A Modular Approach to Non-Deterministic Dynamic Fault Trees

Sascha Müller[1][0000−0002−1913−1719], Adeline Jordon[1][0000−0003−0796−6775], Andreas Gerndt[1,2][0000−0002−0409−8573], and Thomas Noll[3][0000−0002−1865−1798]

[1] Institute for Software Technology
DLR (German Aerospace Center), 38108 Braunschweig, Germany
{Sa.Mueller, Adeline.Jordon, Andreas.Gerndt}@dlr.de
[2] University of Bremen, 8334 Bremen, Germany
[3] Software Modeling and Verification Group
RWTH Aachen University, 52056 Aachen, Germany
Noll@cs.rwth-aachen.de

Abstract. Dynamic Fault Trees (DFTs) are powerful tools for deriving fault-tolerant system designs. However, deterministic approaches to DFTs suffer from semantic struggles with problems such as spare races. In this paper, we discuss the added complexity in the state-space representation of a non-deterministic DFT model and propose a modularized approach for synthesizing recovery automata. Finally, we give an implementation and evaluate it on the Fault tree FOResT (FFORT) benchmark. The results show that non-deterministic semantics with modularization can scale for literature case studies.

Keywords: FDIR · Reliability Engineering · Fault Tree Analysis · Synthesis · Formal Methods.

1 Introduction

Radiation, limited room for human intervention under only partial knowledge, lacking the ability to replace broken hardware – space systems confront reliability engineers with many challenges. They have to ensure that spacecraft can, to a certain degree, continue operation even in the presence of faults. The ability of a system to do so is often measured by Reliability, Availability, Maintainability, and Safety (RAMS) metrics. Fault Detection, Isolation, and Recovery (FDIR) concepts aim to increase these RAMS metrics. In order to derive these concepts and evaluate them, reliability engineers employ Fault Tree Analysis (FTA) [8]. A Fault Tree (FT) is a graphical failure model describing how low-level faults propagate through a system and eventually become a system-wide failure. To strengthen the expressive power of FTs, they were later extended to Dynamic Fault Trees (DFTs), which introduce various features such as temporal dependencies and spare management. However, these DFTs give rise to non-deterministic behavior such as spare races. In these spare races, multiple resources compete simultaneously for a spare, but have no unique semantic resolution.

A methodology presented in [14] aims to overcome this shortcoming. It introduces Non-Deterministic Dynamic Fault Trees (NdDFTs), a non-deterministic extension of DFTs, that drops the inherently rigid rules on how spares should be employed. The methodology foresees transforming NdDFTs to Markov Automata (MA), computing an optimized scheduler for a given objective metric, and then extracting the recovery strategy from said scheduler. This process is referred to as recovery automaton synthesis.

However, as the technique constructs a monolithic state-space representation of the NdDFT, containing an encoding of all possible recovery actions, it suffers severely from classical state-space explosion problems. When employing the technique for industrial benchmarks, we experienced that its naive usage makes it unsuitable for real life applications.

The main contribution of this paper lies in tackling this weakness. For that purpose, we adapt established modularization techniques from the DFT realm. These techniques were originally designed to compute reliability metrics in a compositional way. We transfer them to establish a modular workflow for performing recovery automaton synthesis.

The remainder of the paper is structured as follows. Section 2 gives an overview of other approaches that address non-deterministic semantics in the context of DFTs. Technical background knowledge on the relevant fault tree models is given in Section 3. Section 4 extends the synthesis workflow to integrate modularization. Section 5 investigates the scalability of the approach. Finally, the paper concludes in Section 6, and gives further directions to future work.

2 Related Work

The problem cases induced by the rigid standard fault tree semantics have been considered in other works. The authors of [9] tackle the issue of spare races by employing non-determinism in the propagation semantics of functional dependency gates, while allowing only functional dependencies to cause spare races. The study of the interaction of this approach with spare gates reveals that there are various different, yet sensible, ways in which the resulting semantics can be interpreted. They conclude that there is no "correct" one-fits-all interpretation, and that the fitting variant has to be chosen on a case-by-case basis. This is a concern regarding the applicability of fault trees, as experts in system design are not necessarily experts in fault tree semantics.

The work by the authors of [2] is worthy of particular mention. They apply an approach for converting static, non-deterministic fault tree models, extendible with so-called repair boxes, into Markov decision processes. Resolving the non-determinism gives them an optimal repair strategy. However, the approach does not extend to dynamic gates, which the authors mention as foreseeable future work.

The authors of [18] introduce the concept of fault maintenance trees, which are based on non-deterministic Input-Output Interactive Markov Chains (I/O-IMCs). The semantics are defined compositionally by starting with elementary I/O-IMCs for every gate and by proceeding in bottom-up direction by combining these elementary I/O-IMCs. Recovery strategies for resolving spare races can also be given on the Markovian level. The strategies have to be chosen manually and are then compared to each other using a testing-based approach based on model simulation. In contrast, the NdDFT model employed in this work defines recovery strategies on a higher level and allows them to be computed by resolving the non-determinism on the Markovian level with respect to an optimality criterion.

