SAFEST: Fault Tree Analysis via Probabilistic Model Checking

Matthias Volk, University of Twente
Falak Sher, DGB Technologies
Joost-Pieter Katoen, RWTH Aachen University
Mariëlle Stoelinga, University of Twente

Key Words: dynamic fault trees, fault tree analysis, Markov models, model checking

SUMMARY & CONCLUSIONS

This paper presents *SAFEST*, a powerful tool for modelling and analyzing both static and dynamic fault trees. Dynamic fault trees (DFTs) extend standard fault trees by providing support for faithfully modelling spare management, functional dependencies, and order-dependent failures.

The *SAFEST* tool provides efficient and powerful analysis of DFTs via probabilistic model checking – a rigorous, automated analysis technique for probabilistic systems. The backbone of the analysis is based on efficient state space generation. Several optimization techniques are incorporated, such as exploiting irrelevant failures, symmetries, and independent modules. Probabilistic model checking allows to analyze the resulting state space with respect to a wide range of measures of interest. In addition, an approximation approach is provided that builds only parts of the state space and allows to iteratively refine the computations up to the desired accuracy.

The *SAFEST* tool provides a graphical user interface for creating, generating, simulating, and simplifying fault trees as well as visualizing the results from the fault tree analysis.

SAFEST is state of the art for DFT analysis, as demonstrated by an experimental evaluation and comparison with existing tools. In addition, *SAFEST* and DFT models have been applied in a variety of case studies, including vehicle guidance systems, train operations in railway station areas, and energy systems such as (nuclear) power plants.

1 INTRODUCTION

Probabilistic risk assessment (PRA) is of critical importance to ensure safe and reliable operation of today's systems in areas such as transportation, infrastructure, power generation, space exploration, etc. Fault trees [1], [2] are a popular model in PRA and are recommended by many standards and regulatory bodies.

While standard (or *static*) *fault trees (SFT)* are widely used and well supported by PRA tools, their modelling capabilities are limited. Several extensions to fault trees have been proposed over the years to overcome these limitations [2]. *Dynamic fault trees (DFT)* were introduced by Dugan [3] and are one of the most prominent extensions. DFTs introduce new gate types that allow for more faithful modelling by providing explicit support for spare management, functional dependencies, and order-dependent failures. DFT models have been successfully applied for e.g., aerospace systems [4], autonomous driving [5], railway engineering [6], [7], and analysis of spacecrafts [8] via the *COMPASS* toolset [9]. We refer to [10] for an overview.

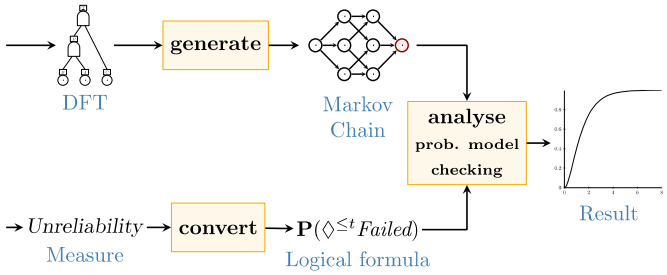

Figure 1: Overview of the DFT analysis approach based on probabilistic model checking (adapted from [31]).

Various analysis approaches for DFTs have been proposed, see [2], [11] for overviews. Example approaches include translations to Markov models [3], [12], [13], Bayesian networks [14], [15], and Petri nets [16], or via Monte Carlo simulation [17], [18]. While in particular Markov model analysis is supported by tools such as *Galileo* [19] and *DFTCalc* [20], mature and modern tools for DFTs are scarce.

We have developed *SAFEST*, a modern, state-of-the-art tool for modelling and analyzing both standard (static) and dynamic fault trees. *SAFEST* comes with a web-based graphical user interface that allows efficient modelling, visualization, simplification and interactive simulation of fault trees using a graphical editor. The analysis of dynamic fault trees is enabled via an efficient translation to Markov models and the use of state-of-the-art techniques from *probabilistic model checking*.

