Rare Event Simulation for Dynamic Fault Trees

Enno Ruijters[1], Daniël Reijsbergen[2], Pieter-Tjerk de Boer[3], Mariëlle Stoelinga[1]

[1] Formal Methods and Tools, University of Twente, The Netherlands
{e.j.j.ruijters, m.i.a.stoelinga}@utwente.nl
[2] Laboratory for Foundations of Computer Science, U. of Edinburgh, Scotland
dreijsbe@staffmail.ed.ac.uk
[3] Design and Analysis of Communication Systems, U. of Twente, The Netherlands
p.t.deboer@utwente.nl

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 two parameterized sets of case studies, showing that our method can handle fault trees that could not be evaluated with either existing analytical techniques, 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 common and widely used extensions is the dynamic fault tree (DFT) [7]. It provides support for common patterns in system design and analysis, at the cost of requiring more memory- and time-intensive analysis techniques. More recently still, maintenance has been integrated into DFTs supporting complex policies of inspections and repairs [9]. This development has again increased the memory and time needed to analyze, to the point where many practical system cannot be analyzed on current systems 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 [19], 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 [12]. 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 otherwise be needed.

We present a novel approach to analyzing DFTs with maintenance through importance sampling. We adapt the recently-developed Path-ZVA algorithm [18] to the settings of DFTs. We retain the existing compositional semantics by Boudali et al. [4] already used in current tools [1]. Using two 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.

Related work. Apart from DFTs and repairs, many more extensions have been developed. For an overview we refer the reader to [20]. Most current FTA formalisms support repairs using per-component repair times [22]. More complicated policies can be specified using repair boxes [2] or the Repairable Fault Tree extension [6], 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 [20]. Standard simulation methods date back to 1970 [23], continuing to be developed until the present day [19]. Rare event simulation has been used to estimate

system reliability since 1980 [13] and is still applied today [16], although, to our surprise, we know of not aware of approach applying rare event simulation specifically to fault trees. An overview of importance sampling techniques in general can be found in [10].

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 is an industry-standard [11], widely used method for graphically modeling systems and conducting reliability and safety analysis [22]. Fault trees (FTs) model how component failures interact to cause system failures. They assist in the evaluation of a wide number of dependability metrics, including the system reliability (i.e., the probability that the system fails within its given mission time) and the availability (i.e., the average percentage of time that a system is up).

An FT is a directed acyclic graph where the leaves describe failures 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.

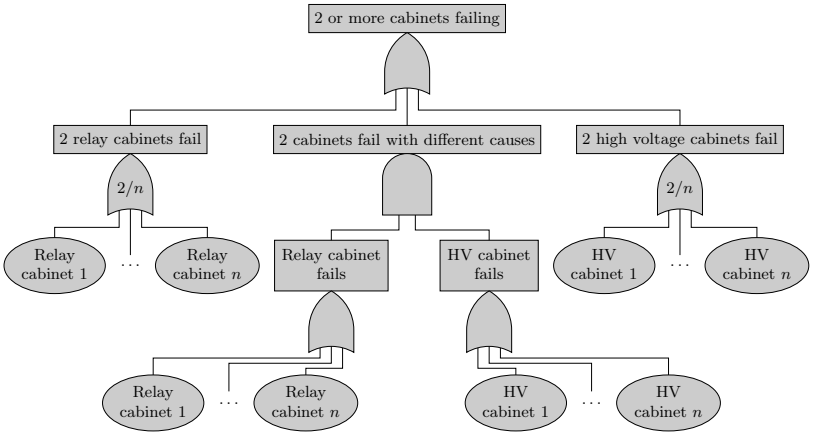

Fig. 1: Example fault tree of the relay cabinet case study. Due to redundancy, the system can survive the failure of any single cabinet, however two failures cause system unavailability. The number of cabinets varies, and is indicated by n.

