Rare Event Simulation for Dynamic Fault Trees[☆]

Enno Ruijters[a,*], Daniël Reijsbergen[b], Pieter-Tjerk de Boer[c], Mariëlle Stoelinga[a]

[a]*Formal Methods and Tools, University of Twente, Zilverling, P.O. Box 217, 7500 AE Enschede, The Netherlands*
[b]*Singapore University of Technology and Design, Singapore*
[c]*Design and Analysis of Communication Systems, University of Twente, Zilverling, P.O. Box 217, 7500 AE Enschede, The Netherlands*

Abstract

Fault trees (FT) are a popular industrial method for reliability engineering, for which Monte Carlo simulation is an important technique to estimate common dependability metrics, such as the system reliability and availability. A severe drawback of Monte Carlo simulation is that the number of simulations required to obtain accurate estimations grows extremely large in the presence of *rare events*, i.e., events whose probability of occurrence is very low, which typically holds for failures in highly reliable systems.

This paper presents a novel method for rare event simulation of dynamic fault trees with complex repairs that requires only a modest number of simulations, while retaining statistically justified confidence intervals. Our method exploits the importance sampling technique for rare event simulation, together with a compositional state space generation method for dynamic fault trees.

We demonstrate our approach using three parameterized sets of case studies, showing that our method can handle fault trees that could not be evaluated with either existing analytical techniques using stochastic model checking, nor with standard simulation techniques.

1. Introduction

The rapid emergence of robots, drones, the Internet-of-Things, self-driving cars and other inventions, increase our already heavy dependence on computer-based systems even further. Reliability engineering is an important field that provides methods, tools and techniques to identify, evaluate and mitigate the risks related to complex systems. Moreover, asset management is currently shifting towards reliability-centered, a.k.a. risk-based, maintenance. This shift also requires a good understanding of the risk involved in the system, and of the effects of maintenance on the reliability. Fault tree analysis (FTA) is one of the most important techniques in that field, and is commonly deployed in industry ranging from railway and aerospace system engineering to nuclear power plants.

A fault tree (FT) is a graphical model that describes how failures propagate through the system, and how component failures lead to system failures. An FT is a tree (or rather, a directed acyclic graph) whose leaves model component failures, and whose gates model how failures propagate through the system, and lead to system failures. Standard (or: static) FTs (SFTs) contain a few basic gates, like AND and OR, making them easy to use and analyze, but also limited in expressivity. To cater for more complex dependability patterns, like spare management and causal dependencies, a number of extensions to FTs have been proposed.

One of the most widely used extensions is the dynamic fault tree (DFT) [1], providing support for common patterns in system design and analysis. More recently, maintenance has been integrated into DFTs supporting complex policies of inspections and repairs [2]. Both of these developments have increased the memory and time needed for analysis, to the point where many practical systems cannot be analyzed on current computers in a reasonable time.

One approach to combat the complexity of analysis is to switch from analytic techniques to simulation. By not constructing the entire state space of the system, but only computing states as they are visited, memory requirements are minimal and computation time can be greatly reduced. This approach can be successfully applied to industrial systems [3], but presents a challenge when dealing with highly reliable systems: If failures are very rare, many simulations are required before observing any at all, let alone observing enough to compute statistically justified error bounds.

This problem in simulating systems with rare events can be overcome through rare event simulation techniques, first developed in the 1950's [4]. By adjusting the probabilities to make failures less rare, and subsequently calculating a correction for this adjustment, statistically justified results can be obtained from far fewer simulations than would

[☆]This article is the extended version of a paper from SafeComp 2017 with the same title.

[*]Principal corresponding author

Email addresses: e.j.j.ruijters@utwente.nl (Enno Ruijters), daniel_reijsbergen@sutd.edu.sg (Daniël Reijsbergen), p.t.deboer@utwente.nl (Pieter-Tjerk de Boer), m.i.a.stoelinga@utwente.nl (Mariëlle Stoelinga) otherwise be needed.

We present a novel approach to analyze DFTs with maintenance through importance sampling. We adapt the recently-developed Path-ZVA algorithm [5] to the setting of DFTs. We retain the existing compositional semantics by Boudali et al. [6] already used in current tools [7]. Using three case studies, we show that our approach can simulate DFTs too large for other tools with events too rare for traditional simulation techniques. Thus, our approach has clear benefits over existing numerical tools, and tools without rare event simulation: We can analyze larger DFTs, producing results quicker and obtain narrow confidence intervals.

Our approach. Our overall approach to rare event simulation for DFTs relies on an on-the-fly conversion of the DFT into a state-space model describing the stochastic behaviour of the DFT. Given this model, we apply the Path-ZVA algorithm for importance sampling to alter the behaviour such that system failures become more probable. We then sample simulation traces from this model, measuring the unavailability of the DFT, and apply a correction for the adjusted probabilities.

More concretely, we take the following steps:

1. Use the DFTCalc tool to compute state-space models for all elements of the DFT. Traditionally, one would compute the composition of these elements to obtain one model describing the behaviour of the DFT. Our approach computes the necessary states of the composition on-the-fly in the following steps.
2. Apply the Path-ZVA algorithm (explained in Sect. 3) to adjust the transition probabilities to preferentially direct the simulations along the most likely paths to failures.
3. Sample traces of the adjusted model, storing how much time of each trace was spent in unavailable (i.e., failed) states, and how much the total probability of each trace was altered by step 3.
4. Average the unavailabilities of the traces, correcting for the altered probability of each trace.

Related Work. Apart from DFTs and repairs, many more extensions have been developed. For an overview we refer the reader to [8]. Most current FTA formalisms support repairs using per-component repair times [9]. More complicated policies can be specified using repair boxes [10] or the Repairable Fault Tree extension [11], however both of these require exponentially distributed failure times of components where our approach allows Erlang distributions.

A wide range of analysis techniques exist as well, again summarized in [8]. Standard simulation methods date back to 1970 [12], continuing to be developed until the present day [3]. Rare event simulation has been used to estimate system reliability since 1980 [13] and is still applied today [14], although, to our surprise, we are not aware of any approach applying rare event simulation specifically to fault trees. An overview of importance sampling techniques in general can be found in [15].

Aside from accelerating the simulation process as described in this paper, various other methods of speeding up (dynamic) fault tree analysis have been proposed. For example, by analyzing static parts of the tree separate from dynamic parts [16, 17]. Such techniques may be applicable in combination with our proposed approach, by using fast, standard methods for the static parts and rare event simulation for the dynamic parts. Similarly, our approach could be adapted to other extensions of (repairable) fault trees that are analyzed using Monte Carlo simulation, such as *state/event fault trees* [18].

Background. The original analysis method for DFTs was a conversion to continuous-time Markov chains (CTMCs) [1]. A CTMC is a state-space model of a sequence of events, where the future evolution of the model depends only on the current state. Formally, a CTMC C is a tuple $C = \langle S, P, E \rangle$, where:

- S is a set of states numbered $s_0, s_1, \ldots, s_n$, of which s_0 is the initial state,
- P is a matrix of transition probabilities, where p_{ij} denotes the probability of transitioning from state s_i to s_j in one step (note: $\forall_i \sum_j p_{ij} = 1$ and $\forall_i p_{ii} = 0$), and
- E is a vector of exit rates, such that the time spent after entering state s_i but before transitioning out of the state is governed by an exponential distribution with rate E_i; i.e., if we denote by T the time until the next transition out of state s_i, then $P(T \leq t) = 1 - e^{-E_i t}$.

To simplify our illustrations, we often use $\lambda_{ij} = p_{ij} E_i$ to denote the transition rates. In this notation, each transition has its own transition time following an exponential distribution with parameter λ_{ij}, and whichever transition out of the current state s_i occurs first is actually taken.

We sometimes abstract away the timed behaviour of a CTMC, in which case we use the embedded discrete-time Markov chain (DTMC) $D = \langle S, P \rangle$ and ignore the times at which transitions are taken.

