Synthesizing FDIR Recovery Strategies From Non-Deterministic Dynamic Fault Trees

Sascha Müller and Andreas Gerndt
DLR (German Aerospace Center), Software for Space Systems and Interactive Visualization, Braunschweig, 38108, Germany

Thomas Noll
Software Modeling and Verification Group, RWTH Aachen University, Aachen, 52056, Germany

Redundancy concepts are an integral part of the design of space systems. Deciding when to activate which redundancy and which component should be replaced can be a difficult task. In this paper, a model of non-deterministic dynamic fault trees is presented and it is shown how appropriate strategies can be synthesized from them. This is achieved by transforming a non-deterministic dynamic fault tree into a Markov Automaton. From the optimized scheduler of this Markov Automaton, an optimal recovery strategy can then be derived. We also introduce the model of Recovery Automata to represent these strategies.

I. Introduction

Reliability engineering is an important discipline in the design of any safety critical system, in particular in the domain of aerospace systems and spacecraft. No matter how well designed a system is, it still has to deal with the presence of faults to some extent. Faults in this context can be events such as equipment failure, wrong sensor readings, external interferences and many more. To raise trust in handling system failures, reliability engineering tries to embed Failure Detection, Isolation and Recovery (FDIR) concepts. These concepts are derived using various tools and methodologies such as Fault Tree Analysis (FTA).[1]

FTA is a methodology commonly used in the industry for performing state-of-the-art failure analysis.[2] The resulting Fault Trees (FT) describe how faults propagate through components and subsystems of a system and eventually lead to a top-level system failure. Graphical representations of these trees are intuitive and easy to understand. On the one hand, FTs can be used to analyze the system qualitatively in terms of fault combinations that lead to system failure. On the other hand, they also enable quantitative analysis of important computable measures such as reliability. Dynamic Fault Trees (DFT) are an extension introducing temporal understanding and new features to analyze redundancy concepts known as spare management. However, there are challenges arising from non-deterministic DFT behavior such as spare races. An example for such race behavior can be seen in a system of two operative memories together with a pool of two spare memories. In case both operative memories fail at the same time it is unclear which backup memory takes over the role of which operational one.

To overcome this shortcoming, a new methodology is presented in this paper. It introduces a model of non-deterministic Dynamic Fault Trees (NdDFT) as an extension to DFT. The new NdDFT does not impose a fixed, rigid order on the spares to be used. As next step, the methodology foresees transforming this NdDFT model into a Markov Automaton (MA) which is suitable for the computation of the aforementioned non-deterministic decisions on spare activations. By optimizing the scheduling of the MA model in terms of reliability of the system, a recovery strategy for the NdDFT can be synthesized. This recovery strategy defines which spare has to be used in which failure state of the system and can therefore guarantee an optimal reliability at all times. This paper is structured as follows. Section II of this paper summarizes the related work relevant to the topic of FTs, MA and synthesis of recovery strategies. Further background on the theory of FTs is given in Section III. The NdDFT model is introduced in Section IV. This section also describes the process of synthesizing recovery strategies from a given NdDFT as well as a model to represent such strategies. Finally the paper concludes in Section V and provides some outlook on future work.

II. Related Work

The goal of FDIR lies in keeping a system in a stable and operational state, even in the presence of faults. While some of the following steps may be omitted in some cases, performing FDIR generally means applying the following procedural approach:[3]

- Monitor the system to detect the occurrence of faults.
- Identify the fault and localize it within the system.
- Isolate the fault and prevent further propagation into other parts of the system.
- Perform recovery actions to reconfigure the system and return it into a stable state.

In order to derive how faults relate to each other and eventually lead to a system wide failure, failure analysis techniques such as FTA can be employed. In general FTs are graphs consisting of two types of nodes representing events and gates. The root node, or top level event (TLE), usually represents the event of a system failure whereas the leaves of the tree model the event of individual components failing. The leaves are also called basic events (BE). They correspond to a Boolean variable where false represents the initial state of no failure. The variable is considered true in case of a failure event. The branches of the trees are represented by the gates performing operations on the events. FTs are directed graphs starting from the BEs pointing over the gates towards the system failure event. They are also free of cycles.

One of the very basic types of FTs are Static Fault Trees (SFT). They employ Boolean algebra to combine various different failure events by AND and OR operations, often graphically represented as gates, until they sum up to the overall system failure. The failure events are usually related to faulty components of the system. Applying this methodology statements such as "The system fails if component A and component B fail" can be modeled and refined to arbitrary levels of precision.

