Boosting Fault Tree Analysis by Formal Methods

Joost-Pieter Katoen[1,2] and Mariëlle Stoelinga[2]

[1] RWTH Aachen University, Germany
[2] University of Twente, the Netherlands

Abstract. Fault trees are a key technique in safety and reliability engineering. Their application includes aerospace, nuclear power, car, and process engineering industries. Various fault tree extensions exist that increase expressiveness while yielding succinct models. Their analysis is a main bottleneck: techniques do not scale and require manual effort. Formal methods have an enormous potential to solve these issues. We discuss a mixture of formal method techniques resulting in a fully automated and scalable approach to analyze Dugan's dynamic fault trees.

1 Introduction

Fault trees are ubiquitous. Fault trees were developed in 1961 at Bell Labs. A few years later Boeing started to use fault tree analysis (FTA, for short) for civil aircraft design. The U.S. Nuclear Regulatory Commission published the NRC Fault Tree Handbook in 1981. Several other industries followed later with their FTA standards. Since the Challenger accident in 1986, NASA considers FTA as a key system reliability and safety analysis technique. The U.S. Federal Aviation Administration's System Safety Handbook (2000) advocates the use of FTA. Fault trees are used on a daily basis by millions of engineers around the world. For example, after the explosion of the (unmanned) Falcon-9 rocket in 2015, the SpaceX CEO posted the following on Twitter [42]:

> "That's all we can say with confidence right now. Will have more to say following a thorough fault tree analysis."

What are fault trees? They are directed acyclic graphs. Leaves model individual component failures or human errors. As errors in FTA are assumed to happen randomly, leaves are equipped with a continuous probability distribution. Internal vertices (a.k.a.: nodes), commonly referred to as gates, model how component failures lead to system failures. Gates are like logical elements in circuits such as AND and OR (but no inverters). FTA amounts to determine the failure probability of the root of the fault tree, called the top-level event. Fault trees that only contain logical gates such as AND and OR are called *static*. Static fault tree analysis can be efficiently done using binary decision diagrams [3]. The key step in the analysis is determining a minimal cut set. This is a set of leaves of

[3] BDDs are succinct representations for switching functions. In 1990, their use in formal methods, in particular formal verification, has been introduced [11].

minimal cardinality whose failures together causes the top-level event to fail. The analysis of static fault trees is simple as the ordering of failure is irrelevant; it only matters whether a leaf has failed or not.

Dynamic fault trees. Static fault trees are too simple for practical systems. This has led to several extensions; for a recent survey see [49]. Dugan's *dynamic* fault trees [16] (DFTs, for short) are the most well-known and commonly used. The behaviour of a DFT not only depends on the set of failed leaves, but also on their order. Thus, DFTs have a richer set of gates and are more expressive than static fault trees. This, however, comes at a price. Their analysis can no longer be done using minimal cut sets. Instead, their behaviour is state-dependent. DFT analysis is typically done by distilling a stochastic process, mostly a continuous-time Markov chain (CTMC, for short), from the DFT. Markov chain analysis is used to obtain information about the probability of the top-level event to fail.

The challenge. Conceptually, this sounds simple. In practice it is not. This leads to the belief that DFT analysis is a difficult problem. For instance, [54] and [43] argue that a state-based approach for dynamic gates is not "realistic" due to the state-space explosion on increasing the DFT size. Indeed, DFTs in practice are large. Hundreds of nodes is not an exception. Their Markov chains consist of millions or even billions of states. State-space generation is a major bottleneck in DFT analysis. This complicates their analysis considerably. As [20] argues:

> "Although DFTs are powerful in modeling systems with dynamic failure behaviors, their quantitative analyses are pretty much troublesome, especially for large scale and complex DFTs."

Or, in the most recent survey paper on fault tree analysis [32]:

> "Although many extensions of fault trees have been proposed, they suffer from a variety of shortcomings. In particular, even where software tool support exists, these analyses require a lot of manual effort."

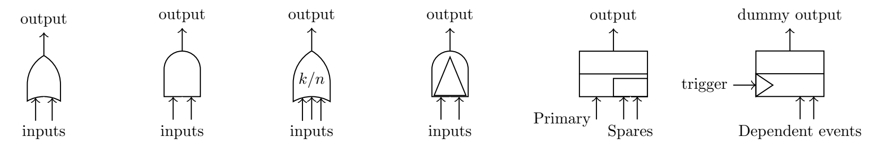
Model checking. In our opinion, this common belief is way too pessimistic! We know that formal methods are not a panacea. However, we argue in this paper that probabilistic and statistical model checking can alleviate the above mentioned "problems" and "shortcomings" to a very large extent. Model checking [4] is a systematic way to analyze the state space with powerful algorithms. It is heavily used in hardware industry to verify IC designs, and the founding fathers of model checking won the prestigious ACM Turing award in 2007.

Probabilistic model checking combines standard model checking techniques with clever stochastic methods to obtain efficient numerical algorithms. Statistical model checking, a state-of-the-art Monte Carlo simulation technique, is more widely applicable and is far less dependent on the state space size. However, it is not an exhaustive technique and requires special treatment of rare events and nondeterminism. In this paper, we show that due to unremitting improvements of state-space generation techniques in the field of probabilistic and statistical model checking, extremely large state spaces can nowadays be treated, both numerically and statistically. In particular, we show how techniques like compositionality, abstraction, partial order reduction, graph rewriting, and abstraction-refinement can be exploited to analyse large DFTs in a matter of minutes.

Take-home message. As such, this paper argues that *FTA is a playground par excellence for formal methods.* Formal methods boost dynamic fault tree analysis significantly and result in a fully automated and software-supported approach.

2 Dynamic Fault Trees in a Nutshell

What are DFTs? Dynamic fault trees (DFTs) [16] are directed acyclic graphs consisting of gates and leaves. A DFT has a distinguished root node, called the top-level event (TLE, for short). DFT leaves represent component failures, called basic events (BEs, for short). DFTs describe how component failures propagate



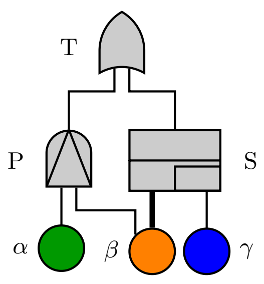
(a) OR (b) AND (c) VOTING (d) PAND (e) SPARE (f) FDEP

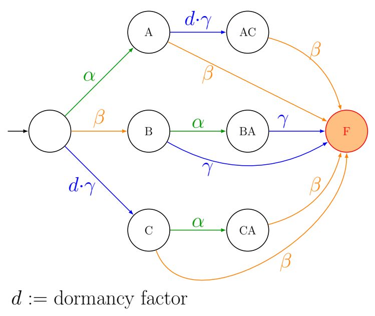
Fig. 1: Gates in dynamic fault trees

through the system. Gates, depicted in Fig. 1, model failure propagation. The static gates OR, AND, VOT(k) fail if respectively one, all or k (out of $n \geq k$) of their inputs fail. The PAND, SPARE, and FDEP are dynamic gates. A PAND-gate fails if the inputs fail from left to right; if the components fail in any other order, then no failure occurs. A SPARE-gate contains one primary, and one or more spare inputs. If the primary input fails, then the leftmost dormant spare takes over its functionality, putting the spare from dormant into active mode. If all spares have failed too, then the SPARE-gate fails. Primary and spares can be entire DFTs, and spares can be shared among several gates. An FDEP-gate contains a trigger input, which instantaneously triggers the failure of all its dependent events.

