 GHz Intel® Xeon® E5520 processor and 24 GB of RAM.

First, we estimate the total failure rate of the compressor over time. We consider NedTrain's fleet of 239 compressors since this model of train began operation in 1994 until 2015. Due to long periods of time when compressors are kept unused in a warehouse, a direct comparison to the model is not possible. Nonetheless, comparing the models prediction to the NedTrain's predicted annual failure rate of compressors in operation, the model's prediction is in agreement to within 50%.

Failure cause	Failure rate
Motor does not start when asked	0.41
De-aeration valve defective	0.025
Radiator obstructed	2.48
Oil thermostat defective	0.40
Low oil level	0.34
Pressure valve leakage	0.22
Air filter obstructed	0
Particle-induced rupture	0.71
Lubrication-induced wear	0.86
Motor/bearings degraded	0.82

Table 8.4: **Listing of the expected failure rates of different causes of compressor failures.** Values are yearly occurrences in a population of 233 compressors.

A graph of the cumulative number of failures over time is shown in Figure 8.4. We observe that the unplanned maintenance events increase almost linearly with time, as they are mostly caused by failures that are not wear-related, and thus occur with exponentially distributed failure times. We only consider the interval between two major overhauls, since the compressor is expected to be as good as new after a major overhaul.

Other maintenance policies. To examine the leading causes of failures, the expected number of occurrences of each failure mode per year was estimated. Table 8.4 shows the annual number of expected failures, averaged over the six-year period between major overhauls. A graphical breakdown of the causes of failures is displayed in Figure 8.5. We see that the failure mode 'radiator obstructed' is by far the leading cause of failure. The current maintenance policy for the radiator is to remove large obstructions when found during visual inspections, and more thoroughly clean it during larger maintenance operations. Our analysis suggests that more frequent cleaning may cheaply reduce failures, although we note that these failures are also usually quickly and cheaply resolved when they do occur.

Next, we consider two possible variations to the maintenance policy: Figure 8.6 shows the number of failures over time for different frequencies of the minor service. We find that this service has a significant effect on the expected failure rate. It is therefore useful to carefully examine the costs associated with this service, to find an optimal balance between servicing and failure costs.

We also consider the possibility of omitting the minor overhaul after three years, and of omitting the major overhaul after six years (instead performing a

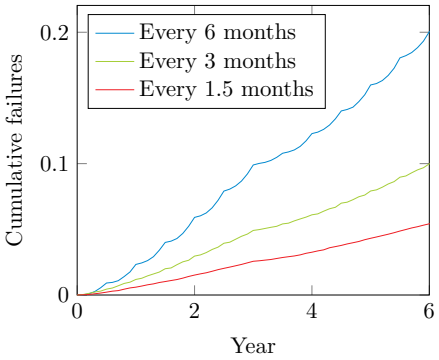


Figure 8.6: Effect of different frequencies of the small service.

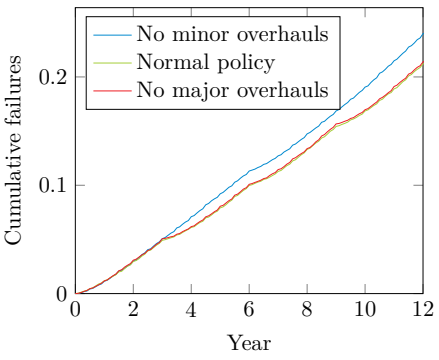


Figure 8.7: Effect of the minor and major overhauls.

minor overhaul at this time). The effects of which are graphed in Figure 8.7. After six years, the minor overhaul has prevented approx. 0.02 failures per compressor. This suggests that the overhaul may not be cost-effective, although this depends strongly on the relative costs of the overhaul and the failure. Furthermore, the effects of replacing the major overhaul by a minor one are too small to be measured by our approach, offering a further possibility for cost savings. We do note, that although we have no indications that the degradation behaviour will be noticeably different after six years, we do not have the data to prove that nonlinear effects such as metal fatigue will not cause more unexpected failures.

Modelling power of FMTs. One of the aims of this case study was to investigate whether FMTs can capture the degradation and maintenance behaviour of the pneumatic compressor. Based on the accuracy of the FMT's predictions under the current maintenance policy, we find that the FMT models most of the relevant behaviour, and that this is sufficient to make predictions about other maintenance policies.

The maintenance policy documented by NS/NedTrain for the compressor easily fits the framework of FMTs, with multiple inspection and repair modules modelling the different types of inspections and overhauls. Also the condition-dependent effects of repairs can be modelled by separating the repair into an inspection and a repair module.

Regarding the degradation of the compressor, two aspects do not easily fit the FMT model: First, the wear of the screws and the motor and bearings is more complicated than can be described in a normal FMT. This degradation is accelerated by (partial) failure of the air filter, oil pollution, and lack of oil. Combinations of these accelerations do not follow the multiplicative effect that

would be modelled by multiple RDEP gates. Therefore, a modified RDEP gate was used with an acceleration factor that depends on the state of multiple trigger events.

Second, the failure mode 'Low oil level' is caused by oil leaks, which can occur with varying severities. A major leak can quickly drain most of the oil, while a small leak causes a negligible increase in the normal rate of oil loss. Including an event 'Oil leak' would not include these different severity levels. As major oil leaks are very rare, we excluded from our analysis, and modelled the effects of small oil leaks as part of the normal degradation rate of the oil level.

8.4 Conclusion

8.4.1 Conclusions on the compressor

We have modelled and analysed several maintenance policies for the compressor via fault maintenance trees. Our analysis concludes that the FMT provides a sufficiently accurate model of the wear and maintenance of the compressor to be useful in its reliability engineering. Concretely, we find that the currently scheduled overhauls have only a small effect on the expected number of annual failures, and are therefore possibly not cost-effective.

Answering the research questions from the introduction of this chapter, we conclude:

1. Section 8.2 describes the modelling process used to obtain the failure and maintenance models, together forming the FMT, of the compressor. While some aspects of the degradation were excluded from the model (notably, the possibility of multiple partial failures combining to cause a system failure), these are believed to be of little impact on the results. This is confirmed by the accuracy of the FMT's predictions under the current maintenance policy.

2. Section 8.2.3 describes the metrics computed in our analysis, and Section 8.3 shows our results. We were able to compute the requested metrics of expected annual number of failures and unplanned maintenance events, demonstrating that FMTs can yield useful metrics. Our validation shows that computed metrics are sufficiently accurate under the current maintenance policy that the predictions for alternative policies are likely also accurate.

3. Section 8.3 shows that the periodic service currently scheduled for every three months has a strong effect on the reliability of the compressor, and is thus a good candidate for optimisation. We also note that the scheduled overhauls have only a small effect on the reliability.

Discussion and future work. In summary, this case study has demonstrated that FMTs can be fruitfully applied to the pneumatic compressor. Furthermore, the similarity between the modelling elements of the compressor and the EI-joint strengthen our belief that these systems are representative of the degradation and maintenance behaviours of systems in the railway industry and beyond. We therefore expect that FMTs can be a useful tool in the analysis and optimisation of maintenance policies in other systems as well.

With regard to the compressor in particular, one aspect was omitted from the analysis to limit our scope: In practice, compressors are installed in a train, operate until maintenance is required, removed for maintenance, and then stored in a warehouse until installed in another train. Thus, a train may have a compressor installed with a certain level of wear, and have it replaced by a much newer/older compressor during maintenance. Our metrics are computed per compressor, and it would be interesting to examine whether this warehousing behaviour can be included to draw useful per-train results.

Next, our model describes failure rate in terms units of 'per operating year' under a normal usage profile. A more detailed description in terms of actual use could allow the FMT to be used to analyse the expected future behaviour of a particular compressor if its actual use is recorded. Such an analysis could be used to schedule maintenance when the future reliability becomes too low, rather than the current time-based schedule.

Lastly, FMTs model all components as degrading in a stepwise fashion (with discrete degradation phases and jumps between them). While this abstraction gives accurate results, it is not particularly realistic, as many failure modes follow gradual degradation curves. In the compressor in particular, the oil level decreases gradually, and the exact level can easily be measured during inspections. A model of such gradual degradation could be more accurate, and would be easier to validate against real-life measurements. An extension of FMTs with continuously-valued degradation is currently being developed as part of the Sequoia project [SHK17].

Part IV

Conclusions

Chapter 9

Conclusions

This thesis set out to integrate maintenance into fault trees, and thereby provide a framework for the integral analysis of maintenance and dependability. In particular, the aim of the ArRangeer project was to develop a methodology that can be used by the Dutch railway infrastructure asset manager ProRail to analyse and optimise their maintenance policy. To that end, we have surveyed the state of the art in fault tree analysis, and developed the formalism of fault maintenance trees (FMTs), extending fault trees with advanced maintenance plans used in the railway industry. We have developed techniques to analyse these FMTs using statistical model checking, including the application of rare event simulation to reduce computation time for high-availability systems. Finally, we have demonstrated that fault maintenance trees can be applied in practice, using two case studies from the railway industry.

9.1 Contributions

The main contribution of this thesis is the development of fault maintenance trees and the analysis thereof. In particular, we have provided the following results:

- We have surveyed the literature on fault trees. We provide an overview of over 150 papers on fault tree analysis, giving an in-depth description of the state of the art in fault trees, their analysis, and extensions. In particular, we introduce static fault trees, and describe the wide range of extensions that have been developed, including dynamic FTs, repairable FTs, and FTs dealing with uncertainty. For both static FTs and extensions, we review the various qualitative and quantitative analysis techniques, with examples to illustrate the different approaches.

- We have introduced the formalism of fault maintenance trees, extending fault trees with multiple phases of degradation for components, inspection and repair modules to undo this degradation, and rate-dependencies modelling dependencies between failure rates of different components. FMTs can provide quantitative insights into the dependability of systems subject to maintenance and, when decorated with costs for maintenance and failures, can be used

to calculate the expected cost of a given maintenance policy, supporting maintenance planners in finding the optimal plan.

- We have provided a translation to the timed automata formalism of the Uppaal-SMC tool, allowing the analysis of FMTs using statistical model checking. This provides statistically justified confidence intervals on quantitative metrics such as reliability, availability, and costs broken down into failure costs, inspection costs, repair costs, etc.

- We developed an analysis method based on rare event simulation for dynamic fault trees extended with the maintenance models of FMTs. This method uses the recently developed Path-ZVA algorithm for importance sampling, reducing the number of simulations (and thus computation time) required to obtain tight confidence intervals for high-availability systems. This analysis method can more quickly compute the availability of FMTs and repairable DFTs, at the expense of approximating any fixed times for maintenance actions by Erlang-distributed times.

- We have demonstrated the practical applicability of FMTs on two case studies from the railway industry:

  - In collaboration with ProRail, we have examined the wear and maintenance of an electrically insulated railway joint. Based on existing documentation and meetings with experts, we have constructed an FMT of the joint and a reference maintenance policy provided by ProRail. We then analysed this model to validate it against historically observed failures and to identify possible improvements to the reference policy. We found that the failures and replacements predicted by the FMT are a good match to reality, and that the reference policy is already close to the cost-optimal policy.

  - Together with the Dutch rolling stock maintenance company NS/NedTrain, we conducted an analysis of the pneumatic compressor used on a particular model of Dutch trains. We helped develop an FMT of its wear and maintenance, and analysed this FMT for validation and improvements. We found that the predicted failure behaviour is close to the actual failure rate, and identified that a routinely-scheduled overhaul has little effect on the reliability, and thus may not be cost-effective.

9.2 Discussion and Future Work

We see several improvements that can be made in future research:

Automatic optimisation. FMTs currently support the analysis of a system under a given maintenance strategy. Optimisation to find the best maintenance policy involves performing multiple FMT analyses using different policies, and selecting the best one. It would be useful to automatically select the optimal policy, given a range of possible maintenance options.

A method has already been developed to synthesise near-optimal strategies for stochastic priced timed games using machine learning [DJL+15]. We suspect that it is possible to apply such an approach to the stochastic timed automata used in FMT analysis to automatically generate cost-optimal maintenance policies.

Adaptation to predictive maintenance. This thesis provides an excellent starting point for the development of more advanced models for predictive maintenance. Current FMTs model a maintenance policy that is fixed in advance, with condition-based decisions providing limited flexibility to reach to observations in the field. A recent trend in maintenance engineering is *predictive maintenance*, where a model of system degradation is constantly updated with information about the current state of the system, allowing the prediction of future degradation and thus better suggestions for maintenance timing.

As FMTs already contain a model of the degradation of the system, we expect that they could be integrated into a predictive maintenance system. Observations could be used to estimate the current degradation phases of the various components, combined with the FMTs own predictions for components that cannot be (accurately) observed. The FMT can then predict future degradation and failure times, as well as how these are affected by various possible maintenance actions. In this way, maintenance decisions can be optimised in real-time.

Exact analysis. The analysis methods for FMTs described in this thesis fundamentally rely on Monte Carlo simulation. While this is an excellent technique to avoid the excessive memory consumption of many state space-based analysis techniques, it has the drawback of yielding only confidence intervals rather than exact results. We see two obstacles that need to be overcome in order to perform an exact stochastic analysis.

First, reduction techniques need to be applied in order to keep the memory requirements of the analysis of practical systems within reasonable bounds. Work has already been done on such techniques for non-repairable dynamic fault trees [VJK18], and we foresee that these techniques can also be applied to fault maintenance trees.

Second is the challenge of performing exact analysis on the types of automata describing the semantics of FMTs. As described in Chapter 5, FMTs support arbitrary probability distributions for failure times, which makes exact analysis extremely difficult. In practice, the case studies in Chapters 7 and 8 used only Erlang-distributed failure times, and exact times for maintenance actions. Such

models correspond to a subset of stochastic timed automata, which likely allows metrics to be computed by stochastic model checking.

Rare event simulation. Chapter 6 presented an importance sampling approach to analysing the availability of fault maintenance trees. We think that this approach can be improved in two, related, ways: supporting metrics other than availability, and supporting non-Markovian probability distributions in the model. These improvements are related, as the introduction of an exactly-timed (i.e., non-Markovian) transition into the model is already sufficient to compute the system reliability. Similarly, the case studies used only exactly-timed transitions for the maintenance modules, so supporting exact times would already greatly improve the applicability of our approach.

Modelling. One of the difficulties in reliability engineering of many systems is the accurate modelling of the failure behaviour of individual components. Also during the case studies in Part III of this thesis, much time was spent estimating the unmaintained failure time distributions of various components.

We envision that this step can be facilitated by automating the extraction of failure time distributions from measured data, e.g., failure databases. Already, the Sequoia project [SHK17] has begun to use big data analysis techniques to automate this step in the modelling. We expect that further development could substantially reduce the time and effort needed to obtain quantitative information for FMTs.