Many extensions have been proposed to the formalism of FTs[2] to increase its expressiveness and enhance its features. A particular extension is the notion of Dynamic Fault Trees (DFT). It introduces temporal understanding and new features to analyze redundancy concepts known as spare management. Accordingly DFTs define a new SPARE gate to model that some faulty component or subsystem is replaced by a spare from a set of redundant parts. In the common understanding of DFTs, the order in which such a spare is chosen is deterministic and defined at design time by the reliability engineer.

With the addition of spares, DFTs also introduce a new node state. In SFTs nodes only have two states: Failed or operational. In DFTs a node can be either failed, active (operational) or dormant (operational). A node that is an unactivated spare is dormant, all other nodes are activated. Together with this state, failure rates for failing actively and failing dormantly can be defined for every BE. Usually the dormant failure rate is lower than the active one, possibly even 0. By using a dormant failure rate of 0 cold redundancy can be modeled. Hot redundancy can be modeled by setting the dormant failure rate to the active failure rate. Likewise, warm redundancy can be modeled by choosing a dormant failure rate between 0 and the active failure rate. These rates are then used for calculating measures of interest such as the probability of the top-level failure after time t (reliability). The reliability can be computed from a given DFT for example by transforming a DFT into a Continuous-Time Markov Chain (CTMC).[4]

Markov Automata[5] are extensions to CTMCs. They are state-based transition systems with two types of transitions: They can contain continuous-time transitions (also called Markovian transitions) that are labeled with rates, that is non-negative real values, as well as immediate, non-deterministic transitions labeled by actions. In the latter case, transitions have to be chosen by a so-called scheduler. The computation of optimal schedulers for Markov Automata with respect to various quantitative objectives, such as state reachability, is discussed in 6.

Computing strategies for recovery purposes from a given fault model has been researched. In 7, a similar approach is taken for repairable fault trees. Repairable fault trees are a FT formalism where failed basic events can be repaired. In this case, this is realized with repair boxes each of them being equipped with a repair policy which states which resources are required to perform the repair, in which order basic events should be repaired, the repair rate and so on. Using such a repairable FT, failure analysis also becomes possible not only for permanent failures, but also for transient ones. The authors consider non-deterministic repair policies where the repair order is not fixed. Optimal repair policies are then computed by converting the repairable fault tree to a Markov decision process, a time discrete version of Markov Automata. However, the authors do not consider DFT models.

In 8, Dynamic Decision Networks (DDN) are employed and their inference capabilities are exploited to create autonomous on-board FDIR systems for spacecraft that can select reactive and preventive recovery actions during run-time. The authors propose in 9 to create the DDN from an extension of the DFT model.

Also set in the domain of space systems, Timed Failure Propagation Graphs are used in 10 to synthesize FDIR components, namely monitors for the purpose of fault detection and recovery plans for every specified combination of fault and mode. Here, the recovery components are created using a planning based approach on predefined actions.

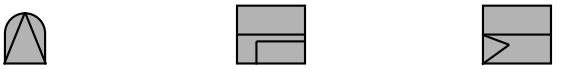
III. Background to Semantics and Notation of Static and Dynamic Fault Trees

Fig. 1 shows the gates and events used in the SFT notation. SFTs use Boolean operations represented by AND and OR gates. There also exist other gates such as the k-VOTE gate, which propagates if at least k inputs have failed. Observe that a 1-VOTE gate corresponds to an OR gate and a k-VOTE gate with k inputs to an AND gate. Implementation wise, all gates can therefore be considered as k-VOTE gates for some fitting k. Some other extensions also introduce a NOT gate. However, this allows the construction of fault trees where the TLE can go from having failed to working again as new failures occur. These fault trees are known as non-coherent fault trees and have been dismissed as being a sign for modeling errors.[11]

(a) BE (b) Non-basic event (c) OR (d) AND (e) k-VOTE

Figure 1: Gates and events in a Static Fault Tree

Fig. 2 depicts the notation to extend SFTs to DFTs introducing three new gates PAND, SPARE and FDEP. The PAND (priority AND) gate propagates in case all inputs fail in sequence from left to right. It does not propagate the failure in case the sequence is not obeyed. The SPARE gate is connected to a primary event and a set of spare events. The spare events can be shared with another SPARE gate, therefore a spare can be claimed by either the one or the other SPARE gate. But there may be no shared elements between the primary input and any spare. We also do allow for nesting of spares, e.g. SPARE gates can be spares. The SPARE gate propagates a failure if the primary input failed and all spares are either claimed or failed themselves. The FDEP (functional dependency) gate sets the outputs to false in case a failure on the left hand input has occurred, even if one of the other input events have still not failed. There are other gate types considered in literature such as the SEQ (sequential) and the POR (priority OR) gate. The SEQ restricts events to happen in a given order only,[4] whereas the POR propagates in case at least one input event fails but still in order from the left to the right input.[12]