Also with regard to modularization techniques, a number of techniques to support compositional analysis of DFTs has been developed. For identifying minimal cut sets, [5] provides an algorithm for finding independent sub-modules of FTs, which can be converted separately to Binary Decision Diagrams (BDDs) and then be analyzed, reducing the computational requirements for handling the entire tree. Our approach to modularization as described in Section 4.2 is based on this work.

For efficiently calculating the reliability of a DFT, [3] provides a compositional semantics for DFT in terms of IMCs, which reduces the combinatorial explosion in many common cases. Moreover, [16] develops an approach to divide a DFT into independent sub-modules for computing reliability. Sub-modules containing only static gates can then be solved using a traditional BDD method, while sub-modules containing dynamic gates can be solved using Markov Chain analysis. The method presented in [7] also modularizes a DFT and uses BDDs for the static sub-modules, but employs the approximation from [1] to solve the dynamic sub-modules. This avoids the state-space explosion problem incurred by conversion to Markov Chains, while retaining a reasonable degree of accuracy. Based on this work, [11] proposes a method to modularize DFTs further, by also collapsing static sub-trees of a dynamic gate, but keeping additional information about the probability distribution of these sub-trees. Finally, [20] provides additional modularization techniques, which can convert static sub-trees and some dynamic sub-trees into equivalent basic events, thus reducing the complexity of further analysis.

3 Background

3.1 Fault Trees

Fault trees are failure propagation models that express how faults start out on low-level components, propagate through the system by combinatorial means, and eventually turn into a high-level, system wide failure. Syntactically, fault trees are directed acyclic graphs with two types of nodes: events and gates. The leaves of a fault tree are called basic events, and the root node is referred to as top-level event. Usually, a basic event is also equipped with a failure rate. In this work, we allow for one additional extension: A basic event (BE) can be equipped either with a failure rate, for describing exponentially distributed behavior, or with a failure probability for instantaneous, uniformly distributed behavior. When a basic event with a uniform distribution is activated, it can fire with its assigned probability. Such an activation occurs at system start and may

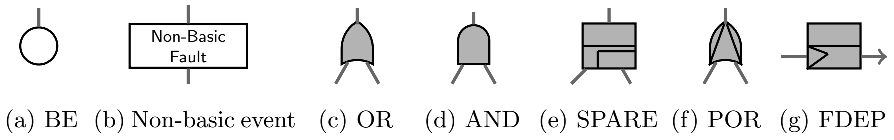


Fig. 1. Relevant gates and events of a fault tree

be triggered again by a gate. We limit ourselves here to the case of permanent failure, i.e., once a BE has failed, it remains in a failed state for all future points in time.

The gates model logical re-combinations of faults, as they propagate through the system. The simplest type of fault tree model, called static fault tree, considers basic logical gate types such as AND and OR. Dynamic Fault Trees (DFT) go further and introduce new gate types enabling various features such as spare management and temporal constraints.

We give a short overview of gates relevant to this paper. Their fault tree notation, together with the notation of events, is depicted in Fig. 1. The OR and AND gates behave as classical, logical gates. If at least one input fails, the OR gate propagates. If all inputs fail, the AND gate propagates.

The SPARE gate has a primary event and a set of spare events, also called the spare pool. Spare events can be shared, and are initially deactivated. However, sub-trees of spare events cannot have any shared nodes. When an input fails, the SPARE gate claims a spare in left-to-right order, and activates it. If the primary input has failed, and all available spares have failed, the SPARE gate fails.

The POR (priority OR) gate propagates if and only if the left-most input occurs before any other input.

The FDEP (functional dependency) gate has a triggering event and any number of dependent basic events. When the triggering event occurs, the dependent basic events are also set to failed. Syntactically, the triggering event and the dependent events are defined to be inputs to the FDEP gate. To prevent semantic confusion, however, the graphical representation uses outgoing edges to connect dependent events.

To illustrate the DFT notation, we consider the example shown in Fig. 2. The depicted DFT consists of four memory components; two primaries and two spares. The two spares are part of a spare pool shared among the two SPARE gates. According to standard DFT semantics, priority is given to claiming Memory3 before Memory4 in case of a failure of Memory1 or Memory2. Moreover, the system is equipped with two hot redundant, always active power sources, Power1 and Power2. Power1 powers both primary components, Memory1 and Memory2 and Power2 powers the spares. Finally, FDEP gates are used to model the functional dependencies between power supplies and memory components. The FDEPs propagate the failure of a power source to the respective memory components.