9.3 Outlook

In conclusion, we believe that FMTs provide a powerful framework for the modelling and analysis of the reliability of systems subject to maintenance. They are an excellent starting point for the development of more advanced formalism for, e.g., predictive maintenance. They also offer a number of interesting challenges for further research, such as automating the quantitative aspects of FMT modelling, and more powerful analysis techniques.

The results of our case studies already demonstrate that FMTs are a valuable tool in the analysis of maintenance policies for the railway industry. We expect that FMTs can be similarly applied in other industries, and see the potential that they can become a standard tool to aid maintenance planning both during and after the design of reliable systems.

Bibliography

[AA04] Suprased V. Amari and Jennifer B. Akers. "Reliability analysis of large fault trees using the vesely failure rate". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 391–396. IEEE, January 2004. DOI: 10.1109/RAMS.2004.1285481, ISBN: 978-0-7803-8215-2.

[AB03] John D. Andrews and Sally Beeson. "Birnbaum's measure of component importance for noncoherent systems". *IEEE Transactions on Reliability*, 52:213–219, June 2003. DOI: 10.1109/TR.2003.809656, ISSN: 0018-9529.

[ABvdB+13] Florian Arnold, Axel Belinfante, Freark van der Berg, Dennis Guck, and Mariëlle Stoelinga. "DFTCalc: A tool for efficient fault tree analysis". In *Proceedings of the 32nd International Symposium on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 8153 of *Lecture Notes on Computer Science*, pages 293–301. Springer, 2013. DOI: 10.1007/978-3-642-40793-2_27, ISBN: 978-3-642-40792-5.

[AD94] Rajeev Alur and David L. Dill. "A theory of timed automata". *Theoretical Computer Science*, 126(2):183–235, April 1994. DOI: 10.1016/0304-3975(94)90010-8, ISSN: 0304-3975.

[AGE03] Suprasad Amari, Dill Glenn, and Howald Eileen. "A new approach to solve dynamic fault trees". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 374–379. IEEE, January 2003. DOI: 10.1109/RAMS.2003.1182018, ISBN: 978-0-7803-7717-2.

[Ake78] Sheldon B. Akers. "Binary decision diagrams". *IEEE Transactions on Computers*, C-27(6):509–516, June 1978. DOI: 10.1109/TC.1978.1675141, ISSN: 0018-9340.

[ALD] Advanced Logistics Development. *Fault Tree Analysis (FTA) Software.* http://aldservice.com/en/reliability-products/fta.html.

[ALRL04] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. "Basic concepts and taxonomy of dependable and secure computing". *IEEE Transactions on Depandable and Secure Computing*, 1:11–33, 2004. DOI: 10.1109/TDSC.2004.2, ISSN: 1545-5971.

[Arm95] Michael J. Armstrong. "Joint reliability-importance of components". *IEEE Transactions on Reliability*, 44:408–412, September 1995. DOI: 10.1109/24.406574, ISSN: 0018-9529.

[Aut08] Automotive Industry Action Group. *SAE J-1739 – Potential Failure Mode & Effects Analysis*, 2008.

[AZ11] Hananeh Aliee and Hamid Reza Zarandi. "Fault tree analysis using stochastic logic: A reliable and high speed computing". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2011. DOI: 10.1109/RAMS.2011.5754466, ISBN: 978-1-4244-8857-5.

[AZ13] Hananeh Aliee and Hamid Reza Zarandi. "A fast and accurate fault tree analysis based on stochastic logic implemented on field-programmable gate arrays". *IEEE Transactions on Reliability*, 62:13–22, March 2013. DOI: 10.1109/TR.2012.2221012, ISSN: 0018-9529.

[BCK+11] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. "Safety, dependability and performance analysis of extended AADL models". *The Computer Journal*, 54:754–775, 2011. DOI: 10.1093/comjnl/bxq024.

[BCR04] Andrea Bobbio and Daniele Codetta-Raiteri. "Parametric fault trees with dynamic gates and repair boxes". In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 459–465. IEEE, 2004. DOI: 10.1109/RAMS.2004.1285491, ISBN: 978-0-7803-8215-2.

[BCRFH08] Marco Beccuti, Daniele Codetta-Raiteri, Giuliana Franceschinis, and Serge Hadded. "Non deterministic repairable fault trees for computing optimal repair strategy". In *Proceedings of the 3rd International Conference on Performance Evaluation, Methodologies and Tools (VALUETOOLS)*, article no. 56, October 2008. DOI: 10.4108/ICST.VALUETOOLS2008.4411, ISBN: 978-963-9799-31-8.

[BCS07a] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. "A compositional semantics for dynamic fault trees in terms of interactive Markov chains". In *Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 4762 of *Lecture Notes on Computer Science*, pages 441–456. Springer, 2007. DOI: 10.1007/978-3-540-75596-8_31, ISBN: 978-3-540-75595-1.

[BCS07b] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. "CORAL: a tool for compositional reliability and availability analysis". In *ARTIST workshop, presented at the 19th International Conference on Computer Aided Verification*, 2007.

[BCS07c] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. "Dynamic fault tree analysis using input/output interactive Markov chains". In *Proceedings of the 37th International Conference on Dependable Systems and Networks (DSN)*, pages 708–717. IEEE, 2007. DOI: 10.1109/DSN.2007.37, ISBN: 978-0-7695-2855-7.

[BCS10] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. "A rigorous, compositional, and extensible framework for dynamic fault tree analysis". *IEEE Transactions on Depandable and Secure Computing*, 7(2):128–143, 2010. DOI: 10.1109/TDSC.2009.45, ISSN: 1545-5971.

[BD05] Hichem Boudali and Joanne Bechta Dugan. "A new Bayesian network approach to solve dynamic fault trees". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 451–456. IEEE, January 2005. DOI: 10.1109/RAMS.2005.1408404, ISBN: 978-0-7803-8824-6.

[BDL+12] P. Bulychev, A. David, K. G. Larsen, M. Mikučionis, D. B. Poulsen, A. Legay, and Z. Wang. "UPPAAL-SMC: Statistical model checking for priced timed automata". In *Proceedings of the 10th workshop on Quantitative Aspects of Programming Languages (QAPL)*, 2012. DOI: 10.4204/EPTCS.85.1.

[BDM02] Simona Bernardi, Susanna Donatelli, and José Merseguer. "From UML sequence diagrams and statecharts to analysable petri net models". In *Proceedings of the 3rd International Workshop on Software and Performance (WOSP)*, pages 35–45, 2002. DOI: 10.1145/584369.584376, ISBN: 978-1-58113-563-3.

[BFCRH09] Marco Beccuti, Giuliana Franceschinis, Daniele Codetta-Raiteri, and Serge Haddad. "Parametric NdRFT for the derivation of optimal repair strategies". In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 399–408, 2009. DOI: 10.1109/DSN.2009.5270312, ISBN: 978-1-4244-4422-9.

[BFCRH14] Marco Beccuti, Giuliana Franceschinis, Daniele Codetta-Raiteri, and Serge Haddad. "Computing optimal repair strategies by means of NdRFT modeling and analysis". *The Computer Journal*, 57(12):1870–1892, December 2014. DOI: 10.1093/comjnl/bxt134, ISSN: 0010-4620.

[BG08] Irad Ben-Gal. "Bayesian networks". *Encyclopedia of Statistics in Quality and Reliability*, I, 2008. DOI: 10.1002/9780470061572.eqr089, ISBN: 978-0-470-06157-2.

[BHD06] Gabriella Budai, Dennis Huisman, and Rommert Dekker. "Scheduling preventive railway maintenance activities". *Journal of the Operational Research Society*, 57:1035–1044, 2006. DOI: 10.1057/palgrave.jors.260208, ISSN: 0160-5682.

[BHHK03] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. "Model-checking algorithms for continuous-time markov chains". *IEEE Transactions on Software Engineering*, 29(6):524–541, June 2003. DOI: 10.1109/TSE.2003.1205180, ISSN: 0098-5589.

[BHMT96] K. Bänsch, A. Hein, M. Malhotra, and K. Trivedi. "Comment/correction: Dependability modeling using Petri nets". *IEEE Transactions on Reliability*, 45(2):272–273, June 1996. DOI: 10.1109/24.510814, ISSN: 0018-9529.

[Bir68] Z. W. Birnbaum. "On the importance of different components in a multicomponent system". Technical report, Department of Mathematics, University of Washington, 1968.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9.

[Blo05] Neil B. Bloom. *Reliability Centered Maintenance (RCM): Implementation Made Simple*. McGraw-Hill, 1 edition, 2005. ISBN: 978-0-07-146069-9.

[BLR05] G. Behrmann, K. G. Larsen, and J. I. Rasmussen. "Priced timed automata: Algorithms and applications". In *Formal Methods for*

Components and Objects, volume 3657 of *Lecture Notes on Computer Science*, pages 162–182. Springer, 2005. DOI: 10.1007/11561163_8, ISBN: 978-3-540-29131-2.

[BMM99] Andrea Bondavalli, Istvan Majzik, and Ivan Mura. "Automatic dependability analysis for supporting design decisions in UML". In *Proceedings of the 4th IEEE International Symposium on High-Assurance Systems Engineering (HASE)*, pages 64–71, November 1999. DOI: 10.1109/HASE.1999.809476, ISBN: 978-0-7695-0418-6.

[BNS09] Hichem Boudali, A. P. Nijmeijer, and Mariëlle I. A. Stoelinga. "DFTSim: A simulation tool for extended dynamic fault trees". In *Proceedings of the 42nd Annual Simulation Symposium (ANSS) at the Spring Simulation Multiconference (SpringSim)*, pages 31:1–31:8, San Diego, California, USA, March 2009.

[Bor12] Carmen Lucia Tancredo Borges. "An overview of reliability models and methods for distribution systems with renewable energy distributed generation". *Renewable and Sustainable Energy Reviews*, 16(6):4008–4015, August 2012. DOI: 10.1016/j.rser.2012.03.055, ISSN: 1364-0321.

[Bou02] Marc Bouissou. "Boolean logic Driven Markov Processes: a powerful new formalism for specifying and solving very large Markov models". In E. J. Bonano, editor, *Proceedings of the 6th International Conference on Probabilistic Safety Assessment and Management (PSAM)*, San Juan, Puerto Rico, USA, 2002. Elsevier. ISBN: 978-0-08-044122-1.

[Bou07] Marc Bouissou. "A generalization of dynamic fault trees through boolean logic driven Markov processes (BDMP)®". In *Proceedings of the 16th European Safety and Reliability Conference*, Stavanger, Norway, 2007. CRC Press. ISBN: 978-0-415-44786-7.

[Bou08] Marc Bouissou. "BDMP (Boolean logic Driven Markov Processes)® as an alternative to Event Trees". In *Proceedings of the European Safety and Reliability Conference*. CRC Press, 2008. ISBN: 978-0-415-48513-5.

[Bou12] M. Bouissou. "BDMP knowledge base for KB3", 2012. http://sourceforge.net/projects/visualfigaro/files/Doc_and_examples/English/.

[BP75] R. E. Barlow and F. Proschan. *Statistical Theory of Reliability and Life Testing*. Holt, Rinehart, & Winstron, 1975. ISBN: 978-0-9606764-0-8.

[BPMC01] Andrea Bobbio, Luigi Portinale, Michele Minichino, and Ester Ciancamerla. "Improving the analysis of dependable systems by mapping fault trees into Bayesian networks". *Reliability Engineering & System Safety*, 71(3):249–260, March 2001. DOI: 10.1016/S0951-8320(00)00077-6, ISSN: 0951-8320.

[Bry92] Randal E. Bryant. "Symbolic boolean manipulation with ordered binary-decision diagrams". *ACM Computing Surveys*, 24(3):293–318, September 1992. DOI: 10.1145/136035.136043.

[BT95] Meera Balakrishnan and Kishor Trivedi. "Componentwise decomposition for an efficient reliability computation of systems with repairable components". In *Digest of Papers 25th International Symposium on Fault-Tolerant Computing (FTCS)*, pages 259–268. IEEE, June 1995. DOI: 10.1109/FTCS.1995.466972, ISBN: 978-0-8186-7079-4.

[Buc99] Kerstin Buchacker. "Combining fault trees and Petri nets to model safety-critical systems". In *Proceedings of the High Performance Computing Symposium (HPC)*, pages 439–444. The Society for Computer Simulation International, 1999.

[Buc00a] Kerstin Buchacker. *Definition und Auswertung erweiterter Fehlerbäume für die Zuverlässigkeitsanalyse technischer Systeme.* PhD thesis, Institut für Informatik Friedrich-Alexander-Universität Erlangen Nürnberg, 2000. In German.

[Buc00b] Kerstin Buchacker. "Modeling with extended fault trees". In *Proceedings of the 5th IEEE International Symposium on High-Assurance Systems Engineering (HASE)*, pages 238–246, November 2000. DOI: 10.1109/HASE.2000.895468, ISBN: 978-0-7695-0927-3.

[BV10] Marco Bozzano and Adolfo Villafiorita. *Design and Safety Assessment of Critical Systems.* CRC Press, 2010. ISBN: 978-1-4398-0331-8.

[BV16] Voestalpine Railpro BV. "Plaatsings-, onderhouds- en sloopvoorschrift energie-las (nrg-las)", 2016.

[BW96] Beate Bollig and Ingo Wegener. "Improving the variable ordering of OBDDs is NP-complete". *IEEE Transactions on Computers*, 45(9):993–1002, September 1996. DOI: 10.1109/12.537122, ISSN: 0018-9340.

[CDFH93] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. "Stochastic well-formed colored nets and symmetric modeling applications". *IEEE Transactions on Computers*, 42(11):1343–1360, 1993. DOI: 10.1109/12.247838, ISSN: 0018-9340.

[CE82] Edmund M. Clarke and E. Allen Emerson. “Design and synthesis of synchronization skeletons using branching time temporal logic”. In *Proceedings of the Workshop on Logic of Programs*, volume 131 of *Lecture Notes on Computer Science*, pages 52–71. Springer, 1982. DOI: 10.1007/BFb0025774, ISBN: 978-3-540-11212-9.

[CES09] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. “Model checking: algorithmic verification and debugging”. *Communications of the ACM*, 52(11):74–84, November 2009. DOI: 10.1145/1592761.1592781.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 1999. ISBN: 978-0-262-03270-4.

[CHR91] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. “A calculus of durations”. *Information Processing Letters*, 40(5):269–276, 1991. DOI: 10.1016/0020-0190(91)90122-X, ISSN: 0020-0190.

[CM93] Olivier Coudert and Jean Christophe Madre. “Fault tree analysis: 10^{20} Prime implicants and beyond”. In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 240–245. IEEE, 1993. DOI: 10.1109/RAMS.1993.296849, ISBN: 978-0-7803-0943-2.

[CM94] Olivier Coudert and Jean Christophe Madre. “MetaPrime: An interactive fault-tree analyzer”. *IEEE Transactions on Reliability*, 43:121–127, March 1994. DOI: 10.1109/24.285125, ISSN: 0018-9529.