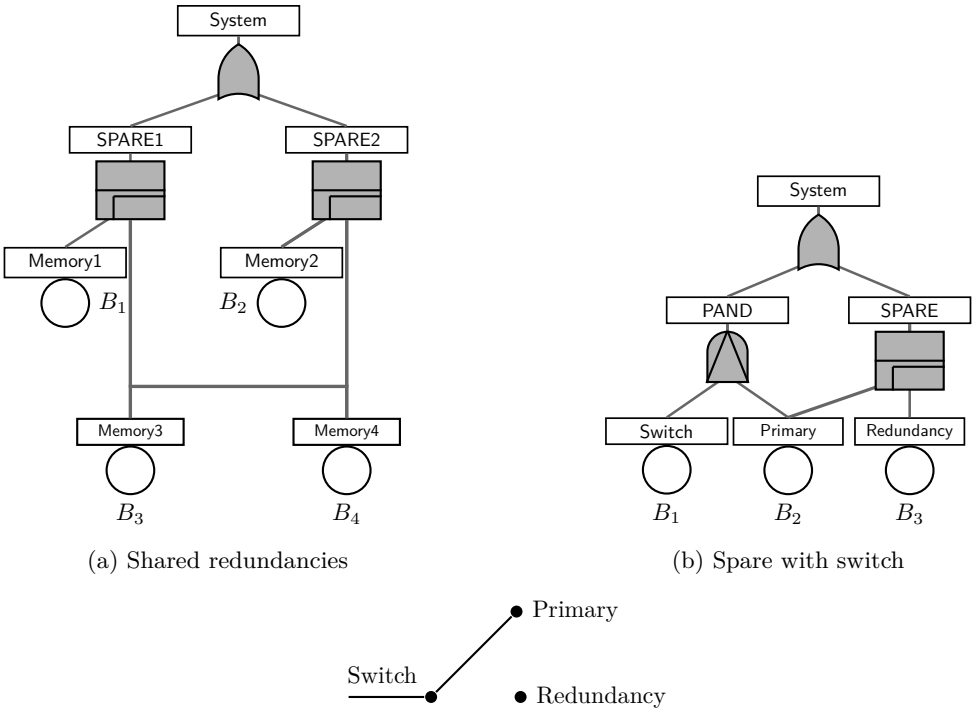


(a) PAND (b) SPARE (c) FDEP

Figure 2: Standard dynamic gates

Generally, basic events will be denoted by $B_1, B_2, \ldots$ and failure rates by $\lambda_1, \lambda_2, \ldots$. As for the association of failure rates with basic events, in the following for any basic event B the active failure rate will be denoted by $F_A(B)$ and the dormant failure rate by $F_D(B)$. In the case of $F_A(B) = F_D(B)$ the subscripts will be dropped and the simplified notation $F(B)$ will be used to denote the failure rate.

Fig. 3 illustrates two examples of the DFT notation. Example 3a shows a system consisting of two memories which are covered by two spare memories for failure. The two spares are shared among the two SPARE gates. According to DFT semantics, Memory3 will be used before Memory4 in case of a failure of Memory1 or Memory2. Example 3b shows a use case for the PAND gate. The system itself is depicted in Fig. 3c and consists of some primary equipment, a cold spare unit and a switch that switches to the cold spare should the primary unit fail. According to DFT semantics, the system is in a non-failing state if the switch fails after the primary component. But should the switch fail before the primary, then we are unable to switch to the redundancy.



(a) Shared redundancies

(b) Spare with switch

(c) A physical system with a cold spare and a switch

Figure 3: (a) and (b) show some example DFTs and (c) is the system represented by (b)

IV. Non-Deterministic Dynamic Fault Trees

As described in the previous sections, DFTs require that spares are activated in a fixed and rigid order. This order cannot be adapted depending on faults that have previously occurred. Additionally, in cases of spare races it is not semantically clear which SPARE gate claims the actual redundancy. To relax on this semantical restriction of the DFT model, an inherently non-deterministic DFT model (NdDFT, following the naming in 7) needs to be defined. This definition introduces a recovery strategy that can be optimized by first transforming the NdDFT into an MA. Computing an optimal scheduler for this MA using standard algorithms allows to derive a so called Recovery Automaton. This Recovery Automaton provides the optimal strategy to react on failures in the NdDFT.