Standard, also called *static*, fault trees have boolean connectors as gates. These are the AND-, OR-, and VOT(k)-gates, which fail when all, any, or at least k of their children fail, respectively. The leaves of the tree are typically described with either simple probabilities describing the probability of failing within a time window of interest or the probability of being failed at any particular time, or with exponential failure rates describing the probability of failure before any given time. If components are repairable, the repair time in a standard fault tree is typically also given as an exponential rate.

An example of such a tree is shown in Fig. 1. This FT describes a case study from [9], modeling part of the interlocking system of a railway corridor. It consists of relay and high voltage cabinets, with redundancy to survive the failure of any single cabinet of each type. In the figure, the TLE is 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.

Classic quantitative analysis techniques for static fault trees include the computation of:

- the probability of the TLE before a given time (called the system *reliability*),
- the expected percentage of time the system is functioning (the *availability*),
- the components that make the largest contributions to these metrics,
- and the sensitivity of these metrics to the parameters of the BEs.

For a more complete overview of analysis techniques, we refer the reader to [20].

2.1 Dynamic and repairable fault trees

Over the years, many extensions to FTs have been developed [20]. One of the most prominent extension is the *dynamic fault trees* (DFT) model [7]. DFTs introduce several new gates to cater for common patterns in dependability models: (1) 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. (2) 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, and can be shared between multiple gates. Shared spare elements can only be used by one gate at any time. (3) 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.

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.

Standard fault trees support only simple policies of independent repairs with exponentially distributed repair times starting immediately upon component failure [22]. Various extensions provide more complex policies, describing that

some repairs occur in sequential order rather than in parallel [2], or complex maintenance policies with preventive inspections and repairs [19].

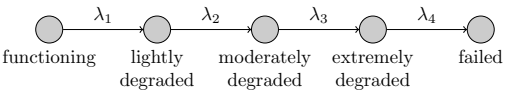

Fig. 2: Basic event with multiple degradation phases.

Dynamic fault trees support both the simple model with independent, exponentially distributed repair times, and more complex policies with periodic inspections and/or repairs [9]. In the more complex policies, BEs can progress through multiple phases of degradation, as depicted in Fig. 2. An inspection periodically checks whether its BEs have degraded beyond some threshold phase, and returns them to their undegraded phase if they have. A periodic replacement simply returns its BEs to their undegraded phase periodically.

2.2 Compositional semantics

The analysis used in this paper follows the compositional semantics in terms of input/output interactive Markov chains given in [4]. 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 minimization helps keep the size of the state-space to a minimum allowing the analysis of larger models.

Input/Output Interactive Markov Chains
I/O-IMCs are a modeling 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 [4].

An I/O-IMC consists of states and transitions between states. The transitions are divided into four categories:

- **Markovian** transitions occur independently of other models. The time at which the transition is taken is governed by an exponential distribution with a given rate. Markovian transitions are denoted by their transition rate, usually denoted by a Greek letter.
- **Internal** actions occur independently of other models, and are taken immediately when their origin state becomes active. They are denoted by a name followed by a semicolon, e.g., 'a;'.
- **Input** actions occur when a parallel model takes the corresponding output action. They are denoted by a name followed by a question mark, e.g., 'b?'.
- **Output** actions can occur immediately when their origin state becomes active. An output transition causes all parallel models to take the corresponding input transition. Output transitions are denoted by a name followed by an exclamation mark, e.g., 'b!'.

If multiple internal or output transitions can be taken from a state, which transition is taken is nondeterministic.

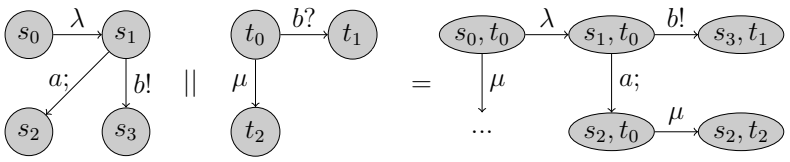

Fig. 3: Example of the partial parallel composition of two I/O-IMCs.