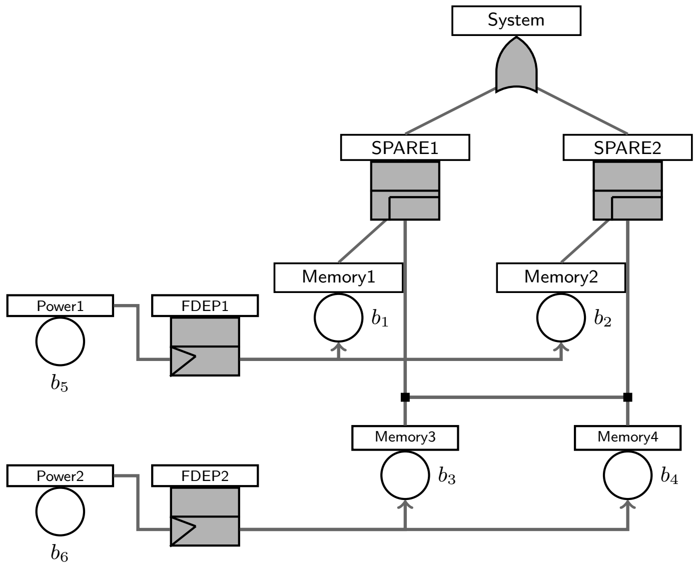


Fig. 2. Example DFT

3.2 Non-Deterministic Dynamic Fault Trees

DFTs impose a fixed and rigid order in which spares are activated. They do not allow to adapt the order depending on the history of occurred faults. This may lead to semantically undesirable consequences:

- A SPARE gate might claim a spare from a spare pool, despite having an already failed parent. This might deny a necessary resource to other SPARE gates that urgently require the spare to recover.
- In the event of spare races, it is not semantically clear which SPARE gate may claim a spare.
- The optimal order for spares has to be known at design time of the fault tree.

Fig. 3 visualizes possible DFT configurations exhibiting the above described semantic complications. Red indicates an incoming failure propagation. Spare claims are marked with thick, black lines.

In order to overcome these issues, [14] introduces an inherently non-deterministic DFT model (NdDFT, following the naming in [2]), which relaxes the semantic restriction of DFT models. Syntactically, the notation of the NdDFT is adopted from the DFT. Semantically, the NdDFT introduces natural non-determinism for spare activations, by allowing a SPARE gate to choose freely which spare to pick. A SPARE gate may also decide to not claim any spares and leave them available for other SPARE gates. In other words, the following recovery actions can be taken:

Definition 1 (Recovery Action). *A recovery action r in an NdDFT $\mathcal{T}$ is an action of the form*

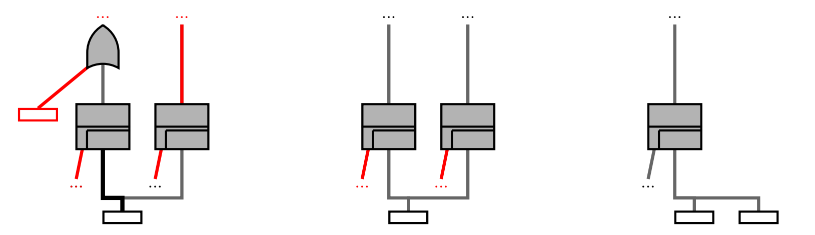


Fig. 3. Example configurations of problematic DFTs

- [] *(empty action) or*
- *CLAIM*(G, S) *(spare gate* G *claims spare* S*, where* S *is a spare of* G*).*

The non-determinism in NdDFT models is then resolved via an object called Recovery Automaton (RA). The RA defines which recovery actions should be taken whenever a set of basic events occur. The reason why sets of events as opposed to single events are used, is due to the ability of FDEPs to cause several basic events to fail simultaneously. Likewise, in order to react to the simultaneous occurrence of basic events, the RA may need to perform not just a single recovery action, but a sequence of recovery actions.

To introduce the formal notion of the RA we formalize the above auxiliary concepts as follows: For that, we denote the set of all recovery actions possible in an NdDFT $\mathcal{T}$ by $R(\mathcal{T})$. Moreover, this definition is extended to the set of recovery action sequences through $RS(\mathcal{T}) := (R(\mathcal{T}) \setminus \{[]\})^*$. For recovery action sequences, the empty action is ignored and considered as the empty word ϵ. The * here denotes the usual Kleene closure. Similarly, we denote the set of all non-empty subsets of basic events of an NdDFT $\mathcal{T}$ by $BES(\mathcal{T})$. We now introduce the RA on a formal level.

Definition 2 (Recovery Automaton). *A Recovery Automaton (RA)* $\mathcal{R}_{\mathcal{T}} = (Q, \delta, q_0)$ *of an NdDFT* $\mathcal{T}$ *is an automaton where*

- Q *is a finite set of states,*
- $q_0 \in Q$ *is the initial state, and*
- $\delta \colon Q \times BES(\mathcal{T}) \to Q \times RS(\mathcal{T})$ *is a deterministic transition function that maps the current state and an observed set of faults to the successor state and a recovery action sequence.*

To illustrate the interaction between RA and NdDFT, we give a simple example in Fig. 4. The system has a cold redundant spare and according to the RA, the redundancy is activated upon failure of the primary unit.