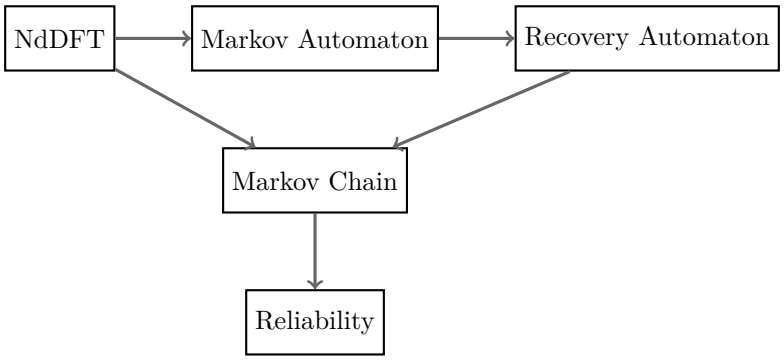


Figure 4: Transformation road map

A. Definition of Non-Deterministic Dynamic Fault Trees

The here defined NdDFT is based on the same semantics as a DFT, except for the activation conditions of spares. The NdDFT drops the requirement that spares are always activated from left to right. The new non-deterministic semantics allow for a SPARE gate to not claim any of the attached spares, thus leaving it available for more important SPARE gates that may also require a spare. The syntax and notation of the NdDFT is completely adopted from the DFT. Whenever a fault event, or more precisely a BE, occurs in a NdDFT the new semantics allow performing any valid recovery action of the following form:

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

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

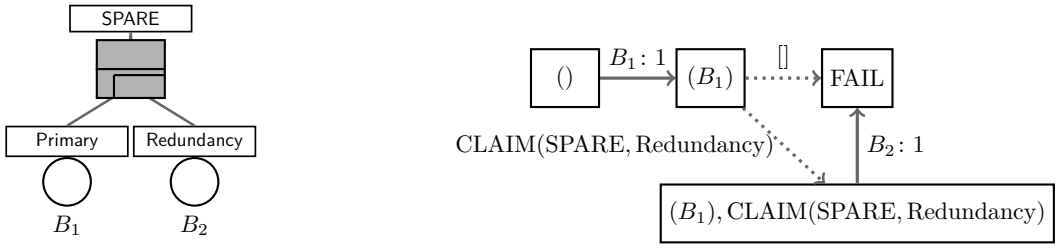
In the NdDFT, the actual recovery action R that is applied is defined by a given recovery strategy. We denote the set of all recovery actions possible in an NdDFT $\mathcal{T}$ by $R(\mathcal{T})$ and likewise the set of all basic events by $BE(\mathcal{T})$. A recovery strategy is then a mapping that, given the observed faults (basic events), returns the recovery actions that should be taken accordingly. The NdDFT considers recovery strategies that only have recovery actions as defined in Def. 1 and is defined as follows:

Definition 2 (Recovery Strategy) *A recovery strategy for an NdDFT $\mathcal{T}$ is a mapping $Recovery : BE(\mathcal{T})^* \to R(\mathcal{T})^*$ such that*

- $Recovery(\varepsilon) = \varepsilon$ *and*
- $Recovery(B_1, \ldots, B_n) = Recovery(B_1, \ldots, B_{n-1}), R_n$ *for some* $R_n \in R(\mathcal{T})$.

B. Transformation of NdDFTs into Markov Automata

In this setting, Markovian transitions will represent basic event transitions with the transition rate being the failure rate of a basic event. The non-deterministic transitions will represent the controlled actions that can be taken. In other words, they will be labeled with the recovery actions. Transforming an NdDFT to a Markov Automaton can be done by adapting traditional algorithms for transforming DFTs to CTMCs. As base algorithm we use the one given in 4. The adapted algorithm operates by memorizing two sets of data in every of its states: First, the history of occurred basic events $(B_1, B_2, \ldots, B_n)$. Second, a mapping from spare gates to the currently claimed spare. The initial empty history of the algorithm is denoted by $()$. Starting with this initial state, all active basic events are used to compute successors for each of them while extending the history accordingly. More precisely, all basic events that are not part of an unactivated spare. The transitions are labeled with the respective failure rate of their basic event. For further computation purposes, they are also labeled with the BE. A special state FAIL is added as well. All transitions that would lead



(a) NdDFT (b) Markov Automaton

Figure 5: Example transformation NdDFT to MA. The active failure rates of the basic events are $F_A(\text{Primary}) = F_A(\text{Redundancy}) = 1$ and the dormant failure rates are $F_D(\text{Primary}) = F_D(\text{Redundancy}) = 0$. Dotted edges represent the non-deterministic transitions and solid lines represent the Markovian transitions. $B : \lambda$ denotes that basic event B occurs (actively or dormant) with rate λ.