DFT leaves can be either dormant, active, or failed. Component failures are governed by continuous distribution functions, e.g., exponential probability distributions. Dormant leaves fail less frequently as they are not in use. Their failure rate λ is reduced by a dormancy factor d in the interval $[0, 1]$. The probability for an active component to fail within time t is $1 - e^{-\lambda \cdot t}$ and $1 - e^{-d \cdot \lambda \cdot t}$ for a dormant component. Fig. 2(a) depicts a simple sample DFT.

The DFT's Markov chain. DFTs have an internal state, e.g., the order in which failures occur influences the internal state, and thus whether the designated top





(a) (b)

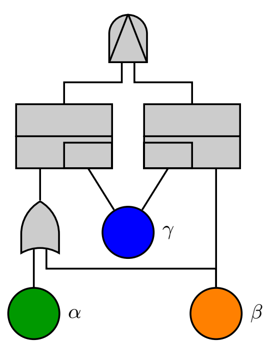
Fig. 2: A (a) sample DFT with three leaves, an OR-gate (top-level event T) and a PAND-gate P and a SPARE-gate S (T's children), and (b) its CTMC.

event has failed. The behaviour of DFTs can be naturally described by CTMCs, where transitions correspond to the failure of a basic event. Fig. 2(b) depicts the CTMC of our sample DFT. Initially, any of the leaves can fail with failure rates α, β, and γ, respectively. As the rightmost leaf is dormant, its failure rate is reduced by d. Once this leaf becomes active, e.g., in CTMC state B, its failure rate becomes γ. In the rightmost CTMC state F, the TLE and thus the entire DFT has failed. Due to the expressive power of DFTs, their interpretation is not always clear; an in-depth discussion on this can be found in [31].

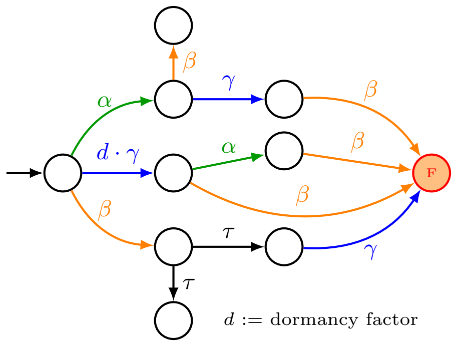
Nondeterminism. Most DFTs are fully probabilistic. They do not exhibit any nondeterminism. Their behaviour can adequately be described by CTMCs. Some DFTs give rise to nondeterminism. An example is provided in Fig. 3(a). The two SPARE gates share a spare. Once the rightmost leaf fails first, the primary child of each SPARE fails. A "race" occurs between the left and right SPARE to use the middle leaf (in blue). This race is nondeterministic. It fundamentally differs from a probabilistic choice as there is no quantitative information available about how to resolve this race. As a result, the underlying model for a DFT is a CTMC with nondeterminism, a so-called *interactive* Markov chain (IMC, for short) [28, 29]. Fig. 3(b) depicts the IMC of the DFT in Fig. 3(a). The nondeterministic choice occurs after the occurrence of the β-transition in the initial state. The two nondeterministic transitions, one for each possible resolution of the race, are labeled with τ. Note that if the race is resolved in favor of the left SPARE-gate, the right SPARE-gate fails, and due to the top-most PAND-gate, the DFT can never fail.

3 Compositional State-Space Generation

A crucial step in DFT analysis is to generate the state space underlying a DFT. Each state records for each BE its status, i.e., whether it is up or down and whether it is operational or not. Key result in [6, 7] is to perform this via *compositional aggregation*, a.k.a. iterative minimization. Rather than generating the



(a)



(b)

Fig. 3: A (a) sample DFT with three leaves, a PAND-gate (top-level event), two SPARE-gates, and an OR-gate, arranged to create nondeterminism, (b) its IMC (assuming the PAND fails on simultaneous failures of its children).

whole state space at once—leading to a procedure that is difficult to understand and modify—[6, 7] generate a Markov model for each DFT element. Recently, Ammar *et al.* [1] advocated the use of compositional model generation combined with probabilistic model checking for DFTs using Markov decision processes (MDPs). The whole state space is then obtained by composing these Markov models in a smart way.

IOIMCs. Standard Markov models cannot be composed in a natural way; i.e., there are no adequate notions to build a larger Markov model from smaller ones. Hence, [5–7] use input/output interactive Markov chains (IOIMCs) [5]. IOIMCs combine CTMCs and labeled transition systems, see Fig. 4 for an example. They feature two types of transitions: *Markovian* transitions are labeled with the parameter λ (a.k.a.: rate) of an exponential distribution. Such a transition can be taken after an exponential waiting time, i.e., the probability to take this transition before time t is given by $1 - e^{-\lambda \cdot t}$. *Interactive* transitions are labeled with action labels and can be used to synchronize two or more IOIMCs. Interactive transitions feature three types of action labels: transitions labeled with *input* labels $a?$ indicate that the IOIMC waits for another component to provide a corresponding output label $a!$. Transitions labeled with input actions are delayable, meaning that the IOIMC can wait as long as needed to take this transition. *Output* actions $a!$ are immediate; i.e., as soon as the output action $a!$ is enabled, it has to be taken. In particular, this means that whenever a state enables both an output action and a Markovian action, the Markovian action is never taken as its probability to be taken immediately is zero. *Internal transitions* are like output actions, and hence immediate, with the difference being that the action label is not visible to the environment. Thus, internal actions are used to model steps that are internal to the component.

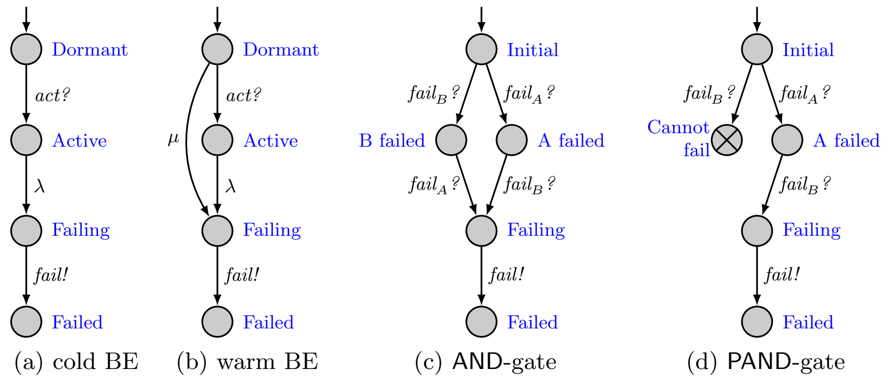


Fig. 4: Examples of the IOIMCs underlying DFTs.

Example 1. Fig. 4(a) depicts the IOIMC underlying a cold BE, a basic event that cannot fail in dormant mode. In the initial state, the IOIMC waits to be activated, i.e., it waits until it has received an input signal *act?* from its environment. If so, it moves to the state named Active. This state has a Markovian transition labeled with λ, indicating that the BE's failure rate is exponentially distributed with parameter λ. After failing, the IOIMC moves to state Failing, which has an outgoing transition labeled with *fail!*. As soon as the BE has failed, the IOIMC sends out a *fail!* signal, so that other components can update their state.

Fig. 4(b) depicts the IOIMC of a warm BE, i.e., a basic event that can fail in dormant mode, but with a reduced rate $\mu = d{\cdot}\lambda$. Now, in the initial state, two things may happen: Either the component is activated, and moves to state Active and the behavior is as before. Alternatively, the component fails before activation, which happens with a reduced failure rate μ, as modeled by the transition Dormant $\xrightarrow{\mu}$ Failing.