A compositional analysis methods for DFTs was developed in [19] in terms of Input/Output Interactive Markov Chains (I/O-IMCs). I/O-IMCs are an extension of Interactive Markov Chains [20] that allow composition by means of input and output actions. An I/O-IMC is defined as a tuple $I = \langle S, Act, \rightarrow, \rightsquigarrow \rangle$, where:

- S is a set of states numbered $s_0, s_1, \ldots, s_n$, of which s_0 is the initial state,
- Act is a finite set of *actions* (also called *signals*), partitioned $Act = Act^I \cup Act^O \cup Act^{int}$ where Act^I are *input* actions, Act^O are *output* actions, and Act^{int} are *internal* actions (all disjoint),

- $\rightarrow \subseteq S \times Act \times S$ is a set of interactive transitions, and
- $\rightsquigarrow \subseteq S \times \mathbb{R}_{>0} \times S$ is a set of Markovian transitions.

A Markovian transition (s_i, λ_{ij}, s_j) of an I/O-IMC behaves like a transition of a CTMC with rate λ_{ij}, while the interactive transitions allow multiple I/O-IMCs to be composed into one larger I/O-IMC, as will be discussed in Sect. 4.1.

As can be seen from the definitions above, all transition times in a CTMC or I/O-IMC follow exponential distributions. Transition times that follow a different probability distribution (e.g., Weibull or truncated normal distributions) can be approximated using a phase-type distribution [21] — this allows us to remain within the CTMC framework at the cost of increasing the state space size. In this paper we often use times governed by Erlang distributions, which are a subset of phase-type distributions, to approximate more general probability distributions. Erlang distributions can be defined as the cumulative time of a sequence of exponential distributions with identical rates: If we let $X_1, X_2, \ldots, X_k$ be independent random variables drawn from an exponential distribution with rate λ, then $X_1 + X_2 + \ldots + X_k$ is governed by a (k, λ)-Erlang distribution. Thus, an Erlang distribution can be easily encoded in a CTMC as a chain of transitions with exponential transition times. We generally use Erlang-distributions to approximate more general probability distributions, using the two parameters (k and λ) to obtain the same mean and variance as in the distribution being fitted. More precise approximations can be obtained using more complex combinations of exponential distributions [21], although we do not use these in this paper.

Some results of the case studies presented in this paper are presented as 95% *confidence intervals*: Intervals computed following a statistical sample such that one can expect that 95% of the intervals constructed in this way will contain the true value being estimated. We compute such intervals using the Central Limit Theorem [22].

Contributions with respect to earlier version. The following contributions of this paper are new since the publication of [23]:

- Sect. 3 and 4 have been expanded to explain our approach in greater detail.
- We present a new algorithm for converting the I/O-IMC of the DFT into a CTMC, which allows the analysis of a greater class of DFTs while guaranteeing that the results are not affected by nondeterminism.

Organization of the Paper. This paper first explains fault trees, DFTs, and repairable DFTs in Sect. 2. Sect. 3 describes rare event simulation, and the Path-ZVA algorithm used in our approach. Next, our adaptation of rare event simulation to DFTs is explained in Sect. 4. Our case studies with their results are shown in Sect. 5, before concluding in Sect. 6.

2. Fault Tree Analysis

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 [24]. 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* (i.e., probability that the system remains functional for the duration of its mission) and *availability* (i.e., the average fraction of the time that the system in functional).

An FT is a directed acyclic graph where the leaves describe failure modes, called *basic events* (BEs), at a component level. *Gates* specify how the failures of their children combine to cause failures of (sub)systems. The root of the FT, called the *top-level event* (TLE), denotes the failure of interest.

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.

Example 1. *Figure 1 shows an example of such a fault tree. It models a case study from [2], 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 are the BEs describing the failures of individual relay and high voltage cabinets.*

2.1. Dynamic Fault Trees

Over the years, many extensions to FTs have been developed [8]. One of the most prominent extensions is the *dynamic fault trees* (DFT) model [1]. DFTs introduce several new gates to cater for common patterns in dependability models:

- The priority-AND (PAND) models order-dependent effects of failures. It fails if and only if its left child fails and then its right child. This is used e.g. to model the difference between a fire detector failing before or after a fire starts.
- The SPARE gate, modeling a primary component with one or more spare elements. The spare elements can have different failure rates when they are in use,

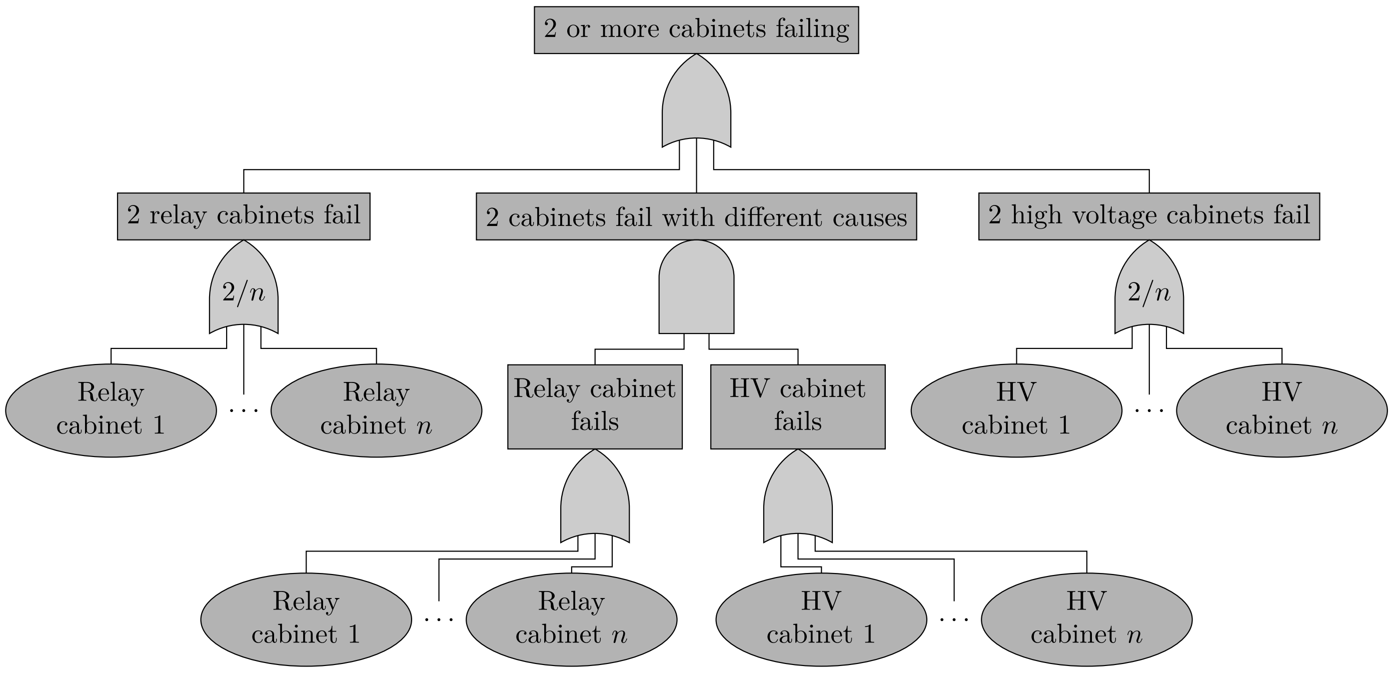


Figure 1: Example fault tree of the relay cabinet case study. Due to redundancy, the system can survive the failure of any single cabinet, however two failures cause system unavailability. The number of cabinets varies, and is indicated by n.

and can be shared between multiple gates. Shared spare elements can only be used by one gate at any time.

- The functional dependency (FDEP) gate which causes all of its children to fail when its trigger fails. It is used e.g. to model common cause failures, such as a power failure disabling many components.