to a state that implies that the top-level event (system failure) has occurred, is then connected to the FAIL state instead. After performing a Markovian transition to a next state, the algorithm generates successors using non-deterministic transitions. Each non-deterministic transition is labeled by a valid recovery action. The algorithm updates the mapping of claimed spares accordingly. This yields an MA with alternating Markovian and non-deterministic transitions, which are respectively labeled by BEs with their failure rates and by recovery actions. For performance purposes, the procedure is also extended to additionally check for each generated state if there is an equivalent (i.e., probabilistically bisimilar[13]) state and reduces the state space accordingly.

Fig. 5 illustrates the aforementioned algorithm. Here, a simple NdDFT consisting only of a spare gate with a primary input and one cold spare is transformed into the corresponding Markov Automaton representation. After B_1 fails, there are two possible recovery actions: the empty action ([]) or the activation of the spare redundancy (Claim(SPARE, Redundancy)). In this simple example it is obvious that immediately activating the Redundancy upon observing B_1 is the correct course of action. A recovery strategy *Recovery* that could be synthesized in this example would thus yield $Recovery(B_1) = \text{Claim}(\text{SPARE}, \text{Redundancy})$.

C. Synthesizing Recovery Automata Based on Markov Automata

Using existing techniques for optimizing the scheduling of a Markov Automaton, it is possible to determine for each state with outgoing non-deterministic transitions, which one should be chosen to maximize the reliability of the system. In order to extract a recovery strategy from a scheduler for a Markov Automaton, a formal way for representing the underlying decision process is required. For this purpose we introduce the concept of a Recovery Automaton. A Recovery Automaton is essentially an automaton that reads the occurred basic events and outputs its next state as well as the recovery action that should be taken. In this sense, it is basically a Mealy Automaton with the input alphabet being the set of basic events and the output alphabet the set of recovery actions. This is formalized with the following definition:

Definition 3 *A Recovery Automaton $\mathcal{R}_\mathcal{T} = (Q, \delta, q_0)$ of an NdDFT $\mathcal{T}$ is an automaton where*

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

By employing such an automaton to decide which spare should be used in an NdDFT, the model can be evaluated deterministically, thus handling spare races in a well defined manner and adapting the allocation of spares according to where they are required. Extracting a Recovery Automaton from a scheduler for a Markov Automaton is achieved by replacing successive pairs of transitions for states s, s', s'' of the form $(s, b\colon \lambda, s'), (s', r, s'')$ where b is a basic event, λ a failure rate, r a recovery action by the transition $\delta(s, b) = (s'', r)$. This applies to all transitions where s'' is the successor computed by the optimized schedule of the Markov Automaton. All other non-deterministic transitions are then discarded. Observe that by construction we have the property that every Markovian transition is followed by a non-deterministic transition or no transition in the special case of the FAIL state. Finally, the algorithm discards all unreachable states and minimizes the resulting automaton for a more compact representation.

D. Example Construction of an Adaptable Recovery Strategy

To illustrate the construction, we consider in the following an example NdDFT and the resulting Recovery Automaton and then compare the reliability between the DFT version and the NdDFT with Recovery Automaton version. To show case the state space construction, a small excerpt of the constructed Markov Automaton will also be shown. As an example NdDFT consider the model shown in Fig. 6. Here, we have a similar case to the one depicted in Fig. 3b, except that now two equipments are dependent on a switch activating their spare. The spare Redundancy is shared among the two spare gates SPARE1 and SPARE2. Should either subsystem fail, then the entire system fails.

Fig. 7b depicts a Recovery Automaton that we have computed for the NdDFT of Fig. 6. Observe that since each event can only occur at most once, each transition including loop transitions can be taken at most once. With $B : r$ we denote the event of basic event B having occurred and that recovery action r should be taken. We consider for the failure rates the (unit less) values