The IOIMC for an AND-gate C with children A and B is given in Fig. 4(c). When it has received failure signals from both its children, the IOIMC sends out a *fail!* signal. Finally, the IOIMC for the PAND-gate is given in Fig. 4(d): if the IOIMC receives a failure signal from A first, and then from B, then the IOIMC sends out a *fail!* signal; otherwise it moves to a sink state (indicated by X) from which it can never fail.

The IOIMCs for the other gates are similar, but more complex.

Smart composition. The CTMC underlying a DFT is obtained by composing all DFT-element IOIMCs via composition aggregation. That is, rather than composing all IOIMCs in one shot, all DFT-element models are composed one-by-one, in an iterative way. After each step, the models are reduced. Thus, the compositional aggregation procedure iteratively performs the following three steps: (1) Pick two IOIMCs and compose these. (2) Hide all actions that are not relevant for other components; i.e., actions that are not used for synchronization by other IOIMCs are made internal. (3) Reduce the model just obtained via minimization techniques such as weak bisimulation [5] or confluence reduction [51]. Action hiding makes that more states are equivalent, enabling stronger reductions. Minimization means that one replaces the model by an equivalent one that is smaller, for instance by grouping states that exhibit the same behavior. The order in which the models are composed does not matter for the end result; however, it impacts the memory footprint, i.e., the size of the intermediate Markov models. Heuristics have been developed to obtain a low peak memory usage [13].

This procedure has been implemented in the tool `CORAL` and its successor `DFTCalc` [2]. An advantage of this technique is its flexibility: adding new gates for instance, is easy, since one only has to provide the IOIMC for that new gate. For example, cyber attacks can easily be incorporated in this way [3, 36], and the same holds for maintenance strategies [24, 25]. A further improvement over existing methods is that the compositional approach is more liberal on the DFTs it can analyze. Earlier methods make rather severe assumptions on the DFTs to analyze, which limits the ability to model and analyze realistic systems. For example, dependent events of FDEP-gates could only be BEs, and the same holds for the spare inputs of a SPARE-gate. `CORAL` was the first tool to alleviate these restrictions.

Experiments. Several experiments have been carried out comparing `DFTCalc`'s predecessor `CORAL` to `Galileo` [50], the state-of-the-art tool at that time. The following case studies were used: Cascaded PAND system (CPS), Cardiac assist system (CAS), fault tolerant parallel processors (FTPP) [16], and a pump system with inherent nondeterminism (NDPS). Table 1 shows the benchmark results in terms of memory footprint (i.e., maximum number of states and transitions encountered during the analysis process) and in terms or running time. It also indicates the DFT's unreliability of the DFT, i.e., the probability that the DFT fails within a deadline. Except for the CAS system which has a very small state space, compositional analysis outperforms `Galileo`, both in time and memory usage. `CORAL` could analyze several variants of the the FTPP case where `Galileo` ran out of memory. Note that the NDPS system cannot be modeled in `Galileo`, since it does not support nondeterminism. Due to the nondeterminism, the unreliability of the NPDS system is an interval and not a single value as for the other cases.

4 Reduce, Reduce, and Reduce More

The previous section described a compositional approach for distilling a CTMC from a DFT. Its main advantage is that each DFT gate and leaf results in a relatively simple CTMC. These CTMCs can be reduced individually and in a pairwise fashion after being put in parallel. This reduces the peak memory consumption. The price is that the CTMC of a DFT gate needs to be equipped

Case study	**Approach**	**Max # of states**	**Max # of transitions**	**Unreliability**	**Run time (sec)**
CPS	`Galileo`	4,113	24,608	0.00135	490
	`CORAL`	133	465	0.00135	67
CPS	`Galileo`	8	10	0.65790	1
	`CORAL`	36	119	0.65790	94
CAS-PH	`CORAL`	40,052	265,442	0.112826	231
FTPP-4	`Galileo`	32,757	426,826	0.01922	13111
	`CORAL`	1,325	13,642	0.01922	65
FTPP-5	`CORAL`	43,105	643,339	0.00306	309
FTPP-6	`CORAL`	1,180,565	22,147,378	0.000453	1989
FTPP-C	`CORAL`	653,303	12,220,653	0.02136	1806
FTPP-A	`Galileo`	32,757	426,826	0.0167	13111
	`CORAL`	19,367	154,566	0.0167	240
NDPS	`CORAL`	61	169	[0.00586, 0.00598]	266

Table 1: Results of `CORAL` and `Galileo` (taken from [7]).

with extra transitions to enable its parallel composition with CTMCs of other DFT gates. They thus are slightly more complex due to the fact that they need to be composed. Another drawback is that the CTMCs of each DFT gate are "context free". That is to say, their behaviour does not take into account the context in which they are put. This is good on the one hand, as it means equal gates yield equal CTMCs, which can be exploited. On the other hand, it is bad as certain parts of the CTMCs might not be reachable if the context would be taken into account. For instance, if a given sub-tree can only become active once other parts of the DFT have failed, then parts of the sub-tree might not be relevant any more.

Revive the original approach. An alternative is to take the original `Galileo` [50] approach—the first tool for DFT analysis; it treats a DFT as monolithic entity—and modernise it using techniques to shrink the state space prior or during its generation. Techniques that can be adopted are: *symmetry reduction*, *partial-order reduction*, and *don't care detection*. Symmetry reduction recognises isomorphic sub-trees and stochastic independences among sub-trees by a static analysis of the DFT. It thus is *not* a symmetry reduction at state-space level, but rather at the DFT level. Sub-trees that become obsolete (don't care) after the occurrence of some failures in the DFT are pruned. Finally, one can detect superfluous nondeterminism such that certain failure orderings are irrelevant. A detailed account of this approach is given in [52, 53].

Monolithic state-space generation with don't care propagation. The principle is rather straightforward. Each gate and leaf in the DFT is provided with a status such as e.g., operational (OP), failed (F), fail-safe (FS), or don't care (X). For each SPARE-gate, one has to do some extra bookkeeping. One needs to keep track of the currently used child (a.k.a.: spare) (CUC). In addition, for each spare one needs information about whether it is active (A) or dormant (D). All this information together constitutes the state space of a DFT. A state is thus the status of each gate and leaf, plus some extra information for each SPARE-gate and its spares. State changes originate from the failure of one of the DFT leaves [4]. These state changes not only involve a status change of the just failed BE. It may affect the CUC of a SPARE-gate, may give rise to gates to become fail-safe (FS) as they cannot fail in the future anymore, and involves don't care propagation—a top-down pass over the DFT determining whether nodes have become don't care (X) as all their parents are F or FS.

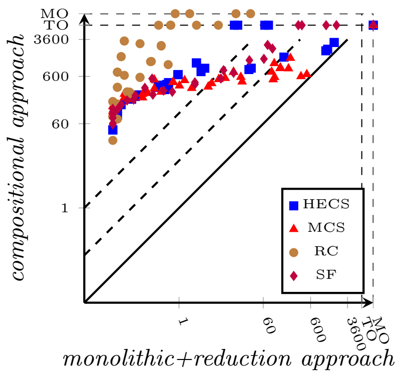
Partial-order reduction. In many DFTs, the actual order in which subsets of leaves fail is not crucial. This is exploited for FDEP-gates, where instead of exploring all interleavings over the triggered events one aims to only explore a single order. This can be done applying static partial order reduction [4, Ch. 8.2.4] to DFTs. A static analysis of the DFT identifies which dependencies can be executed in arbitrary order. If so, only a canonical order is expanded.

