# DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper)

Dennis Guck[1], Jip Spel[1] and Mariëlle Stoelinga[1]

[1] Formal Methods and Tools, University of Twente, The Netherlands
d.guck@utwente.nl, j.j.spel@student.utwente.nl, m.i.a.stoelinga@utwente.nl

**Abstract.** Reliability, availability, maintenance and safety (RAMS) analysis is essential in the evaluation of safety critical systems like nuclear power plants and the railway infrastructure. A widely used methodology within RAMS analysis are fault trees, representing failure propagations throughout a system. We present DFTCalc, a tool-set to conduct quantitative analysis on dynamic fault trees including the effect of a maintenance strategy on the system dependability.

**Keywords:** Dynamic fault trees, maintenance, reliability, context-dependent reduction

## 1 Introduction

Maintenance is a crucial aspect in reliability engineering: good maintenance, consisting of timely inspections, spare management, renewals and repairs, reduces the number of failures and extends the system life time. The trend in maintenance is reliability-centered. To achieve higher reliability and reduce costs, it is commonly agreed that essential components should be maintained more intensively than less crucial ones, rather than the usual practice of subjecting all components the same maintenance regime. Challenge here is to identify the crucial components and determine the optimal maintenance strategy. The tool DFTCalc provides important support here: given an advanced maintenance strategy and a system model given as a fault tree, DFTCalc computes standard reliability measures like the system reliability, availability, and mean time to failures.

Technically, DFTCalc is realized via stochastic model checking of interactive Markov chains, yielding a flexible and efficient framework by exploiting state space generation via bisimulation minimisation. A first version of DFTCalc was reported in [1] concerning fault tree analysis only. This paper reports the extensions of DFTCalc with preventive and corrective maintenance models and their analysis. To handle the additional complexity, we have implemented context-dependent model-generation, which significantly reduces the state space. We show the application of DFTCalc on standard case studies from the literature, as well as industrial cases from railway engineering.

**Paper organization.** Section 2 introduces fault trees and maintenance, Section 3 their analysis in DFTCalc, Section 4 the case studies, and Section 5 concludes the paper.

## 2 Fault trees and maintenance

**Fault trees.** Fault trees are a popular graphical method for RAMS (reliability, availability, maintenance, and security) analysis [14]. A fault tree (FT) is a tree (or rather a directed acyclic graph, since sub-trees can be shared) describing how component failures propagate through a system and may lead to system failures. The FT leaves represent component failures, called *basic events* (BEs), and are equipped with probability