It should be noted that several different semantics for DFTs have been developed over the years, some of which contradict each other in some cases [25]. In this paper, we use the semantics of the DFTCalc tool [2]. In particular, we do not support the sequence-enforcing gate provided in the original definition.

2.2. Repairable Fault Trees

Many practical systems are not just built and then left on their own, instead repairs and maintenance are often performed to keep a system functioning correctly and correct failures when they occur. This maintenance is crucial to the dependability of the system, as it can prevent or delay failures. It is therefore important to consider the maintenance policy when performing reliability analysis.

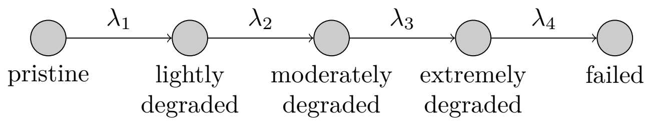


Figure 2: CTMC describing a basic event with multiple degradation phases.

Standard fault trees support only simple policies of independent repairs with exponentially distributed repair times starting immediately upon component failure [9]. Various extensions provide more complex policies, describing that some repairs occur in sequential order rather than in parallel [10], or complex maintenance policies with preventive inspections and repairs [3, 26].

Dynamic fault trees support both the simple model with independent, exponentially distributed repair times, and have been extended with complex policies with periodic inspections and/or repairs [2]. We use this extension in this paper. A key component of the implementation of more advanced 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 such DFTs can progress through multiple phases of degradation, as depicted in Figure 2. 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 degraded too much. Periodic replacements simply return their BEs to their undegraded phase periodically.

3. 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 [27]. 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 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 $6.8 \cdot 10^{-9}$ [28]. 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.

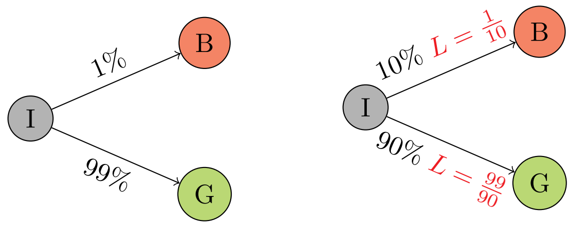
The main approaches to rare event simulation can be divided into two categories: *importance splitting* and *importance sampling*. Both of these were developed in the early days of computing [4].

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. The simulation engine then begins simulating normally, but starts additional runs from states of high importance. This way, the additional simulation runs are more likely to eventually reach the rare event.

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 [29].

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 [15]. 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



(a) Original system **(b)** Modified system

Figure 3: Example of a change of measure for importance sampling for a discrete-time model. The event of interest is reaching state B. In the original system this event has a probability of 1%, while a possible modification for importance sampling increases this probability to 10%, giving a likelihood ratio of $L = \frac{1\%}{10\%} = \frac{1}{10}$.

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 (formally: if the i'th simulation run observes trace π, then $L_i = \frac{\mathbb{P}_{orig}(\pi)}{\mathbb{P}_{IS}(\pi)}$). Details of the computation of L_i depend on the system being simulated.

Example 2. *We consider the system in Figure 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 $L_{I \to B} = \frac{0.01}{0.1} = 0.1$.*

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 artificially decreased probability ($L_i > 1$), it has a higher impact on the estimate. Should the event be reached via a path with an unchanged probability ($L_i = 1$), its effect on the estimate is also unchanged compared to normal MC simulation.

Example 3. *Figure 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 B at all. In the most likely case (1 observed instance), the 95% confidence interval for the probability is* $[0.0024, 0.0545]$ *(using the Clopper-Pearson method [30]).*

If one makes the rare event 10 times as likely, as shown in Figure 3b, the same 100 simulations will observe far more runs reaching B. In the most likely case of observing 10 runs reaching G, a 95% confidence interval of the original system (compensating for the increased likelihood) is $[0.0049, 0.0176]$*, over four times as precise as the original estimate.*

Change of Measure. While the general idea behind importance sampling is simple, making the interesting but rare event less rare (i.e., increasing the *probability measure* evaluated at the event) 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) 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. In other words, the likelihood ratio of transitions moving towards the goal should be below 1, while transition moving away from the goal should be above 1. 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) [4]. 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 $\frac{1}{N}I_iL_i = \frac{1}{N}1P_o$, 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 [31]. The Path-ZVA algorithm used in our approach builds such an approximation (hence the name: zero-variance approximation based on dominant paths).

Example 4. *Figure 4a shows a discrete-time Markov chain, in which we estimate the probability of reaching the goal (rightmost) state. The actual probability is clearly* $0.9^3 = 0.729$.

Figure 4b shows a zero-variance estimator of this probability. Every sample run will reach the goal 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 goal state, so we simply force this to always be the path sampled.*

Figure 4c illustrates the downside of a poorly chosen change of measure: if we do not use our knowledge of the path to the goal state, and simply make each transition equally likely, we end up reducing the likelihood of reaching the goal. We thus obtain a greater variance than in the original system.

4. Our approach: FTRES

Our overall approach to rare event simulation for DFTs, implemented in our tool FTRES (Fault Tree Rare Event Simulator), relies on a conversion of the DFT into an input/output interactive Markov chain (I/O-IMC). This I/O-IMC is a Markovian model describing the behaviour of the DFT. Given a DFT, our analysis technique consists of the following steps:

1. Use the DFTCalc tool to compute I/O-IMCs for all elements of the DFT.
2. Apply the steps of the Path-ZVA algorithm, as explained in Sect. 4.3, 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 completes a cycle (i.e., returns to the initial state), storing the likelihood ratio L_i and time spent in unavailable (i.e., failed) states Z_i.
4. Sample traces of the original model, again one cycle per trace, storing the total time of the cycle D_i,
5. Average the total time $\overline{D}$ and unavailable time $\overline{ZL}$ of the traces, multiplied by the likelihood ratios. Now $\overline{ZL}/\overline{D}$ is the output estimated unavailability.

In our approach, we follow the semantics of [6], which describes the behaviour of dynamic fault trees as I/O-IMCs. These semantics were extended in [32] 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 DFT.

4.1. Compositional Fault Tree Semantics

The analysis used in this paper follows the compositional semantics in terms of input/output interactive

Markov chains given in [6], with subsequent extensions for maintainable systems [32]. This compositional approach 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.

For repairs, we follow the extension introduced for Fault Maintenance Trees (FMTs) [3], in which inspection and repair modules periodically examine the condition of attached basic events, and perform maintenance as needed depending on the conditions of the BEs.

Example 5. *Figure 5 shows the I/O-IMC of an inspection module (IM). The dashed transitions denote Markovian transition governing the times at which inspections are performed. The solid-lined transitions are decorated with input and output actions that allow the IM to communicate with its attached BE whether the BE has reached the maintenance threshold, and with a repair module to conduct a repair.*

Figure 6 shows the I/O-IMC of a repairable basic event. When the BE degrades from the 'okay' to the 'degraded' state, it communicates to the associated IM that the BE needs repair. Subsequently, it will either be repaired, receiving the 'repair?' signal from a repair module, or eventually fail.

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 [6].

Example 6. *An example of this composition is shown in Figure 7. 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 this 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.