Symmetry reduction. The symmetry in a DFT, i.e., the presence of isomorphic sub-trees, can be exploited [12]. Faults in isomorphic sub-trees have the same effect if they are only connected to the remaining DFT via a single static node, i.e., an AND, OR, or a VOT($-$) gate. The states of symmetric sub-trees can then be swapped. Thus it is not important to administer which nodes in symmetric sub-trees have failed, but only how many and reduces the number of states.

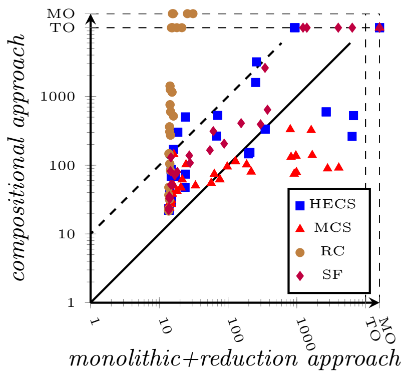
Experiments. Together with compact bit-level state representations and manipulations, bisimulation minimisation of the resulting CTMC, and modularisation, the above approach results in a well-performing modern version of `Galileo`. Modularisation [26] is a DFT technique that identifies independent sub-trees in the DFT, analyses them separately, and combines the results to a final result. Experiments show that intermediate state spaces are often ten times smaller compared to the compositional approach. This results in boosting the state-space generation by up to several orders of magnitude [5]. For some case studies, the compositional approach yields a smaller peak memory usage. For 164 problem instances, this approach solved 11% more cases (for analysing the DFT's reliability) than the compositional approach. Plots indicating the difference between the compositional and the revived `Galileo` approach are provided in Fig. 5. These are log-log scale plots. Fig. 5(a) compares the time consumption (which includes state-space generation and analysis). The lower dashed line indicates an advantage of our tool by a factor ten, the upper of a factor 100. The outer lines

[4] As failure probabilities are continuous probability distributions, the probability that two or more leaves fail simultaneously is zero.

[5] It is fair to say, that some of these effects are also due to a different implementation of the state-space generation process; the compositional approach as realised in the tool `DFTCalc` [2] is based on the CADP tool-box [19], whereas the monolithic approach is implemented [52] on top of the `storm` model checker [15].



(a) run time (seconds)



(b) memory footprint (MB)

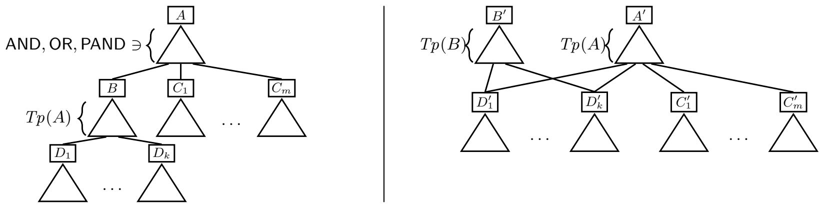
Fig. 5: Overview of the experimental results on four different benchmark sets (taken from [52, 53]).

indicate time-outs (TO) and memory-outs (MO), respectively. Fig. 5(b) indicates the peak memory consumption. Here one sees that for several benchmarks it is beneficial to employ compositional state-space generation and reduction, whereas for others it is not. The following benchmarks were used: Hypothetical Example Computer System (HECS) from the NASA FT handbook, Multiprocessor Computing System (MCS) [40], Railway Crossing (RC) [24] and the Sensor Filter (SF) [8]. The sizes of the corresponding CTMCs vary to up to one million states.

5 Fault Tree Rewriting and All That

Whereas the previous reductions work on the underlying CTMC or IOIMC, one can also reduce the DFT before any state is generated, thus obtaining a "slim" fault tree. This is the idea behind the paper *Fault trees on a diet* [30]: one rewrites a DFT into another one that is equivalent—in the sense that important measures-of-interest such as reliability and mean time-to-failure are preserved—, but whose state space is much smaller. Note that this does not necessarily mean that the fault tree itself is smaller.

Since fault trees are graphs, it is natural to use graph transformation systems for that. Graph transformation systems rewrite one graph into another one via transformation rules. These rules look for patterns in a graph, and if such a pattern is found, then it can be replaced by another pattern. In this way, nodes and edges can be removed or added, and attributes such as failure rates can be changed. For example, if two OR-gates are stacked on top of each other, then these gates can be glued into one large OR-gate as depicted as follows: In total, [30] has developed a set of 29 transformation rules, which have been implemented



Input: $\{C_i \mapsto C'_i\}_{i=1}^{m} \cup \{D_i \mapsto D'_i\}_{i=1}^{k}$
Output: $\{A \mapsto A', B \mapsto B'\}$

Rewrite rule 1: Left-flattening of AND-/OR-/PAND-gates

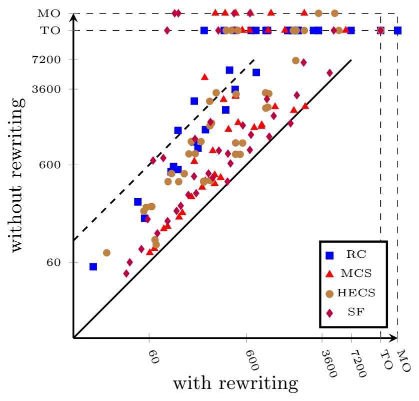
in the graph transformation tool GROOVE [22], and can be used in combination with any DFT analysis tool.

Experiments. The effect of rewriting was analyzed on 183 benchmarks, obtained by instantiating seven different, mostly industrial, case studies with different parameter values [30]. We investigated the influence of rewriting on (1) the number of nodes in the DFT, (2) the peak memory consumption, (3) the total analysis time (including model generation, rewriting, and analysis), as well as (4) the size of the resulting Markov chain, see Fig. 6(a)-(d). The base setting is the compositional minimization approach as realised in the tool DFTCalc without rewriting. These plots clearly show that rewriting DFTs improves the performance for all these criteria in almost all cases. Improvements of upto several orders of magnitude were obtained. In particular, 49 cases could be analysed that yielded a time-out (TO, two hours) or out-of-memory (MO, 8000 MB) in the base setting without rewriting. A more detailed analysis reveals that the graph rewriting with GROOVE is very fast, typically between 7 and 12 sec. Most time is devoted to the Markov chain construction and bisimulation minimisation. The analysis time of the resulting Markov chain using probabilistic model checking (see Section 3) is negligible.

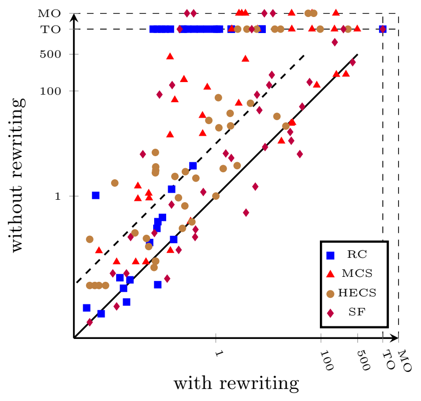
6 Abstract, Check, and Refine

Partial state-space generation. The approaches so far focused on the analysis of the DFT after the entire CTMC has been generated. This has the advantage that all information is available to get an exact [6] account of the DFT's measures-of-interest. In many cases, however, one is not interested in the exact mean time to failure (MTTF) or the exact probability that the top-level event fails within a certain time deadline (a.k.a.: reliability). Instead, in practice one often wants

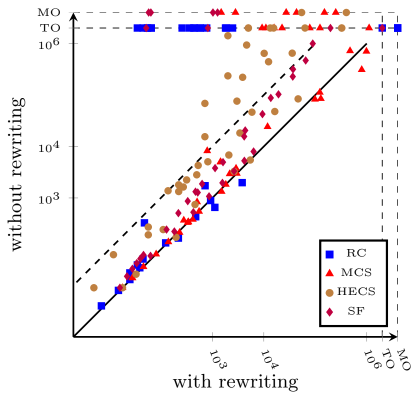
[6] Up to some numerical or simulative evidence.