[CM11] Sergio Contini and Vaidas Matuzas. “New methods to determine the importance measures of initiating and enabling events in fault tree analysis”. *Reliability Engineering & System Safety*, 96(7):775–784, 2011. DOI: 10.1016/j.ress.2011.02.001, ISSN: 0951-8320.

[Cn99] Juan A. Carrasco and Víctor Su né. “An algorithm to find minimal cuts of coherent fault-trees with event-classes using a decision tree”. *IEEE Transactions on Reliability*, 48:31–41, March 1999. DOI: 10.1109/24.765925, ISSN: 0018-9529.

[CP91] Danny I. Cho and Mahmut Parlar. “A survey of maintenance models for multi-unit systems”. *European Journal of Operational Research*, 51(1):1–23, March 1991. DOI: 10.1016/0377-2217(91)90141-H, ISSN: 0377-2217.

[CR05a] Daniele Codetta-Raiteri. “The conversion of dynamic fault trees to stochastic petri nets, as a case of graph transformation”. In *Proceedings of the Workshop on Petri Nets and Graph Transformations*

(PNGT), volume 127(2) of *Electronic Notes in Theoretical Computer Science*, pages 45 – 60, March 2005. DOI: 10.1016/j.entcs.2005.02.005.

[CR05b] Daniele Codetta-Raiteri. *Extended Fault Trees Analysis supported by Stochastic Petri Nets*. PhD thesis, Università degli Studi di Torino, 2005.

[CR06] Daniele Codetta-Raiteri. "BDD based analysis of parametric fault trees". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 442–449. IEEE, January 2006. DOI: 10.1109/RAMS.2006.1677414, ISBN: 978-1-4244-0007-2.

[CR11] Daniele Codetta-Raiteri. "Integrating several formalisms in order to increase fault trees' modeling power". *Reliability Engineering & System Safety*, 96(5):534–544, 2011. DOI: 10.1016/j.ress.2010.12.027, ISSN: 0951-8320.

[CRFIV04] Daniele Codetta-Raiteri, Giuliana Franceschinis, Mauro Iacono, and Valeria Vittorini. "Repairable fault tree for the automatic evaluation of repair policies". In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 659–668. IEEE, 2004. DOI: 10.1109/DSN.2004.1311936, ISBN: 978-0-7695-2052-0.

[CRL+11] Pierre-Yves Chaux, Jean-Marc Roussel, Jean-Jacques Lesage, Gilles Deleuze, and Marc Bouissou. "Qualitative analysis of a BDMP by finite automaton". In *Proceedings of the European Safety and Reliability Conference*, pages 2050–2057. CRC Press, 2011. ISBN: 978-0-415-68379-1.

[CRL+12] Pierre-Yves Chaux, Jean-Marc Roussel, Jean-Jacques Lesage, Gilles Deleuze, and Marc Bouissou. "Systematic extraction of minimal cut sequences from a BDMP model". In *Proceedings of the European Safety and Reliability Conference*, pages 3344–3351, 2012.

[CRL+13] Pierre-Yves Chaux, Jean-Marc Roussel, Jean-Jacques Lesage, Gilles Deleuze, and Marc Bouissou. "Towards a unified definition of minimal cut sequences". In *Proceedings of the 4th IFAC Workshop on Dependable Control of Discrete Systems (DCDS)*, volume 46(22) of *IFAC Proceedings Volumes*, pages 1–6. Elsevier, September 2013. DOI: 10.3182/20130904-3-UK-4041.00013, ISBN: 978-3-902823-49-6.

[Cro71] Paul A. Crosetti. "Fault tree analysis with probability evaluation". *IEEE Transactions on Nuclear Science*, 18(1):465–471, February 1971. DOI: 10.1109/TNS.1971.4325911, ISSN: 0018-9499.

[CW01] Carlos Carreras and Ian D. Walker. "Interval methods for fault-tree analysis in robotics". *IEEE Transactions on Reliability*, 50:3–11, 2001. DOI: 10.1109/24.935010, ISSN: 0018-9529.

[CY88] Costas Courcoubetis and Mihalis Yannakis. "Verifying temporal properties of finite-state probabilistic programs". In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 338–345. IEEE, 1988. DOI: 10.1109/SFCS.1988.21950, ISBN: 978-0-8186-0877-3.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. "Interface automata". In *Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 109–120. ACM Press, September 2001. DOI: 10.1145/503209.503226, ISBN: 978-1-58113-390-5.

[DBB90] Joanne Bechta Dugan, Salvatore J Bavuso, and Mark A Boyd. "Fault trees and sequence dependencies". In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 286–293. IEEE, 1990. DOI: 10.1109/ARMS.1990.67971.

[DBB92] Joanne Bechta Dugan, Salvatore J. Bavuso, and Mark A. Boyd. "Dynamic fault-tree models for fault-tolerant computer systems". *IEEE Transactions on Reliability*, pages 363–377, September 1992. DOI: 10.1109/24.159800, ISSN: 0018-9529.

[DCC+02] Daniel D. Deavours, Graham Clark, Tod Courtney, David Daly, Salem Derisavi, Jay M. Doyle, William H. Sanders, and Patrick G. Webster. "The Möbius framework and its implementation". *IEEE Transactions on Software Engineering*, 28(10):956–969, October 2002. DOI: 10.1109/TSE.2002.1041052, ISSN: 0098-5589.

[Dek95] Rommert Dekker. "On the use of operations research models for maintenance decision making". *Microelectronics Reliability*, 34(9–10):1321–1331, 1995. DOI: 10.1016/0026-2714(95)99380-2, ISSN: 0026-2714.

[Dek96] Rommert Dekker. "Applications of maintenance optimization models: a review and analysis". *Reliability Engineering & System Safety*, 51(3):229–240, March 1996. DOI: 10.1016/0951-8320(95)00076-3, ISSN: 0951-8320.

[Dem86] W. E. Deming. *Our of the Crisis*. MIT Press, 1986. ISBN: 978-0-262-54115-2.

[dJKTT15] Bram de Jonge, Warse Klingenberg, Ruud Teunter, and Tiedo Tinga. "Optimum maintenance strategy under uncertainty in the lifetime distribution". *Reliability Engineering & System Safety*, 133:59–67, January 2015. DOI: 10.1016/j.ress.2014.09.013, ISSN: 0951-8320.

[dJKTT16] Bram de Jonge, Warse Klingenberg, Ruud Teunter, and Tiedo Tinga. "Reducing costs by clustering maintenance activities for multiple critical units". *Reliability Engineering & System Safety*, 145:93–103, January 2016. DOI: 10.1016/j.ress.2015.09.003, ISSN: 0951-8320.

[DJKV17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. "A STORM is coming: A modern probabilistic model checker". In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, volume 10427 of *Lecture Notes on Computer Science*, pages 592–600. Springer, 2017. DOI: FIXME, ISBN: 978-3-319-63390-9.

[DJL+15] Alexandre David, Peter Gjøl Jensen, Kim Gulstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. "Uppaal stratego". In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9035 of *Lecture Notes on Computer Science*, pages 206–211. Springer, 2015. DOI: 10.1007/978-3-662-46681-0_16, ISBN: 978-3-662-46680-3.

[DP07] Salvatore Distefano and Antonio Puliafito. "Dynamic reliability block diagrams: Overview of a methodology". In *Proceedings of the European Safety and Reliability Conference*, volume 7, pages 1059–1068, 2007. ISBN: 978-0-415-44786-7.

[DR96] Yves Dutuit and Antoline B. Rauzy. "A linear-time algorithm to find modules of fault trees". *IEEE Transactions on Reliability*, 45:422–425, September 1996. DOI: 10.1109/24.537011, ISSN: 0018-9529.

[DR01] Yves Dutuit and Antoine B. Rauzy. "Efficient algorithms to assess component and gate importance in fault tree analysis". *Reliability Engineering & System Safety*, 72(2):213–222, 2001. DOI: 10.1016/S0951-8320(01)00004-7, ISSN: 0951-8320.

[DR05] Yves Dutuit and Antoline B. Rauzy. "Approximate estimation of system reliability via fault trees". *Reliability Engineering & System*

Safety, 87(2):163–172, 2005. DOI: 10.1016/j.ress.2004.02.008, ISSN: 0951-8320.

[DRGSR+09] K. Durga Rao, V. Gopika, V. V. S. Sanyasi Rao, H. S. Kushwaha, Ajit Kumar Verma, and Ajit Srividya. "Dynamic fault tree analysis using monte carlo simulation in probabilistic safety assessment". *Reliability Engineering & System Safety*, 94(4):872–883, April 2009. DOI: 10.1016/j.ress.2008.09.007, ISSN: 0951-8320.

[Dro15] Peter Drolenga. "Fault maintenance tree analysis in train systems", 2015. In: M. Peters, B. Huisman, and S. Hoekstra (supervisors) Industrial internship report.

[DVG97] Joanne Bechta Dugan, Bharath Venkataraman, and Rohit Gulati. "DIFtree: A software package for the analysis of dynamic fault tree models". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 64–70. IEEE, January 1997. DOI: 10.1109/RAMS.1997.571666, ISBN: 978-0-7803-3783-1.

[DWvdDS97] Rommert Dekker, Ralph E. Wildeman, and Frank A. van der Duyn Schouten. "A review of multi-component maintenance models with economic dependence". *Mathematical Methods of Operations Research*, 45(3):411–435, October 1997. DOI: 10.1007/BF01194788, ISSN: 1432-2994.

[DX06] Salvatore Distefano and Liudong Xing. "A new approach to modeling the system reliability: dynamic reliability block diagrams". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2006. DOI: 10.1109/RAMS.2006.1677373, ISBN: 978-1-4244-0007-2.

[Eas84] N. W. Eason. "Data as an asset". In *Proceedings of the 6th National Conference on Computers for Maintenance Managers*, 1984.

[Ebe97] Charles E. Ebeling. *An Introduction to Reliability and Maintainability Engineering*. McGraw-Hill, 1997. ISBN: 978-1-57766-625-7.

[EC06] "Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006 on machinery", 2006.

[EEC89] "EEC council directive 89/391/EEC of 12 June 1989 on the introduction of measures to encourage improvements in the safety and health of workers at work", 1989.

[EHZ10] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. "On probabilistic automata in continuous time". In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 342–351, 2010. DOI: 10.1109/LICS.2010.41, ISBN: 978-1-4244-7588-9.

[EIO98] Aly El-Iraki and Emmanual R. Odoom. "Fuzzy probist reliability assessment of repairable systems". In *Proceedings of the Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, pages 96–100. IEEE, August 1998. DOI: 10.1109/NAFIPS.1998.715544, ISBN: 978-0-7803-4453-2.

[Eme16] Emerson. "How manufacturers achieve top quartile performance", 2016.

[ENN13] Bernhard Ern, Viet Yen Nguyen, and Thomas Noll. "Characterization of failure effects on AADL models". In Bitsch F., Guiochet J., and Kaâniche M., editors, *Proceedings of the International Symposium on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 8153 of *Lecture Notes on Computer Science*, pages 241–252. Springer, 2013. DOI: 10.1007/978-3-642-40793-2_22, ISBN: 978-3-642-40792-5.

[EPR] EPRI. *CAFTA*. http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=000000000001015514.

[Eri99] Clifton A. Ericson. "Fault Tree Analysis – a history". In *Proceedings of the 17th International System Safety Conference*, pages 1–9, Orlando, Florida, USA, 1999.

[FAA98] Federal Aviation Administration, U.S. Department of Transportation. *FAA Order 8040.4: Safety Risk Management*, 1998.

[FAA00] Federal Aviation Administration, U.S. Department of Transportation. *System Safety Handbook*, 2000.

[FAA05] U.S. Department of Transportation Federal Aviation Administration. *AC 431.35-2A – Reusable launch and reeintry vehicle system safery process*, 2005.

[FAA18] Federal Aviation Administration. *Maintenance, Preventive Maintenance, Rebuilding, and Alteration*, chapter 1(C), Part 43. Government Publishing Office, 2018.

[FG12] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, 2012. ISBN: 978-0-13-420889-3.

[FHM74] J. B. Fussell, E. B. Henry, and N. H. Marshall. "MOCUS: a computer program to obtain minimal sets from fault trees". Technical report, Aerojet Nuclear Co., Idaho Falls, 1974.

[Fis96] George Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer Series in Operations Research and Financial Engineering. Springer, 1996. DOI: 10.1007/978-1-4757-2553-7, ISBN: 978-0-387-94527-9.

[FK06] Marc Förster and Bernhard Kaiser. "Increased efficiency in the quantitative evaluation of state/event fault trees". In *Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing*, volume 39(3) of *IFAC Proceedings Volumes*, pages 255–260. Elsevier, 2006. DOI: 10.3182/20060517-3-FR-2903.00143.

[FMC09] Igor Nai Fovino, Marcelo Masera, and Alessio De Cian. "Integrating cyber attacks within fault trees". *Reliability Engineering & System Safety*, 94(9):1394–1402, 2009. DOI: 10.1016/j.ress.2009.02.020, ISSN: 0951-8320.

[FMI+14] Francesco Flammini, Stefano Marrone, Mauro Iacono, Nicola Mazzocca, and Valeria Vittorini. "A multiformalism modular approach to ERTMS/ETCS failure modelling". *International Journal of Reliability, Quality and Safety Engineering*, 21(1), February 2014. DOI: 10.1142/S0218539314500016, ISSN: 0218-5393.

[FMIM05] Francesco Flammini, Nicola Mazzocca, Mauro Iacono, and Stefano Marrone. "Using repairable fault trees for the evaluation of design choices for critical repairable systems". In *Proceedings of the IEEE International Symposium on High-Assurance Systems Engineering (HASE)*, pages 163–172. IEEE, 2005. DOI: 10.1109/HASE.2005.26, ISBN: 0-7695-2377-3.

[FS84] Hitoshi Furuta and Naruhito Shiraishi. "Fuzzy importance in fault tree analysis". *Fuzzy Sets and Systems*, 12(3):205–213, 1984. DOI: 10.1016/0165-0114(84)90068-X, ISSN: 0165-0114.

[FT09] Marc Forster and Mario Trapp. "Fault tree analysis of software-controlled component systems based on second-order probabilities". In *Proceedings of the 20th IEEE International Symposium on*

Software Reliability Engineering (ISSRE), pages 146–154. IEEE, November 2009. DOI: 10.1109/ISSRE.2009.22.

[Fus75] J. B. Fussell. "How to hand-calculate system reliability and safety characteristics". *IEEE Transactions on Reliability*, R-24(3):169–174, August 1975. DOI: 10.1109/TR.1975.5215142, ISSN: 0018-9529.

[GCdSeS+95] A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg, and K. S. Trivedi. "The system availability estimator". In *Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS), Highlights from Twenty-Five Years*, pages 182–187, June 1995. DOI: 10.1109/FTCSH.1995.532632, ISBN: 978-0-8186-7150-0.