Figure 3 shows a part of the parallel composition of two I/O-IMCs. In the composed model, the component models all begin in their initial states. From t_0 the transition 'b?' cannot be taken unless the output transition 'b!' is also taken, so both initial states can only perform their Markovian transitions. Assuming the leftmost model takes its transition with rate λ first, the composition enters state s_1, t_0. From here, two options are possible: (1) the internal action 'a;' from s_1 to s_2 can be taken, leaving the rightmost model in state t_0, or (2) the output transition 'b!' from s_1 to s_3 can be taken together with the input transition 'b?' from t_0 to t_1. In the latter case, the composed model takes a transition 'b!' allowing it to be composed with yet more models, and enters state s_3, t_1, from which neither component model can take further transitions. If the internal action was taken, the transition from t_0 to t_2 with rate μ remains possible, leading to the terminal state s_2, t_2.

3 Rare event simulation

Performance measures in practical problems often depend on events which occur only rarely. Estimating probabilities of such rare events using standard stochastic simulation is not efficient: with a limited amount of available simulation runs, the event of interest will either not be observed at all, or not sufficiently often to draw statistically sound conclusions. To deal with this, rare-event simulation techniques have been developed, which modify the model or the simulation procedure in such a way that the event of interest occurs more frequently, and then compensate mathematically for this modification.

There are two main approaches to rare event simulation, namely *splitting* and *importance sampling*, both of which go back to the early days of computing [12]. In this paper, we use importance sampling; see [10] for a survey. In importance sampling, the probability distributions of the random variables in the model are modified to make the target event occur more frequently. Every time the simulator draws a sample from a random variable, a so-called *likelihood ratio* is updated, to keep track of the error being made. In standard simulation, the estimator for the target probability would be $\hat{\gamma} = \sum_{i=1}^{N} I_i$, where the sum is over all N samples or sample paths drawn, and I_i is the indicator of the target event having occurred on the ith sample(path). In importance sampling, the estimator is changed to $\hat{\gamma} = \sum_{i=1}^{N} I_i L_i$. Here L_i is the likelihood ratio of the ith sample(path), defined as its probability under the original probability measure divided by its probability under the modified measure.

Change of measure. The challenge in importance sampling is to find a good way to change the probability distribution, also called a *change of measure* (CoM). Generally, transitions (e.g., component failures) that bring the system closer to the target state (e.g., system failure), should be made more likely and vice versa. However, a bad choice can lead to an estimator which is worse than the standard simulation, e.g., by putting too much emphasis on parts of the state space that are not relevant; this can even lead to estimators that are biased or have infinite variance. On the other hand, the theoretically optimal choice leads to an estimator with zero variance. Calculating a zero variance estimator, however, requires that we already know the probability of interest, and is therefore unfeasible.

The Path-ZVA algorithm. Many different methods have been proposed to find a good change of measure; in this paper, we exploit the Path-ZVA algorithm [18,17]. This algorithm is well suited for DFT simulation since it works fully automated for a large class of Markov chain models with provably good performance, and it does not require the entire state space to be constructed. Rather, the only input needed is a function which, given a state, returns to the outgoing transitions, together with the associated rates. The simulator can then estimate probabilities of events of the form "reaching state (or set of states) A, starting from state B, and before returning to state C", where C must be a state that is reached frequently. Also (but related) it can estimate the fraction of time the system spends in states (or set of states) A. In either case, an estimate and a confidence interval are returned. As such, these capabilities of Path-ZVA fit very well to estimating the unavailability of a system composed of several components, as typically described using dynamic fault trees, under the restrictions that there are repairs (so state C in the above can be the state where all components are up), and that all failure and repair processes are Markovian.

Like many other recent algorithms, the Path-ZVA algorithm starts with a numerical approximation of the probability of interest, and then computes a CoM from that approximation, using the formula that could be used for computing the zero-variance CoM if the true probability of interest were known. Some other examples of this approach in the context of modeling highly reliable Markovian systems include [3] and [15]. All of this builds on parameterizing the model's rates in terms of powers of some rarity parameter ϵ, an idea that goes back to [21] where a heuristic CoM was proposed.