Model checking [21] is a rigorous technique for checking whether a given model satisfies a specification given as a logical formula. Model checking uses highly optimized techniques to efficiently analyze the state space. *Probabilistic model checking* [22] considers probabilistic systems that capture random behavior, such as Markov models. The approach uses tailored numerical algorithms and answers queries such as "what is the probability of a system failure

2024 Annual Reliability and Maintainability Symposium (RAMS) | 979-8-3503-0769-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/RAMS51492.2024.10457719

within a year?" or "what is the expected time to failure when the system has entered a degraded state?".

We apply probabilistic model checking to DFT analysis. The approach is illustrated in Figure 1. The input is a DFT and a measure of interest, for instance the *unreliability* or the *mean-time-to-failure (MTTF)*. From the DFT, a Markov chain is generated. The measure is converted to a logical formula. Both the Markov chain and the logical formula are fed into a probabilistic model checker which computes the results. The analysis via probabilistic model checking is supported by mature and efficient tool support [23], [24].

The use of probabilistic model checking has several advantages for the DFT analysis. First, model checking supports a wide range of complex logical formulas that can be checked out of the box. Thus, we use a one-for-all analysis approach instead of having to develop algorithms tailored to each type of measure. Second, our approach uses model checking as a black box. This means we can easily change the underlying analysis tools and directly incorporate and benefit from recent advances in model checking algorithms without changing the overall approach. Third, unlike approaches based on Monte Carlo simulation, probabilistic model checking yields exact results and, in particular, is agnostic to rare events.

The key innovation of our approach is an efficient generation of the state space to combat possible state space explosion. To this end, we developed several optimizations that exploit irrelevant failures of events, symmetric structures and independent modules in the DFT [13]. Furthermore, we incorporated efficient analysis techniques for static fault trees based on binary decision diagrams [25]. Finally, we developed an approximation approach based on only building the most relevant parts of the state space [13]. All of these techniques allow for efficient analysis of DFTs.

Our experimental evaluation shows that our tool significantly outperforms existing DFT analysis tools and can handle DFTs with several hundred elements. We have demonstrated the performance of our tool and the modelling capabilities of DFTs in several industrial case studies. Examples include DFT models for vehicle guidance systems and analyzing the impact of infrastructure failures on train operations in railway station areas.

SAFEST is available at www.safest.dgbtek.com.

2 DYNAMIC FAULT TREES

Fault trees [1] model how failures occur and propagate through a system. *Basic events (BE)* represent atomic components which can fail according to an associated probability distribution. Logical *gates* model how failures in the inputs (also called children) propagate upwards. A failure of the top-level event – the root of the tree – represents a failure of the modelled system.

Figure 2 depicts the node types present in fault trees. We summarize the different types in the following and refer to [26] for details on the semantics.

Basic events (BE) represent atomic components which are not further subdivided. The failure of a BE is governed by a probability distribution, most commonly an exponential distribution with a given failure rate.

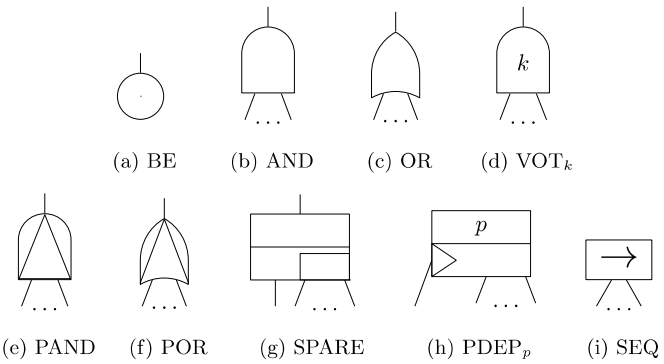

Figure 2: Node types in SFTs (top) and DFTs (bottom).

Static fault trees (SFT) support logical gates. The *AND-gate* fails if all inputs fail, the *OR-gate* fails if at least one input fail and the *voting gate* with threshold *k* fails if at least *k* inputs fail. As an example, consider the fault tree in Figure 3 which models a laptop computer. For example, the laptop fails if both *Processors*, i.e., *CPU1* and *CPU2*, fail.

Dynamic fault trees (DFT) introduce additional dynamic gates which extend the modelling capabilities.

Order-dependent failures are modelled with the *priority AND-gate (PAND)* and the *priority-OR-gate (POR)*. They fail if the inputs fail in order from left to right. If an input fails out of order, the priority gate becomes "*failsafe*" and cannot fail. In the laptop example, if first the *Backup* and then the *Installed OS* fails, the *OS* fails. However, if the *Installed OS* fails first, it can be restored from the *Backup* and the *OS* will not fail.