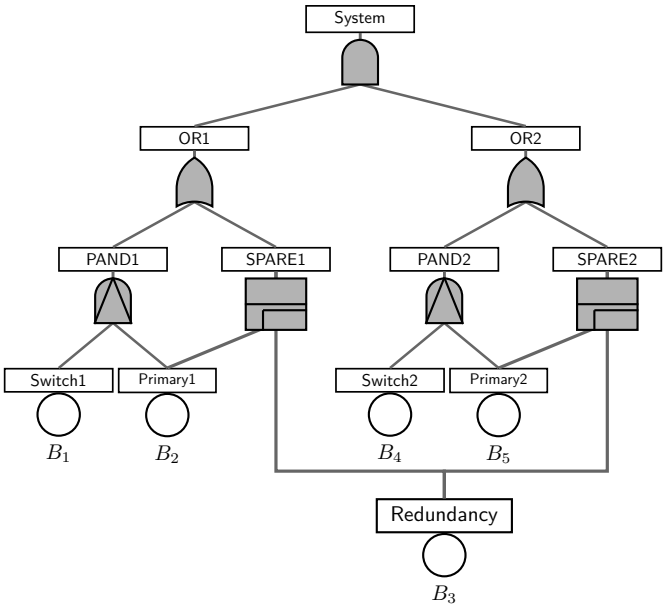
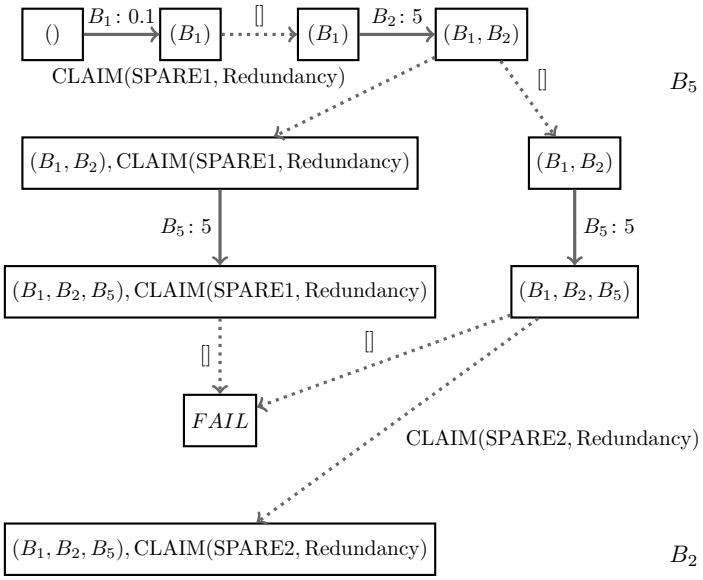


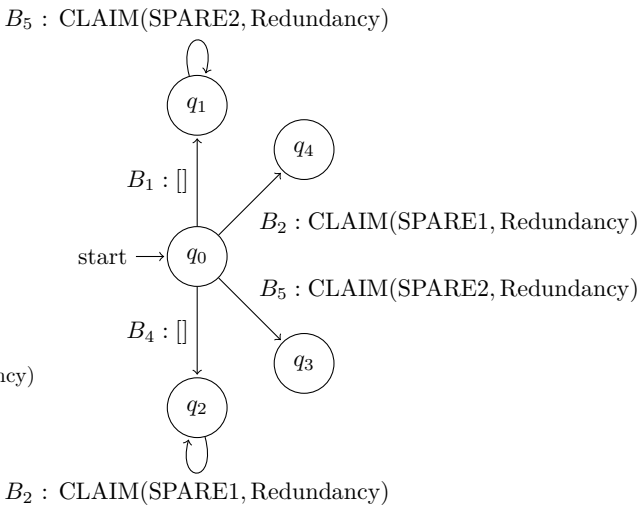
Figure 6: DFT with a shared redundancy and two switches

- $F(B_2) = F(B_5) = F_A(B_3) = 5$ (equal active failure rates for all equipments),
- $F_D(B_3) = 0$ (the spare is a cold redundancy) and
- $F(B_1) = F(B_4) = 0.1$ (low switch failure rate).

Fig. 7a show cases a small excerpt of the constructed Markov Automaton. In the Recovery Automaton we can find the corresponding fragment in the states q_0 and q_1. In essence, the synthesized Recovery Automaton states that the spare gates SPARE1, SPARE2 should not allocate the redundancy if the switch has already failed.



(a) Excerpt from the Markov Automaton for the NdDFT in Fig. 6 for the fault sequence (B_1, B_2, B_5)



(b) Synthesized Recovery Automaton for switch system

Figure 7: Example extraction of Recovery Automaton

Fig. 8 shows the reliability measures firstly when considering the fault tree in Fig. 6 as a standard DFT and secondly when considering it as an NdDFT and equipping it with the Recovery Automaton from Fig. 7b. For the time frame a unit less mission time of 1 has been chosen. We can see that even though both converge towards 0, employing an adaptive strategy for activating spares yields a reliability curve that is consistently better than the fixed order strategy of standard DFT.