In the case of Path-ZVA, the approximation of the probability of interest consists of summing the contribution of only the most important paths to the event of interest; hence the name Path-ZVA: zero-variance approximation based on exploring these dominant paths. Each possible path to the event of interest consists of a number of transitions of the Markov chain, each of which has a rate parameterized by ϵ. The dominant paths are those whose transitions have the lowest total power of ϵ and are thus dominant in the limit of small ϵ. These are found by running a graph analysis algorithm, which needs to explore only a small subset of the state space (typically several orders of magnitude smaller than the full state space). For more details see [18]. Under mild conditions, it can be proven that the method leads to estimators having the nice property of

Bounded Relative Error. 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 with the probability of interest, making its *relative* error bounded (cf. [14]).

Other approaches. Compared to other importance sampling based approaches, (e.g. [21], [15], [5]) the Path-ZVA has the advantage of dealing well with models having so-called high-probability cycles and having provable efficiency properties in the limit of very small ϵ, thus avoiding the issues of bias and large or infinite variance mentioned above. Splitting-based approaches have not been considered in this paper because they tend to be less suitable for models where the rare target event is reached via only a few transitions each having a low rate, since such models provide fewer points where sample paths can be split.

4 Rare event simulation for fault trees

To develop a rare event simulation technique for repairable FTs and DFTs, we need to convert the FT into a Markov chain. For this purpose, we follow the semantics of [4], describing the behavior of a DFT as an I/O-IMC. A major benefit of this approach is that the I/O-IMC is not constructed as one large model, but as a parallel composition of many smaller I/O-IMCs, each modeling one element (i.e., gate, basic event, or maintenance module) of the DFT.

Overall, given a DFT, our analysis technique consists of the following steps:

1. Use DFTCalc to compute I/O-IMCs for all elements of the DFT.
2. Perform a breadth-first search of the Markovianized composition (explained in Sections 2.2 and 4.1) of these elements to identify the smallest number of rare transitions needed to reach a failed state, called d.
3. Continue the breadth-first search to find all paths that reach a failed state within d rare transitions.
4. Apply the Path-ZVA algorithm (explained in Section 3) to adjust the transition probabilities and compute the corresponding likelihood ratios. Since only the above-mentioned paths receive altered probabilities, the rest of the model can be computed on-the-fly.
5. Sample traces of the adjusted model, ending each trace when it returns to the initial state, storing the likelihood ratio, total time, and time spent in unavailable (i.e., failed) states.
6. Average the total time $\overline{D}$ and unavailable time $\overline{Z}$ of the traces, multiplied by the likelihood ratios. Now $\overline{Z}/\overline{D}$ is the output estimated unavailability.

4.1 Reducing I/O-IMCs to Markov Chains

In most settings, I/O-IMCs are analyzed by computing the parallel composition of the full system, and analyzing this model using a standard model checker [1]. Our setting often produces models too large to compute the full parallel composition, so we use Monte Carlo simulation in which we can compute visited states on-the-fly.

Our technique requires that the (composed) I/O-IMC be reduced to a Markov Chain. We do so by performing the following steps for each visited state:

1. Collect the outgoing transitions from all component models from their current states.
2. Merge input and output transitions that synchronize, leaving the resulting transitions an output transition.
3. Remove any remaining input transitions, as they cannot be taken without the corresponding output transition.
4. Remove any non-Markovian transitions that produce a cycle.
5. If any non-Markovian transitions remain, take this transition and return to step 1 from the new state.

On the conclusion of this procedure, we have reached a state with only Markovian transitions, which can be used as an input for the Monte Carlo simulation.

This process also resolves any nondeterminism in the model, although it is undefined which transition is taken in nondeterministic states. In most reasonable DFT models, the only source of nondeterminism is the order in which gates fail when an element has multiple parents. Such nondeterminism is spurious, in that it has no effect on the outcome of the analysis. It is therefore acceptable to leave the exact resolution undefined.

In models where nondeterminism can actually affect the results of the analysis, our determinisation is clearly not desirable. We therefore apply our analysis only on DFTs in which a syntactic check rules out the possibility of non-spurious nondeterminism.