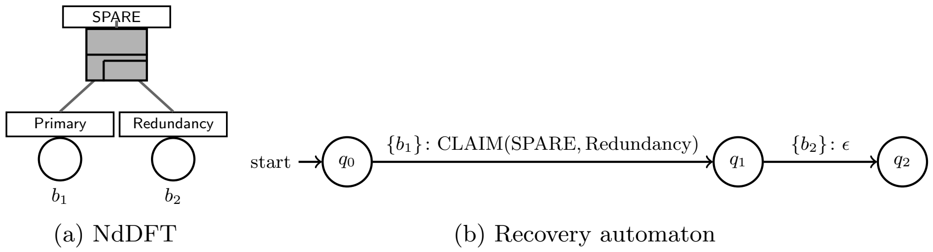


Fig. 4. Example of (a) NdDFT and (b) RA

3.3 Synthesizing Recovery Strategies

We sketch the key steps for recovery automaton synthesis. Further details are available in [14]. Initially, the algorithm converts an NdDFT model into a so-called Markov Automaton (MA) [6]. An MA is a transition system with continuous-time, non-deterministic, and probabilistic transitions. The MA contains all possible decisions on spare activations.

The transformation of an NdDFT into an MA is obtained by adapting traditional state-space generation algorithms for transforming DFTs to Continuous-Time Markov Chains (CTMCs). The base algorithm adapted here is given in [4]. The adapted algorithm operates in the following manner:

- Each state tracks a history of occurred basic event sets $(B_1, B_2, \ldots, B_n)$ and a mapping from spare gates to the currently claimed spares.
- The algorithm starts with the initial state denoted by $()$.
- BEs which are activated or have a dormant failure rate > 0 are considered as enabled events. Enabled events are used to compute the Markovian successors of a state. The history of the successor state is extended accordingly.
- The basic event set is obtained by taking a failing enabled event and computing the transitive closure according to FDEPs.
- Markovian transitions are labeled by the failure rate of the failing BE.
- All transitions that would lead to a state implying the top-level event are instead redirected to a special FAIL state.
- For each Markovian successor, non-deterministic successors are then computed, each of them corresponding to an enabled recovery action sequence.

The optimal recovery strategy, represented by an RA, can then be obtained by optimizing the scheduling of the generated MA with respect to an objective metric. The RA is then further reduced using both common state-space reduction methods based on trace equivalence and techniques exploiting the domain knowledge about the occurrence of faults [15]. Finally, by performing model checking queries on the Markov Chain (MC) obtained from the RA, enriched with the corresponding failure rates of the NdDFT, the desired RAMS metrics can be computed. A summary of the workflow and simple examples of all involved semantic objects can be found in Fig. 5.

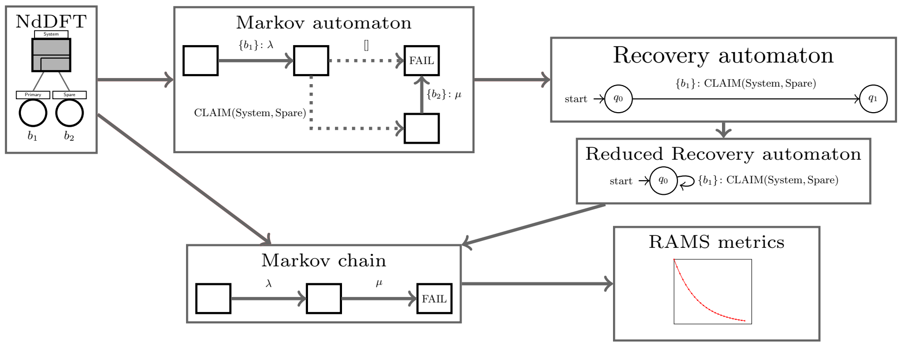


Fig. 5. Transformation road map

The metrics computed from the MC may be the optimization objective, but might also be any other metric of interest. The RA ensures that a consistent recovery strategy is applied for all possible queries. Relevant metrics of interest are for example:

– **Reliability After Time** t, which describes the probability that a system is still functional after a time span t.
– **Mean Time To Failure** (MTTF), which describes the expected time span that will pass until the system-level failure occurs.

The concrete metric itself is interchangeable. However, in this work, we focus on optimizing with regard to MTTF. This gives the advantage of dropping the time parameter t. For the MA, maximizing the MTTF corresponds to maximizing the expected long-term reachability property of the FAIL state.

4 Modular synthesis of recovery automata

The Markovian state space generated from a fault tree can be massive. In general, its size can grow exponentially with the number of nodes in a fault tree. The problem of an exponentially increasing state space is commonly known as the *state-space explosion problem*. In conventional dynamic fault trees, the blow-up can be attributed mostly to the interleaving occurrence of basic events. In the case of non-deterministic DFTs, the state-space explosion problem gains an additional dimension: The non-determinism caused by the selection of an appropriate recovery action generates an additional source of exponential blow-up.

Ensuring scalability while synthesizing recovery strategies for large fault trees with hundreds of basic events is nearly impossible using the previous, naive workflow. In the following, we consider how existing modular approaches for deterministic DFTs can be leveraged to solve the synthesis problem.