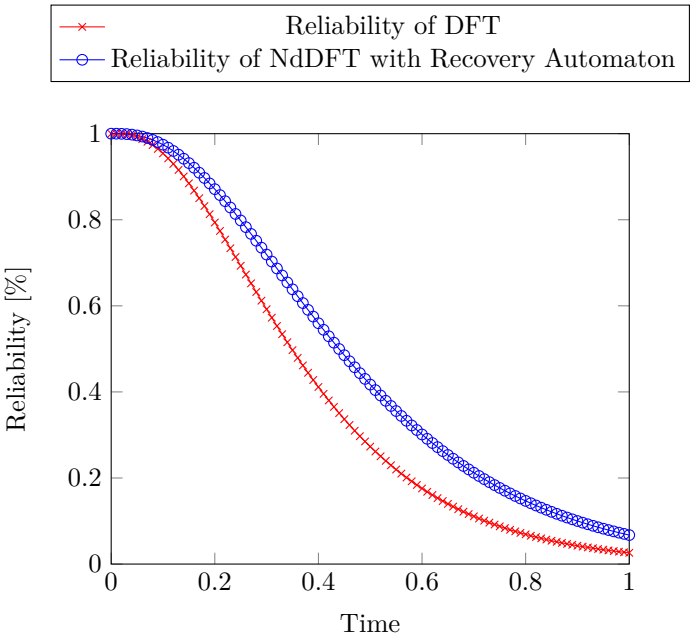


Figure 8: Reliability of DFT vs. reliability of NdDFT with Recovery Automaton

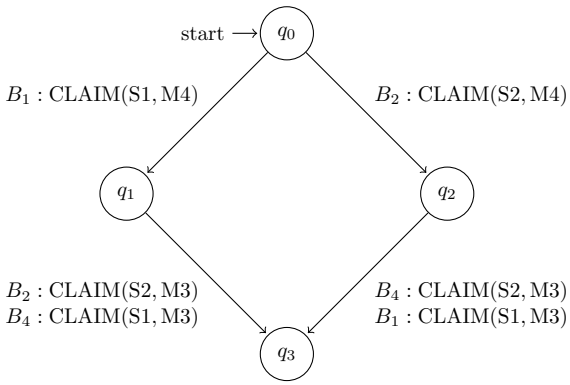
E. Example of Optimized Spare Ordering

As a second example use case, reconsider the redundant memory system shown in Fig. 3a. With the default DFT semantics, Memory3 would always be employed before Memory4. However, as will be shown in the following, depending on the failure rates this might yield a suboptimal strategy. Consider for example as failure rates

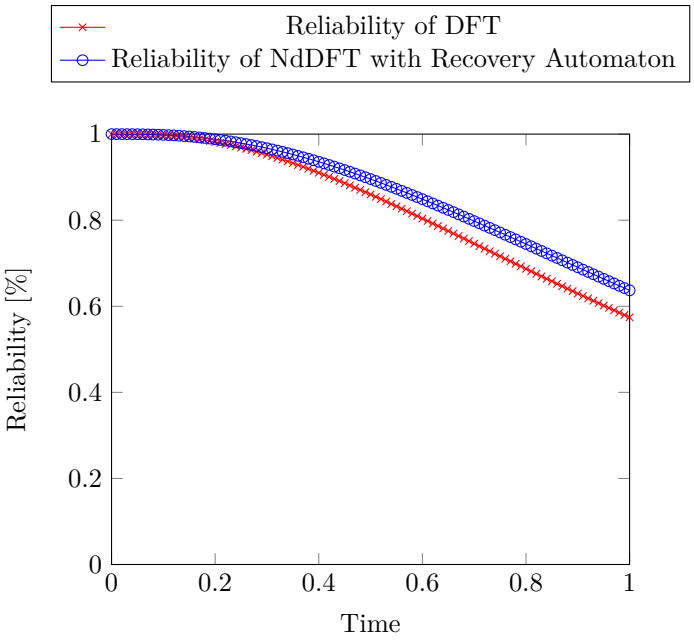
- $F(B_1) = F(B_2) = 1$,
- $F_A(B_3) = 5$ (Modeling a low quality spare),
- $F_A(B_4) = 0.5$ (Modeling a high quality spare) and
- $F_D(B_4) = F_D(B_3) = 0$ (The spares are cold redundancies).

Hence, according to DFT semantics the low quality spare will always be activated first. However, as will be seen in the following this ordering is not optimal. For improved readability in the following figures, the items SPARE1, SPARE2 and Memory1, ..., Memory4 will be abbreviated by S1, S2 and M1, ..., M4 respectively. The synthesized Recovery Automaton is depicted in Fig. 9a. It basically states that the system should always first activate Memory4, that is the high quality spare, and then the low quality spare Memory3.