4.2 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. This way, we ensure that our semantics are identical to those used in the existing analysis. Fig. 4 shows the operation of DFTCalc and our FTRES tool. Before DFTCalc calls the CADP [8] tool to compute the composition, we use CADP to translate each element into a parsable model file. We then combine these models and apply the importance sampling algorithm to compute the unavailability of the model.

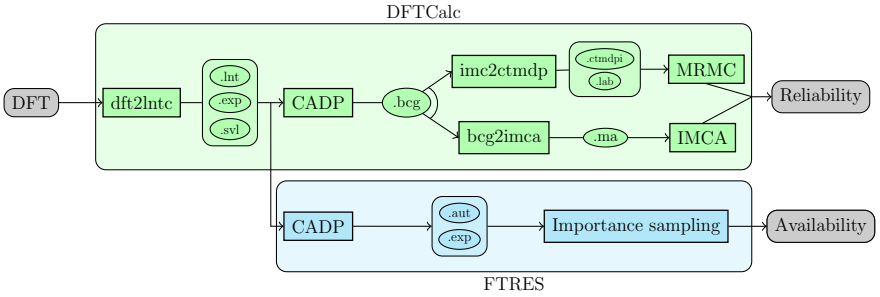

Fig. 4: The DFTCalc and FTRES tool-chains.

5 Case studies

To investigate the effectiveness our method, we compare FTRES to both a standard Monte Carlo simulator (MC) without importance sampling built into FTRES, and to the DFTCalc tool, which evaluates DFTs numerically via stochastic model checking. We analyze two case studies, for a number of different parameters. The first case is an industrial case study from railway signaling [9]. The second case models a fault-tolerant parallel processor (FTPP) and is a well-known benchmark from the literature [7].

Experimental setup. For each of the cases, we compute the unavailability (exact for DFTCalc, 95% confidence interval for FTRES and MC). We measure the time taken with a time-out of 48 hours, and the memory consumption in number of states (which is negligible for MC). For DFTCalc we measure both peak and final memory consumption. Simulations by FTRES and MC were performed for 10 minutes.

For both cases, we model the the failure times of the basic events via an Erlang distribution where the number of phases P is a parameter ranging from 1 to 3 phases; obviously, $P = 1$ corresponds to the exponential distribution.

All experiments were conducted on a dual 2.26 GHz Intel® Xeon® E5520 processor and 24 GB of RAM.

5.1 Railway cabinets

This case models a redundant system of relays and high-voltage cabinets used in railway signaling and provided by the railroad consultancy company Movares.

The FT, shown in Fig. 1, describes the relays and high voltage systems controlling a railway section. The relays are used to interface the electrically-powered systems such as switch motors with remote operating stations. They are also crucial for the safety of the trains, as they prevent multiple signals allowing trains onto already-occupied tracks, switches moving while trains are passing, and other safety violations. The high voltage cabinets provide power for local systems such as switches and signals.

We consider several variants of the FT for given parameter values. We augment the FT with a periodic inspection restoring any degraded basic events to perfect conditions. The time between executions of this action is governed by an Erlang distribution with two phases, and a mean time of half a year. We vary the number of cabinets in the system from 2 to 4.

Table 1 shows the results of the FTRES and DFTCalc and the MC tool. We note that, whenever DFTCalc is able to compute an analytic result, this result lies within the confidence interval computed by FTRES. We further see that the 2-phase models with 4 cabinets, and the 3-phase models with 3 or 4 cabinets could not be computed by DFTCalc within the time-out (times shown in Fig. 5), 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 larger confidence intervals. For the largest

Table 1: Comparison of the unavailabilities computed by DFTCalc, FTRES, and MC simulation for the case studies with N cabinets/processor groups.