4.1 Modular Workflow

To tackle the state-space explosion problem for calculating RAMS metrics on deterministic DFTs, previous works have considered employing modularization techniques. These primarily involve detecting independent sub-trees in a fault tree – referred to as modules –, evaluating the metrics on the individual modules, and then composing them into the total metric for the original fault tree. For example, the total reliability after a certain time span for two modules connected via an AND gate can be obtained by means of multiplication.

Commonly, this approach faces a significant issue: Not all metrics can be computed in a compositional manner, but instead require the full state space for computation. In particular, highly interesting metrics such as the MTTF are not compositional [19]. However, in the context of the recovery strategy synthesis problem, the problem of compositionality changes. Even though the metric to be optimized may not be compositional, to determine the recovery automaton it is fortunately not necessary to compute the actual metric for the complete tree. Instead, the objects that require composition are the already optimized recovery automata. Automata composition in turn is a common problem that can be solved using standard techniques. We therefore exploit a two-stage approach by first synthesizing recovery automata, and then employing them during the computation of the actual metrics. In this manner, the non-determinism can be resolved modularly during the synthesis step. In greater detail, we apply the following approach:

1. Modularization: determine the modules in the fault tree.
2. Trimming: discard modules without non-determinism.
3. Synthesis: compute the optimal RA for each module, and reduce it.
4. Composition: assemble the overall recovery automaton from the modular RA.

As noted previously, basic events are a major driver for exponential blow-up. Therefore, events that do not affect the resolution of the non-determinism are taken out of the equation. Finally, as the non-determinism has already been resolved before the evaluation step, this particular source of exponential blow-up is absent during the computation of the metrics. Trimmed modules are only discarded for the purpose of the RA synthesis. This ensures that unnecessary information is safely removed, but properly considered during the metrics computation. The new workflow incorporating modularization is visualized in Fig. 6.

4.2 Modularization

We base our modularization approach on the pre-existing algorithm given in [5]. It applies a depth-first search on the fault tree, traversing all nodes while keeping track of the first and last visiting time of each node. These visiting times are then used to identify the modules using the following criterion: Given a node which is suspected of being the root of a module, if its descendants' visit dates – both first and last – all lie within the first and last visit dates of that node, then the node and all of its descendants form a module. In addition to this basic rule,

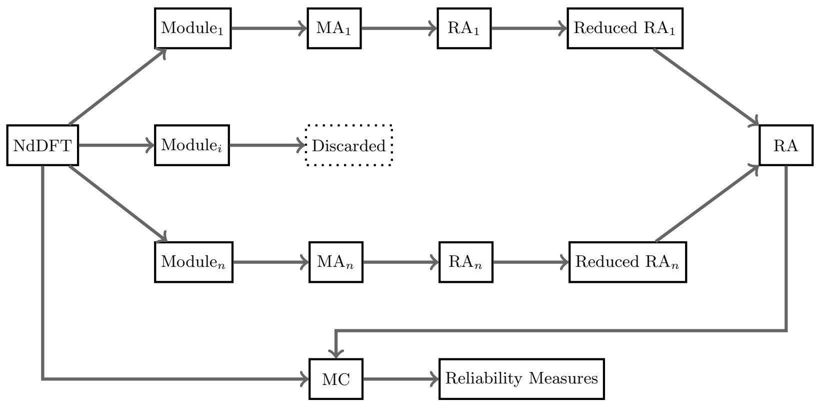


Fig. 6. Transformation road map with modularization

further restrictions have to be applied to obtain the desired compositionality property for the recovery automata.

There are two special cases which have to be considered: SPARE gates and all types of priority gates. Priority gates are road blockers to the desired compositionality property, as they may change the optimization direction. Consider for example a POR gate. In the case of the first input being a SPARE gate, the optimal strategy for maximizing the MTTF would also be to maximize the MTTF of the SPARE gate. In other words, claiming its available spares is the best course of action. On the other hand, were a SPARE gate the second input to a POR gate, then suddenly this simple relationship changes: Now minimizing the MTTF of the SPARE gate will lead to a scenario where the POR gate is more inclined to become fail-safe. The two scenarios are visualized in Fig. 7.

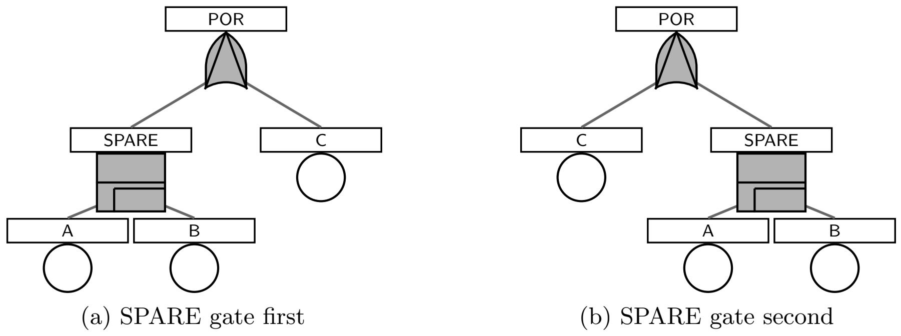


Fig. 7. Non-compositionality of priority gates as they change optimization direction.