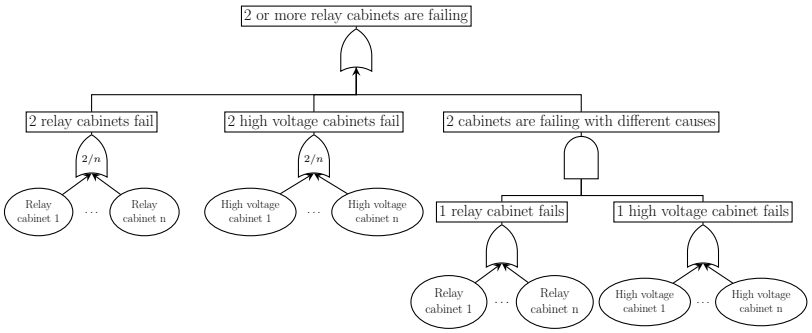



**Fig. 1.** Example FT of relay cabinet failures.

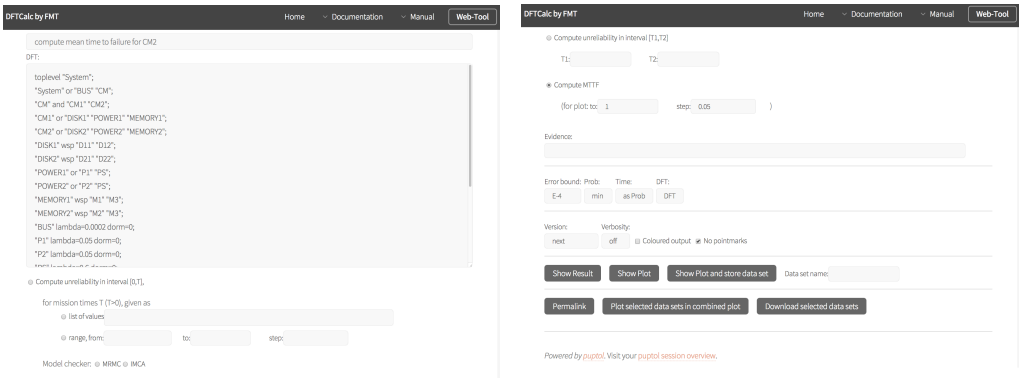
distributions modelling the component's failure behaviour over time. Failure times are often modelled as exponential probability distributions; that is, the probability that the component fails within time $t$ is given by $\mathbf{P}[X < t] = 1 - e^{-\lambda t}$. Here, the parameter $\lambda \in \mathbb{R}^+$ is known as the component's *failure rate*. Additionally, leaves are given a dormancy factor $\alpha \in [0, 1]$ that reduces the failure rate of a component when dormant, i.e. not in use. Thus, the probability for dormant component to fail within time $t$ is given by $\mathbf{P}[X < t] = 1 - e^{-\alpha\lambda t}$. Apart from exponential distributions, DFTCalc supports phase-types, i.e., probability distributions given by absorption times in Markov chains, which can be used to approximate any probability distributions with arbitrary precision.

The FT *gates* model failure propagation. The static gates OR, AND, VOT($k$) model respectively that a gate fails if one, all, or $k$ of their inputs fail. The dynamic gates PAND, SPARE, FDEP provide support for common reliability patterns like sequencing, spare management and functional dependencies, and are known as dynamic fault trees (DFTs). Their behaviour is as follows: A PAND-gate fails if the inputs fail from left to right, otherwise no failure occurs. A SPARE-gate contains a primary input, and one or more spare inputs. If the primary input fails, then a spare gets activated and takes over its functionality. If all spares have failed as well, then the SPARE-gate fails. An FDEP-gate contains a trigger input, which triggers the failure of all its dependent events.

A wide variety of qualitative and quantitative DFT analysis techniques are available, see [12] for an overview. Qualitative techniques include cut sets and cut sequences; quantitative techniques compute important system measures like the system *reliability* (What is the probability that the system fails during its mission time $T$?); the *availability* (What is the percentage of time that the system is up in the long run?), and *mean time to first failure* (What is the expected time until the first failure occurs?).

*Example 1.* The fault tree in Fig. 1 describes an instance of the failure behaviour of a redundant relay cabinet system, used on an operated railway track [7]. It has been provided by the RAMS consultant Movares. The top level OR-gate describes the disruption of several relays on an operated track. The system fails, if there are at least 2 relay or high voltage cabinet failures, as modelled by the VOT(2)-gates. Besides, the system can also fail if a combination of one relay and high voltage cabinet failure occurs, as modelled by the AND-gate.

**Maintenance.** Maintenance comprises a combination of inspections, repairs, renewals and spare management. Two types are distinguished: *preventive maintenance* refers to actions that prevent failures. Components are inspected, and based on their condition, (partial) renewals or repairs are performed, putting the component in a better condition. Preventive maintenance can be further divided into condition-based (e.g., the replacement of car tires when their profile is too low) and usage-based maintenance (e.g.




(a) DFT input interface. (b) Dependability measures interface.

**Fig. 2.** DFTCalc web interface.

inspection every 10.000 car miles). *Corrective maintenance* is carried out after a failure has occurred, replacing or repairing the broken component. Corrective maintenance may trigger preventive maintenance. For example, when a broken train is in the garage, additional inspections commonly take place. Regular fault trees, whether static or dynamic, do not incorporate such maintenance strategies: the component failure rates assume a certain maintenance regime and once a BE fails, it remains failed.