	N	P	Unavailability DFTCalc	FTRES	MC
Railway cabinets	2	1	$4.25685 \cdot 10^{-4}$	$[4.256; 4.258] \cdot 10^{-4}$	$[4.239; 4.280] \cdot 10^{-4}$
	3	1	$7.71576 \cdot 10^{-4}$	$[7.713; 7.716] \cdot 10^{-4}$	$[7.694; 7.751] \cdot 10^{-4}$
	4	1	$1.99929 \cdot 10^{-3}$	$[1.998; 2.000] \cdot 10^{-3}$	$[1.999; 2.004] \cdot 10^{-4}$
	2	2	$4.55131 \cdot 10^{-8}$	$[4.548; 4.555] \cdot 10^{-8}$	$[1.632; 4.387] \cdot 10^{-8}$
	3	2	$6.86125 \cdot 10^{-8}$	$[6.846; 6.873] \cdot 10^{-8}$	$[0.673; 1.304] \cdot 10^{-7}$
	4	2	—	$[2.358; 2.394] \cdot 10^{-7}$	$[2.282; 3.484] \cdot 10^{-7}$
	2	3	$5.97575 \cdot 10^{-12}$	$[5.714; 6.252] \cdot 10^{-12}$	—
	3	3	—	$[5.724; 7.914] \cdot 10^{-12}$	—
	4	3	—	$[0.337; 1.871] \cdot 10^{-11}$	—
FTPP	1	1	$2.18303 \cdot 10^{-10}$	$[2.182; 2.184] \cdot 10^{-10}$	—
	2	1	$2.19861 \cdot 10^{-10}$	$[2.198; 2.199] \cdot 10^{-10}$	—
	3	1	$2.21420 \cdot 10^{-10}$	$[2.213; 2.215] \cdot 10^{-10}$	—
	4	1	$2.22979 \cdot 10^{-10}$	$[2.229; 2.230] \cdot 10^{-10}$	$[0; 2.140] \cdot 10^{-8}$
	1	2	$1.76174 \cdot 10^{-20}$	$[1.761; 1.763] \cdot 10^{-20}$	—
	2	2	$1.76178 \cdot 10^{-20}$	$[1.756; 1.770] \cdot 10^{-20}$	—
	3	2	—	$[1.673; 1.856] \cdot 10^{-20}$	—
	4	2	—	$[1.257; 2.553] \cdot 10^{-20}$	—

models, the MC simulator observed no failures at all, and thus computed an unavailability of 0.

Figure 6 shows the generated state spaces for both tools. Since FTRES only needs an explicit representation of the shortest paths to failure, it can operate in substantially less memory than DFTCalc. Although the final model computed by DFTCalc is smaller due to its bisimulation minimization, the intermediate models are often much larger. An interesting observation is that minimized state spaces for the $P = 1$ models are identical except for the transition rates. This is due to an implicit counting abstraction where the model only needs to track the number of failed cabinets of each type before a system failure, while all states after failure can be collapsed into one 'wait until repair' state.

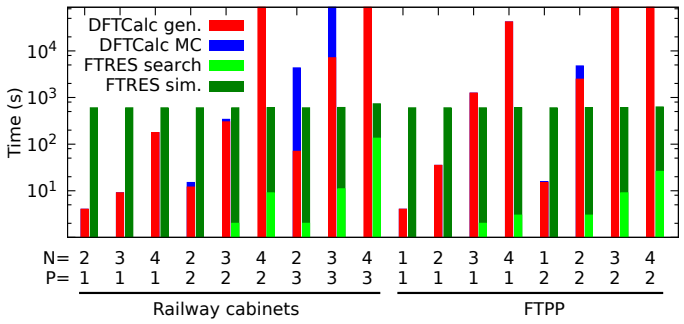

Fig. 5: Processing times for the different tools: Times for model generation and model checking for DFTCalc, and for the graph search and simulation for FTRES. Bars reaching the top of the graph reached the time-out.

5.2 Fault-tolerant parallel processor

The second case study is taken from the DFT literature [7], and concerns a fault-tolerant parallel computer system. This system consists of four groups of processors, labeled 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 system also has several workstations, each of which contains one processor of each group. A workstation normally uses processors A, B, and C. Processor S is used as a spare when one of the others fails. If more than one processor fails, the workstation is down.