Spare management is modelled by the *SPARE-gate*. The SPARE-gate initially uses its first input – e.g., in our example, the *Power* is supplied by the *Plug*. Other inputs are dormant and can have a reduced failure rate (specified by a dormancy factor). If the currently used spare element fails, the SPARE-gate switches to the next spare and starts using it. In our example, the *Power* would switch from the *Plug* to the *Battery* if the *Plug* failed. A spare element can be available to multiple SPARE-gates, but only one SPARE-gate can end up using it.

Functional dependencies between components can be modelled by a *probabilistic dependency (PDEP)* with probability *p*. If the trigger of the PDEP fails, the failure is forwarded to the dependent events with probability *p*. In our example, a failure of the *Hard disk* triggers a failure of both the *Backup* and *Installed OS* through the dependency with probability one.

Restrictors allow only certain failure sequences. The *sequence enforcer-gate (SEQ)* ensures that failures of its inputs occur from left to right. This is different from a PAND-gate where out-of-order failures lead to "failsafe" but can still occur.

3 ANALYSIS VIA MODEL CHECKING

Our approach for DFT analysis is based on probabilistic model checking. An overview of the approach is given in Figure 1. The main bottleneck of the approach is the state space generation of the Markov chain. The state space corresponding to a DFT can grow exponentially in the worst case, leading to the common belief that state space-based techniques are

infeasible for DFTs, see e.g., [18]. Therefore, it is crucial to employ smart state space generation that keeps the resulting Markov chain as small as possible. To this end, we have developed several optimization techniques.

(1) Smart state space generation [13]. First, we exploit the fact that in (non-repairable) DFTs failures of certain events render other events irrelevant. For instance, in Figure 3, if the *Laptop* is failed due to a failure in *Processors*, subsequent failures of other components such as the *Power* are irrelevant and we *Don't Care* about them as they do not change the outcome. Not considering failures of Don't Care events can already drastically reduce the resulting state space. In addition, we exploit *symmetries* which are often present in fault trees, for example due to redundancies. In Figure 3, *CPU1* and *CPU2* are symmetric and we do not need to distinguish which CPU exactly failed. Such symmetry reduction can additionally reduce the resulting state space.

(2) Hybrid analysis via modularization [25]. We apply modularization [27] to individually analyze independent parts of a fault trees. Modularization enables efficient analysis of static and dynamic submodules with the techniques best suited for them. Static submodules are efficiently analyzed via binary decision diagrams (BDD) [28]. Dynamic submodules are analyzed via Markov models. Here, it suffices to generate the (smaller) Markov chain corresponding to a submodule.

(3) Approximation via partial state space generation [13]. As an alternative to generating the full state space, we have also developed an approximation approach based on only building the "most important" parts of the state space. The unexplored parts can then be assumed to be either all failed – providing a pessimistic bound – or to never fail – providing an optimistic bound. The approximation thus yields upper and lower bounds on the measure of interest and the exact result – which would be obtained by building the complete state space – is guaranteed to lie within these bounds. This hard guarantee is in contrast to, e.g., Monte Carlo simulation, where the computed result only holds with a given confidence. The bounds of the approximation approach can be iteratively refined up to a user-desired precision by exploring more parts of the state space. The exploration of the state space can be guided by various heuristic, such as the number of consecutive failures or the probability of reaching a state. The approximation approach combines the best of two worlds: it provides guaranteed bounds on the measure of interest, while being memory efficient by only building a small part of the entire state space.

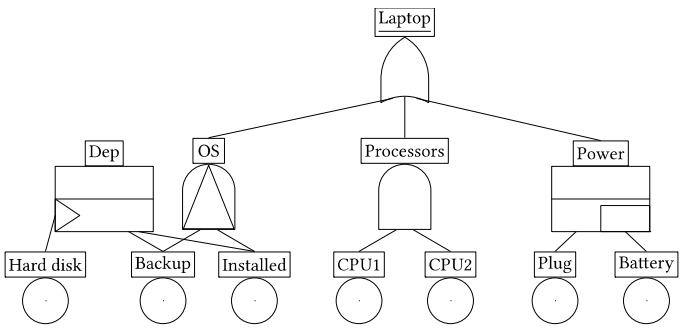

Figure 3: Dynamic fault tree model of an example laptop [31]

Implementation. The analysis approaches and optimization techniques are implemented in the probabilistic model checker *Storm* [24]. *Storm* is an open-source, state-of-the-art tool for the analysis of probabilistic systems and provides efficient algorithms for Markov chain analysis. We implemented the efficient state-space generation for DFTs in *Storm* and use *Storm*'s existing algorithms to analyze the resulting Markov chain with respect to a given logical formula.