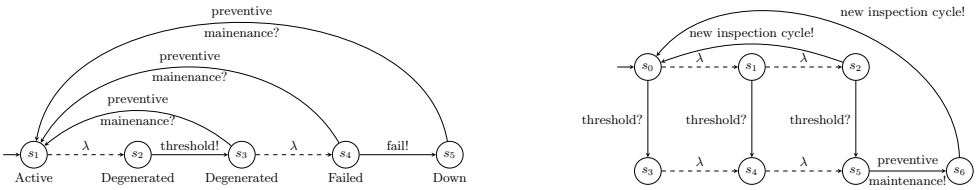
## 3 DFTCalc

DFTCalc provides efficient tool support for quantitative analysis of dynamic fault trees and is available at `http://fmt.ewi.utwente.nl/puptol/dftcalc` via a web interface, depicted in Fig. 2. The tool takes as inputs a DFT in Galileo format [13] and a maintenance model. It computes the most common dependability metrics, being the system reliability over a time interval $[T_1, T_2]$, the system availability, and the mean time to first failure, which are outputted textually as well as graphically. Technically, DFTCalc is realized via stochastic model checking, via a compositional translation of each DFT element to an input/output interactive Markov chain where the dependability metrics are internally expressed as CSL formulas.

**Maintenance in DFTCalc.** We include maintenance in the FT framework by redefining the BEs behaviour. We handle condition-based maintenance, inspections, repairs, and spare management.

The *condition* of a component is modelled by different degeneration phases, similar to extended fault trees [5]. In Fig. 3(a), the first phase $s_1$ represents the component in perfect condition; subsequent phases represent degraded conditions, until the component is broken in $s_n$. A threshold is given after which an inspection will trigger a maintenance procedure. *Inspections* are modelled by an inspection module (IM), which handles several BEs, see Fig. 3(b). Thus, if the IM inspects a BE and applies a maintenance procedure, then the BE is set back to a less degraded mode — in Fig. 3(a) to its initial condition. We handle *repairs* via repair units (RUs) as presented in [7]. A RU caters for several BEs, and determines in which order the BEs are getting repaired. Further, the BE gets a repair time assigned which is described by an exponential distributed delay.

**Analysis.** As formal semantics of DFTs, and thus the basis for the quantitative analysis, input/output interactive Markov chains (I/O-IMCs) are used. I/O-IMCs are an extension of interactive Markov chains (IMCs) [9] by adding input and output signals to the




(a) Basic component with inspection threshold. (b) Inspection module.

**Fig. 3.** Inspection modules as I/O-IMCs.

action set. An I/O-IMC consists of a number of states which are connected via two types of transitions, interactive and Markovian. The interactive transitions are labelled with signals, which are used to communicate between components. The Markovian transitions are labelled with rates $\lambda$ representing an exponential distributed delay.

**State space generation** DFTCalc implements the *compositional aggregation* approach presented in [2]. Compositional aggregation provides a method to translate a DFT into an I/O-IMC while keeping the state space small. First, each element of the DFT is translated into an I/O-IMC. Then the obtained I/O-IMCs are iteratively composed and minimised until a single I/O-IMC remains.

This approach enables us to define I/O-IMCs for corrective and preventive maintenance. The repairable BEs as well as the RU are equivalent to the models presented in [7]. The IM as well as the new BEs for preventive maintenance are depicted in Fig. 3.

**Context-dependent state space generation.** The state space generation by compositional aggregation provides already a scalability of several orders of magnitude [2]. However, for large industrial case studies there is a demand for even more reduction. Therefore, we investigate context-dependencies in the component translation from DFT modules to I/O-IMCs. Instead of translating a DFT element directly into an I/O-IMC based on the semantics from [2], we differentiate between active and inactive elements and eliminate superfluous signals beforehand. Consider an AND-gate with two inputs. In the standard semantics, the full behaviour will be described, including the inactive behaviour of the components. However, if the component is active from the start, all the inactive behaviour can be discarded, which reduces the state space of the component.

**Implementation.** DFTCalc combines several state-of-the-art model checkers to provide a DFT analysis tool, see Fig. 4. The generation of the DFT, including the compositional aggregation, is done using the CADP tool set [6]. The generated I/O-IMC can be translated to the Markov Reward Model Checker (MRMC) [10], or to the Interactive Markov Chain Analyzer (IMCA) [8]. Finally, the requested dependability metrics, which are (a) the reliability for one or more mission times $T$, or (b) the probability on a system failure during an interval $[T_1, T_2]$, or (c) the mean time to failure, can be computed. A complete description of DFTCalc can be found in [1].