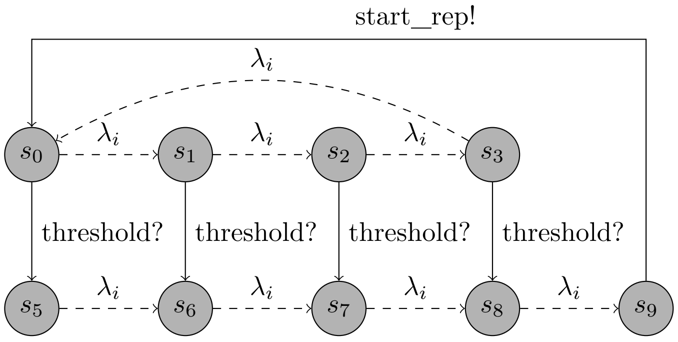
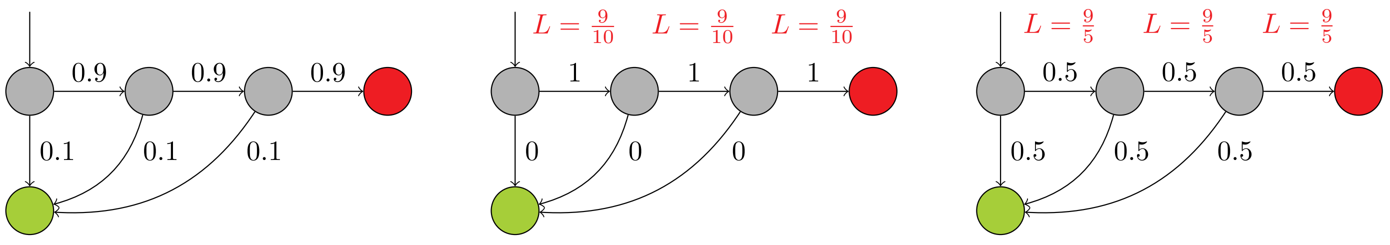


Figure 5: I/O-IMC of an Inspection module with Erlang-distributed time between inspections. As time progresses, the module advances towards the states on the right, until an inspection is performed returning the module to state s_0. If no threshold signal is received, the model remains in the top row of states, and the transition $s_3 \to s_0$ performs no action during the inspection. When a threshold signal is received, the model moves to the bottom row, waits until the time an inspection is performed (i.e., state s_9 is reached), signals that a repair is needed, and moves back to the initial state.

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.

4.2. 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 DFT. 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 an interactive transition, we apply the maximal progress assumption[33] and take this transition. If multiple interactive transitions



(a) Original system: Variance ≈ 0.198 **(b)** Zero-variance estimator: Variance = 0 **(c)** Worse estimator: Variance ≈ 3.72

Figure 4: Examples of different changes of measure in a discrete-time system and their effects variance of the estimator of the probability to reach the red state.

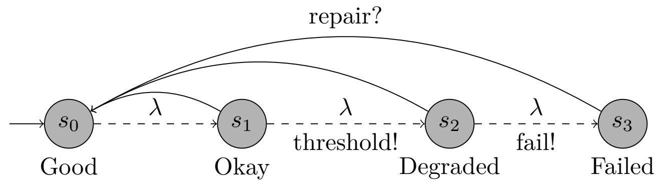


Figure 6: Illustration of the I/O-IMC of a repairable basic event including communication signals (interactive and Markovian transition combined into one transition for brevity).

can be taken, we verify that all paths of only interactive transitions lead to the same (Markovian) state. Thus we are only left with states with only Markovian transitions, which can be used as an input for the Monte Carlo simulation.

In more detail, we check after every Markovian transition that the directed graph formed by interactive transitions from the current state always leads to the same set of Markovian transitions: From the current state, we apply Tarjan's algorithm [34] to identify the bottom strongly connected components (BSCCs), excluding Markovian transitions. We then verify that the exit rates and outgoing (Markovian) probability distributions are the same for all states in these BSCCs. There are now three possibilities:

- One or more BSCCs has no outgoing Markovian transitions. In this case, we abort the analysis since the model is ill-defined.
- Different states in the BSCCs have different exit rates or outgoing probability distributions. Here, we also abort the analysis with an error message that possibly non-spurious nondeterminism has been detected.
- All states in the BSCCs have the same exit rate and outgoing probability distributions. In this case, we replace the outgoing transitions from the current state by this rate and distribution.

We note that this approach does not exclude IMCs with interactive cycles (i.e., *Zeno runs*), we merely require that every state on such a cycle has the same Markovian behaviour.

Compared to the algorithm described in the previous version of this paper [23], this algorithm has two advantages:

- We can analyze a larger set of models, since we previously excluded any DFT for which a syntactic check of the DFT identified possible nondeterminism. Our current analysis is still conservative, i.e., we still exclude some DFTs in which nondeterminism does not affect the result, but we now allow models that turn out not to have nondeterminism within the states reached by the simulation.
- We verify that any nondeterminism is definitely spurious, thereby ensuring that we have not missed any potential problem cases in the syntactic checks on DFTs.

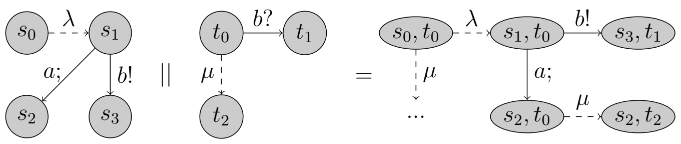


Figure 7: Example of the partial parallel composition of two I/O-IMCs.

4.3. 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 [5, 35]. 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).

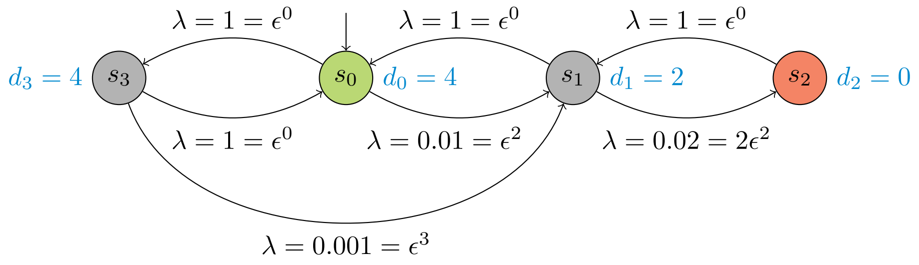
Path-ZVA produces a CoM suitable for estimating the probabilities of events of the form "reaching set of states A (*goal states*), starting from state B (*initial state*), and before reaching a state in set C (*taboo states*)", where the system must frequently visit some states in C. In our setting, 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 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 typically measured as the total rarity of the 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 likely, and states further away from the target become less likely.

This model relies on the transition rates being described 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



$\lambda = 1 = \epsilon^0$
$\lambda = 1 = \epsilon^0$
$\lambda = 1 = \epsilon^0$
$d_3 = 4$
s_3
s_0
$d_0 = 4$
s_1
$d_1 = 2$
s_2
$d_2 = 0$
$\lambda = 1 = \epsilon^0$
$\lambda = 0.01 = \epsilon^2$
$\lambda = 0.02 = 2\epsilon^2$
$\lambda = 0.001 = \epsilon^3$

Figure 8: Illustration of the Path-ZVA model in a CTMC. 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.

powers k of ϵ are smallest. In the limit of $\epsilon \downarrow 0$, these paths dominate the total probability of reaching the target.

Example 7. *Figure 8 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 $s_0s_1s_2$ have rarity 2, so the distance is $d_0 = 2 + 2 = 4$. Finally, The most probable path from s_3 is the path $s_3s_0s_1s_2$, giving distance $d_3 = 0 + 2 + 2 = 4$. Note that the path $s_3s_1s_2$ is shorter in number of transitions, but has a higher total rarity (5), and is therefore not the most likely path (for $\epsilon << 1$).*

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 paper, we fix a value of $\epsilon < 1$ (typically $\epsilon = 0.01$), 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 distances of 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 $p^{IS}_{ik} = \frac{\epsilon^2}{\epsilon^2+\epsilon^3}$ and $p^{IS}_{il} = \frac{\epsilon^3}{\epsilon^2+\epsilon^3}$ (for $\epsilon = 0.1$, this means $p^{IS}_{ik} \approx 0.9$ and $p^{IS}_{il} \approx 0.1$). 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. Since we are calculating steady-state unavailability, the probability of reaching an unavailable state is determined only by the relative probabilities of the transitions, not by the total exit rate, so the unchanged exit rate does not affect the performance of the estimation.

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 [5].

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 $t_n = T$, $\forall_i : p_{t_it_j} > 0$, 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 [5]. 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. [36]). 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.