4 THE SAFEST TOOL

We have developed a web-based graphical user interface for our DFT analysis algorithms called *SAFEST*. *SAFEST* supports both the modelling and analysis of static and dynamic fault trees.

4.1 *Modelling*

Graphical editor. SAFEST provides a web-based drag-and-drop editor for creating and editing fault trees, supporting all static (AND, OR, VOT) and dynamic (FDEP, PAND, POR, SEQ, SPARE) gates documented in the literature. Fault trees can be modelled hierarchically in the editor and divided into modules (independent sub-trees). Each fault tree can be exported in different formats (Galileo [19], JSON formats and LaTeX), simplified module by module, and reused. Fault trees can be displayed in graphical, tabular, and Galileo formats.

SysML import. Given SysML 2.0 models that have been annotated with safety data, e.g. functional dependencies and redundancies, *SAFEST* can automatically generate DFTs. This allows to perform reliability analysis of systems concurrently with their design and development, helping to explore the design space to identify the best design w.r.t. multiple metrics.

Parameters. SAFEST allows the specification of model attributes as parameters, which can be constants, real-valued expressions, or probability distributions. Examples include failure rates and dormancy factors for basic events. Different variants of fault trees can be constructed and compared at the time of analysis by simply altering the values of the parameters.

Failure distributions. For SFTs, *SAFEST* offers exponential, Erlang, Weibull, and log-normal distributions as failure distributions for basic events; for DFTs, it supports exponential and Erlang distributions. Failure distributions can be either (1) directly provided, (2) automatically constructed using statistical methods from input failure data of components, or (3) composed as a combination of distributions, i.e., the weighted average of other distributions.

Simulator. SAFEST comes with an interactive simulator to help better understand the fault tree models (see Figure 4). Users of the step-by-step simulator can choose which BE to fail next. The effect of the failure on the status of each fault tree element is then demonstrated and visualized. The interactive simulator helps identify modelling errors early on and gives users better insight into their models.

Simplification. SAFEST can automatically simplify the structure of a DFT – even with parameters – without affecting its behavior. The simplification, based on [29], modifies the DFT to make it more suitable for effective analysis.

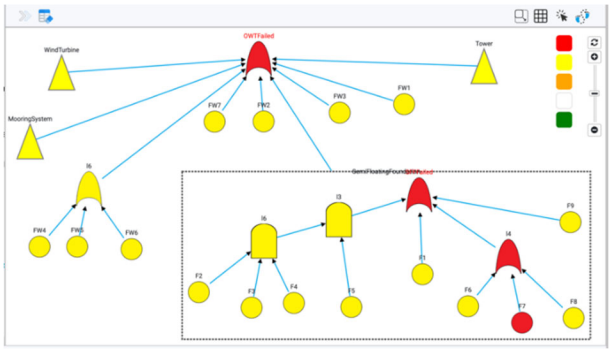

Figure 4: Interactive simulator in SAFEST.

4.2 *Analysis*

The fault tree analysis in *SAFEST* is based on the techniques presented in the previous Section 3.

Metrics. All key quantitative dependability metrics, such as system *reliability*, *mean-time-to-failure (MTTF)*, and component *criticality* based on importance, are supported out-of-the-box by *SAFEST*. Experienced users can specify their own specific measures of interest, even expressing complex metrics in mathematical logic.

Plots. To compare several DFT variants during the design phase, *SAFEST* allows to plot the results of the dependability metrics (see Figure 5). This also makes it possible to compare the criticality of different components of a system at different time points during the life cycle.