(a) run time (seconds)



(b) memory footprint (MB)



(c) # states in MC

	solved		Σ time (h)			red.
	bs	rw	bs	rw$^{(1)}$	rw$^{(2)}$	$\frac{\lvert V_{\mathrm{rw}}\rvert}{\lvert V_{\mathrm{bs}}\rvert}$
HECS(44)	34	43	11.8	3.3	9.1	1.4
MCS(44)	30	43	9.3	3.7	8.2	1.1
RC(36)	15	31	7.3	5.1	9.3	2.1
SF(39)	31	38	10.1	5.3	7.1	1.5
MOV(8)	3	7	2.3	0.6	0.7	3.4
HCAS(8)	8	8	0.4	0.3	0.3	1.2
SAP(4)	4	4	0.1	0.1	0.1	1.7
total(183)	125	174	41.3	18.4	34.8	1.6

(1) time on instances solved by all.
(2) time on all instances solved.

(d) timing (bs = base)

Fig. 6: Overview of the experimental results on four different benchmark sets (taken from [30]).

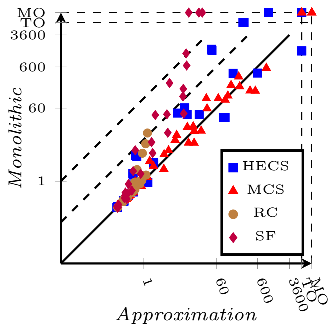
to know whether the reliability is below a given threshold or, similarly, whether the MTTF is beyond a certain value. To answer these queries, it suffices to analyse DFTs by considering their *partial* state space only. The simple idea is to generate only a—hopefully small—fragment of the DFT's CTMC. This goes along the way described in Section 4 except that one stops the state-space generation at a certain point, e.g., if a certain fraction of the DFT has been considered, a certain size of the CTMC has been reached, or similar. Inspired by the ISO 26262 standard where "high-order" failures are ignored, bounded depth exploration is a good possible termination criterion: any states encoding up to k-point failures are considered. This CTMC fragment is now used to obtain lower and upper bounds on the measure-of-interest, say MTTF. A detailed account of this approach can be found in [53].

Pessimistic abstraction. To obtain a lower bound on the MTTF, the DFTs failure probability is overestimated. Correspondingly, it is assumed that the failure of *any* additional leaf results in a TLE failure. This is easily realised by mildly adapting the state space fragment: a transition is added in the CTMC from each unexplored state to a failed state on the failure of any additional DFT leaf. The rate of such transition is the sum of the failure rates of the operational leaves. The resulting CTMC can be viewed as a *pessimistic abstraction* of the DFT. This results in a lower bound on the MTTF as it corresponds to the worst possible scenario. The true MTTF can not be worse. The lower bound thus is safe.

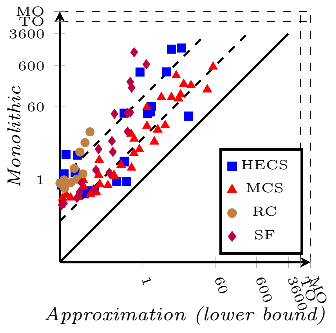
Optimistic abstraction. Symmetrically, an optimistic view is obtained by assuming that *all* of the unconsidered DFT leaves have to fail to cause the TLE to fail. This uses the mild assumption that the TLE always fails if all (fallible) leaves fail regardless of the order in which they fail [7]. This yields a safe upper bound, as the true MTTF can not be larger. The realisation of this optimistic perspective is somewhat more involved though. The failure rate of the DFT is given by the maximum of all failure rates of the operational leaves. We add a transition to each state in the CTMC fragment. Its rate μ is chosen such that the expected time of an exponential distribution with rate μ equals the expected time of the maximum over the failure distributions of the operational leaves. The resulting CTMC can be viewed as an *optimistic abstraction* of the DFT.

Refinement. So far, so good. Assume now the DFT's MTTF is required to exceed some threshold M, say. If the lower bound lb is at most M, the DFT satisfies the requirement; if the upper bound ub is below M, it refutes. In all other cases, the result is inconclusive. In that case, a heuristic can be employed to refine the two abstractions. This can be done such that earlier analysis results can be partially re-used. The MTTF analysis by means of probabilistic model

[7] It can be automatically checked whether a DFT satisfies this assumption by encoding it in difference logic, a fragment of linear integer arithmetic, and check this encoding using SMT solvers; for further details see [53].



(a) run time (seconds)



(b) run time (seconds)

Fig. 7: Abstract-reduce-refine versus the monolitic approach on four different benchmark sets (taken from [53]).

checking (see section 7), provides bounds on the MTTF for each state. This can be exploited in a simple heuristic: states that are reachable with a high probability and whose gap between lower and upper bound is wide, are explored first.

Abstract-reduce-refine. Altogether, this results in an iterative abstraction-refinement approach. It stops whenever it can be decided whether the MTTF is beyond or below M. Or, if one is nonetheless interested in more precise information about the MTTF, one can also terminate the abstraction-refinement process whenever the gap between lower and upper bound is sufficiently tight. While obtaining the partial state spaces, symmetry and partial-order reduction, as well as don't care propagation (see Section 4) can be exploited.

Experiments. This approach works very well for larger models: some DFTs which result in an out-of-memory for the monolithic approach of Section 4 are now solved within minutes. Results are provided in Fig. 7a where a precision of 10% is used. That is to say, the abstract-reduce-refine algorithm terminates with lower bound lb and upper bound ub if $ub - lb < 0.1 \cdot \frac{ub+lb}{2}$. The approximation comes at some overhead. It requires some internal bookkeeping for the state-space generation and constructing the upper bound is costly. Whenever the upper bound is too pessimistic, an almost complete state space is required leading to a decreased performance. Lower bounds are easier to estimate and are not really influenced by low probability paths. The on-the-fly algorithm updates the approximation after each iteration, and the lower bounds quickly becomes accurate. Let x be the true MTTF: For Fig. 7b the runtime until the (unchanged) procedure certified that the MTTF was at least $0.95 \cdot x$ is given. This is *always* very fast. Thus, 90% to 99% of the computation time is spent making the upper bound tighter.

7 Probabilistic Model Checking

The quantitative analysis of the resulting DFT's CTMC can be done using probabilistic model checking [34, 37]. This is *not* the branch of computer-aided verification that exploits randomized algorithms for verification but rather the area that focuses on the model checking of probabilistic models such as Markov chains and variations thereof. This field is not new. Soon after the birth of model checking in 1981, the first papers on probabilistic model checking (though not called that way) appeared. Whereas initial works focused on almost-sure events—does a phenomenon happen with probability one?—, later quantitative queries could be handled by combining model-checking algorithms with algorithms from numerical mathematics and operations research. Powerful tools such as `Prism` [38], `MRMC` [35] and `storm` [15] together with the development of various efficient verification algorithms have led to an enormous impulse to the field. It is fair to say that probabilistic model checking extends and complements long-standing analysis techniques for Markov processes.