Example 8. *To summarize our approach, Figure 9 illustrates the steps on a simple DFT with two components, and periodic repair.*

1. *We convert every element of the DFT in Figure 9a into an I/O-IMC shown in Figure 9b.*
2. *We compose these I/O-IMCs and remove the non-Markovian transitions, obtaining the model shown in Figure 9c. 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 (s_1) in one transition, which is only the blue transition ($s_0 \to s_1$) in Figure 9c.*
4. *Applying Path-ZVA, we increase the likelihood of the transitions along the previously identified path, resulting in the model shown in Figure 9d.*
5. *We draw simulation traces from the adjusted model. Each trace contains one cycle, and we keep track of the time spent in unavailable states, and the likelihood ratio of the trace (the product of the likelihood ratios of the transitions in the trace). For example, we can draw three traces (in reality one would draw many thousands of traces):*
   (a) *$t_0 t_0$ ($L_1 = \frac{5}{5+3\epsilon} / \frac{\frac{5}{8}(5+3\epsilon)}{5+3\epsilon} = \frac{5}{\frac{5}{8}(5+3\epsilon)} \approx 7.5$) with no time in unavailable states ($Z_1 = 0$).*
   (b) *$t_0 t_2 t_0$ ($L_2 \approx 0.23$) with no unavailable time ($Z_2 = 0$).*
   (c) *$t_0 t_1 t_0$ ($L_3 \approx 0.15$) with unavailable time $Z_3 = 0.196$. (sampled from an exponential distribution with mean $\frac{1}{5}$ for the unavailable state t_1).*
6. *We draw simulation traces from the original model, and we keep track of the total time of the cycle. We again draw three traces:*
   (a) *$s_0 s_0$ with total time $D_1 = 0.035$ (sampled from an exponential distribution with mean $\frac{1}{5.3}$).*
   (b) *$s_0 s_0$ with total time $D_2 = 0.301$.*
   (c) *$s_0 s_2 s_0$ with total time $D_3 = 0.033 + 0.123 = 0.156$.*
7. *Finally, we combine the samples to obtain our average unavailability. For the samples drawn above, we would obtain $\hat{ZL} = \frac{1}{3}(L_1 Z_1 + L_2 Z_2 + L_3 Z_3) \approx 0.029$, $\hat{D} = \frac{1}{3}(D_1 + D_2 + D_3) \approx 0.164$, and $\hat{U} = \frac{\hat{ZL}}{\hat{D}} \approx 0.18$. More detailed statistical measures, such as confidence intervals, can also be computed.*

4.4. 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 [37] tool to generate the composed IMC which can be analysed by a stochastic model checker such as IMCA [38] or MRMC [39].

Our tool, 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 10 shows how the various programs interact to obtain numerical metrics from a (repairable) DFT. First, a DFT is input to DFTCalc, and its dft2lntc program converts it into a three-part state-space model:

- Each element of the DFT is specified as a LOTOS NT [40] '.lnt' file.
- A '.exp' file [41] provides a specification for the composition of the elements (i.e., which signals synchronize in which models).
- A '.svl' [42] file specifying options regarding the composition process (e.g., which state-space minimization steps to perform).

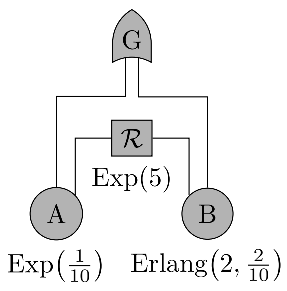
For the analytic solution, DFTCalc then uses the CADP toolset [37] to compose the models into one I/O-IMC (the '.bcg' file in the diagram), and translates this model into the input to a model-checking tool which computes the desired metric.

FTRES does not generate the full composed state-space, but rather uses CADP to generate each element's state-space separately (stored in a '.aut' file), and keeps the composition specification in the '.exp' file. These are then used to generate the needed states on the state-space on the fly during the importance sampling process.

5. Case Studies

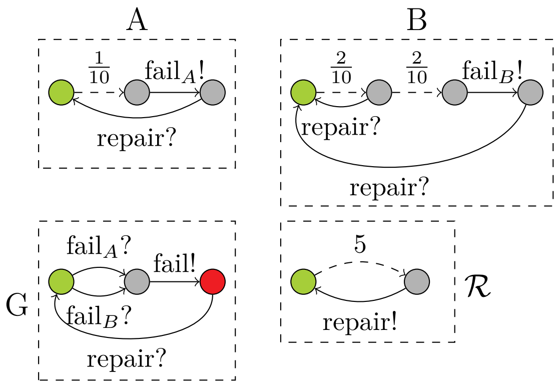
We evaluate the effectiveness of the importance sampling analysis method described in this paper on three parameterized case studies. We compare our FTRES tool to the DFTCalc tool, which evaluates DFTs numerically through stochastic model checking [7], and to a standard Monte Carlo simulator (MC) built 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 [2]. The other two cases are the fault-tolerant parallel processor (FTPP) [1] and the hypothetical example computer system (HECS) [43].

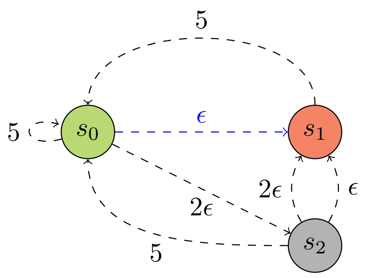


G
$\mathcal{R}$
Exp(5)
A
B
$\text{Exp}\left(\frac{1}{10}\right)$
$\text{Erlang}\left(2, \frac{2}{10}\right)$

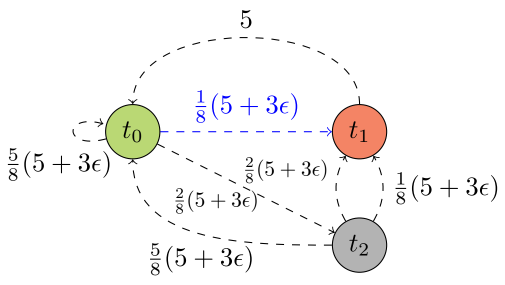
(a) Repairable fault tree (the repair box $\mathcal{R}$ periodically repairs connected components)



(b) I/O-IMCs



(c) Composed model converted to CTMC ($\epsilon = \frac{1}{10}$, all failed states collapsed into s_1)



5
$\frac{1}{8}(5+3\epsilon)$
t_0
t_1
$\frac{5}{8}(5+3\epsilon)$
$\frac{2}{8}(5+3\epsilon)$
$\frac{2}{8}(5+3\epsilon)$
$\frac{1}{8}(5+3\epsilon)$
$\frac{5}{8}(5+3\epsilon)$
t_2

(d) Adjusted CTMC after performing the calculations of step 6 of Path-ZVA

Figure 9: Example of the steps to apply importance sampling to a DFT. Dashed arrows indicate transitions with exponentially distributed delay times, solid arrows indicate transitions with no delay or delayed by synchronizing actions.

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 [43]), 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 24 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 2.10 GHz Intel® Xeon® E5-2683 v4 processor and 256 GB of RAM.

5.1. Railway Cabinets

This case, provided by the consulting company Movares in [2], is a model of a redundant system of relay and high-voltage cabinets used in railway signalling.