[GeE99] Antonio C. F. Guimarẽes and Nelson F. F. Ebecken. "FuzzyFTA: A fuzzy fault tree system for uncertainty analysis". *Annals of Nuclear Energy*, 26(6):523–532, April 1999. DOI: 10.1016/S0306-4549(98)00070-X, ISSN: 0306-4549.

[GFK02] Scott M. Goldman, Edna R. Fiedler, and Raymond E. King. "General aviation maintenance-related accidents: A review of ten years of ntsb data". Technical Report DOT/FAA/AM-02/23, Civil Aerospace Medical Institute, Federal Aviation Administration, 2002.

[GHK+07] Todd L. Graves, Michael S. Hamada, Richard Klamann, Andrew Koehler, and Harry F. Martz. "A fully Bayesian approach for combining multi-level information in multi-state fault tree quantification". *Reliability Engineering & System Safety*, 92(10):1476–1483, 2007. DOI: 10.1016/j.ress.2006.11.001, ISSN: 0951-8320.

[Git92] C. W. Gits. "Design of maintenance concepts". *International Journal of Production Economics*, 24(3):217–226, March 1992. DOI: 10.1016/0925-5273(92)90133-R, ISSN: 0925-5273.

[GJK+17a] Majdi Ghadhab, Sebastian Junges, Joost-Pieter Katoen, Matthias Kuntz, and Matthias Volk. "Model-based safety analysis for vehicle guidance systems". In *Proceedings of the International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 10488 of *Lecture Notes on Computer Science*, pages 3–19. Springer, September 2017. DOI: 10.1007/978-3-319-66266-4_1, ISBN: 978-3-319-66265-7.

[GJK+17b] Majdi Ghadhab, Sebastian Junges, Joost-Pieter Katoen, Matthias Kuntz, and Matthias Volk. "Model-based safety analysis for vehicle guidance systems". In *Proceedings of the International Symposium*

on Computer Safety, Reliability, and Security (SAFECOMP), volume 10488 of *Lecture Notes on Computer Science*, pages 3–19. Springer, 2017. DOI: 10.1007/978-3-319-66266-4, ISBN: 978-3-319-66265-7.

[GKS+14] Dennis Guck, Joost-Pieter Katoen, Mariëlle I. A. Stoelinga, Ted Luiten, and Judi Romijn. "Smart railroad maintenance engineering with stochastic model checking". In *Proceedings of the 2nd International Conference on Railway Technology: Research, Development and Maintenance (Railways)*, volume 104 of *Civil-Comp Proceedings*, article no. 299. Civil-Comp Press, Stirlingshire, UK, April 2014. DOI: 10.4203/ccp.104.299.

[GLMS13] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. "CADP 2011: a toolbox for the construction and analysis of distributed processes". *International Journal on Software Tools for Technology Transfer*, 15(2):89–107, April 2013. DOI: 10.1007/s10009-012-0244-z, ISSN: 1433-2779.

[Glu07] Pawel Gluchowski. "Duration calculus for analysis of fault trees with time dependencies". In *Proceedings of the 2nd International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX)*, pages 107–114. IEEE, June 2007. DOI: 10.1109/DEPCOS-RELCOMEX.2007.19, ISBN: 0-7695-2850-3.

[Gro98] EEIG ERTMS Users Group. "Ertms/etcs rams requirements specification, chapter 2 - ram". Technical Report 02S1266-, UIC, 1998.

[GSS15] Dennis Guck, Jip Spel, and Mariëlle I. A. Stoelinga. "DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper)". In *Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM)*, volume 9407 of *Lecture Notes on Computer Science*, pages 304–311. Springer, November 2015. DOI: 10.1007/978-3-319-25423-4_19.

[GTH+14] Dennis Guck, Mark Timmer, Hassan Hatefi, Enno Ruijters, and Mariëlle Stoelinga. "Modelling and analysis of markov reward automata". In Franck Cassez and Jean-François Raskin, editors, *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 8837 of *Lecture Notes on Computer Science*, pages 168–185. Springer, November 2014. DOI: 10.1007/978-3-319-11936-6_13, ISBN: 978-3-319-11935-9.

[Guc17] Dennis Guck. *Reliable Systems: Fault Tree Analysis via Markov Reward Automata*. PhD thesis, University of Twente, the Netherlands, 2017. ISBN: 978-90-365-4291-3.

[HBA08] E. E. Hurdle, L. M. Bartlett, and J. D. Andrews. "System fault diagnostics using fault tree analysis". *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 221(1):43–55, 2008.

[Hei95] Philip Heidelberger. "Fast simulation of rare events in queueing and reliability models". *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 5(1):43–85, January 1995. DOI: 10.1145/203091.203094, ISSN: 1049-3301.

[Her02] Holger Hermanns. *Interactive Markov chains: and the quest for quantified quality*, volume 2428 of *Lecture Notes on Computer Science*. Springer, 2002. DOI: 10.1007/3-540-45804-2, ISBN: 978-3-540-44261-5.

[HGH11] Wei Han, Weigang Guo, and Zhiqiang Hou. "Research on the method of dynamic fault tree analysis". In *Proceedings of the 9th International Conference on Reliability, Maintainability and Safety (ICRMS)*, pages 950–953. IEEE, IEEE, June 2011. DOI: 10.1109/ICRMS.2011.5979422, ISBN: 978-1-61284-667-5.

[HHH14] Ernst Moritz Hahn, Arnd Hartmanns, and Holger Hermanns. "Reachability and reward checking for stochastic timed automata". *Electronic Communications of the EASST*, 70, 2014. DOI: 10.14279/tuj.eceasst.70.968, ISSN: 1863-2122.

[Hix68] A. F. Hixenbaugh. *Fault Tree for Safety*. The Boeing Company, 1968.

[HJ94] Hans Hansson and Bengt Jonsson. "A logic for reasoning about time and reliability". *Formal Aspects of Computing*, 6(5):512–535, September 1994. DOI: 10.1007/BF01211866, ISSN: 0934-5043.

[HL93] Jung Sik Hong and Chang Hoon Lie. "Joint reliability-importance of two edges in an undirected network". *IEEE Transactions on Reliability*, 42:17–23, March 1993. DOI: 10.1109/24.210266, ISSN: 0018-9529.

[HLS+14] Ernst Moritz Hahn, Yi Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. "Iscasmc: A web-based probabilistic model checker". In *Proceedings of the International Symposium on Formal Methods*

(FM), volume 8442 of *Lecture Notes on Computer Science*, pages 312–317. Springer, 2014. DOI: 10.1007/978-3-319-06410-9_22, ISBN: 978-3-319-06409-3.

[Hoo10] Paul Hoogerkamp. *Praktijkgids Risicobeoordeling Machinerichtlijn*. Nederlands Normalisatie-instituut, 2010.

[HTZ04] Hong-Zhong Huang, Xin Tong, and Ming J Zuo. "Posbist fault tree analysis of coherent systems". *Reliability Engineering & System Safety*, 84(2):141–148, 2004. DOI: 10.1016/j.ress.2003.11.002, ISSN: 0951-8320.

[IEC06a] "IEC 60812: Analysis techniques for system reliability - procedure for failure mode and effects analysis (FMEA)", 2006.

[IEC06b] "IEC 61025: Fault tree analysis", 2006.

[iM93] Shin ichi Minato. "Zero-suppressed BDDs for set manipulation in combinatorial problems". In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 272–277. ACM New York, June 1993. DOI: 10.1145/157485.164890, ISBN: 978-0-89791-577-9.

[Iso] Isograph. *FaultTree+*. www.isograph.com/software/reliability-workbench/fault-tree-analysis/.

[ISO11] "ISO 26262:2011: Road vehicles – functional safety", 2011.

[ITEM] ITEM Software. *ITEM Toolkit: Fault Tree Analysis (FTA)*. www.itemsoft.com/fault_tree.html.

[Jac83] Peter S. Jackson. "On the s-importance of elements and prime implicants of non-coherent systems". *IEEE Transactions on Reliability*, R-32(1):21–25, April 1983. DOI: 10.1109/TR.1983.5221464, ISSN: 0018-9529.

[JGKS16] Sebastian Junges, Dennis Guck, Joost-Pieter Katoen, and Mariëlle Stoelinga. "Uncovering dynamic fault trees". In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 299–310. IEEE, 2016. DOI: 10.1109/DSN.2016.35, ISBN: 978-1-4673-8891-7.

[JT88] M. A. Johnson and M. R. Taaffe. "The denseness of phase distributions". Technical report, School of Industrial Engineering Research Memoranda 88-20, Purdue University, 1988.

[Kai05] Bernhard Kaiser. “Extending the expressive power of fault trees”. In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 468–474. IEEE, January 2005. DOI: 10.1109/RAMS.2005.1408407, ISBN: 0-7803-8824-0.

[KG04] Bernhard Kaiser and Catharina Gramlich. “State-event-fault-trees - a safety analysis model for software controlled systems”. In *Proceedings of the International Symposium on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 3219 of *Lecture Notes on Computer Science*, pages 195–209. Springer, 2004. DOI: 10.1007/978-3-540-30138-7_17, ISBN: 978-3-540-23176-9.

[KGF07] Bernhard Kaiser, Catharina Gramlich, and Marc Förster. “State/event fault trees – a safety analysis model for software-controlled systems”. *Reliability Engineering & System Safety*, 92(11):1521–1537, November 2007. DOI: 10.1016/j.ress.2006.10.010, ISSN: 0951-8320.

[KH51] Herman Kahn and T. E. Harris. “Estimation of particle transmission by random sampling”. In *Monte Carlo method; Proceedings of the Symposium held June 29, 30, and July 1 1949*, volume 12 of *National Bureau of Statistics, Applied Mathematics Series*, pages 27–30, 1951.

[KJG96] CE Kim, YJ Ju, and M Gens. “Multilevel fault tree analysis using fuzzy numbers”. *Computers & Operations Research*, 23(7):695–703, 1996. DOI: 10.1016/0305-0548(95)00070-4, ISSN: 0305-0548.

[Kle99] Trevor A. Kletz. *Hazop & Hazan: Identifying and Assessing Process Industry Hazards*. CRC Press, fourth edition, 1999. ISBN: 978-1-56032-858-2.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. “PRISM 4.0: Verification of probabilistic real-time systems”. In *Computer Aided Verification*, volume 6806 of *Lecture Notes on Computer Science*, pages 585–591. Springer, 2011. DOI: 10.1007/978-3-642-22110-1_47, ISBN: 978-3-642-22109-5.

[Kom02] Kari Komonen. “A cost model of industrial maintenance for profitability analysis and benchmarking”. *International Journal of Production Economics*, 79(1):15–31, September 2002. DOI: 10.1016/S0925-5273(00)00187-0, ISSN: 0925-5273.

[KP14] Kailash C. Kapur and Michael Pecht. *Reliability Engineering*. John Wiley & Sons, 2014. DOI: 10.1002/9781118841716, ISBN: 978-1-118-14067-3.

[KPCS14] Barbara Kordy, Ludovic Piètre-Cambacèdés, and Patrick Schweitzer. "Dag-based attack and defense modeling: Don't miss the forest for the attack trees". *Computer Science Review*, 13–14:1–38, November 2014. DOI: 10.1016/j.cosrev.2014.07.001, ISSN: 1574-0137.

[KPP08] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. "The epsilon transformation language". In *Proceedings of the International Conference on Theory and Practice of Model Transformations (ICMT)*, volume 5063 of *Lecture Notes on Computer Science*, pages 46–60. Springer, 2008. DOI: 10.1007/978-3-540-69927-9_4, ISBN: 978-3-540-69926-2.

[KRS15] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. "Quantitative attack tree analysis via priced timed automata". In Sriram Sankaranarayanan and Enrico Vicario, editors, *Proceedings of the 13th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 9268 of *Lecture Notes on Computer Science*, pages 156–171. Springer, September 2015. DOI: 10.1007/978-3-319-22975-1_11, ISBN: 978-3-31922974-4.

[KS17] Joost-Pieter Katoen and Mariëlle Stoelinga. "Boosting fault tree analysis by formal methods". In *ModelEd, TestEd, TrustEd*, volume 10500 of *Lecture Notes on Computer Science*, pages 368–389. Springer, September 2017. DOI: 10.1007/978-3-319-68270-9_19, ISBN: 978-3-319-68269-3.

[KSR+18] Rajesh Kumar, Stefano Schivo, Enno Ruijters, Buğra M. Yildiz, David Huistra, Jacco Brandt, Arend Rensink, and Mariëlle Stoelinga. "Effective analysis of attack trees: a model-driven approach". In *Proceedings of the 21st International Conference on Fundamental Approaches to Software Engineering (FASE)*, volume 10802 of *Lecture Notes on Computer Science*, pages 56–73. Springer, 2018. DOI: 10.1007/978-3-319-89363-1_4, ISBN: 978-3-319-89362-4.

[KZE09] Chakib Kara-Zaitri and Enver Ever. "A hardware accelerated semi analytic approach for fault trees with repairable components". In *Proceedings of the 11th International Conference on Computer Modelling and Simulation (UKSIM)*, pages 146–151. IEEE, March 2009. DOI: 10.1109/UKSIM.2009.83, ISBN: 978-1-4244-3771-9.

[Lam10] Mariapia Lampis. *Application of Bayesian Belief Networks to System Fault Diagnostics*. PhD thesis, Loughborough University, 2010.

[LBTG10] Pierre L'Ecuyer, Jose H. Blanchet, Bruno Tuffin, and Peter W. Glynn. "Asymptotic robustness of estimators in rare-event simulation". *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 20(1), 2010. DOI: 10.1145/1667072.1667078, ISSN: 1049-3301.

[LDB10] Axel Legay, Benoit Delahare, and Saddek Bensalem. "Statistical model checking: An overview". In *Proc. 1st Int. Conf. on Runtime Verification (RV)*, volume 6418 of *Lecture Notes on Computer Science*, pages 122–135. Springer, November 2010. DOI: 10.1007/978-3-642-16612-9_11, ISBN: 978-3-642-16611-2.

[LGTL85] Wen-Shing Lee, D. L. Grosh, Frank A. Tillman, and Chang H. Lie. "Fault tree analysis, methods, and applications — A review". *IEEE Transactions on Reliability*, R-34(3):194–203, 1985. DOI: 10.1109/TR.1985.5222114, ISSN: 0018-9529.

[Li08] Deng-Feng Li. "A note on "using intuitionistic fuzzy sets for fault-tree analysis on printed circuit board assembly"". *Microelectronics Reliability*, 48(10):1741, October 2008. DOI: 10.1016/j.microrel.2008.07.059, ISSN: 0026-2714.

[LJ07] Lixuan Lu and Jin Jiang. "Joint failure importance for noncoherent fault trees". *IEEE Transactions on Reliability*, pages 435–443, September 2007. DOI: 10.1109/TR.2007.898574, ISSN: 0018-9529.