Therefore, given recovery automata for two modules connected by a POR node, we cannot obtain the overall recovery automaton by means of composition. In addition to priority gates, SPARE gates also prohibit further modularization of their sub-trees. Due to the semantic definition of a SPARE gate, any basic event contained in a sub-tree may trigger a recovery action, and thus requires a representation within the Markovian state space. Bundling these observations, we obtain the following restrictions of the modularization rules:

– A SPARE gate that is a descendant of a priority gate cannot be the root of a module.
– A node that has a SPARE gate as a descendant and that is a descendant of a priority gate cannot be the root of a module.
– A descendant of a SPARE gate cannot be the root of a module.

Finally, an example application of the algorithm with the additional rules is given in Fig. 8. The algorithm proceeds in a leftmost order. Each node is labeled by the first and last visiting time, and the computed modules are indicated by dotted boxes.

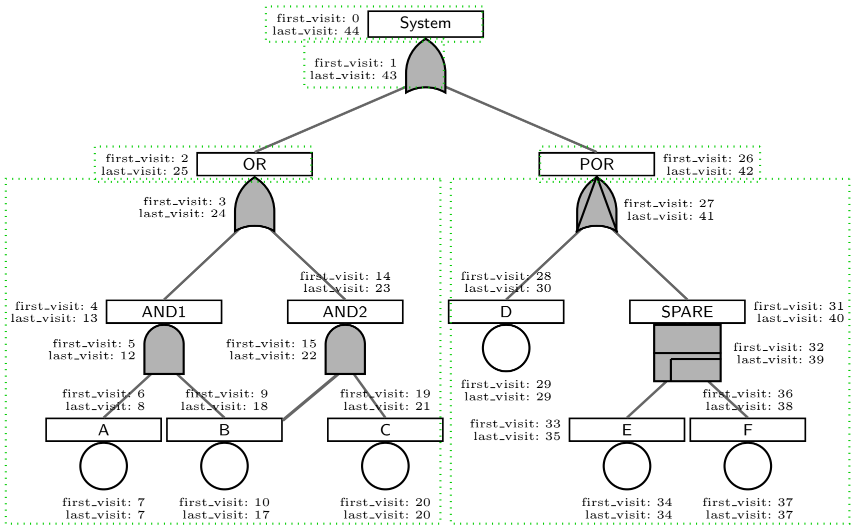


Fig. 8. Example application of the modularization algorithm

5 Case Studies

The proposed non-deterministic semantics and the modular workflow have been implemented within our application Virtual Satellite 4 FDIR (VirSat FDIR) [13].

Virtual Satellite 4 is an Eclipse-based modeling framework intended for Model-Based Systems Engineering of spacecraft [10]. VirSat FDIR is an application employing the framework to provide capabilities for modeling FDIR.

The FFORT benchmark set introduced in [17] was used as a source of fault tree benchmarks to evaluate our proposed techniques. FFORT is an online fault tree database with fault trees collected from scientific literature for the primary purpose of benchmarking. From the FFORT benchmark set, we have selected fault trees which contain at least one SPARE gate but do not employ the authors' custom fault tree extension of inspection modules (IM). Therefore, we can guarantee that all experiments contain some non-determinism. The following fault tree families from the FFORT benchmark fulfilled the selection criteria. (The graphic symbols refer to the evaluation charts shown in Fig. 9 and 10.)

- **Active Heat Rejection System (AHRS ○).** The AHRS is made up of thermal rejection units of which only one is needed for the system to function.
- **Cardiac Assist System (CAS ●).** The CAS models a hypothetical cardiac assist system with redundant CPUs, motors, and pumps.
- **Electro-Mechanical Actuator (EM _).** The model focuses on common-cause failures in an electro-mechanical actuator.
- **Hypothetical Example Computer System (HECS □).** The HECS fault trees model computer systems including their processors, memory modules, buses, consoles, operators, and software.
- **Hypothetical Example Multi-Phase System (HEMPS ■).** The HEMPS model is a demonstrator of a system designed for a multi-phase mission.
- **Mission Avionics System (MAS $*$).** The MAS models represent mission- and safety-critical systems with high redundancy. Components include hardware, software and vehicle control subsystems, and system management.
- **Multiprocessor Computing System (MCS ×).** The MCS model computers with power supplies, memory modules, hard disks, and connecting buses. The benchmarks have been enriched with instances from [19].
- **Nuclear Power Plant Water Pumping System (NPPW ⊗).** The model represents a nuclear power plant system.
- **Railway Crossing (RC ☆).** The RC fault tree collection models level railway crossings with sensors, motors, and controllers. The models come in two variations (sc and hc), representing the controller being a single basic event or hypothetical example computer system, respectively.
- **Vehicle Guidance System (VGS △).** The VGS models are industrial case studies dealing with variants of safety concepts for vehicle guidance systems.