The model, shown in Figure 1, 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 signal 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 an inspection module monitoring all the BEs in the FT. If the degradation phase of any BE exceeds the threshold phase (1 phase before the failure) at the time of inspection, a repair is triggered to

			Unavailability		
	N	**P**	**DFTCalc**	**FTRES**	**MC**
Railway cabinets	2	1	$4.25685 \cdot 10^{-4}$	$[4.256; 4.258] \cdot 10^{-4}$	$[4.253; 4.278] \cdot 10^{-4}$
	3	1	$7.71576 \cdot 10^{-4}$	$[7.712; 7.718] \cdot 10^{-4}$	$[7.706; 7.743] \cdot 10^{-4}$
	4	1	$1.99929 \cdot 10^{-3}$	$[1.991; 2.000] \cdot 10^{-3}$	$[1.975; 2.003] \cdot 10^{-3}$
	2	2	$4.55131 \cdot 10^{-8}$	$[4.547; 4.569] \cdot 10^{-8}$	$[3.214; 5.599] \cdot 10^{-8}$
	3	2	$6.86125 \cdot 10^{-8}$	$[6.752; 7.046] \cdot 10^{-8}$	$[5.092; 8.682] \cdot 10^{-7}$
	4	2	$2.38069 \cdot 10^{-7}$	$[2.275; 2.434] \cdot 10^{-7}$	$[1.889; 4.991] \cdot 10^{-7}$
	2	3	$5.97575 \cdot 10^{-12}$	$[5.757; 6.408] \cdot 10^{-12}$	—
	3	3	$7.51512 \cdot 10^{-12}$	$[4.637; 7.042] \cdot 10^{-12}$	—
	4	3	—	$[3.272; 8.620] \cdot 10^{-12}$	—
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	—	$[2.226; 2.232] \cdot 10^{-10}$	—
	1	2	$1.76174 \cdot 10^{-20}$	$[1.761; 1.762] \cdot 10^{-20}$	—
	2	2	$1.76178 \cdot 10^{-20}$	$[1.761; 1.763] \cdot 10^{-20}$	—
	3	2	—	$[1.761; 1.762] \cdot 10^{-20}$	—
	4	2	—	$[1.760; 1.763] \cdot 10^{-20}$	—
	N	**k**			
HECS	1	1	$4.12485 \cdot 10^{-5}$	$[4.124; 4.126] \cdot 10^{-5}$	$[4.079; 4.156] \cdot 10^{-5}$
	2	1	$3.02469 \cdot 10^{-9}$	$[3.022; 3.026] \cdot 10^{-9}$	$[0; 9.040] \cdot 10^{-9}$
	2	2	$8.24940 \cdot 10^{-5}$	$[8.247; 8.251] \cdot 10^{-5}$	$[8.218; 8.338] \cdot 10^{-5}$
	3	1	$3.11891 \cdot 10^{-13}$	$[3.103; 3.128] \cdot 10^{-13}$	—
	3	2	$9.07344 \cdot 10^{-9}$	$[9.060; 9.076] \cdot 10^{-9}$	$[8.153; 20.70] \cdot 10^{-9}$
	3	3	$1.23736 \cdot 10^{-4}$	$[1.236; 1.238] \cdot 10^{-4}$	$[1.234; 1.251] \cdot 10^{-4}$
	4	1	—	$[3.902; 4.364] \cdot 10^{-17}$	—
	4	2	—	$[1.239; 1.252] \cdot 10^{-12}$	—
	4	3	—	$[1.813; 1.818] \cdot 10^{-8}$	$[0; 8.352] \cdot 10^{-9}$
	4	4	—	$[1.648; 1.651] \cdot 10^{-4}$	$[1.621; 1.657] \cdot 10^{-4}$

Table 1: Comparison of the unavailabilities computed by DFTCalc, FTRES, and MC simulation for the case studies with N cabinets/processor groups/HECS replications, P degradation phases per BE for the railway cabinets and FTPP cases, and k required functional replications for the HECS case.

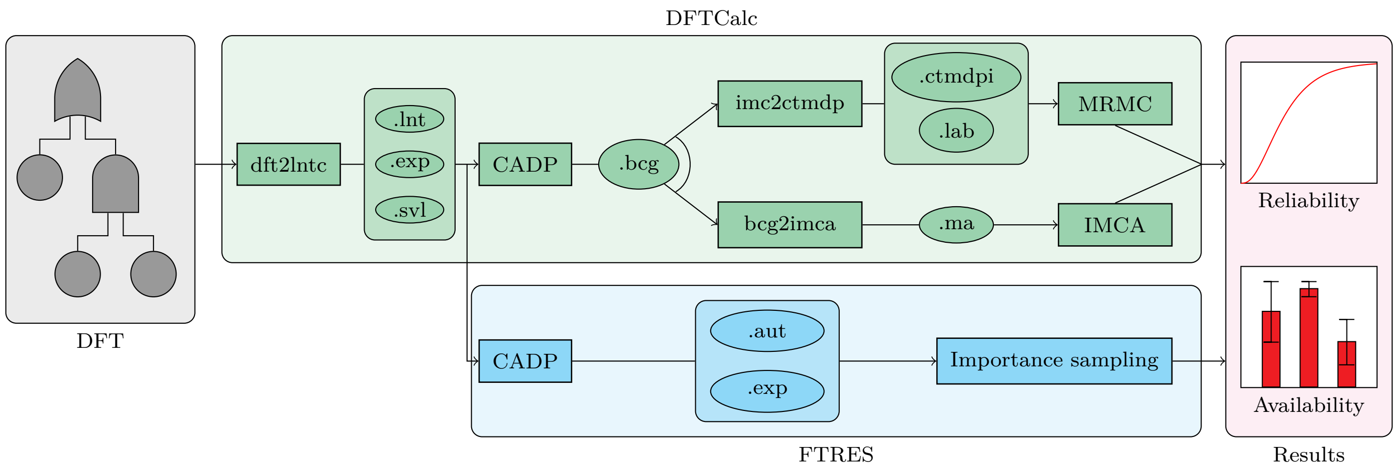


Figure 10: Diagram of the workflow of FTRES and DFTCalc.

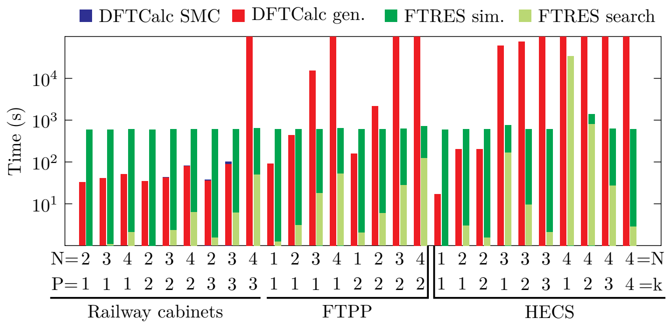


Figure 11: Processing times for the different tools: Times for model generation (gen.) and stochastic model checking (SMC, negligible compared to model generation time) for DFTCalc, and for the graph search and simulation phases for FTRES (calculation for the change-of-measure are performed during the simulation). Bars reaching the top of the graph reached the time-out of 24 hours.

replace all degraded BEs. The time between inspections 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 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 3-phase model with 4 cabinets could not be computed by DFTCalc within the time-out (times shown in Figure 11), 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.

Figure 14 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 usually smaller due to its bisimulation minimisation, the intermediate models are often much larger.

5.2. Fault-Tolerant Parallel Processor

The second case study is taken from the DFT literature [1], and describes a fault-tolerant parallel computer system illustrated in Figure 12. 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.

The numerical results and computation times for this case study can be found in Table 1 and Figure 11 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 three largest cases while FTRES still performs well. The standard MC simulation observed no failures for most of the models.

Figure 14 lists the generated state spaces for both tools. Again, FTRES requires less peak memory than DFTCalc.

5.3. Hypothetical Example Computer System

Our final example is the classic benchmark DFT of the Hypothetical Example Computer System (HECS), described in [43] 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,
- 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 13.