We exploit the modular framework of DFTCalc and provide new templates for the IM, RU, and BEs with phases, inspection signals, repair functionality and context-

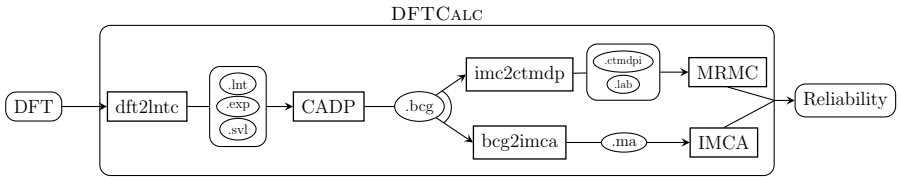
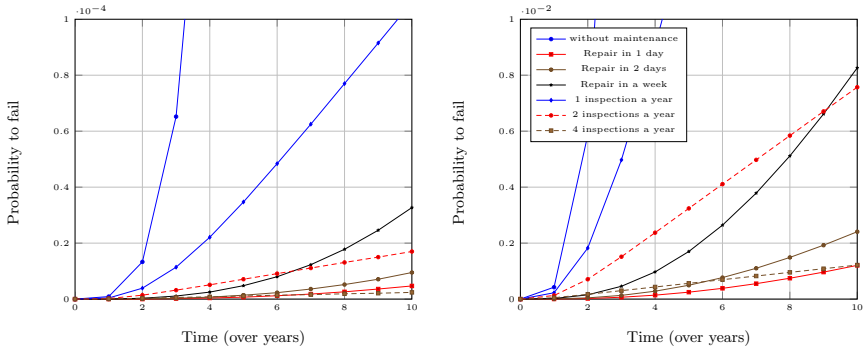



**Fig. 4.** The DFTCalc tool-chain.




(a) Relay cabinets (b) Switches

**Fig. 5.** System reliability over time for different inspection intervals.

dependencies. Further, we adapted the generation of those FTs in the `dft2lntc` tool. The inspection and repair functionality is implemented for static FTs, with AND- OR- and VOT($k$)-gates. The context-dependent behaviour detection based on the active and inactive modes works for DFTs without maintenance. The RU is defined by the new keyword `ru`, and the corresponding repair time of a BE is specified with `repair=`$\mu$, where $\mu$ is the repair rate. The IM has the keyword `KinspN` where `K` is defined as the number of inspection phases and `N` the rate of each phase. The threshold in the BE is specified with `interval=`$n$, where $n$ is the threshold. Further, each BE has a keyword `phases=`$k$, where $k$ represents the number of degeneration phases.

## 4 Experiments

We have conducted several case studies to demonstrate the applicability of DFTCalc; all were run on a single core of a 2 GHz Intel Xeon with 22GB RAM running on Linux.

**Modelling of maintenance strategies.** Fig. 5 shows the effect of different maintenance strategies on a set of FTs constructed by the RAMS consultancy firm Movares, concerning a part of a major railway corridor in the Netherlands [7]. We consider two systems, the relay cabinet whose abstract version is given in Fig. 1, and a railway switch. To investigate the effect of corrective and preventive maintenance, each group of cabinets is assigned to either a RU or a IM, following the following strategies: (a) without maintenance; (b) corrective maintenance with repair times of one, two and seven days; (c) preventive maintenance with inspection frequencies of once, twice, and four times a year. We calculated the system reliability for a mission time of 10 years. The results depicted in Fig. 5 show that increasing the inspection frequency significantly improves the reliability. Lowering the repair times helps as well, but with less effect.