The benchmarks were carried out with a Intel i7-6600U CPU, 4GB of RAM, and a timeout of 600s. The software, the experiment setup, all experiments, and all results can be found at [12]. The number of solved instances, the number of timeouts, the number of out-of-memories (OOMs), and the total solving time were logged. A summary of the results is given in Tab. 1.

As hypothesized in the beginning of the paper, not applying modularization leads to massive state-space explosion, causing many cases of OOMs. Applying

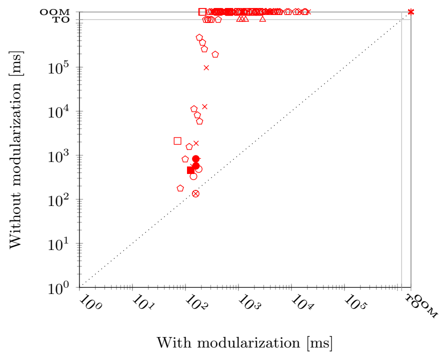
Table 1. Summary of benchmark results

modularization	solved	timeouts	ooms	solveTime [s]
no	22/156	10	124	1429
yes	142/156	0	14	292

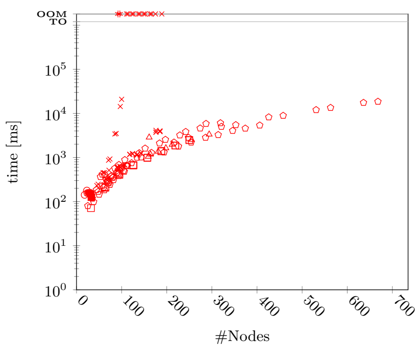
modularization, on the other hand, yields a major speed-up, enabling us to synthesize RA for nearly all instances. An interesting observation is also that the number of timeout events is rather small, compared to the large number of OOMs. This suggests that there is room for investing more computation time into further techniques for state-space reduction, and hence also reducing the memory consumption.

The following charts give a closer look at the results of the experiments. Fig. 9a shows a detailed time comparison between the synthesizer with and without a modularizer, respectively. How the algorithm scales overall as the total number of fault-tree nodes increases, is shown in Fig. 9b. The dashed line marks where both algorithms require the same time. Timeouts and out-of-memory results have been placed on the outer lines and are labeled with TO and OOM, respectively.

As described before, two major drivers for state-space explosion are basic events and SPARE gates. Fig. 10a gives a breakdown on how the number of BEs impact the synthesis. Likewise, Fig. 10b gives the breakdown in reference to the number of SPARE gates. Modularization is enabled in both cases. The data shows that the speed-up gained from modularization is overall crucial to obtain scalability, but also heavily depends on the fault tree family. The families causing OOMs are primarily MAS and MCS. A closer look into Fig. 9b and Fig. 10b reveals that these have a relatively small number of nodes, while at the same time having a relatively large number of SPARE gates.

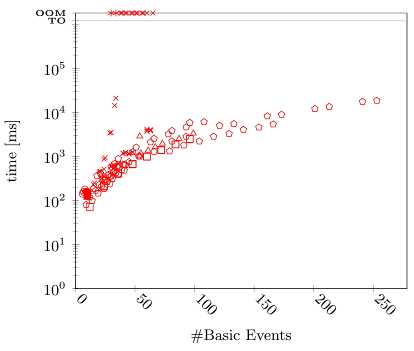


(a) Modularization vs no modularization

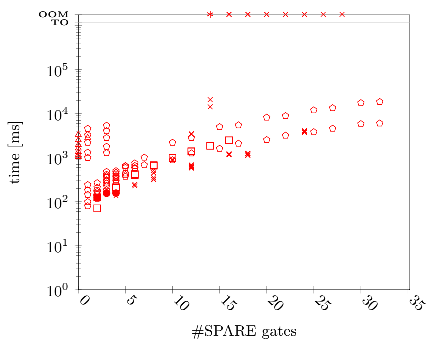


(b) Runtimes with #Nodes

Fig. 9. Time measurements of modularization approach



(a) Basic Events



(b) Spares

Fig. 10. Time measurement break-down for basic events and SPARE gates

6 Conclusions and Future Work

In this paper, we investigated a modular approach approach to recovery automata synthesis for non-deterministic DFTs. In order to deal with the increasing complexity due to the semantic extension, modularization approaches were employed. Both their necessity and effectiveness were demonstrated on case studies coming from the FFORT benchmark set. However, it was also shown that the semantics still yield a severe level of state-space explosion, causing many out-of-memories but only few timeouts.

Further techniques for dealing with larger modules and modules with a high degree of non-determinism are therefore desirable. In the past, symmetry reduction techniques have proved useful to combat the state-space explosion problem in deterministic DFTs [19]. In the future, we hope to investigate how those approaches can be leveraged to NdDFTs to further improve the efficiency of our approach.