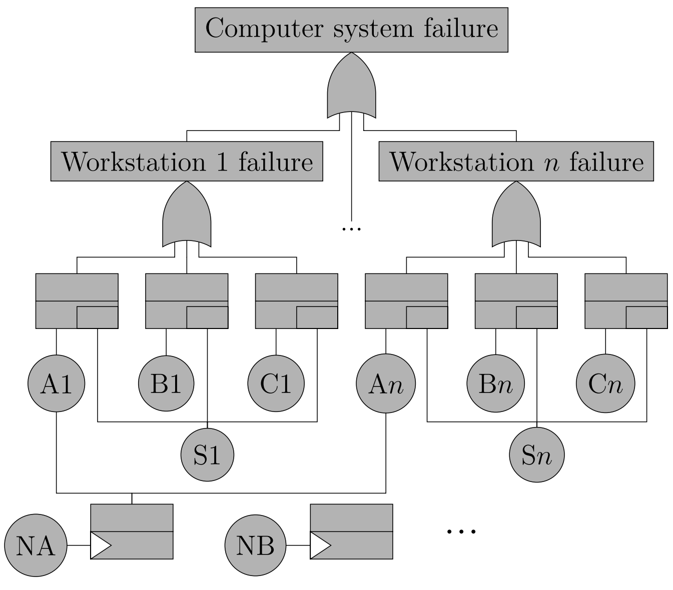


Figure 12: DFT of the fault-tolerant parallel processor. Connections between the FDEP for B omitted for clarity, as well as the FDEPs for groups C and S.

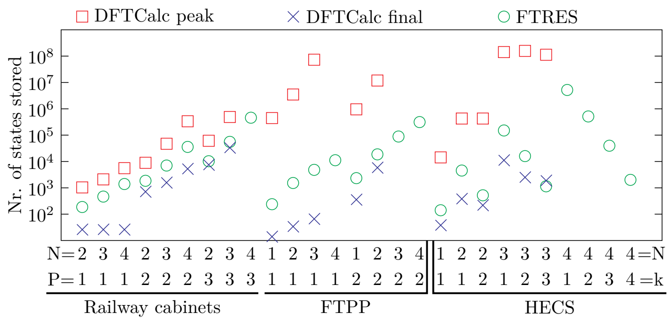


Figure 14: Numbers of states stored in memory for the different cases with N cabinets/processor groups. For DFTCalc, both the largest intermediate (peak) and the minimised (final) state spaces are given.

We parameterize this example by replicating the HECS N times, and requiring 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 1 lists the unavailabilities computed by the tools, while Figures 11 and 14 show the processing time and state spaces computed, respectively. We notice that for the 4-replication models, DFTCalc is unable to compute the state space in the available time, and the MC simulator in many cases failed to observe any failures, and sometimes produces very wide confidence intervals in the cases where it did. FTRES, on the other hand, produced reasonable confidence intervals for all cases.

5.4. 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.
- gives confidence intervals up to an order of magnitude tighter than those estimated by MC in similar processing time.

6. 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 paper 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 [6] and [32], providing flexibility and extensibility in the semantics of the models. By deploying the Path-ZVA algorithm [5], we only need to explore a small fraction of the entire state space, substantially reducing the state-space explosion

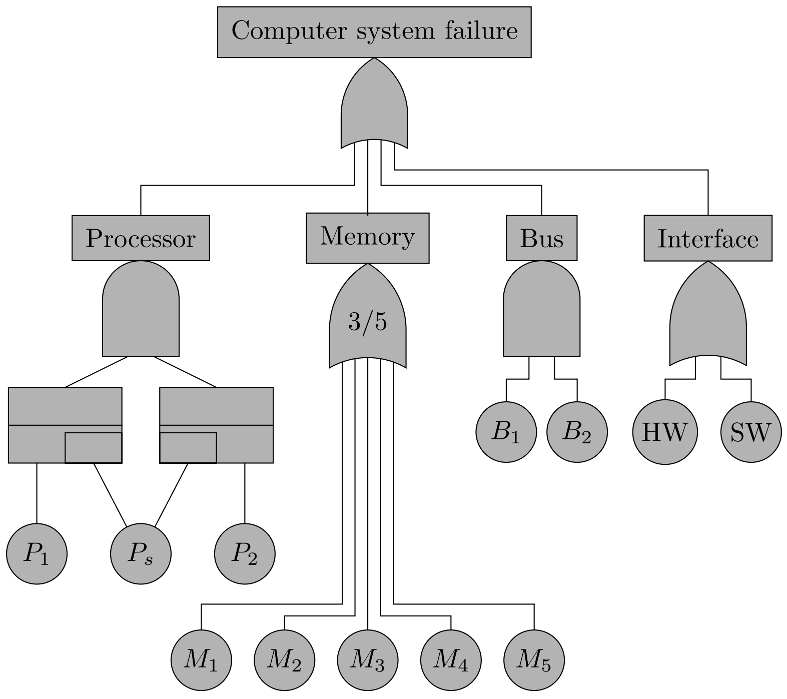


Figure 13: DFT of the hypothetical example computer system.

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 the stochastic model checking approach used by 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 (i.e., exponential probability distributions for transition times and no non-spurious nondeterminism), which means we can only approximate the semantics described in [3]. 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. [6] 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.

Acknowledgments. This research was partially funded by STW and ProRail under project ArRangeer (grant 12238) with participation by Movares, STW project SEQUOIA (15474), NWO project BEAT (612001303), NWO project SamSam (50918239), and the EU project grant SUCCESS (102112).

[1] J. B. Dugan, S. J. Bavuso, M. A. Boyd, Fault trees and sequence dependencies, in: Proc. Annu. Reliability and Maintainability Symp., IEEE, 1990, pp. 286–293. `doi:10.1109/ARMS.1990.67971`.

[2] D. Guck, J. Spel, M. I. A. Stoelinga, DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper), in: Proc. 17th Int. Conf. Formal Engineering Methods (ICFEM), Vol. 9407 of LNCS, 2015, pp. 304–311. `doi:10.1007/978-3-319-25423-4_19`.

[3] E. Ruijters, D. Guck, P. Drolenga, M. I. A. Stoelinga, Fault maintenance trees: reliability centered maintenance via statistical model checking, in: Proc. IEEE 62nd Annu. Reliability and Maintainability Symposium (RAMS), 2016. `doi:10.1109/RAMS.2016.7447986`.

[4] H. Kahn, T. Harris, Estimation of particle transmission by random sampling, in: Monte Carlo method; Proc. Symp. held June 29, 30, and July 1, 1949, Vol. 12 of Nat. Bur. Standards Appl. Math. Series, 1951, pp. 27–30.

[5] D. Reijsbergen, P. T. de Boer, W. Scheinhardt, S. Juneja, Path-ZVA: general, efficient and automated importance sampling for highly reliable Markovian systems, ACM Transactions on Modeling and Computer Simulation (TOMACS) 28 (3). `doi:10.1145/3161569`.

[6] H. Boudali, P. Crouzen, M. I. A. Stoelinga, A rigorous, compositional, and extensible framework for dynamic fault tree analysis, IEEE Trans. Dependable Secure Comput. 7 (2) (2010) 128–143. `doi:10.1109/TDSC.2009.45`.

[7] F. Arnold, A. Belinfante, D. G. Freark van der Berg, M. I. A. Stoelinga, DFTCalc: A tool for efficient fault tree analysis, in: Proc. 32nd Int. Conf. Computer Safety, Reliability and Security (SAFECOMP), Vol. 8153 of LNCS, 2013, pp. 293–301. `doi:10.1007/978-3-642-40793-2_27`.

[8] E. Ruijters, M. I. A. Stoelinga, Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools, Computer Science Review 15–16 (2015) 29–62. `doi:10.1016/j.cosrev.2015.03.001`.

[9] W. E. Vesely, F. F. Goldberg, N. H. Roberts, D. F. Haasl, Fault Tree Handbook, Office of Nuclear Regulatory Reasearch, U.S. Nuclear Regulatory Commision, 1981.

[10] A. Bobbio, D. Codetta-Raiteri, Parametric fault trees with dynamic gates and repair boxes, in: Proc. 2004 Annu. IEEE Reliability and Maintainability Symp. (RAMS), 2004, pp. 459–465. `doi:10.1109/RAMS.2004.1285491`.

[11] D. Codetta-Raiteri, G. Franceschinis, M. Iacono, V. Vittorini, Repairable fault tree for the automatic evaluation of repair policies, in: Proc. Annu. IEEE/IFIP Int. Conf. Dependable Systems and Networks (DSN), 2004, pp. 659–668. `doi:10.1109/DSN.2004.1311936`.