Model checking DFTs. Probabilistic model checking can be directly applied to the CTMCs underlying DFTs. This does not require any additional means. It can be used as a black box. Measures-of-interest such as MTTF and reliability can be readily cast as formulas in stochastic temporal logics such as some form of probabilistic CTL. Alternatively, automata can be used. In fact, this is not quite right. Logics allow for specifying constraints on such measures. Examples are e.g., the MTTF is at least M, or the probability that the TLE fails is below 10^{-9}. Verifying these logical formulas is typically very fast and requires a negligible amount of time compared to the state-space generation for DFTs. The aforementioned tools enable the automated verification of models with several millions of states within a couple of minutes.

The benefits of probabilistic model checking DFTs. Is that all? Not quite. Given the rich plethora of functional correctness properties that can be described in temporal logics, the functional correctness of DFTs can be checked as well. Properties such as: can it ever happen that gates A and B both fail? or: if the leaves fail in a certain order, does that cause a TLE failure? can be automatically checked using model checking too. No dedicated algorithms are needed for that. Using the same machinery for validating the measures of interest, many functional properties can be checked.

The use of logics and automata for specifying DFT's properties offers, in addition, a high degree of expressiveness and flexibility. Most standard measures such as MTTF, reliability, and availability are readily covered. Nesting formulas yields a simple mechanism to specify complex measures in a succinct manner. A complex property like " the probability that once a certain set of gates have failed soon with high probability (say, within 10 time units with at least probability 0.99), the TLE will fail within 1,000 time units when in addition gates A and B have failed is low" can be captured by a succinct formula. The main benefit though is the use of model checking as a fully algorithmic approach toward measure evaluation. Even better, it provides a single computational technique for any possible measure that can be written. This applies from simple properties to complicated, nested, and possibly hard-to-grasp formulas. This is radically different from common practice in DFT evaluation where tailored and new algorithms are developed for "new" measures.

Measure-specific computation. All algorithmic details, all detailed and non-trivial numerical computation steps are hidden to the user. Without any expert knowledge on, say, numerical analysis techniques for CTMCs, measure evaluation is possible. Even better: the algorithmic analysis is measure-driven. That is to say, the stochastic process can be tailored to the measure of interest prior to any computation, avoiding the consideration of parts of the state space that are irrelevant for the property of interest. In this way, computations must be carried out only on the fragments of the state space that are relevant to the property of interest.

Nondeterminism. Finally, probabilistic model checking is applicable to models with nondeterminism. This is relevant for DFTs too, as some DFTs may give rise to nondeterminism, see Fig. 3 (left). In these models the future behaviour is not always determined by a unique probability distribution, but by selecting one from a set of them. Rather than providing exact probabilities, the measures are subject to the resolution of the nondeterminism. As a result, bounds on the measures are obtained: lower bounds typically correspond to the "worst" possible resolution of nondeterminism, whilst upper bounds correspond to the most favourable resolution of the nondeterminism. For DFTs, the recent advances in model checking of Markov automata [17], a nondeterministic extension of CTMCs is of relevance. Efficient algorithms have been developed for objectives such as expected reward (and time, a.k.a.: MTTF), long-run rewards, timed reachability, and combinations of such objectives; for algorithmic details we refer to [23].

8 Statistical Model Checking

Statistical model checking [39] relies on Monte Carlo simulation, and can be seen as a modern form of discrete event simulation. Rather than exploring the whole state space and numerically computing the probability on a certain event, statistical model checking takes (a large number of) random samples from a statistical model and estimates the metric of interest.

Advantages. Statistical model checking has two important advantages over numerical model checking. First, it can handle very large state spaces, enabling the analysis of DFTs with many and/or complex elements, which cannot be tackled with numerical methods due to the size of the underlying state space. The memory footprint of statistical techniques is extremely low, and this method can trivially be parallelized on multi-core computer clusters.

Second, statistical methods can handle (almost) any probability distribution. Numerical computations of models with non-exponential probability distributions is difficult, especially when various types of distributions are combined. One can however, approximate arbitrary distributions with combinations of exponentials, using (acyclic) phase type distributions, but this comes at the cost of a larger state space. This is particularly true for the frequently occurring deterministic distributions and Gaussian distributions. Statistical methods do not suffer from this problem of combining different probability distributions. Therefore, they have been fruitfully applied in a number of case studies. These include the evaluation of complex maintenance strategies and their effect on system reliability [45, 46, 44]. Here, the failure rates are Erlang-distributed, whereas repair times and inspection frequencies are deterministic.

Drawbacks and remediations. Statistical methods have also their drawbacks. First, they yield a stochastic estimate upto a certain confidence level, rather than an exact value. It is, however, a subject of debate whether this is a true disadvantage, since the failure rates and other numerical values appearing in DFTs are often estimates themselves, obtained via measurements or expert opinions. Second, statistical methods have a hard time supporting nondeterminism, however, recent progress has been made in [14].

Finally, statistical methods require many samples for rare events, i.e., events whose probability is low, which is typically the case for safety-critical systems. For example, if the probability for a failure to happen is $\frac{1}{1000}$, we need 1,000 samples on average to see the event once, and for statistically significant results, even more samples are needed, e.g., 10,000. To remedy this problem, rare event simulation techniques have been invented [33]. These techniques increase the probability for the rare events to happen, and then compensate the end result for it. Two major classes of rare event simulation exist: importance sampling [27] and importance splitting [41], and both have also been applied to DFTs, respectively in [47] and [10].

9 Industrial Applications

Railway engineering. We have conducted a series of case studies [24, 44, 46, 48] in close collaboration with stakeholders from railroad engineering, namely asset manager ProRail, rolling stock maintenance company NS/NedTrain, and consultancy firm Movares. All these case studies focussed on maintenance and studied the effect of different maintenance policies in terms of their performance benefits (i.e., increased availability or reliability) and costs (broken down into cost for planned and unplanned downtime, and corrective and preventive maintenance).

More specifically, the maintenance strategies were modeled in the leaves of the fault trees, leading to fault maintenance trees [45]. Both probabilistic and statistical model checking were deployed.

The paper [24] analyzed a railway safety system of a railroad trajectory a major crossing-points in the Netherlands. The goal of the analysis is to verify that the rail trajectory fulfils the railway system specifications. Here, the focus lies on the availability of the systems on the rail trajectory, defined by three failure categories: Severe disruption in both directions, such that no train can ride; severe disruption in one direction, such that no train can ride; and minor disruption which leads to dispunctuality. These yield fault trees containing 25 to 350 BEs.

The paper considers two different repair strategies: a dedicated repair procedure for each component, i.e., each component can be repaired at any time. This is the strategy Movares has considered for their analysis. A second strategy considers one repair per group of components, which is more realistic in practice.

The paper [44] studies the effect of different maintenance strategies on a pneumatic compressor, which produces compressed air used to operate, among other things, the doors and brakes of trains. This compressor is critical to the operation of the train, and a failure can lead to a lengthy and expensive disruption. Within the rolling stock maintenance company NedTrain, [44] modelled this compressor as a fault maintenance tree (FMT), i.e., a fault tree augmented with maintenance aspects. We have shown how this FMT naturally models complex maintenance plans including condition-based maintenance with regular inspections. The analysis demonstrates that FMTs can be used to model the compressor, a practical system used in industry, including its maintenance policy. We validate this model against experiences in the field, compute the importance of performing minor services at a reasonable frequency, and find that the currently scheduled overhaul may not always be cost-effective.