To evaluate the recovery strategy induced by the Recovery Automaton, in Fig. 9b the reliability curves of the fault tree models are plotted. It can be seen that employing the strategy proposed by the synthesized Recovery Automaton yields a slight edge over the reliability curve of the standard DFT. While in this simple example the DFT model could yield the same performance by correcting the spare ordering, this can become exceedingly difficult as the complexity of spares, which may also be modeled by complex fault trees, increases.



(a) Synthesized Recovery Automaton



(b) Reliability of DFT vs. reliability of NdDFT with Recovery Automaton

Figure 9: Analysis of memory system in Fig. 3a

V. Conclusion and Future Work

We have considered in this paper the problem of synthesizing recovery strategies, focusing on the aspect of scheduling spare activations and resolving spare races. For this purpose, the formalism of non-deterministic DFTs was introduced and motivated. An algorithm for converting NdDFTs to Markov Automata was given and it was discussed how recovery strategies can be extracted from the optimized scheduler of the generated Markov Automaton.

Additionally, a representation for recovery strategies in form of the also newly introduced model of Recovery Automata was given as well. Furthermore, we considered two examples of how equipping an NdDFT with a Recovery Automaton can noticeably improve the reliability curve. While the two examples considered two separate use cases, one where the strategy had to adapt dynamically to events and one where the strategy had to provide an optimized ordering for spares, complex combinations of these use cases are of course also possible and can be handled in the NdDFT model.

In the future we want to consider that not all basic events may be observable and instead that we can only react to events for which we have monitors that can indeed observe them. Also, we would like to examine expanding the set of possible recovery actions, for example by including notions of repair, such as restarts, or mode changes, such as switching to a Safe Mode.

Another interesting direction for research is the consideration how the synthesized Recovery Automata can be employed to improve existing system design and further improve redundancy concepts. For example by simplifying redundancy concepts without reducing reliability outside of a given margin and without breaking potential other additional FDIR requirements such as having a 3-fault tolerant system remain 3-fault tolerant.

References

[1]International Electrotechnical Commission et al. International standard iec 61025: Fault tree analysis, 2006.

[2]Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15:29–62, 2015.

[3]A Wander and R Förstner. *Innovative fault detection, isolation and recovery strategies on-board spacecraft: State of the art and research challenges*. Deutsche Gesellschaft für Luft-und Raumfahrt-Lilienthal-Oberth eV, 2013.

[4]Joanne Bechta Dugan, Salvatore J Bavuso, and Mark A Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, 1992.

[5]Christian Eisentraut, Holger Hermanns, and Lijun Zhang. On probabilistic automata in continuous time. In *Logic in Computer Science*, pages 342–351. IEEE, 2010.

[6]Dennis Guck, Hassan Hatefi, Holger Hermanns, Joost-Pieter Katoen, and Mark Timmer. Modelling, reduction and analysis of Markov automata. In *Quantitative Evaluation of Systems*, volume 8054 of *LNCS*, pages 55–71. Springer, 2013.

[7]Marco Beccuti, Giuliana Franceschinis, Daniele Codetta-Raiteri, and Serge Haddad. Computing optimal repair strategies by means of ndrft modeling and analysis. *The Computer Journal*, 57(12):1870–1892, 2014.

[8]Daniele Codetta-Raiteri and Luigi Portinale. Dynamic Bayesian networks for fault detection, identification, and recovery in autonomous spacecraft. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):13–24, 2015.

[9]Luigi Portinale and Daniele Codetta-Raiteri. Arpha: an FDIR architecture for autonomous spacecrafts based on dynamic probabilistic graphical models. In *Proc. ESA Workshop on AI in Space@IJCAI*, 2011.

[10]Benjamin Bittner, Marco Bozzano, Alessandro Cimatti, Regis De Ferluc, Marco Gario, Andrea Guiotto, and Yuri Yushtein. An integrated process for FDIR design in aerospace. In *Model-Based Safety and Assessment*, volume 8822 of *LNCS*, pages 82–95. Springer, 2014.

[11]William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. Fault tree handbook. Technical report, DTIC Document, 1981.

[12]Ernest Edifor, Martin Walker, and Neil Gordon. Quantification of priority-or gates in temporal fault trees. In *International Conference on Computer Safety, Reliability, and Security*, volume 7612 of *LNCS*, pages 99–110. Springer, 2012.

[13]Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT press, 2008.