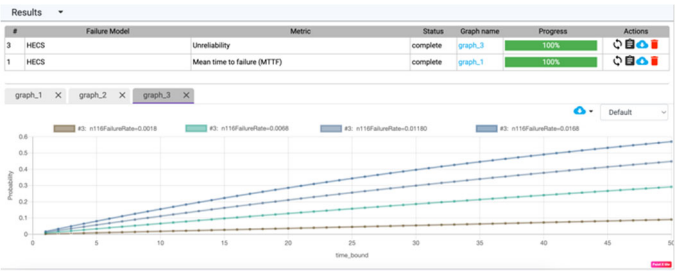

Figure 5: Comparison of multiple DFT variants in SAFEST.

5 *EVALUATION AND CASE STUDIES*

5.1 *Comparison to existing DFT tools*

We evaluated the analysis capabilities of our tool *SAFEST* on the DFTs from the *FFORT benchmark suite* [10]. Our evaluation considered 22 different fault trees in 174 variants and we computed both the unreliability and the MTTF. In total, we considered 369 benchmarks. The evaluation was performed on a single core with 2.1 GHz processor with 16 GB of memory. We compare our analysis approach with two existing tools: *DFTCalc* and *DFTRES*. *DFTCalc* [20] is based on compositional state space generation while *DFTRES* [17] performs rare event (Monte Carlo) simulation (with a relative error of 10^{-4}). We run *SAFEST* in several configurations: *Plain* without any optimizations, *DC* which exploits Don't Cares, *DC+Sym* which additionally exploits symmetries and the *Approximation* approach with an error bound of 10^{-4}.

Validation. The analysis results of *SAFEST* coincide with the results of the existing tools *DFTCalc* and *DFTRES*. Moreover, *SAFEST* employs model checking techniques which have been validated in extensive tool comparisons, cf. [30].

Runtimes. Figure 6 compares the runtime of the different tools in a quantile plot. A point *(x,y)* indicates that the *x* fastest benchmarks could be solved within at most *y* seconds each. First, we can see that even the plain approach in *SAFEST* without optimizations performs comparable to both existing tools *DFTCalc* and *DFTRES*. Only for larger DFTs the existing tools perform better. However, when the smart state space generation with Don't Care and exploiting symmetries is enabled, our approach performs significantly better. In fact, *SAFEST* (*DC+Sym*) solves more benchmarks in 1 second than both other tools in 1 hour. Thus, our smart state space generation is crucial for efficient DFT analysis. Finally, using the *approximation* approach again leads to a runtime improvement and allows to solve 82% of the benchmarks within the 1-hour time limit. A detailed performance evaluation can be found in [31].

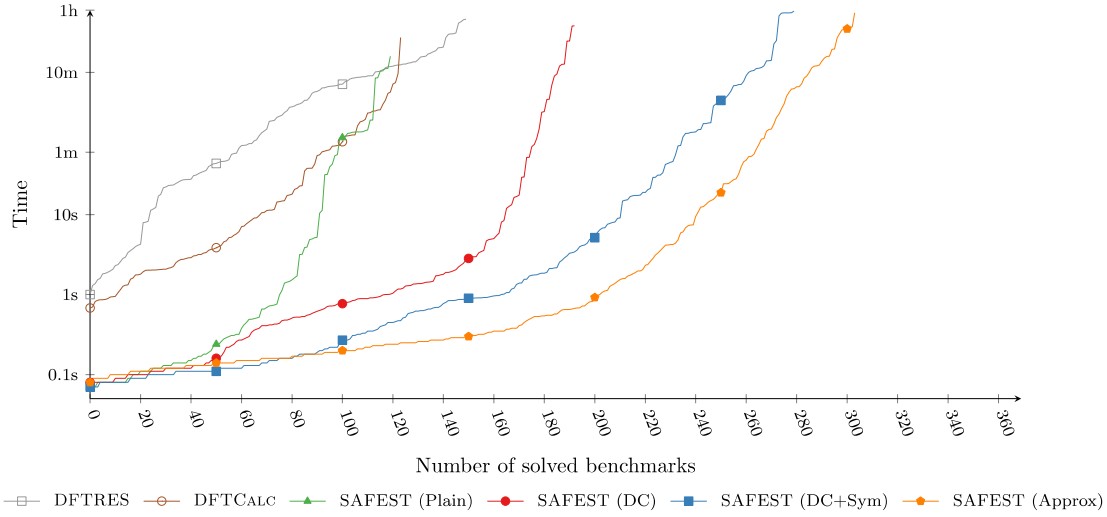

Figure 6: Comparison of different tools for DFT analysis (adapted from [31]). Closer to bottom right is better.