The paper [46] studies the effect of different maintenance strategies on the electrically insulated railway joint (EI-joint), a critical asset in railroad tracks for train detection, and a relative frequent cause for train disruptions. Together with experts in maintenance engineering, [46] modeled the EI-joint as a fault maintenance tree (FMT). Again, complex maintenance concepts, such as condition-based maintenance with periodic inspections, were naturally modeled by FMTs, and several key performance indicators, such as the system reliability, number of failures, and costs, can easily be analysed.

The analysis shows that the current maintenance policy is close to cost-optimal. It is possible to increase joint reliability, e.g., by performing more inspections, but the additional maintenance costs outweigh the reduced cost of failures.

The faithfulness of quantitative analyses heavily depends on the accuracy of the parameter values in the models. Here, we have been in the unique situation that extensive data could be collected, both from incident registration databases, as well as from interviews with domain experts from several companies. This made that we could construct a model that faithfully predicts the expected number of failures at system level.

Automotive industry. For the car manufacturer BMW, we have carried out a large case study on the design-phase safety analysis of vehicle guidance systems [21]. Its aim is to model a variety of safety concepts and E/E architectures for drive automation. Several DFTs have been automatically generated from system descriptions and combined (in an automated manner) with hardware failure models for several mappings of functions on hardware. The DFT state-space generation has been done according to the monolithic approach using abstraction-refinement to obtain bounds. The DFT analysis focused on investigating the effect of different hardware partitionings on a range of metrics. These metrics include e.g., the mean time from degradation to failure and the minimal degraded reliability. DFTs with more than 300 nodes resulting in a CTMC of about 4 million states and 66 million transitions have been generated and successfully analysed in a matter of minutes.

Aerospace industry. This paper focused on exploiting formal methods in state-space generation and DFT analysis. Formal methods can however also help to *synthesise* fault trees from system description languages such as AADL or SysML, see the recent survey [32]. The key idea here is to exploit the structure of the system architecture so as to generate a fault tree in a fully automated manner. In a case study with ESA [18], this technique has been successfully applied to obtain a (static) fault tree of 66 nodes explaining the behaviour of a severe failure in a complex satellite. The interesting aspect here is that the satellite design team developed this FT manually, whereas using the `compass` tool-set [8] that supports AADL, it could be generated in a fully automated manner within two hours. The FT generation algorithm is described in detail in [9].

10 Epilogue

Summary. This paper concentrated on the *analysis* of (dynamic) fault trees. This includes the generation of stochastic state-based models from DFTs as well as their quantitative analysis. We argued why formal methods can substantially boost this. In a nutshell, the main benefits are: (1) probabilistic model checking is mostly faster than competitive DFT analyses especially when several dynamic gates are involved; (2) it enables the treatment of a larger class of DFT, namely also those giving rise to nondeterminism; (3) it supports a large set of measures of interest that go beyond the classical DFT measures; (4) compositionality, abstraction, and reduction techniques improve the scalability of DFT analysis; and (5) flexibility: attack trees can be treated in a similar way, extensions with maintenance aspects, and other DFT elements are possible.

Future work. This paper concentrated on state-space generation for DFTs and the analysis of the resulting stochastic (decision) processes. Open research challenges are to improve the process of obtaining DFTs for systems at hand. There are effective ways to obtain fault trees from architecture description languages such as AADL and SysML in an automated manner. Formal methods play an important role here too as recently surveyed in [32]. The current approaches do however not support the full expressiveness of DFTs but rather concentrate on a subclass of DFTs. More importantly though is how to obtain trustworthy information about the system at hand, such as failure rates, repair strategies and so on. We believe that big data analysis can be exploited to help out. An alternative direction is to consider parametric DFTs in which rates or even the redundancy of components is left open. The key issue is then to synthesize parameter values for which the resulting DFT ensures to satisfy a given reliability.

Acknowledgement. We thank the anonymous reviewers for their valuable feedback. A big thanks goes to our co-workers on fault trees in academia: Hichem Boudali, Pepijn Crouzen, Dennis Guck, Sebastian Junges, Viet Yen Nguyen, Bart Postma, Enno Ruijters, and Matthias Volk, and to our industrial partners: Peter Drolenga (NS/NedTrain), Jaap van Ekris (Delta Pi), Bob Huisman (NS), Madji Ghadhab (BMW), Gea Kolk (Movares), Matthias Kuntz (BMW), Martijn van Noort (ProRail), Margot Peters (NS/NedTrain), Wietske Postma (Nuclear Research Group), Judi Romijn (Movares), and Yuri Yushstein (ESA).

We thank Ed Brinksma for his guidance and inspiration over the many years. This survey paper is a birthday salute to him. His belief in formal methods, especially the elegance of compositionality and his strong view on narrowing the gap between formal methods and industrial practice have influenced our work to an enormous extent. About 25 years ago, Ed was one of the creative minds to aim at developing a framework for the *integrated* modelling and analysis of functional and performance aspects of reactive systems. This survey gives a short account about what one can achieve along these lines in a by tradition completely different research field—reliability analysis. Last but not least, we thank Ed for his eloquence, his view on culture, art, books, and good food. And, as a Rector Magnificus of the University of Twente, his role in establishing a branch of Starbucks on campus, almost next to our offices.

References

1. Marwan Ammar, Ghaith Bany Hamad, Otmane Aït Mohamed, and Yvon Savaria. Efficient probabilistic fault tree analysis of safety critical systems via probabilistic model checking. In *Proc. of FDL*. IEEE, 2016.
2. Florian Arnold, Axel Belinfante, Freark van der Berg, Dennis Guck, and Mariëlle Stoelinga. DFTCalc: A tool for efficient fault tree analysis. In *Proc. of SAFECOMP*, volume 8153 of *LNCS*, pages 293–301. Springer, 2013.
3. Florian Arnold, Dennis Guck, Rajesh Kumar, and Mariëlle Stoelinga. Sequential and parallel attack tree modelling. In *Proc. of SAFECOMP*, volume 9338 of *LNCS*, pages 291–299, 2015.
4. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.
5. Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. A compositional semantics for dynamic fault trees in terms of interactive Markov chains. In *Proc. of ATVA*, volume 4762 of *LNCS*, pages 441–456. Springer, 2007.
6. Hichem Boudali, Pepijn Crouzen, and Mariëlle I. A. Stoelinga. Dynamic fault tree analysis using input/output interactive Markov chains. In *Proc. of DSN*, pages 708–717, 2007.