[LLLT09] Pierre L'Ecuyer, François Le Gland, Pascal Lezaud, and Bruno Tuffin. *Splitting techniques*, chapter 3, pages 39–61. John Wiley & Sons, 2009.

[LP07] Helge Langseth and Luigi Portinale. "Bayesian networks in reliability". *Reliability Engineering & System Safety*, 92(1):92–108, January 2007. DOI: doi:10.1016/j.ress.2005.11.037, ISSN: 0951-8320.

[LRCRS] Lloyd's Register Consulting. *RiskSpectrum*. www.riskspectrum.com/en/risk.

[LSH00] W Long, Y Sato, and M Horigome. "Quantification of sequential failure logic for fault tree analysis". *Reliability Engineering & System Safety*, 67(3):269–274, 2000. DOI: 10.1016/S0951-8320(99)00075-7, ISSN: 0951-8320.

[LT11] Pierre L'Ecuyer and Bruno Tuffin. "Approximating zero-variance importance sampling in a reliability setting". *Annals of Operations Research*, 189(1):277–297, September 2011. DOI: 10.1007/s10479-009-0532-5, ISSN: 1572-9338.

[LW97] Ching-Torng Lin and Mao-Jiun J Wang. "Hybrid fault tree analysis using fuzzy sets". *Reliability Engineering & System Safety*, 58(3):205–213, December 1997. DOI: 10.1016/S0951-8320(97)00072-0, ISSN: 0951-8320.

[LXL+10] Dong Liu, Lei Xiong, Zhi Li, Peng Wang, and Honglin Zhang. "The simplification of cut sequence set analysis for dynamic systems". In *Proceedings of the 2nd International Conference on Computer and Automation Engineering (ICCAE)*, volume 3, pages 140–144. IEEE, February 2010. DOI: 10.1109/ICCAE.2010.5451831, ISBN: 978-1-4244-5569-0.

[LXZ+07] Dong Liu, Weiyan Xing, Chunyuan Zhang, Rui Li, and Haiyan Li. "Cut sequence set generation for fault tree analysis". In *Embedded Software and Systems*, volume 4523 of *Lecture Notes on Computer Science*, pages 592–603. Springer, 2007. DOI: 10.1007/978-3-540-72685-2_55, ISBN: 978-3-540-72684-5.

[LY77] Howard E. Lambert and George Yadigaroglu. "Fault trees for diagnosis of system fault conditions". *Nuclear Science and Engineering*, 62:20–34, 1977. DOI: 10.13182/NSE77-A26936.

[LYZL09] Xiaofeng Liang, Hong Yi, Yufang Zhang, and Dan Li. "A numerical simulation approach for reliability analysis of fault-tolerant repairable system". In *Proceedings of the 8th International Conference on Reliability, Maintainability and Safety (ICRMS)*, pages 191–196. IEEE, July 2009. DOI: 10.1109/ICRMS.2009.5270210, ISBN: 978-1-4244-4903-3.

[LZX+07] Dong Liu, Chunyuan Zhang, Weiyan Xing, Rui Li, and Haiyan Li. "Quantification of cut sequence set for fault tree analysis". In *Proceedings of the International Conference on High Performance Computing and Communications*, volume 4782 of *Lecture Notes on Computer Science*, pages 755–765. Springer, 2007. DOI: 10.1007/978-3-540-75444-2_70, ISBN: 978-3-540-75443-5.

[MAV+13] Y. A. Mahmood, A. Ahmadi, A. K. Verma, A. Srividya, and U. Kumar. "Fuzzy fault tree analysis: a review of concept and application". *International Journal of System Assurance Engineering and Management*, 4(1):19–32, March 2013. DOI: 10.1007/s13198-013-0145-x, ISSN: 0975-6809.

[May60] Raymond R. Mayer. "Problems in the application of replacement theory". *Management Science*, 6(3):303–310, 1960. DOI: 10.1287/mnsc.6.3.303, ISSN: 0025-1909.

[MCC+14] G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, and N. Trapani. "Conception of repairable dynamic fault trees and resolution by the use of raatss, a matlab®toolbox based on the ats formalism". *Reliability Engineering & System Safety*, 121:250–262, January 2014. DOI: 10.1016/j.ress.2013.09.002, ISSN: 0951-8320.

[Mer10] Guillaume Merle. *Algebraic modelling of Dynamic Fault Trees, contribution to qualitative and quantitative analysis*. PhD thesis, École normale supérieure de Cachan, 2010. https://tel.archives-ouvertes.fr/tel-00502012v1.

[MKK09] Mohammad Modarres, Mark Kaminskiy, and Vasiliy Krivtsov. *Reliability Engineering and Risk Analysis: A Practical Guide*. CRC Press, third edition, 2009. ISBN: 978-1-4987-4587-1.

[MNTL13] Zuoyu Miao, Ru Niu, Tao Tang, and Jieyu Liu. "A new generation algorithm of fault tree minimal cut sets and its application in CBTC system". In *Proceedings of the International Conference on Intelligent Rail Transportation (ICIRT)*, pages 221–226. IEEE, August 2013. DOI: 10.1109/ICIRT.2013.6696297, ISBN: 978-1-4673-5278-9.

[Mo14] Yuchang Mo. "A multiple-valued decision-diagram-based approach to solve dynamic fault trees". *IEEE Transactions on Reliability*, 63(1):81–93, March 2014. DOI: 10.1109/TR.2014.2299674, ISSN: 0018-9529.

[Mob02] R. Keith Mobley. *An introduction to predictive maintenance*. Elsevier, 2002. ISBN: 978-0-08-047869-2.

[Mou97] John Moubray. *Reliability centered maintenance*. Industrial Press, 1997. ISBN: 978-0-8311-3146-3.

[MPB05a] Stefania Montani, Luigi Portinale, and Andrea Bobbio. "Dynamic Bayesian networks for modeling advanced fault tree features in dependability analysis". In *Proceedings of the European Safety and Reliability Conference*, pages 1415–1422. CRC Press, 2005. ISBN: 978-0-415-38340-0.

[MPB+05b] Stefania Montani, Luigi Portinale, Andrea Bobbio, M. Varesio, and Daniele Codetta-Raiteri. "DBNet, a tool to convert dynamic fault trees into dynamic Bayesian networks". Technical report, Dip. di Informatica, Univ. del Piemonte Orientale, August 2005.

[MPBCR08] Stefania Montani, Luigi Portinale, Andrea Bobbio, and Daniele Codetta-Raiteri. "Radyban: A tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks". *Reliability Engineering & System Safety*, 93(7):922–932, 2008. DOI: 10.1016/j.ress.2007.03.013, ISSN: 0951-8320.

[MR07] Guillaume Merle and Jean-Marc Roussel. "Algebraic modelling of fault trees with priority AND gates". In *Proceedings of the 1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS)*, volume 40(6) of *IFAC Proceedings Volumes*, pages 31–36. Elsevier, June 2007.

[MRL11] Guillaume Merle, Jean-Marc Roussel, and Jean-Jaques Lesage. "Dynamic fault tree analysis based on the structure function". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2011. DOI: 10.1109/RAMS.2011.5754452, ISBN: 978-1-4244-8857-5.

[MRLB10] Guillaume Merle, Jean-Marc Roussel, Jean-Jaques Lesage, and Andrea Bobbio. "Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events". *IEEE Transactions on Reliability*, 59(1):250–261, March 2010. DOI: 10.1109/TR.2009.2035793, ISSN: 0018-9529.

[MS95] Krishna B. Misra and K. P. Soman. "Multi state fault tree analysis using fuzzy probability vectors and resolution identity". *Reliability and Safety Analyses under Fuzziness*, 4:113–125, 1995. DOI: 10.1007/978-3-7908-1898-7_7, ISBN: 978-3-662-12913-5.

[MT95] Manish Malhotra and Kishor S. Trivedi. "Dependability modeling using Petri-nets". *IEEE Transactions on Reliability*, 44(3):428–440, September 1995. DOI: 10.1109/24.406578, ISSN: 0018-9529.

[MZ00] Marzio Marseguerra and Enrico Zio. "Optimizing maintenance and repair policies via a combination of genetic algorithms and monte carlo simulation". *Reliability Engineering & System Safety*, 68(1):69–83, April 2000. DOI: 10.1016/S0951-8320(00)00007-7, ISSN: 0951-8320.

[ND08] Robin P. Nicolai and Rommert Dekker. "Optimal maintenance of multi-component systems: A review". *Complex System Maintenance Handbook*, pages 263–286, 2008. DOI: 10.1007/978-1-84800-011-7_11, ISBN: 978-1-84800-010-0.

[Ned14] NedTrain. *Naslagwerk onderhoud & storingen VIRM 2/3*, 2014. Internal document.

[NTX13] Jun Ni, Wencheng Tang, and Yan Xing. "A simple algebra for fault tree analysis of static and dynamic systems". *IEEE Transactions on Reliability*, 62:846–861, December 2013. DOI: 10.1109/TR.2013.2285035, ISSN: 0018-9529.

[OD00] Yong Ou and Joanne Bechta Dugan. "Sensitivity analysis of modular dynamic fault trees". In *Proceedings of the IEEEInternational Computer Performance and Dependability Symposium (IPDS)*, pages 35–43. IEEE, March 2000. DOI: 10.1109/IPDS.2000.839462, ISBN: 978-0-7695-0553-4.

[Ope] *OpenFTA*. www.openfta.com/.

[OSH94] Occupational Safety and Health Administration, U.S. Department of Labor. *OSHA 3133: Process Safety Management Guidelines for Compliance*, 1994.

[PA13] Darren Prescott and John Andrews. "Modelling maintenance in railway infrastructure management". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, 2013. DOI: 10.1109/RAMS.2013.6517678, ISBN: 978-1-4673-4709-9.

[Pal02] Girish Keshav Palshikar. "Temporal fault trees". *Information and Software Technology*, 44(3):137–150, 2002. DOI: 10.1016/S0950-5849(01)00223-3, ISSN: 0950-5849.

[PBCRM07] Luigi Portinale, Andrea Bobbio, Daniele Codetta-Raiteri, and Stefania Montani. "Compiling dynamic fault trees into dynamic Bayesian nets for reliability analysis: the RADYBAN tool". In *Proceedings of the Applications Workshop of the 5th Uncertainty in Artificial Intelligence Conference (UAI-AW)*, pages 47–54, 2007.

[PCRM10] Luigi Portinale, Daniele Codetta-Raiteri, and Stefania Montani. "Supporting reliability engineers in exploiting the power of dynamic Bayesian networks". *International Journal of Approximate Reasoning*, 51(2):179–195, January 2010. DOI: 10.1016/j.ijar.2009.05.009, ISSN: 0888-613X.

[PD96] Laura L. Pullum and Joanne Bechta Dugan. "Fault tree models for the analysis of complex computer-based systems". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 200–207. IEEE, January 1996. DOI: 10.1109/RAMS.1996.500663, ISBN: 978-0-7803-3112-9.

[PDAC05] Petar Popic, Dejan Desovski, Walid Abdelmoez, and Bojan Cukic. "Error propagation in the reliability analysis of component based systems". In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pages 52–62. IEEE, November 2005. DOI: 10.1109/ISSRE.2005.18, ISBN: 978-0-7695-2482-5.

[PG92] L. M. Pintelon and L. F. Gelders. "Maintenance management decision making". *European Journal of Operational Research*, 58:301–317, May 1992. DOI: 10.1016/0377-2217(92)90062-E, ISSN: 0377-2217.

[PP94] Lavon B. Page and Jo Ellen Perry. "Standard deviation as an alternative to fuzziness in fault tree models". *IEEE Transactions on Reliability*, 43(3):402–407, September 1994. DOI: 10.1109/24.326434, ISSN: 0018-9529.

[Pro15] ProRail. "Netverklaring 2016, Gemengde net [in Dutch]", 2015.

[PSC75] Pradip K. Pande, Michael E. Spector, and Purnendu Chatterjee. "Computerized fault tree analysis: TREEL and MICSUP". Technical report, Operation Research Centre, University of California, Berkeley, 1975.

[PTC] PTC. *Windchill FTA*. www.ptc.com/product/relex/fault-tree.

[QS82] Jean-Pierre Queille and Joseph Sifakis. "Specification and verification of concurrent systems in cesar". In *Proceedings of the International Symposium on Programming*, volume 137 of *Lecture Notes on Computer Science*, pages 337–351. Springer, 1982. DOI: 10.1007/3-540-11494-7_22, ISBN: 978-3-540-11494-9.

[RA06] R. Remenyte and J. D. Andrews. "A simple component connection approach for fault tree conversion to binary decision diagram". In *Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES)*, pages 449–456, April 2006. DOI: 10.1109/ARES.2006.17, ISBN: 978-0-7695-2567-9.

[Rau93] Antoine B. Rauzy. "New algorithms for fault tree analysis". *Reliability Engineering & System Safety*, 40(3):203–211, 1993. DOI: 10.1016/0951-8320(93)90060-C, ISSN: 0951-8320.

[Rau08] Antoine Rauzy. *Binary Decision Diagrams for Reliability Studies*, pages 381–396. Springer London, 2008.

[Rau11] Antoine B. Rauzy. "Sequence algebra, sequence decision diagrams and dynamic fault trees". *Reliability Engineering & System Safety*, 96(7):785–792, 2011. DOI: 10.1016/j.ress.2011.02.005, ISSN: 0951-8320.

[RBM91] Don E. Ross, Kenneth M. Butler, and M. Ray Mercer. "Exact ordered binary decision diagram size when representing classes of symmetric functions". *Journal of Electronic Testing*, 2(3):243–259, August 1991. DOI: 10.1007/BF00135441, ISSN: 0923-8174.

[RD97] Antoine Rauzy and Yves Dutuit. "Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within Aralia". *Reliability Engineering & System Safety*, 58(2):127–144, November 1997. DOI: 10.1016/S0951-8320(97)00034-3, ISSN: 0951-8320.

[RdBSH15] Daniël Reijsbergen, Pieter-Tjerk de Boer, Werner Scheinhardt, and Boudewijn Haverkort. "On hypothesis testing for statistical model checking". *International Journal on Software Tools for Technology Transfer*, 17(4):377–395, August 2015. DOI: 10.1007/s10009-014-0350-1, ISSN: 1433-2779.

[RdBSJ18] Daniël Reijsbergen, Pieter-Tjerk de Boer, Werner R. W. Scheinhardt, and Sandeep Juneja. "Path-ZVA: general, efficient and automated importance sampling for highly reliable markovian systems". *ACM Transactions on Modeling and Computer Simulation*, 2018. Accepted for publication.

[Rei13] Daniël Reijsbergen. *Efficient simulation techniques for stochastic model checking*. PhD thesis, University of Twente, Enschede, December 2013. ISBN: 978-90-365-3586-1.

[Rel] ReliaSoft. *BlockSim*. www.reliasoft.com/BlockSim/index.html.