5.2 *Case studies*

We have demonstrated the benefit of using DFT models and the feasibility of our analysis approaches on a variety of industrial case studies. We refer to the FFORT benchmark collection [10] for additional DFT models.

We modelled a *vehicle guidance system* for autonomous driving as a DFT in [5]. We modelled different safety concepts and a corresponding partitioning of functions on the hardware. We analyzed these different variants with respect to various complex measures. In particular, we investigated the behavior after degradation, i.e., when some functionality has already failed. We compared the different variants in order to find the best partitioning. The DFT models consist of up to 300 elements and are, to the best of our knowledge, the largest real-world DFTs in the literature. Nevertheless, our analysis was able to compute results within minutes.

In [7], we modelled train routing options in *railway station areas* and linked these routes to the physical infrastructure elements (tracks, signals, switches, etc.) required for train operation. Our DFT models allowed us to investigate how train operations are disrupted by infrastructure failures. In particular, our analysis helped identify the most critical switches.

We have used our tool to model and analyze the emergency power supply in a *nuclear power plant* [32]. In [33], we showed the advantages of using DFT models on the example of *fire sprinkler systems* in shopping centers. Our tool has also been used to analyze *floating offshore wind turbines*, based on the DFT models from [34].

Recently, we modelled and analyzed a *Bipolar 12-Pulse Ultra High Voltage DC Transmission System,* based on [35]. The system has numerous redundant components, such as transformers, smoothing reactors, converter voltage groups, etc. The DFT has 252 BEs, 157 static and 64 dynamic gates, and is one of the largest DFTs to date. We ranked the criticality of components based on significance metrics – the Birnbaum importance index, criticality importance, risk achievement / reduction worth, and diagnostic importance factor.

6 *CONCLUSION*

We presented *SAFEST*, a modelling and analysis tool for static and dynamic fault trees. The graphical user interface enables the generation, construction, simulation and simplification of fault trees. The fault tree analysis is based on probabilistic model checking and efficient optimization techniques such as exploiting DFT structures, hybrid analysis through modularization and approximation via partial state space generation. The advantages of DFT models and the applicability of *SAFEST* have been demonstrated in a number of industrial case studies.

Future work. We aim to continuously improve the user interface based on the feedback from industrial practitioners. We plan to provide better integration into existing workflows by e.g., supporting import and export of common file formats.