7. Hichem Boudali, Pepijn Crouzen, and Mariëlle I. A. Stoelinga. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. *IEEE Trans on Dependable Secure Comput*, 7(2):128–143, 2010.
8. 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.
9. Marco Bozzano, Alessandro Cimatti, and Francesco Tapparo. Symbolic fault tree analysis for reactive systems. In *Proc. of ATVA*, volume 4762 of *LNCS*, pages 162–176. Springer, 2007.
10. Carlos E. Budde, Pedro R. D'Argenio, and Holger Hermanns. Rare event simulation with fully automated importance splitting. In *Proc. of EPEW*, volume 9272 of *LNCS*, pages 275–290. Springer, 2015.
11. Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. of LICS*, pages 428–439. IEEE Computer Society, 1990.
12. Edmund M. Clarke, E. Allen Emerson, Somesh Jha, and A. Prasad Sistla. Symmetry reductions in model checking. In *Proc. of CAV*, volume 6605 of *LNCS*, pages 147–158. Springer, 1998.
13. Pepijn Crouzen and Frédéric Lang. Smart reduction. In *Proc. of FASE*, volume 6603 of *LNCS*, pages 111–126. Springer, March 2011.
14. Pedro R. D'Argenio, Arnd Hartmanns, Axel Legay, and Sean Sedwards. Statistical approximation of optimal schedulers for probabilistic timed automata. In *Proc. of IFM*, volume 9681 of *LNCS*, pages 99–114. Springer, 2016.
15. Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *Proc. of CAV*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.
16. Joanne Bechta Dugan, Salvatore J. Bavuso, and Mark A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Trans Rel*, 41(3):363–377, 1992.
17. Christian Eisentraut, Holger Hermanns, and Lijun Zhang. On probabilistic automata in continuous time. In *Proc. of LICS*, pages 342–351. IEEE CS, 2010.
18. Marie-Aude Esteve, Joost-Pieter Katoen, Viet Yen Nguyen, Bart Postma, and Yuri Yushtein. Formal correctness, safety, dependability, and performance analysis of a satellite. In *Proc. of ICSE*, pages 1022–1031. IEEE Computer Society, 2012.
19. Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *Int'l J on Software Tools for Technology Transfer*, 15(2):89–107, 2013.
20. Daochuan Ge, Meng Lin, Yanhua Yang, Ruoxing Zhang, and Qiang Chou. Quantitative analysis of dynamic fault trees using improved sequential binary decision diagrams. *Reliab Eng Syst Safe*, 142:289 – 299, 2015.
21. Majdi Ghadhab, Sebastian Junges, Joost-Pieter Katoen, Matthias Kuntz, and Matthias Volk. Model-based safety analysis for vehicle guidance systems. In *Proc. of SAFECOMP*, volume 10488 of *LNCS*. Springer, 2017. (To appear).
22. Amir Hossein Ghamarian, Maarten de Mol, Arend Rensink, Eduardo Zambon, and Maria Zimakova. Modelling and analysis using GROOVE. *STTT*, 14(1):15–40, 2012.
23. Dennis Guck, Hassan Hatefi, Holger Hermanns, Joost-Pieter Katoen, and Mark Timmer. Analysis of timed and long-run objectives for Markov automata. *LMCS*, 10(3), 2014.

24. Dennis Guck, Joost-Pieter Katoen, Mariëlle I. A. Stoelinga, Ted Luiten, and Judi Romijn. Smart railroad maintenance engineering with stochastic model checking. In *Proc. of RAILWAYS*, volume 104 of *Civil-Comp Proceedings*, pages 299–314. Civil-Comp Press, 2014.
25. Dennis Guck, Jip Spel, and Mariëlle I. A. Stoelinga. DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper). In *Proc. of ICFEM*, volume 9407 of *LNCS*, pages 304–311, 2015.
26. Rohit Gulati and Joanne Bechta Dugan. A modular approach for analyzing static and dynamic fault trees. In *Proc. of RAMS*, pages 57–63, 1997.
27. Philip Heidelberger. Fast simulation of rare events in queueing and reliability models. *ACM Trans. Modeling and Computer Simulation*, 5(1):43–85, 1995.
28. Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer, 2002.
29. Holger Hermanns and Joost-Pieter Katoen. The how and why of interactive markov chains. In *Proc. of FMCO*, volume 6286 of *LNCS*, pages 311–337. Springer, 2009.
30. Sebastian Junges, Dennis Guck, Joost-Pieter Katoen, Arend Rensink, and Mariëlle Stoelinga. Fault trees on a diet: automated reduction by graph rewriting. *Formal Asp. Comput.*, 29(4):651–703, 2017.
31. Sebastian Junges, Dennis Guck, Joost-Pieter Katoen, and Mariëlle I. A. Stoelinga. Uncovering dynamic fault trees. In *Proc. of DSN*, pages 299–310. IEEE CS, 2016.
32. Sohag Kabir. An overview of fault tree analysis and its application in model based dependability analysis. *Expert Syst. Appl.*, 77:114–135, 2017.
33. Herman Kahn and T. E. Harris. Estimation of particle transmission by random sampling. In *Monte Carlo method; Proc. Symp. held June 29, 30, and July 1, 1949*, volume 12 of *Nat. Bur. Standards Appl. Math. Series*, pages 27–30, 1951.
34. Joost-Pieter Katoen. The probabilistic model checking landscape. In *Proc. of LICS*, pages 31–45. ACM, 2016.
35. Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.*, 68(2):90–104, 2011.
36. Rajesh Kumar and Mariëlle Stoelinga. Quantitative security and safety analysis with attack-fault trees. In *Proc. of HASE*, pages 25–32. IEEE, 2017.
37. Marta Z. Kwiatkowska. Model checking for probability and time: from theory to practice. In *Proc. of LICS*, pages 351–360. IEEE Computer Society, 2003.
38. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. of CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
39. Kim G. Larsen and Axel Legay. On the power of statistical model checking. In *Proc. of ISoLA (2)*, volume 9953 of *LNCS*, pages 843–862, 2016.
40. Stefania Montani, Luigi Portinale, Andrea Bobbio, and Daniele Codetta-Raiteri. Automatically translating dynamic fault trees into dynamic Bayesian networks by means of a software tool. In *Proc. of ARES*, pages 804–809, 2006.
41. Jérôme Morio, Rudy Pastel, and François Le Gland. An overview of importance splitting for rare event simulation. *Eur. J. of Physics*, 31(5):1295–1303, August 2010.
42. Elon Musk, 2015. https://twitter.com/elonmusk/status/615185689999765504.
43. K. Durga Rao, V. Gopika, V.V.S. Sanyasi Rao, H.S. Kushwaha, A.K. Verma, and A. Srividya. Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment. *Reliab Eng Syst Safe*, 94(4):872 – 883, 2009.

44. 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 *Proc. of QEST*, volume 9826 of *LNCS*, pages 331–347. Springer, 2016.
45. Enno Ruijters, Dennis Guck, Peter Drolenga, and Mariëlle Stoelinga. Fault maintenance trees: reliability centered maintenance via statistical model checking. In *Proc. of RAMS*. IEEE, 2016.
46. 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 *Proc. of DSN*, pages 662–669. IEEE, 2016.
47. Enno Ruijters, Daniël Reijsbergen, Pieter-Tjerk de Boer, and Mariëlle Stoelinga. Rare event simulation for dynamic fault trees. In *Proc. of SAFECOMP*, volume 10488 of *LNCS*, 2017. (Accepted for publication).
48. Enno Ruijters and Mariëlle Stoelinga. Better railway engineering through statistical model checking. In *Proc. of ISoLA*, volume 9952 of *LNCS*, pages 151–165, 2016.
49. Enno Ruijters and Mariëlle I. A. 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.
50. Kevin J. Sullivan, Joanne Bechta Dugan, and David Coppit. The Galileo fault tree analysis tool. In *Proc. of FTCS*, pages 232–235, 1999.
51. Mark Timmer, Joost-Pieter Katoen, Jaco van de Pol, and Mariëlle Stoelinga. Confluence reduction for Markov automata. In *Theor Comput Sci*, volume 655, pages 193–219, 2016.
52. Matthias Volk, Sebastian Junges, and Joost-Pieter Katoen. Advancing dynamic fault tree analysis. In *Proc. of SAFECOMP*, volume 9922 of *LNCS*, pages 253–265. Springer, 2016.
53. Matthias Volk, Sebastian Junges, and Joost-Pieter Katoen. Fast dynamic fault tree analysis by model checking techniques. *IEEE Trans on Industrial informatics*, 2017. (To appear) DOI: 10.1109/TII.2017.2710316.
54. T. Yuge and S. Yanagi. Quantitative analysis of a fault tree with priority AND gates. *Reliab Eng Syst Safe*, 93(11):1577–83, 2008.