**Analysis efficiency.** Table 1 shows the impact of context-dependent state-space generation. Here, we use standard DFTs from literature: the multiprocessor computing system (MCS) [11], the cardiac assist system (CAS) [3], the fault-tolerant parallel processor (FTPP) [2], cascaded PAND system (CPS) [2] and an instance of a Sensorfilter (SF) [4] case study. The results are shown in Table 1. While the final state spaces are small, it is the size of the largest intermediate models that matter, since these determine the amount of memory required. We observe that the maximal state space reduction for the intermediate state space during generation lies between 72% and 96%, and the state space reduction for the final state space lies between 3% and 33%. This points the significance of distinguishing the active and inactive DFT part beforehand.

| Model | Gates | BEs | Smart detection | States | Transitions | Max States | Max transitions |
|---|---|---|---|---|---|---|---|
| CAS | 10 | 10 | without | 16 | 36 | 84 | 304 |
| | | | with | 14 | 34 | 49 | 133 |
| MCS | 10 | 11 | without | 18 | 37 | 6438 | 32202 |
| | | | with | 12 | 31 | 220 | 803 |
| FTPP-4 | 21 | 20 | without | 72 | 312 | 45823 | 230596 |
| | | | with | 66 | 306 | 7020 | 32200 |
| CPS | 5 | 12 | without | 39 | 71 | 918 | 3140 |
| | | | with | 38 | 70 | 134 | 291 |
| SF | 6 | 7 | without | 15 | 36 | 383 | 1500 |
| | | | with | 14 | 35 | 64 | 138 |

**Table 1.** Context-dependent state space reductions.

## 5 Conclusions and future work

This paper presented an extension of DFTCalc with preventive and corrective maintenance as well as a way to reduce the state space w.r.t. context-dependent reductions. We believe that the context-dependent generation will also have a high impact on the state space of DFTs with maintenance. Further, future work is needed to apply maintenance modules to the dynamic gates as well as to incorporate costs into the framework, to optimise maintenance strategies w.r.t. reliability as well as costs.

**Acknowledgement.** This work has been supported by the STW-ProRail partnership program ExploRail under the project ArRangeer (12238). We acknowledge our cooperation with Movares in the ArRangeer project.

## References

1. F. Arnold, A. Belinfante, F. Van der Berg, D. Guck, and M. Stoelinga. DFTCalc: a tool for efficient fault tree analysis. In *Computer Safety, Reliability and Security (SAFECOMP)*, volume 8153 of *LNCS*, pages 293–301. Springer, 2013.
2. H. Boudali, P. Crouzen, and M. Stoelinga. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. *IEEE Transactions on Dependable and Secure Computing*, 7:128–143, 2010.
3. H. Boudali and J. B. Dugan. A Bayesian network reliability modeling and analysis framework. *IEEE Transactions on Reliability*, 55:86–97, 2005.
4. M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri. Safety, dependability and performance analysis of extended AADL models. *The Computer Journal*, 54:754–775, 2011.
5. K. Buchacker. Modeling with extended fault trees. In *Proc. of the 5th Int. Symp. on High Assurance Systems Engineering (HASE)*, pages 238–246, Nov. 2000.
6. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: A toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2012.
7. D. Guck, J. P. Katoen, M. I. A. Stoelinga, T. Luiten, and J. Romijn. Smart railroad maintenance engineering with stochastic model checking. In *Railway Technology: Research, Development and Maintenance*, volume 104 of *Civil-Comp*, page 299, 2014.
8. D. Guck, M. Timmer, H. Hatefi, E. J. J. Ruijters, and M. I. A. Stoelinga. Modelling and analysis of Markov reward automata. In *Automated Technology for Verification and Analysis (ATVA)*, volume 8837 of *LNCS*, pages 168–184. Springer, 2014.
9. H. Hermanns. *Interactive Markov Chains: And the Quest for Quantified Quality*. Springer-Verlag, Berlin, Heidelberg, 2002.
10. J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Perf. Eval.*, 68(2):90–104, 2011.
11. S. Montani, L. Portinale, A. Bobbio, M. Varesio, and D. Codetta-Raiteri. A tool for automatically translating dynamic fault trees into dynamic Bayesian networks. In *RAMS*, pages 434–441, 2006.
12. E. J. J. Ruijters and M. I. A. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Elsevier Computing Surveys*, 2015.
13. K. J. Sullivan, J. Bechta Dugan, and D. Coppit. The Galileo fault tree analysis tool. In *29th Annual Int. Symp. on Fault-Tolerant Computing*, pages 232–235. IEEE, 1999.
14. W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. Office of Nuclear Regulatory Reasearch, U.S. Nuclear Regulatory Commision, 1981.