[RG12] Dev G. Raheja and Louis J. Gullo, editors. *Design for Reliability*. John Wiley & Sons, 2012.

[RGD10] Duan Rongxing, Wan Guochun, and Dong Decun. "A new assessment method for system reliability based on dynamic fault tree". In *Proceedings of the International Conference on Intelligent Computation Technology and Automation (ICICTA)*, pages 219–222. IEEE, May 2010. DOI: 10.1109/ICICTA.2010.237, ISBN: 978-1-4244-7279-6.

[RGD+16] Enno Ruijters, Dennis Guck, Peter Drolenga, Margot Peters, and Mariëlle Stoelinga. "Maintenance analysis and optimization via statistical model checking: Evaluation of a train's pneumatic compressor".

In *Proceedings of the 13th International Conference on Quantitative Evaluation of SysTems (QEST)*, volume 9826 of *Lecture Notes on Computer Science*, pages 331–347. Springer, August 2016. DOI: 10.1007/978-3-319-43425-4_22, ISBN: 978-3-319-43424-7.

[RGDS16] Enno Ruijters, Dennis Guck, Peter Drolenga, and Mariëlle Stoelinga. "Fault maintenance trees: reliability contered maintenance via statistical model checking". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2016. DOI: 10.1109/RAMS.2016.7447986, ISBN: 978-1-5090-0248-1.

[RGvNS16] Enno Ruijters, Dennis Guck, Martijn van Noort, and Mariëlle Stoelinga. "Reliability-centered maintenance of the electrically insulated railway joint via fault tree analysis: A practical experience report". In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 662–669, 2016. DOI: 10.1109/DSN.2016.67, ISBN: 978-1-4673-8891-7.

[RH04] Marvin Rausand and Arnljot Hoylan. *System Reliability Theory. Models, Statistical Methods, and Applications*. Wiley series in probability and statistics. John Wiley & Sons, 2004. ISBN: 978-0-471-47133-2.

[Rij12] Rijkswaterstaat. *Leidraad RAMS—sturen op prestaties van systemen*. Ministerie van Verkeer en Waterstaat, 2012. In Dutch.

[RJ10] Amir Rajabzadeh and Mohammed S. Jahangiry. "Hardware-based reliability tree (HRT) for fault tree analysis". In *Proceedings of the 15th CSI International Symposium on Computer Architecture and Digital Systems (CADS)*, pages 171–172. IEEE, September 2010. DOI: 10.1109/CADS.2010.5623587, ISBN: 978-1-4244-6267-4.

[RJB04] James Rumbaugh, Ivar Jabobson, and Grady Booch. *The Unified Modeling Language Referance manual*. Pearson Higher Education, second edition, 2004. ISBN: 978-0-321-71895-2.

[RK11] Yi Ren and Leixing Kong. "Fuzzy multi-state fault tree analysis based on fuzzy expert system". In *Proceedings of the 9th International Conference on Reliability, Maintainability and Safety (ICRMS)*, pages 920–925. IEEE, June 2011. DOI: 10.1109/ICRMS.2011.5979415, ISBN: 978-1-61284-667-5.

[RL13] Michael Roth and Peter Liggesmeyer. "Qualitative analysis of state/event fault trees for supporting the certification process of software-intensive systems". In *Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 353–358, November 2013. DOI: 10.1109/ISSREW.2013.6688920, ISBN: 978-1-4799-2553-7.

[RPA08] R. Remenyte-Prescott and J.D. Andrews. "An enhanced component connection method for conversion of fault trees to binary decision diagrams". *Reliability Engineering & System Safety*, 93(10):1543–1550, October 2008. DOI: 10.1016/j.ress.2007.09.001, ISSN: 0951-8320.

[RRdBS17] Enno Ruijters, Daniël Reijsbergen, Pieter-Tjerk de Boer, and Mariëlle Stoelinga. "Rare event simulation for dynamic fault trees". In *Proceedings of the International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 10488 of *Lecture Notes on Computer Science*, pages 20–35. Springer, September 2017. DOI: 10.1007/978-3-319-66266-4_2, ISBN: 978-3-319-66265-7.

[RS15] Enno Ruijters and Mariëlle Stoelinga. "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools". *Computer Science Review*, 15–16:29–62, 2015. DOI: 10.1016/j.cosrev.2015.03.001, ISSN: 1574-0137.

[RS16] Enno Ruijters and Mariëlle Stoelinga. "Better railway engineering through statistical model checking". In *Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, volume 9952 of *Lecture Notes on Computer Science*, pages 151–165. Springer, October 2016. DOI: 10.1007/978-3-319-47166-2_10, ISBN: 978-3-319-47165-5.

[RSSR17] Enno Ruijters, Stefano Schivo, Mariëlle Stoelinga, and Arend Rensink. "Uniform analysis of fault trees through model transformations". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2017. DOI: 10.1109/RAM.2017.7889759, ISBN: 978-1-5090-5284-4.

[Rus85] Ali M. Rushdi. "Uncertainty analysis of fault-tree outputs". *IEEE Transactions on Reliability*, R-34(5):458–462, December 1985. DOI: 10.1109/TR.1985.5222232, ISSN: 0018-9529.

[SA96] Roslyn M. Sinnamon and John D. Andrews. "Fault tree analysis and binary decision diagrams". In *Proceedings of the Reliability and*

Maintainability Symposium (RAMS), pages 215–222. IEEE, 1996. DOI: 10.1109/RAMS.1996.500665, ISBN: 978-0-7803-3112-9.

[SBR96] P. V. Suresh, A. K. Babar, and V. Venkat Raj. "Uncertainty in fault tree analysis: A fuzzy approach". *Fuzzy Sets and Systems*, 83(2):135–141, 1996. DOI: 10.1016/0165-0114(95)00386-X, ISSN: 0165-0114.

[Sca97] Philip A. Scarf. "On the application of mathematical models in maintenance". *European Journal of Operational Research*, 99(3):493–506, June 1997. DOI: 10.1016/S0377-2217(96)00316-5, ISSN: 0377-2217.

[SCC06] Ming-Hung Shu, Ching-Hsue Cheng, and Jing-Rong Chang. "Using intuitionistic fuzzy sets for fault-tree analysis on printed circuit board assembly". *Microelectronics Reliability*, 46(12):2139–2148, 2006. DOI: 10.1016/j.microrel.2006.01.007, ISSN: 0026-2714.

[SCD+03] William H. Sanders, Tod Courtney, Danial D. Deavours, David Daly, Salem Derisavi, and Vinh Lam. "Multi-formalism and multi-solution-method modeling frameworks: The Möbius approach". In *Proceedings of the Symposium on Performance Evaluation - Stories and Perspectives*, pages 241–256, Vienna, Austria, December 2003.

[Sch90] Winfrid G. Schneeweiss. "SyRePa'89–a package of programs for systems reliability evaluations". Technical report, Informatik-Rep. 91, Fern Universität, 1990.

[Sch98] Winfrid G. Schneeweiss. "On the polynomial form of boolean functions: Derivations and applications". *IEEE Transactions on Computers*, 47:217–221, February 1998. DOI: 10.1109/12.663768, ISSN: 0018-9340.

[SDC99] Kevin J. Sullivan, Joanne Bechta Dugan, and David Coppit. "The galileo fault tree analysis tool". In *Proceedings of the Annual International Symposium on Fault-Tolerant Computing (FTCS)*. IEEE, June 1999. DOI: 10.1109/FTCS.1999.781056, ISBN: 978-0-7695-0213-7.

[SHK17] Mariëlle Stoelinga, Djoerd Hiemstra, and Joost-Pieter Katoen. "Sequoia: Smart maintenance optimization". http://fmt.cs.utwente.nl/research/projects/SEQUOIA/, 2017.

[Sin90] D. Singer. "A fuzzy set approach to fault tree and reliability analysis". *Fuzzy Sets and Systems*, 34(2):145–155, 1990. DOI: 10.1016/0165-0114(90)90154-X, ISSN: 0165-0114.

[Sin96] Roslyn Mary Sinnamon. *Binary Decision Diagrams for Fault Tree Analysis*. PhD thesis, Loughborough University, 1996.

[SM93] K. P. Soman and Krishna B. Misra. "Fuzzy fault tree analysis using resolution identity and extension principle". *Journal of Fuzzy Mathematics*, 1:193–212, 1993.

[Soc17] Society of Automotive Engineers. *AS5506C: Architecture Analysis & Design Language*, 2017.

[SPN18] Stichting voor de Technische Wetenschappen, ProRail, and Nederlandse Organisatie voor Wetenschappelijk Onderzoek. "Explorail - english summary". `http://explorail.verdus.nl/1334`, 2018.

[Ste86] Karl Stecher. "Evaluation of large fault-trees with repeated events using an efficient bottom-up algorithm". *IEEE Transactions on Reliability*, 35:51–58, April 1986. DOI: `10.1109/TR.1986.4335344`, ISSN: 0018-9529.

[STR02] Gerhard Schellhorn, Andreas Thums, and Wolfgang Reif. "Formal fault tree semantics". In *Proceedings of the 6th World Conference on Integrated Design and Process Technology*, 2002.

[SVD+02] Michael Stamatelatos, William Vesely, Joanne Bechta Dugan, Joseph Fragola, Joseph Minarick, and Jan Railsback. *Fault Tree Handbook with Aerospace Applications*. Office of safety and mission assurance NASA headquarters, 2002.

[SWK95a] Dragan A. Savic, Godfrey A. Walters, and Jezdimir Knezevic. "Optimal opportunistic maintenance policy using genetic algorithms 1: formulation". *Journal of Quality in Maintenance Engineering*, 1(2):34–49, 1995. DOI: `10.1108/13552519510089574`, ISSN: 1355-2511.

[SWK95b] Dragan A. Savic, Godfrey A. Walters, and Jezdimir Knezevic. "Optimal opportunistic maintenance policy using genetic algorithms 2: analysis". *Journal of Quality in Maintenance Engineering*, 1(3):25–34, 1995. DOI: `10.1108/13552519510096378`, ISSN: 1355-2511.

[SyD11] Anil Sharma, G. S. yadava, and S. G. Deshmukh. "A literature review and future perspectives on maintenance optimization". *Journal of Quality in Maintenance Engineering*, 17(1):5–25, 2011. DOI: `10.1108/13552511111116222`, ISSN: 1355-2511.

[SYR+17] Stefano Schivo, Buğra M. Yildiz, Enno Ruijters, Christopher Gerking, Rajesh Kumar, Stefan Dziwok, Arend Rensink, and Mariëlle Stoelinga. "How to efficiently build a front-end tool for UPPAAL: A model-driven approach". In *Proceedings of the Symposium on Dependable Software Engineering: Theories, Tools and Appliations (SETTA)*, volume 10606 of *Lecture Notes on Computer Science*, pages 319–336. Springer, 2017. DOI: 10.1007/978-3-319-69483-2_19, ISBN: 978-3-319-69482-5.

[TD04] Zhihua Tang and Joanne Bechta Dugan. "Minimal cut set/sequence generation for dynamic fault trees". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 207–213. IEEE, January 2004. DOI: 10.1109/RAMS.2004.1285449, ISBN: 978-0-7803-8215-2.

[TFLT83] Hideo Tanaka, LT Fan, FS Lai, and K Toguchi. "Fault-tree analysis by fuzzy probability". *IEEE Transactions on Reliability*, 32(5):453–457, 1983. DOI: 10.1109/TR.1983.5221727, ISSN: 0018-9529.

[Tin10] Tiedo Tinga. "Application of physical failure models to enable usage and load based maintenance". *Reliability Engineering & System Safety*, 95(10):1061–1075, October 2010. DOI: 10.1016/j.ress.2010.04.015, ISSN: 0951-8320.

[TJ13] Tiedo Tinga and Rene Janssen. "The interplay between deployment and optimal maintenance intervals for complex multi-component systems". *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 227(3):227–240, June 2013. DOI: 10.1177/1748006X13480743, ISSN: 1748-006X.

[TRSS00] D. W. Twigg, A. V. Ramesh, U. R. Sandadi, and T. C. Sharma. "Modeling mutually exclusive events in fault trees". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*, pages 8–13. IEEE, January 2000. DOI: 10.1109/RAMS.2000.816276, ISBN: 0-7803-5848-1.

[U.S49] U.S. Department of Defense. *Procedures for performing a failure mode, effects and criticality analysis (MIL-P-1629)*, 1949.

[U.S90] U.S. Department of Defense. *Procedures for performing a failure mode, effects and criticality analysis (MIL-STD-1629A)*, 1990.

[UT12] University of Twente. "Arrangeer: Smart railroad maintenance engineering with stochastic model checking". http://fmt.cs.utwente.nl/research/projects/ArRangeer/, 2012.

[Van91] S. G. Vanneste. “A markov model for opportunity maintenance”. Technical Report FEW 476, Faculteit der Economische Wetenschappen, Tilburg University, 1991.

[Vau02] Jussi K. Vaurio. “Treatment of general dependencies in system fault-tree and risk analysis”. *IEEE Transactions on Reliability*, 51:278–287, September 2002. DOI: 10.1109/TR.2002.801848, ISSN: 0018-9529.

[Ves70] W. E. Vesely. “A time-dependent methodology for fault tree evaluation”. *Nuclear Engineering and Design*, 13(2):337–360, August 1970. DOI: 10.1016/0029-5493(70)90167-6.

[VG06] Erin Vadala and Christen Graham. “Downtime costs auto industry $22k/minute - survey”. https://news.thomasnet.com/companystory/downtime-costs-auto-industry-22k-minute-survey-481017, 2006.

[VGRH81] 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.

[VJK18] Matthias Volk, Sebastian Junges, and Joost-Pieter Katoen. “Fast dynamic fault tree analysis by model checking techniques”. *IEEE Transactions on Industrial Informatics*, 14(1):370–379, January 2018. DOI: 10.1109/TII.2017.2710316, ISSN: 1551-3202.

[VN70] W. E. Vesely and R. E. Narum. “PREP and KITT: computer codes for the automatic evaluation of a fault tree”. Technical report, Idaho Nuclear Corp., Idaho Falls, 1970.

[WBP07] Martin Walker, Leonardo Bottaci, and Yiannis Papadopoulos. “Compositional temporal fault tree analysis”. In *Proceedings of the International Symposium on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 4680 of *Lecture Notes on Computer Science*, pages 106–119. Springer, September 2007. DOI: 10.1007/978-3-540-75101-4_12, ISBN: 978-3-540-75100-7.

[Wes69] G. Westinghouse. “Improvement in steam-power-brake devices”, 1869. US Patent 88,929.

[WH00] Yuan-Shun Way and Der-Yu Hsia. “A simple component-connection method for building binary decision diagrams encoding a fault tree”. *Reliability Engineering & System Safety*, 70(1):59–70, October 2000. DOI: 10.1016/S0951-8320(00)00048-X, ISSN: 0951-8320.