References

1. Amari, S., Dill, G., Howald, E.: A new approach to solve dynamic fault trees. In: Annual Reliability and Maintainability Symposium. pp. 374–379. IEEE (2003). https://doi.org/10.1109/RAMS.2003.1182018
2. Beccuti, M., Franceschinis, G., Codetta-Raiteri, D., Haddad, S.: Computing optimal repair strategies by means of NdRFT modeling and analysis. The Computer Journal **57**(12), 1870–1892 (2014). https://doi.org/10.1093/comjnl/bxt134
3. Boudali, H., Crouzen, P., Stoelinga, M.: A compositional semantics for dynamic fault trees in terms of interactive markov chains. In: Automated Technology for Verification and Analysis. LNCS, vol. 4762, pp. 441–456. Springer (2007). https://doi.org/10.1007/978-3-540-75596-8_31
4. Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Transactions on Reliability **41**(3), 363–377 (1992). https://doi.org/10.1109/24.159800

5. Dutuit, Y., Rauzy, A.: A linear-time algorithm to find modules of fault trees. IEEE Transactions on Reliability **45**(3), 422–425 (1996). https://doi.org/10.1109/24.537011
6. Guck, D., Hatefi, H., Hermanns, H., Katoen, J.P., Timmer, M.: Modelling, reduction and analysis of Markov automata. In: Quantitative Evaluation of Systems. LNCS, vol. 8054, pp. 55–71. Springer (2013). https://doi.org/10.1007/978-3-642-40196-1_5
7. Han, W., Guo, W., Hou, Z.: Research on the method of dynamic fault tree analysis. In: Int. Conf. on Reliability, Maintainability and Safety. pp. 950–953. IEEE (2011). https://doi.org/10.1109/ICRMS.2011.5979422
8. International Electrotechnical Commission, Geneva, Switzerland: Fault Tree Analysis (FTA) (2006)
9. Junges, S., Guck, D., Katoen, J.P., Stoelinga, M.: Uncovering dynamic fault trees. In: Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on. pp. 299–310. IEEE (2016)
10. Lange, C., Grundmann, J.T., Kretzenbacher, M., Fischer, P.M.: Systematic reuse and platforming: Application examples for enhancing reuse with model-based systems engineering methods in space systems development. Concurrent Engineering **26**(1), 77–92 (2018). https://doi.org/10.1177/1063293X17736358
11. Liu, D., iei Xiong, Li, Z., iang, P., Zhang, H.: The simplificafion of cuf sequence sef analysis for dynamic sysfems. In: Int. Conf. on Computer and Automation Engineering. pp. 140–144. IEEE (2010). https://doi.org/10.1109/ICCAE.2010.5451831
12. Müller, S.: virtualsatellite/VirtualSatellite4-FDIR: Release 4.12.1 (Oct 2020). https://doi.org/10.5281/zenodo.4075576
13. Müller, S., Gerndt, A.: Towards a conceptual data model for fault detection, isolation and recovery in Virtual Satellite. In: SECESA 2018. European Space Agency (2018), `https://elib.dlr.de/122061/`
14. Müller, S., Gerndt, A., Noll, T.: Synthesizing FDIR recovery strategies from non-deterministic dynamic fault trees. In: 2017 AIAA SPACE Forum. vol. AIAA 2017-5163. American Institute of Aeronautics and Astronautics (2017). https://doi.org/10.2514/6.2017-5163
15. Müller, S., Mikaelyan, L., Gerndt, A., Noll, T.: Synthesizing and optimizing FDIR recovery strategies from fault trees. Science of Computer Programming **196**, 102478 (2020). https://doi.org/https://doi.org/10.1016/j.scico.2020.102478
16. Pullum, L., Dugan, J.: Fault tree models for the analysis of complex computer-based systems. In: Annual Reliability and Maintainability Symposium. pp. 200–207. IEEE (1996). https://doi.org/10.1109/RAMS.1996.500663
17. Ruijters, E., Budde, C.E., Nakhaee, M.C., Stoelinga, M.I.A., Bucur, D., Hiemstra, D., Schivo, S.: FFORT: A benchmark suite for fault tree analysis. In: 29th European Safety and Reliability Conference. pp. 878–885. Singapore Research Publishing (2019). https://doi.org/10.3850/978-981-11-2724-3_0641-cd
18. Ruijters, E., Guck, D., Drolenga, P., Stoelinga, M.: Fault maintenance trees: reliability centered maintenance via statistical model checking. In: 2016 Annual Reliability and Maintainability Symposium (RAMS). pp. 1–6. IEEE (2016). https://doi.org/10.1109/RAMS.2016.7447986
19. Volk, M., Junges, S., Katoen, J.P.: Advancing dynamic fault tree analysis-get succinct state spaces fast and synthesise failure rates. In: International Conference on Computer Safety, Reliability, and Security. LNCS, vol. 9922, pp. 253–265. Springer (2016). https://doi.org/10.1007/978-3-319-45477-1_20
20. Yevkin, O.: An improved modular approach for dynamic fault tree analysis. In: Annual Reliability and Maintainability Symposium. pp. 1–5. IEEE (2011). https://doi.org/10.1109/RAMS.2011.5754437