[12] W. E. Vesely, R. E. Narum, PREP and KITT: computer codes for the automatic evaluation of a fault tree, Tech. rep., Idaho Nuclear Corp. (1970).

[13] H. Kumamoto, K. Tanaka, K. Inoue, E. J. Henley, Dagger-sampling Monte Carlo for system unavailability evaluation, IEEE Trans. Rel. R-29 (2) (1980) 122–125. `doi:10.1109/TR.1980.5220749`.

[14] M. Ramakrishnan, Unavailability estimation of shutdown system of a fast reactor by Monte Carlo simulation, Ann. Nuclear Energy 90 (2016) 264–274. `doi:10.1016/j.anucene.2015.11.031`.

[15] P. Heidelberger, Fast simulation of rare events in queueing and reliability models, ACM Trans. Modeling and Computer Simulation (TOMACS) 5 (1) (1995) 43–85. `doi:10.1145/203091.203094`.

[16] R. Gulati, J. B. Dugan, A modular approach for analyzing static and dynamic fault trees, in: Proc. Annu. IEEE Reliability and Maintainability Symp. (RAMS), 1997, pp. 57–63. `doi:10.1109/RAMS.1997.571665`.

[17] K. D. Rao, V. Gopika, V. V. S. Sanyasi Rao, H. S. Kushwaha, A. K. Verma, A. Srividya, Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment, Reliability Engineering and System Safety 94 (4) (2009) 872–883. `doi:10.1016/j.ress.2008.09.007`.

[18] B. Kaiser, C. Gramlich, M. Förster, State/event fault trees – a safety analysis model for software-controlled systems, Reliablity Engineering and System Safety 92 (11) (2007) 1521–1537. `doi:10.1016/j.ress.2006.10.010`.

[19] H. Boudali, P. Crouzen, M. I. A. Stoelinga, A compositional semantics for dynamic fault trees in terms of interactive Markov chains, in: Proc. 5th Int. Symp. on Automated Technology for Verification and Analysis (ATVA), Vol. 4762, 2007, pp. 441–456. `doi:10.1007/978-3-540-75596-8_31`.

[20] H. Hermanns, Interactive Markov Chains, Vol. 2428 of LNCS, Springer, 2002. `doi:10.1007/3-540-45804-2`.

[21] M. R. Pulungan, Reduction of acyclic phase-type representations, Ph.D. thesis, Universität des Saarlandes, Saarbrücken (2009).

[22] A. M. Law, Simulation modeling and analysis, 4th Edition, McGraw-Hill New York, 2007.

[23] E. Ruijters, D. Reijsbergen, P. T. de Boer, M. Stoelinga, Rare event simulation for dynamic fault trees, in: Proc. Int. Conf. Computer Safety, Reliability, and Security (SAFECOMP), Vol. 10488 of LNCS, Springer, 2017, pp. 20–35. `doi:10.1007/978-3-319-66266-4_2`.

[24] ISO 26262:2011: Road vehicles – functional safety (2011).

[25] S. Junges, D. Guck, J.-P. Katoen, M. I. A. Stoelinga, Uncovering dynamic fault trees, in: Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Systems and Networks (DSN), 1990, pp. 299–310. `doi:10.1109/DSN.2016.35`.

[26] E. Ruijters, D. Guck, P. Drolenga, M. Peters, M. Stoelinga, Maintenance analysis and optimization via statistical model checking: Evaluation of a train's pneumatic compressor., in: Proc. 13th Int. Conf. Quantitative Evaluation of SysTems (QEST), Vol. 9826 of Lecture Notes in Computer Science, 2016, pp. 331–347.

doi:10.1007/978-3-319-43425-4_22.

[27] G. 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.

[28] EEIG ERTMS Users Group, ERTMS/ETCS RAMS requirements specification, chapter 2 - RAM, Tech. Rep. 02S1266-, UIC (1998).

[29] P. L'Ecuyer, F. Le Gland, P. Lezaud, B. Tuffin, Rare Event Simulation using Monte Carlo Methods, John Wiley & Sons, 2009, Ch. 3 – Splitting techniques, pp. 39–61. doi:10.1002/9780470745403.ch3.

[30] J. C. Clopper, E. S. Pearson, The use of confidence or fiducial limits illustrated in the case of the binomial, Biometrika 26 (4) (1934) 404–413. doi:10.1093/biomet/26.4.404.

[31] P. L'Ecuyer, B. Tuffin, Approximating zero-variance importance sampling in a reliability setting, Ann. Operations Research 189 (1) (2011) 277–297. doi:10.1007/s10479-009-0532-5.

[32] D. Guck, J.-P. Katoen, M. I. A. Stoelinga, T. Luiten, J. Romijn, Smart railroad maintenance engineering with stochastic model checking, in: Proc. 2nd Int. Conf. Railway Technology: Research, Development and Maintenance (Railways), Vol. 104 of Civil-Comp Proceedings, Civil-Comp Press, 2014, article no. 299. doi:10.4203/ccp.104.299.

[33] C. Eisentraut, H. Hermanns, L. Zhang, On probabilistic automata in continuous time, in: Proc. 25th Annual IEEE Symposium on Logic in Computer Science (LICS), 2010, pp. 342–351. doi:10.1109/LICS.2010.41.

[34] R. Tarjan, Depth-first search and linear graph algorithms, SIAM Journal on Computing 1 (2) (1972) 146–160. doi:10.1137/0201010.

[35] D. Reijsbergen, Efficient simulation techniques for stochastic model checking, Ph.D. thesis, University of Twente, Enschede (December 2013). doi:10.3990/1.9789036535861.

[36] P. L'Ecuyer, J. Blanchet, B. Tuffin, P. Glynn, Asymptotic robustness of estimators in rare-event simulation, ACM Trans. Modeling and Computer Simulation (TOMACS) 20 (1) (2010) nr. 6. doi:10.1145/1667072.1667078.

[37] H. Garavel, F. Lang, R. Mateescu, W. Serwe, CADP 2011: a toolbox for the construction and analysis of distributed processes, Int. J. Software Tools for Technology Transfer 15 (2) (2013) 89–107. doi:10.1007/s10009-012-0244-z.

[38] D. Guck, T. Han, J.-P. Katoen, M. R. Hauhäußer, Quantitative timed analysis of interactive Markov chains, in: Nasa Formal Methods (NFM), Vol. 7226 of LNCS, 2012, pp. 8–23. doi:10.1007/978-3-642-28891-3_4.

[39] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, D. N. Jansen, The ins and outs of the probabilistic model checker MRMC, Performance Evaluation 68 (2) (2011) 90–104. doi:10.1016/j.peva.2010.04.001.

[40] D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, F. Lang, C. McKinty, V. Powazny, W. Serwe, G. Smeding, Reference manual of the lnt to lotos translator (version 6.7), Tech. rep., INRIA/VASY - INRIA/CONVECS (2018).
URL http://cadp.inria.fr/publications/Champelovier-Clerc-Garavel-et-al-10.html

[41] F. Lang, Exp.open 2.0: A flexible tool integrating partial order, compositional, and on-the-fly verification methods, in: Proc. 5th Int. Conf. Integrated Formal Methods (IFM), Vol. 3771 of LNCS, 2005, pp. 70–88. doi:10.1007/11589976_6.

[42] H. Garavel, F. Lang, SVL: a scripting language for compositional verification, in: Proc. 21st Int. Conf. Formal Techniques for Networked and Distributed Systems (FORTE), Vol. 69 of IFIP International Federation for Information Processing, 2001, pp. 377–394. doi:10.1007/0-306-47003-9_24.

[43] M. Stamatelatos, W. Vesely, J. B. Dugan, J. Fragola, J. Minarick, J. Railsback, Fault Tree Handbook with Aerospace Applications, Office of safety and mission assurance NASA headquarters, 2002.