REFERENCES

1. M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. Minarick, and J. Railsback, “Fault tree handbook with aerospace applications,” 2002.
2. E. Ruijters and M. Stoelinga, “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools,” *Comput. Sci. Rev.*, vol. 15–16, pp. 29–62, 2015.
3. J. B. Dugan, S. J. Bavuso, and M. A. Boyd, “Dynamic fault-tree models for fault-tolerant computer systems,” *IEEE Trans. Reliab.*, vol. 41, no. 3, pp. 363–377, 1992.
4. J. B. Dugan and T. Assaf, “Dynamic Fault Tree Analysis of a Reconfigurable Software System”, *Internat. System Safety Conference,* 2001.
5. M. Ghadhab, S. Junges, J.-P. Katoen, M. Kuntz, and M. Volk, “Safety analysis for vehicle guidance systems with dynamic fault trees,” *Reliab. Eng. Syst. Saf.*, vol. 186, pp. 37–50, 2019.
6. E. Ruijters, D. Guck, M. v Noort, and M. Stoelinga, “Reliability-Centered Maintenance of the Electrically Insulated Railway Joint via Fault Tree Analysis: A Practical Experience Report,” in *Internat. Conference on Dependable Systems and Networks (DSN)*, 2016, pp. 662–669.
7. N. Weik, M. Volk, J.-P. Katoen, and N. Nießen, “DFT modeling approach for operational risk assessment of railway infrastructure,” *Int. J. Softw. Tools Technol. Transf.*, vol. 24, no. 3, pp. 331–350, 2022.
8. M. Bozzano, A. Cimatti, J.-P. Katoen, P. Katsaros, K. Mokos, V.Y. Nguyen, T. Noll, B. Postma, and M. Roveri, “Spacecraft early design validation using formal methods,” *Reliab. Eng. Syst. Saf.*, vol. 132, pp. 20–35, 2014.
9. M. Bozzano, H. Bruintjes, A. Cimatti, J.-P. Katoen, T. Noll, and S. Tonetta, “COMPASS 3.0,” in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS, Springer, 2019, pp. 379–385.
10. E. Ruijters, C. E. Budde, M. C. Nakhaee, M.I.A. Stoelinga, D. Bucur, D. Hiemstra, and S. Schivo, “FFORT: A benchmark suite for fault tree analysis,” in *ESREL*, p. 8, 2019.
11. K. Aslansefat, S. Kabir, Y. Gheraibia, and Y. Papadopoulos, “Dynamic Fault Tree Analysis: State-of-the-Art in Modeling, Analysis, and Tools,” in *Reliability Management and Engineering*, CRC Press, 2020.
12. H. Boudali, P. Crouzen, and M. Stoelinga, “A Rigorous, Compositional, and Extensible Framework for Dynamic Fault Tree Analysis,” *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 2, pp. 128–143, 2010.
13. M. Volk, S. Junges, and J. P. Katoen, “Fast Dynamic Fault Tree Analysis by Model Checking Techniques,” *IEEE Trans. Ind. Inform.*, vol. 14, no. 1, pp. 370–379, 2018.
14. S. Montani, L. Portinale, A. Bobbio, and D. Codetta-Raiteri, “Radyban: A tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks,” *Reliab. Eng. Syst. Saf.*, vol. 93, no. 7, pp. 922–932, 2008.
15. H. Boudali and J. B. Dugan, “A new Bayesian network approach to solve dynamic fault trees,” in *Proc. of Reliability and Maintainability Symp.,* 2005, pp. 451–456.
16. A. Bobbio and D. C. Raiteri, “Parametric fault trees with dynamic gates and repair boxes,” in *Proc. of Reliability*

and Maintainability Symposium (RAMS), 2004, pp. 459–465.
17. C. E. Budde, E. Ruijters, and M. Stoelinga, "The Dynamic Fault Tree Rare Event Simulator," in *Quantitative Evaluation of Systems*, LNCS, Springer, 2020, pp. 233–238.
18. K. Durga Rao, V. Gopika, V. V. S. Sanyasi Rao, H. S. Kushwaha, A. K. Verma, and A. Srividya, "Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment," *Reliab. Eng. Syst. Saf.*, vol. 94, no. 4, pp. 872–883, 2009.
19. K. J. Sullivan, J. B. Dugan, and D. Coppit, "The Galileo fault tree analysis tool," in *Internat. Symposium on Fault-Tolerant Computing,* 1999, pp. 232–235.
20. F. Arnold, A. Belinfante, F. V. der Berg, D. Guck, and M. Stoelinga, "DFTCalc: A Tool for Efficient Fault Tree Analysis," in *Computer Safety, Reliability, and Security*, LNCS, no. 8153, Springer, 2013, pp. 293–301.
21. C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.
22. J.-P. Katoen, "The Probabilistic Model Checking Landscape," in *Proc. of the Symposium on Logic in Computer Science (LICS)*, ACM, 2016, pp. 31–45.
23. M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of Probabilistic Real-Time Systems," in *Computer Aided Verification (CAV)*, LNCS, Springer, 2011, pp. 585–591.
24. C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk, "The probabilistic model checker Storm," *Int J Softw Tools Technol Transf*, vol. 24, no. 4, pp. 589–610, 2022.
25. D. Basgöze, M. Volk, J.-P. Katoen, S. Khan, and M. Stoelinga, "BDDs Strike Back," in *NASA Formal Methods*, LNCS, Springer, 2022, pp. 713–732.
26. S. Junges, J.-P. Katoen, M. Stoelinga, and M. Volk, "One Net Fits All," in *Application and Theory of Petri Nets and Concurrency*, LNCS. Springer, 2018, pp. 272–293.
27. R. Gulati and J. B. Dugan, "A modular approach for analyzing static and dynamic fault trees," in *Proc. of Reliability and Maintainability Symp.,* 1997, pp. 57–63.
28. A. Rauzy, "New algorithms for fault trees analysis," *Reliab. Eng. Syst. Saf.*, vol. 40, no. 3, pp. 203–211, 1993.
29. S. Junges, D. Guck, J.-P. Katoen, A. Rensink, and M. Stoelinga, "Fault trees on a diet: automated reduction by graph rewriting," *Form. Asp. Comput.*, vol. 29, no. 4, pp. 651–703, 2017.
30. C. E. Budde, A. Hartmanns, M. Klauck, J. Křetínský, D. Parker, T. Quatmann, A. Turrini, and Z. Zhang, "On Correctness, Precision, and Performance in Quantitative Verification," in *Leveraging Applications of Formal Methods (ISoLA)*, LNCS, Springer, 2021, pp. 216–241.
31. M. Volk, "Dynamic fault trees: semantics, analysis and applications," PhD thesis, RWTH Aachen University, 2022.
32. S. Khan, J.-P. Katoen, M. Volk, and M. Bouissou, "Synergizing Reliability Modeling Languages: BDMPs without Repairs and DFTs," in *Pacific Rim Internat. Symp. on Dependable Computing (PRDC)*, 2019, pp. 266–275.
33. S. Khan, J.-P. Katoen, M. Volk, A. Zafar, and F. Sher, "Modelling and Analysis of Fire Sprinklers by Verifying Dynamic Fault Trees," in *Latin-American Symposium on Dependable Computing (LADC)*, 2021, pp. 1–10.
34. X. Zhang, L. Sun, H. Sun, Q. Guo, and X. Bai, "Floating offshore wind turbine reliability analysis based on system grading and dynamic FTA," *J. Wind Eng. Ind. Aerodyn.*, vol. 154, pp. 21–33, 2016.
35. K. Xie, B. Hu, and C. Singh, "Reliability Evaluation of Double 12-Pulse Ultra HVDC Transmission Systems," *IEEE Trans. Power Deliv.*, vol. 31, no. 1, pp. 210–218, 2016