[WM00] Pathirage Gamini Wijayarathna and Mamoru Maekawa. “Extending fault trees with an AND-THEN gate”. In *Proceedings of the 11th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pages 283–292. IEEE, October 2000. DOI: 10.1109/ISSRE.2000.885879, ISBN: 978-0-7695-0807-8.

[WP09] Martin Walker and Yiannis Papadopoulos. “Qualitative temporal analysis: Towards a full implementation of the Fault Tree Handbook”. *Control Engineering Practice*, 17(10):1115–1125, October 2009. DOI: 10.1016/j.conengprac.2008.10.003, ISSN: 0967-0661.

[WP10] Martin Walker and Yiannis Papadopoulos. “A hierarchical method for the reduction of temporal expressions in pandora”. In *Proceedings of the First Workshop on Dynamic Aspects in Dependability Models for Fault-Tolerant Systems (DYADEM-FTS)*, pages 7–12. ACM New York, April 2010. DOI: 10.1145/1772630.1772634, ISBN: 978-1-60558-916-9.

[WS00] Jim Walters and Robert Sumwalt. *Aircraft Accident Analysis: Final Reports*. McGraw Hill Professional, 2000. ISBN: 978-0-07-137984-7.

[WSB08] Max Walter, Markus Siegle, and Arndt Bode. “Opensesame: the simple but extensive, structured availability modeling environment”. *Reliability Engineering & System Safety*, 93(6):857–873, June 2008. DOI: 10.1016/j.ress.2007.03.034, ISSN: 0951-8320.

[WvG14] W. Wagner and P. H. A. J. M. van Gelder. “Applying ramssheep analysis for risk-driven maintenance”. In *Safety, Reliability and Risk Analysis: Beyond the Horizon*, pages 703–713. Taylor & Francis Group, 2014. ISBN: 978-1-138-00123-7.

[WX12] Yanfu Wang and Min Xie. “Approach to integrate fuzzy fault tree with Bayesian network”. *Procedia Engineering*, 45:131–138, 2012. DOI: 10.1016/j.proeng.2012.08.133, ISSN: 1877-7058.

[WXNM11] Yan Fu Wang, Min Xie, Kien Ming Ng, and Yi Fei Meng. “Quantitative risk analysis model of integrating fuzzy fault tree with Bayesian network”. In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 267–271, July 2011. DOI: 10.1109/ISI.2011.5984095, ISBN: 978-1-4577-0082-8.

[XHH+13] Bingfeng Xu, Zhiqiu Huang, Jun Hu, Ou Wei, and Yu Zhou. “Minimal cut sequence generation for state/event fault trees”. In *Proceedings of the 2013 Middleware Doctoral Symposium*, article no. 3. ACM New York, 2013. DOI: 10.1145/2541534.2541592, ISBN: 978-1-4503-2548-6.

[XYM+11] Jianwen Xiang, Kazuo Yanoo, Yoshiharu Maeno, Kumiko Tadano, Fumio Machida, Atsushi Kobayashi, and Takao Osaki. "Efficient analysis of fault trees with voting gates". In *Proceedings of the 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pages 230–239. IEEE, December 2011. DOI: 10.1109/ISSRE.2011.23, ISBN: 978-1-4577-2060-4.

[Yev11] Olexandr Yevkin. "An improved modular approach for dynamic fault tree analysis". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2011. DOI: 10.1109/RAMS.2011.5754437, ISBN: 978-1-4244-8857-5.

[Zad75] L. A. Zadeh. "The concept of a linguistic variable and its application to approximate reasoning". *Information Sciences*, 8(3):199–249, 1975. DOI: 10.1016/0020-0255(75)90036-5, ISSN: 0020-0255.

[ZMFW09] Xiaojie Zhang, Qiang Miao, Xianfeng Fan, and Dong Wang. "Dynamic fault tree analysis based on Petri nets". In *Proceedings of the 8th International Conference on Reliability, Maintainability and Safety (ICRMS)*, pages 138–142. IEEE, July 2009. DOI: 10.1109/ICRMS.2009.5270223, ISBN: 978-1-4244-4903-3.

[ZWST03] Xinyu Zang, Dazhi Wang, Hairong Sun, and Kishor S. Trivedi. "A BDD-based algorithm for analysis of multistate systems with multistate components". *IEEE Transactions on Computers*, 52(12):1608–1618, December 2003. DOI: 10.1109/TC.2003.1252856, ISSN: 0018-9340.

[ZZLL11] Hong-Lin Zhang, Chun-Yuan Zhang, Dong Liu, and Rui Li. "A method of quantitative analysis for dynamic fault tree". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2011. DOI: 10.1109/RAMS.2011.5754471, ISBN: 978-1-4244-8857-5.

Part V

Appendices

Appendix A

Questionnaire on EI-joint

To elicit more information from experts at ProRail and several contractors, we sent out the following questionnaire (Translation from Dutch in italics):

Beantwoord alstublieft voor elke regel in het IHC: *Please answer for each line of the FMECA:*

1. Wat is de gemiddelde standtijd van deze faalvorm als geen onderhoud wordt gepleegd? *How long would it take, on average, for an EI-joint to exhibit this failure mode? Assume no maintenance is performed.*

2. Welk van Figure A.1 t/m A.5 geeft het degeneratie-/slijtgedrag van deze faalvorm het beste weer? In de grafieken gebeurt ongeveer 70% van de storingen tussen de lijnen 'snel' en 'langzaam'. *Which of Figures A.1 through A.5 best represents the wear/failure process of this failure mode? In the graphs approx. 70% of the failures occurs between the lines 'fast' and 'slow'.*

3. Als een inspection wordt uitgevoerd rond de helft van de standtijd, is het dan waarschijnlijk dat er duidelijke tekenen van slijtage gevonden worden? *If an inspection is performed around half the time before the expected time to failure, is it probable that significant signs of wear are observed?*

4. Als bij inspectie op de standtijd geen slijtage van deze faalvorm wordt opgemerkt, is het dan waarschijnlijk dat de faalvorm pas veel later dan geschat zal optreden? *If an inspection is performed at the expected time to failure, and no wear is observed, does this make it likely that the failure mode will only occur much later than expected?*

5. Treedt deze faalvorm regelmatig op kort na installatie? Bijvoorbeeld door fabricage- of installatiefouten. *Does this failure mode regularly occur shortly after installation? For example due to errors in manufacturing or installation.*

6. Hoe vaak treedt deze faalvorm op voor de helft van de standtijd? *How often does this failure mode occur before half of the expected time to failure?*

(a) Heel vaak (>50% van de gevallen).
Very often (>50% of the cases).

(b) Vaak (25 – 50% van de gevallen).
Often (25 – 50% of the cases).

(c) Soms (10 – 25% van de gevallen).
Sometimes (10 – 25% of the cases).

(d) Zelden (1 – 10% van de gevallen).
Rarely (1 – 10% of the cases).

(e) Vrijwel nooit (<1% van de gevallen).
Almost never (<1% of the cases).

7. Hoe vaak treedt deze faalvorm pas later dan 1,5 keer de standtijd op? *How often does this failure mode occur only after 1.5 times the expected time to failure?*

(a) Heel vaak (>50% van de gevallen).
Very often (>50% of the cases).

(b) Vaak (25 – 50% van de gevallen).
Often (25 – 50% of the cases).

(c) Soms (10 – 25% van de gevallen).
Sometimes (10 – 25% of the cases).

(d) Zelden (1 – 10% van de gevallen).
Rarely (1 – 10% of the cases).

(e) Vrijwel nooit (<1% van de gevallen).
Almost never (<1% of the cases).

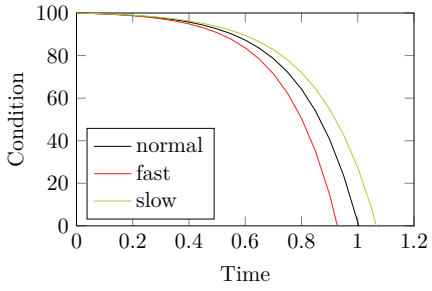


Figure A.1: **Nonlineaire slijtage, kleine spreiding.** *Nonlinear wear, small variance.*

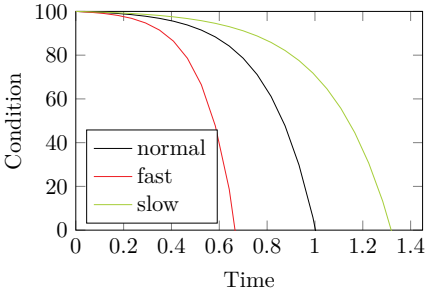


Figure A.2: **Nonlineaire slijtage, grote spreiging.** *Nonlinear wear, large variance.*

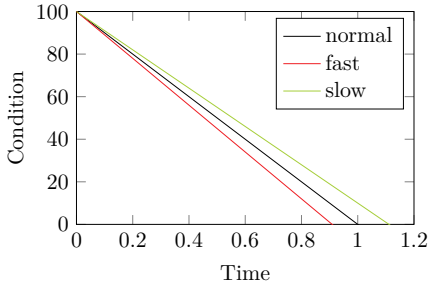


Figure A.3: **Lineaire slijtage, kleine spreiding.** *Linear wear, small variance.*

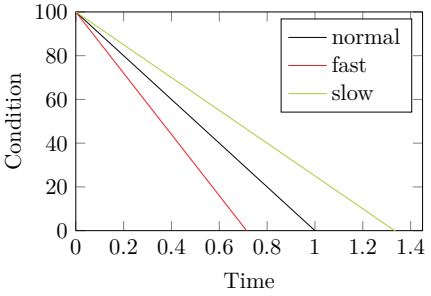


Figure A.4: **Lineaire slijtage, grote spreiding.** *Linear wear, large variance.*

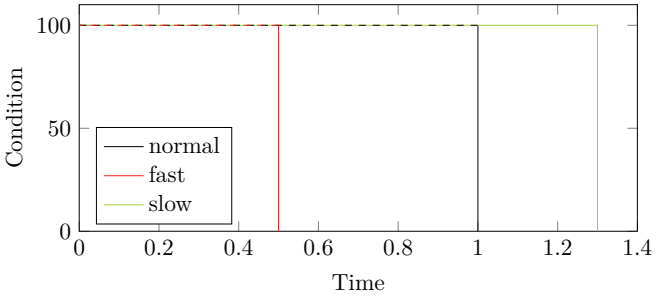


Figure A.5: **Geheugenloos (random) falen.** *Memoryless (random) failures.*

Appendix B

Numerical data used for plots

	Model		Computation time (s)		
			DFTCalc		
	N	**P**	**Generation**	**Model-checking**	**FTRES**
Railway cabinets	2	1	4	<0.01	600
	3	1	9	<0.01	600
	4	1	176	<0.01	601
	2	2	12	3	600
	3	2	301	38	602
	4	2	>7200	>7200	609
	2	3	70	4261	602
	3	3	7114	>7200	611
	4	3	>7200	>7200	734
FTPP	1	1	4	<0.01	600
	2	1	35	<0.01	601
	3	1	1242	<0.01	602
	4	1	41714	<0.01	603
	1	2	15	.7	601
	2	2	2469	2296	603
	3	2	>7200	>7200	609
	4	2	>7200	>7200	626
	N	**k**			
HECS	1	1	10.3	<0.01	600.71
	2	1	>7200	>7200	603.07
	2	2	>7200	>7200	601.19
	3	1	>7200	>7200	702.71
	3	2	>7200	>7200	610.95
	3	3	>7200	>7200	602.04
	4	1	>7200	>7200	5328
	4	2	>7200	>7200	1063.22
	4	3	>7200	>7200	634.54
	4	4	>7200	>7200	603.33

Table B.1: Computation times as shown in Figure 6.12 (FTRES times include 600 seconds computation time). Time-out occurred when all remaining programs had executed at least 7200 seconds.

	Model		Number of states		
			DFTCalc		
	N	**P**	**Peak**	**Final**	**FTRES**
Railway cabinets	2	1	741	26	187
	3	1	5279	26	465
	4	1	133993	26	1395
	2	2	17987	712	1851
	3	2	504183	1580	7021
	4	2	-	-	35803
	2	3	165693	7493	10231
	3	3	11475195	33383	55641
	4	3	-	-	459307
FTPP	1	1	783	14	193
	2	1	15965	34	589
	3	1	310791	115	2299
	4	1	5998865	115	2299
	1	2	17059	356	1327
	2	2	1626451	5830	5575
	3	2	-	-	16495
	4	2	-	-	39811
	N	**k**			
HECS	1	1	10277	41	141
	2	1	47054585	-	3845
	2	2	34954881	-	521
	3	1	-	-	103069
	3	2	-	-	12619
	3	3	-	-	1143
	4	1	-	-	2780937
	4	2	-	-	322121
	4	3	-	-	29511
	4	4	-	-	2007

Table B.2: Numbers of states as shown in Figure 6.15. Absent numbers correspond to time-outs in the computation. On HECS(2,1) and HECS(2,2), DFTCalc timed out during the final minimisation, which is why these entries have peak values but no final values.

Publications by the author

- Dennis Guck, Mark Timmer, Hassan Hatefi, Enno Ruijters, and Mariëlle Stoelinga. "Modelling and analysis of markov reward automata". In Franck Cassez and Jean-François Raskin, editors, *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 8837 of *Lecture Notes on Computer Science*, pages 168–185. Springer, November 2014. DOI: 10.1007/978-3-319-11936-6_13, ISBN: 978-3-319-11935-9.

- Enno Ruijters and Mariëlle Stoelinga. "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools". *Computer Science Review*, 15–16:29–62, 2015. DOI: 10.1016/j.cosrev.2015.03.001, ISSN: 1574-0137.

- Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. "Quantitative attack tree analysis via priced timed automata". In Sriram Sankaranarayanan and Enrico Vicario, editors, *Proceedings of the 13th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 9268 of *Lecture Notes on Computer Science*, pages 156–171. Springer, September 2015. DOI: 10.1007/978-3-319-22975-1_11, ISBN: 978-3-31922974-4.

- Enno Ruijters, Dennis Guck, Peter Drolenga, and Mariëlle Stoelinga. "Fault maintenance trees: reliability contered maintenance via statistical model checking". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2016. DOI: 10.1109/RAMS.2016.7447986, ISBN: 978-1-5090-0248-1.

- Enno Ruijters, Dennis Guck, Martijn van Noort, and Mariëlle Stoelinga. "Reliability-centered maintenance of the electrically insulated railway joint via fault tree analysis: A practical experience report". In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 662–669, 2016. DOI: 10.1109/DSN.2016.67, ISBN: 978-1-4673-8891-7.

- Enno Ruijters, Dennis Guck, Peter Drolenga, Margot Peters, and Mariëlle Stoelinga. "Maintenance analysis and optimization via statistical model checking: Evaluation of a train's pneumatic compressor". In *Proceedings of the 13th International Conference on Quantitative Evaluation of SysTems (QEST)*, volume 9826 of *Lecture Notes on Computer Science*, pages 331–347. Springer, August 2016. DOI: 10.1007/978-3-319-43425-4_22, ISBN: 978-3-319-43424-7.