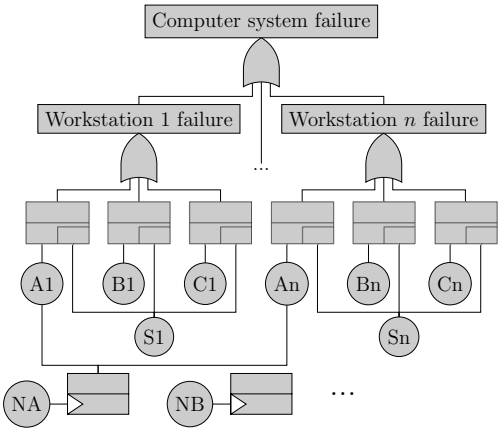

Fig. 7: DFT of the fault-tolerant parallel processor. Connections between the FDEP for B omitted for clarity, as well as the FDEPs for groups C and S.

Repairs are conducted by a periodic replacement which restores any degraded components to perfect condition. This replacement occurs at times following an Erlang distribution of four phases, with a mean time of 0.5 for each phase.

The numerical results and computation times for this case study can be found in Table 1 and Fig. 5 respectively. We note that the unavailability varies little when increasing the number of computer groups, as the dominant sources of failures are the network elements which do not increase with N. We again see that FTRES continues to perform well after DFTCalc runs out of time. We do see wider confidence intervals for the larger models, however the results remain

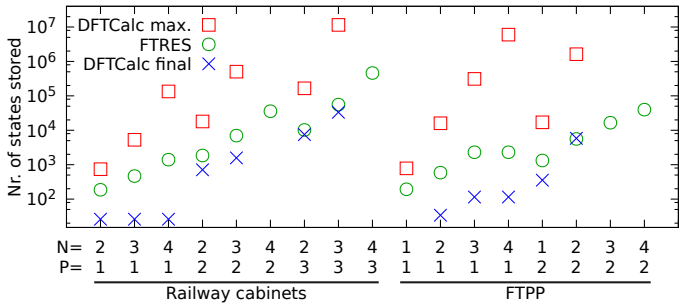

Fig. 6: Numbers of states stored in memory for the different cases with N cabinets/processor groups.

usable for practical purposes. The standard MC simulation observed no failures for most of the models.

Figure 6 lists the generated state spaces for both tools. Again, FTRES requires less peak memory than DFTCalc.

5.3 Conclusions on case studies

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 memory than DFTCalc in every case, and requires less time for large models, while still achieving high accuracy.
- can analyze models larger than DFTCalc can handle.
- gives substantially more accurate results than MC in similar processing time.

6 Conclusion

We have presented FTRES, an efficient and novel approach for rare-event simulation of dynamic fault trees through importance sampling. We follow the compositional semantics of Boudali et al. [4] providing flexibility and extensibility. Our use of the Path-ZVA [18] algorithm allows us to store only a small fraction of the state space, ameliorating the problem of the state space explosion.

We have demonstrated through two case studies that our approach has clear benefits over existing numerical tools, and tools without rare event simulation: We can analyze larger DFTs, produce results quicker and obtain narrow confidence intervals.

As future work, we intend to extend the tool to support non-spurious nondeterminism, allowing the analysis of the full space of DFTs.

Acknowledgments

This research has been partially funded by STW and ProRail under the project ArRangeer (grant 12238) with participation by Movares, STW under the project SEQUOIA (grant 15474), NWO under the project BEAT (grant 612001303), NWO under the project SamSam (grant 50918239) and the EU under the project grant SUCCESS (grant 651.002.001 / 1467).

References

1. Arnold, F., Belinfante, A., Freark van der Berg, D.G., Stoelinga, M.I.A.: DFTCalc: A Tool for Efficient Fault Tree Analysis. In: Proc. 32nd Int. Conf. Computer Safety, Reliability and Security (SAFECOMP). LNCS, vol. 8153, pp. 293–301 (2013)
2. Bobbio, A., Codetta-Raiteri, D.: Parametric fault trees with dynamic gates and repair boxes. In: Proc. 2004 Annu. IEEE Reliability and Maintainability Symp. (RAMS). pp. 459–465 (2004)
3. de Boer, P.T., L'Ecuyer, P., Rubino, G., Tuffin, B.: Estimating the probability of a rare event over a finite time horizon. In: Proc. 2007 Winter Simulation Conference. pp. 403–411. IEEE Press (2007)
4. Boudali, H., Crouzen, P., Stoelinga, M.I.A.: A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE Trans. Dependable Secure Comput. 7(2), 128–143 (2010)