BIOGRAPHIES

Dr. Matthias Volk
University of Twente, Formal Methods & Tools
Drienerlolaan 5, 7522 NB Enschede, The Netherlands

e-mail: m.volk@utwente.nl

Matthias Volk is a postdoctoral researcher at the University of Twente. He received his PhD degree from RWTH Aachen University. His research focuses on the analysis of safety-critical systems through formal methods, most notably (dynamic) fault tree analysis, and model checking of probabilistic systems.

Dr. Falak Sher
DGB Technologies, Formal Methods Group
476 James Way, Wyckoff, New Jersey 07481, USA

e-mail: chfalak@dgbtek.com

Falak Sher is a Formal Method Expert at DGB Technologies. He received his PhD from RWTH Aachen University and was a post-doctoral researcher at Fortiss GmbH, Germany. His areas of expertise include analysis of reactive, stochastic, real-time, and hybrid systems, and formal software verification.

Prof. Dr. Ir. Dr. h. c. Joost-Pieter Katoen
RWTH Aachen University, Software Modeling & Verification
Ahorstraße 55, 52056 Aachen, Germany

e-mail: katoen@cs.rwth-aachen.de

Joost-Pieter Katoen is a Distinguished Professor with RWTH Aachen University and holds a part-time professorship at the University of Twente. He received the honorary doctorate degree from Aalborg University, Denmark. His research interests include formal methods, model checking, concurrency theory, and probabilistic computation. He coauthored the book on "Principles of Model Checking." Prof. Katoen is an ACM Fellow and a member of Academia Europaea. He holds an highly remunerated ERC Advanced Grant.

Prof. Dr. Mariëlle Stoelinga
University of Twente, Formal Methods & Tools
Drienerlolaan 5, 7522 NB Enschede, The Netherlands

e-mail: m.i.a.stoelinga@utwente.

Mariëlle Stoelinga is a full professor of risk management for

high-tech systems, both at the University of Twente and Radboud University Nijmegen, the Netherlands. She leads a research team that develops quantitative risk assessments methods. Her special interests are predictive maintenance, human factors and the interplay between safety and security. She is the programme leader of executive Master on Risk Management, holds a prestigious ERC consolidator grant on the integration of Safety and Security risks, and leads a large Dutch consortium on Predictive Maintenance.