- Enno Ruijters and Mariëlle Stoelinga. "Better railway engineering through statistical model checking". In *Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, volume 9952 of *Lecture Notes on Computer Science*, pages 151–165. Springer, October 2016. DOI: 10.1007/978-3-319-47166-2_10, ISBN: 978-3-319-47165-5.

- Enno Ruijters, Stefano Schivo, Mariëlle Stoelinga, and Arend Rensink. "Uniform analysis of fault trees through model transformations". In *Proceedings of the Reliability and Maintainability Symposium (RAMS)*. IEEE, January 2017. DOI: 10.1109/RAM.2017.7889759, ISBN: 978-1-5090-5284-4.

- Enno Ruijters, Daniël Reijsbergen, Pieter-Tjerk de Boer, and Mariëlle Stoelinga. "Rare event simulation for dynamic fault trees". In *Proceedings of the International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 10488 of *Lecture Notes on Computer Science*, pages 20–35. Springer, September 2017. DOI: 10.1007/978-3-319-66266-4_2, ISBN: 978-3-319-66265-7.

- Stefano Schivo, Buğra M. Yildiz, Enno Ruijters, Christopher Gerking, Rajesh Kumar, Stefan Dziwok, Arend Rensink, and Mariëlle Stoelinga. "How to efficiently build a front-end tool for UPPAAL: A model-driven approach". In *Proceedings of the Symposium on Dependable Software Engineering: Theories, Tools and Appliations (SETTA)*, volume 10606 of *Lecture Notes on Computer Science*, pages 319–336. Springer, 2017. DOI: 10.1007/978-3-319-69483-2_19, ISBN: 978-3-319-69482-5.

- Rajesh Kumar, Stefano Schivo, Enno Ruijters, Buğra M. Yildiz, David Huistra, Jacco Brandt, Arend Rensink, and Mariëlle Stoelinga. "Effective analysis of attack trees: a model-driven approach". In *Proceedings of the 21st International Conference on Fundamental Approaches to Software Engineering (FASE)*, volume 10802 of *Lecture Notes on Computer Science*, pages 56–73. Springer, 2018. DOI: 10.1007/978-3-319-89363-1_4, ISBN: 978-3-319-89362-4.

Titles in the IPA Dissertation Series since 2015

G. Alpár. *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World*. Faculty of Science, Mathematics and Computer Science, RU. 2015-01

A.J. van der Ploeg. *Efficient Abstractions for Visualization and Interaction*. Faculty of Science, UvA. 2015-02

R.J.M. Theunissen. *Supervisory Control in Health Care Systems*. Faculty of Mechanical Engineering, TU/e. 2015-03

T.V. Bui. *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness*. Faculty of Mathematics and Computer Science, TU/e. 2015-04

A. Guzzi. *Supporting Developers' Teamwork from within the IDE*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

T. Espinha. *Web Service Growing Pains: Understanding Services and Their Clients*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

S. Dietzel. *Resilient In-network Aggregation for Vehicular Networks*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

E. Costante. *Privacy throughout the Data Cycle*. Faculty of Mathematics and Computer Science, TU/e. 2015-08

S. Cranen. *Getting the point — Obtaining and understanding fixpoints in model checking*. Faculty of Mathematics and Computer Science, TU/e. 2015-09

R. Verdult. *The (in)security of proprietary cryptography*. Faculty of Science, Mathematics and Computer Science, RU. 2015-10

J.E.J. de Ruiter. *Lessons learned in the analysis of the EMV and TLS security protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2015-11

Y. Dajsuren. *On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems*. Faculty of Mathematics and Computer Science, TU/e. 2015-12

J. Bransen. *On the Incremental Evaluation of Higher-Order Attribute Grammars*. Faculty of Science, UU. 2015-13

S. Picek. *Applications of Evolutionary Computation to Cryptology*. Faculty of Science, Mathematics and Computer Science, RU. 2015-14

C. Chen. *Automated Fault Localization for Service-Oriented Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-15

S. te Brinke. *Developing Energy-Aware Software*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-16

R.W.J. Kersten. *Software Analysis Methods for Resource-Sensitive Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2015-17

J.C. Rot. *Enhanced coinduction.* Faculty of Mathematics and Natural Sciences, UL. 2015-18

M. Stolikj. *Building Blocks for the Internet of Things.* Faculty of Mathematics and Computer Science, TU/e. 2015-19

D. Gebler. *Robust SOS Specifications of Probabilistic Processes.* Faculty of Sciences, Department of Computer Science, VUA. 2015-20

M. Zaharieva-Stojanovski. *Closer to Reliable Software: Verifying functional behaviour of concurrent programs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-21

R.J. Krebbers. *The C standard formalized in Coq.* Faculty of Science, Mathematics and Computer Science, RU. 2015-22

R. van Vliet. *DNA Expressions – A Formal Notation for DNA.* Faculty of Mathematics and Natural Sciences, UL. 2015-23

S.-S.T.Q. Jongmans. *Automata-Theoretic Protocol Programming.* Faculty of Mathematics and Natural Sciences, UL. 2016-01

S.J.C. Joosten. *Verification of Interconnects.* Faculty of Mathematics and Computer Science, TU/e. 2016-02

M.W. Gazda. *Fixpoint Logic, Games, and Relations of Consequence.* Faculty of Mathematics and Computer Science, TU/e. 2016-03

S. Keshishzadeh. *Formal Analysis and Verification of Embedded Systems for Healthcare.* Faculty of Mathematics and Computer Science, TU/e. 2016-04

P.M. Heck. *Quality of Just-in-Time Requirements: Just-Enough and Just-in-Time.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2016-05

Y. Luo. *From Conceptual Models to Safety Assurance – Applying Model-Based Techniques to Support Safety Assurance.* Faculty of Mathematics and Computer Science, TU/e. 2016-06

B. Ege. *Physical Security Analysis of Embedded Devices.* Faculty of Science, Mathematics and Computer Science, RU. 2016-07

A.I. van Goethem. *Algorithms for Curved Schematization.* Faculty of Mathematics and Computer Science, TU/e. 2016-08

T. van Dijk. *Sylvan: Multi-core Decision Diagrams.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2016-09

I. David. *Run-time resource management for component-based systems.* Faculty of Mathematics and Computer Science, TU/e. 2016-10

A.C. van Hulst. *Control Synthesis using Modal Logic and Partial Bisimilarity – A Treatise Supported by Computer Verified Proofs.* Faculty of Mechanical Engineering, TU/e. 2016-11

A. Zawedde. *Modeling the Dynamics of Requirements Process Improvement.*

Faculty of Mathematics and Computer Science, TU/e. 2016-12

F.M.J. van den Broek. *Mobile Communication Security*. Faculty of Science, Mathematics and Computer Science, RU. 2016-13

J.N. van Rijn. *Massively Collaborative Machine Learning*. Faculty of Mathematics and Natural Sciences, UL. 2016-14

M.J. Steindorfer. *Efficient Immutable Collections*. Faculty of Science, UvA. 2017-01

W. Ahmad. *Green Computing: Efficient Energy Management of Multiprocessor Streaming Applications via Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-02

D. Guck. *Reliable Systems – Fault tree analysis via Markov reward automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-03

H.L. Salunkhe. *Modeling and Buffer Analysis of Real-time Streaming Radio Applications Scheduled on Heterogeneous Multiprocessors*. Faculty of Mathematics and Computer Science, TU/e. 2017-04

A. Krasnova. *Smart invaders of private matters: Privacy of communication on the Internet and in the Internet of Things (IoT)*. Faculty of Science, Mathematics and Computer Science, RU. 2017-05

A.D. Mehrabi. *Data Structures for Analyzing Geometric Data*. Faculty of Mathematics and Computer Science, TU/e. 2017-06

D. Landman. *Reverse Engineering Source Code: Empirical Studies of Limitations and Opportunities*. Faculty of Science, UvA. 2017-07

W. Lueks. *Security and Privacy via Cryptography – Having your cake and eating it too*. Faculty of Science, Mathematics and Computer Science, RU. 2017-08

A.M. Şutîi. *Modularity and Reuse of Domain-Specific Languages: an exploration with MetaMod*. Faculty of Mathematics and Computer Science, TU/e. 2017-09

U. Tikhonova. *Engineering the Dynamic Semantics of Domain Specific Languages*. Faculty of Mathematics and Computer Science, TU/e. 2017-10

Q.W. Bouts. *Geographic Graph Construction and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2017-11

A. Amighi. *Specification and Verification of Synchronisation Classes in Java: A Practical Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-01

S. Darabi. *Verification of Program Parallelization*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-02

J.R. Salamanca Tellez. *Coequations and Eilenberg-type Correspondences*. Faculty of Science, Mathematics and Computer Science, RU. 2018-03

P. Fiterău-Broştean. *Active Model Learning for the Analysis of Network Protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2018-04

D. Zhang. *From Concurrent State Machines to Reliable Multi-threaded Java Code*. Faculty of Mathematics and Computer Science, TU/e. 2018-05

H. Basold. *Mixed Inductive-Coinductive Reasoning Types, Programs and Logic*. Faculty of Science, Mathematics and Computer Science, RU. 2018-06

A. Lele. *Response Modeling: Model Refinements for Timing Analysis of Runtime Scheduling in Real-time Streaming Systems*. Faculty of Mathematics and Computer Science, TU/e. 2018-07

N. Bezirgiannis. *Abstract Behavioral Specification: unifying modeling and programming*. Faculty of Mathematics and Natural Sciences, UL. 2018-08

M.P. Konzack. *Trajectory Analysis: Bridging Algorithms and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2018-09

E.J.J. Ruijters. *Zen and the art of railway maintenance: Analysis and optimization of maintenance via fault trees and statistical model checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-10

Samenvatting

Onderhoud is cruciaal voor de werking van moderne systemen. Tijdige inspecties, reparaties, en vervangingen helpen dure storingen te voorkomen, en verzekeren dat systemen goed en veilig blijven functioneren.

Tegelijkertijd kost dit onderhoud veel geld. Er is personeel nodig, reserveonderdelen moeten gekocht worden, en vaak is het systeem niet bruikbaar terwijl er een inspectie of reparatie wordt uitgevoerd. Te veel onderhoud is verspilling van geld, hindert het gebruik van het systeem, en kan zelfs ongelukken veroorzaken als het onderhoud onjuist wordt uitgevoerd. Het is dus belangrijk om een *onderhoudsbeleid* te vinden met een goede balans van kosten en betrouwbaarheid.

Om deze balans te vinden, moeten we begrijpen hoe een systeem mettertijd slijt, en wat de gevolgen zijn van mogelijke handelingen om deze slijtage te verhelpen of voorkomen. Dit proefschrift presenteert *onderhoudsfoutenbomen* (FMTs, Engels: *fault maintenance trees*), een nieuw formalisme voor kwantitatieve analyse van de gevolgen van onderhoud op kosten en systeembetrouwbaarheid, om de analyse en verbetering van onderhoudsbeleid te ondersteunen.

FMTs zijn gebaseerd op de industriële standaard *foutenbomen*, die al decennia gebruikt worden om de betrouwbaarheid van veiligheidskritische system, zoals kerncentrales en vliegtuigen, te bestuderen. Foutenbomen zijn in de jaren '60 ontwikkeld, en sindsdien is er een scala aan uitbreidingen en varianten ontwikkeld. Deze ondersteunen de analyse van systemen met tijdsafhankelijke gevolgen van storingen, onzekere faalkansen, en diverse andere eigenschappen. Het eerste deel van dit proefschrift brengt het oerwoud aan uitbreidingen van foutenbomen in kaart, met een overzicht van meer dan 150 artikelen over dit onderwerp.

Het tweede deel van dit proefschrift introduceert FMTs, die foutenbomen uitbreiden met onderhoudsacties zoals inspecties en vervangingen van onderdelen. Met deze informatie kunnen we de kans van een systeemstoring berekenen gegeven een bepaald onderhoudsplan. FMTs bevatten ook informatie over de kosten van verschillende onderhoudsacties en storingen, waarmee de verwachte totale kosten van een gegeven onderhoudsplan berekend kunnen worden. Hiermee maken FMTs het mogelijk om verschillende onderhoudsplannen te vergelijken wat betreft hun effecten op systeembetrouwbaarheid en -kosten, waardoor FMTs ondersteuning bieden bij het kiezen van het onderhoudsbeleid met de beste balans hiertussen.

Technisch worden FMTs doorgerekend door middel van *statistisch model checking* (SMC), een moderne techniek om complexe systemen te analyseren zonder de grote hoeveelheden geheugen te gebruiken die veel andere analysetechnieken voor uitgebreide foutenbomen nodig hebben. SMC maakt het mogelijk om statistisch gegronde betrouwbaarheidsintervallen te berekenen van kwantitatieve maten zoals kosten, systeembetrouwbaarheid en het verwachte aantal storingen per tijdseenheid.

SMC werkt voor veel systemen goed, maar heeft een nadeel dat in onze context erg merkbaar is: Nauwkeurige schattingen van lage kansen hebben veel reken-

tijd nodig. Daarom presenteren we ook een tweede analysetechniek voor FMTs, gebaseerd op het recent ontwikkelde Path-ZVA algoritme voor de simulatie van zeldzame gebeurtenissen (Engels: *rare event simulation*). Hoewel deze techniek nu beperkt is tot het berekenen van de gemiddelde beschikbaarheid van een systeem, is hierbij veel minder rekentijd nodig dan voor SMC bij systemen met een hoge beschikbaarheid, terwijl de statistische garanties van SMC bewaard blijven.

Ten slotte willen we dat FMTs ook in de praktijk toepasbaar zijn. Hiervoor presenteren we in het derde deel van dit proefschrift twee casussen uit de spoorwegindustrie: een elektrische scheidingslas en een luchtcompressor. Deze casussen zijn uitgevoerd in nauwe samenwerking met onze industriële partners, en laten zien dat FMTs echte systemen en onderhoudsplannen nauwkeurig kunnen modelleren, alsmede inzichten kunnen opleveren die kunnen helpen om onderhoudsplannen te verbeteren.

Maintenance is crucial for the operation of modern systems, to prevent failures and ensure that systems function properly and safely.

This maintenance is also costly. Too much maintenance wastes time and parts, and prevents the system from being used for its intended purpose. It is therefore important to find a good maintenance policy that balances cost and dependability.

This thesis introduces a novel formalism to assess the cost and effectiveness of a maintenance policy: Fault Maintenance Trees. By combining the industry-standard technique of fault tree analysis with advanced maintenance models, we can calculate how much the maintenance will cost, and how many failures it prevents. Thus, we can compare different maintenance strategies and choose the one that best fits the needs of a particular system.