5. Carrasco, J.A.: Failure transition distance-based importance sampling schemes for the simulation of repairable fault-tolerant computer systems. IEEE Trans. Rel. 55(2), 207–236 (2006)
6. Codetta-Raiteri, D., Franceschinis, G., Iacono, M., Vittorini, V.: Repairable fault tree for the automatic evaluation of repair policies. In: Proc. 2004 Annu. IEEE/IFIP Int. Conf. Dependable Systems and Networks (DSN). pp. 659–668 (2004)
7. Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Fault trees and sequence dependencies. In: Proc. 1990 Annu. IEEE Reliability and Maintainability Symp. (RAMS) (1990)
8. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. Int. J. Software Tools for Technology Transfer 15(2), 89–107 (Apr 2013)
9. Guck, D., Spel, J., Stoelinga, M.I.A.: DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper). In: Proc. 17th Int. Conf. Formal Engineering Methods (ICFEM). LNCS, vol. 9407, pp. 304–311 (Nov 2015)
10. Heidelberger, P.: Fast simulation of rare events in queueing and reliability models. ACM Trans. Modeling and Computer Simulation (TOMACS) 5(1), 43–85 (1995)
11. International Electrotechnical Commission: IEC 61025: Fault tree analysis (2006)
12. Kahn, H., Harris, T.: Estimation of particle transmission by random sampling. In: Monte Carlo method; Proc. Symp. held June 29, 30, and July 1, 1949. Nat. Bur. Standards Appl. Math. Series, vol. 12, pp. 27–30 (1951)
13. Kumamoto, H., Tanaka, K., Inoue, K., Henley, E.J.: Dagger-Sampling Monte Carlo for System Unavailability Evaluation. IEEE Trans. Rel. R-29(2), 122–125 (Jun 1980)
14. L'Ecuyer, P., Blanchet, J., Tuffin, B., Glynn, P.: Asymptotic robustness of estimators in rare-event simulation. ACM Trans. Modeling and Computer Simulation (TOMACS) 20(1), nr. 6 (2010)
15. L'Ecuyer, P., Tuffin, B.: Approximating zero-variance importance sampling in a reliability setting. Ann. Operations Research 189(1), 277–297 (2011)
16. Ramakrishnan, M.: Unavailability estimation of shutdown system of a fast reactor by Monte Carlo simulation. Ann. Nuclear Energy 90, 264–274 (Apr 2016)
17. Reijsbergen, D.P.: Efficient simulation techniques for stochastic model checking. Ph.D. thesis, University of Twente, Enschede (December 2013)
18. Reijsbergen, D.P., de Boer, P., Scheinhardt, W., Juneja, S.: Path-ZVA: general, efficient and automated importance sampling for highly reliable markovian systems. Submitted.
19. Ruijters, E., Guck, D., Drolenga, P., Stoelinga, M.I.A.: Fault maintenance trees: reliability contered maintenance via statistical model checking. In: Proc. IEEE 62nd Annu. Reliability and Maintainability Symposium (RAMS). IEEE (Jan 2016)
20. Ruijters, E., Stoelinga, M.I.A.: Fault tree Analysis: A survey of the state-of-the-art in modeling, analysis and tools. Computer Science Review 15–16, 29–62 (2015)
21. Shahabuddin, P.: Importance sampling for the simulation of highly reliable Markovian systems. Management Science pp. 333–352 (1994)
22. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault tree handbook. Office of Nuclear Regulatory Reasearch, U.S. Nuclear Regulatory Commision (1981)
23. Vesely, W.E., Narum, R.E.: PREP and KITT: computer codes for the automatic evaluation of a fault tree. Tech. rep., Idaho Nuclear Corp. (1970)