Synthesizing Failure Detection, Isolation, and Recovery Strategies from Nondeterministic Dynamic Fault Trees

Sascha Müller* and Andreas Gerndt†
DLR, German Aerospace Center, 38108 Brunswick, Germany
and
Thomas Noll‡
RWTH Aachen University, 52056 Aachen, Germany

DOI: 10.2514/1.I010669

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 nondeterministic dynamic fault trees is presented. It is shown how appropriate recovery strategies can be synthesized from them. This is achieved by transforming a nondeterministic dynamic fault tree into a Markov automaton. From the optimized scheduler of this Markov automaton, an optimal recovery strategy can then be derived. The model of recovery automata is also introduced to represent these strategies. Finally, how these synthesized strategies can help improve overall system reliability is discussed.

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 the fault tree analysis (FTA) [1].

The FTA is a methodology commonly used in the industry for performing state-of-the-art failure analysis [2]. The resulting fault trees (FTs) describe how faults propagate through the 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 the quantitative analysis of important computable measures such as reliability. Dynamic fault trees (DFTs) are an extension introducing temporal understanding and new features to analyze redundancy concepts known as spare management. However, there are challenges arising from nondeterministic 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 nondeterministic dynamic fault trees (NDDFTs) as an extension to DFTs. In contrast to the latter, the new NDDFT does not impose a fixed, rigid order on the spares to be used. As a next step, the methodology foresees transforming this NDDFT model into a Markov automaton (MA) that is suitable for the computation of the aforementioned nondeterministic 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 it 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 topics of FTs, MAs, and the synthesis of recovery strategies. Further background on the theory of FTs is given in Sec. III. The NDDFT model is introduced in Sec. IV. This section also describes the process of synthesizing recovery strategies from a given NDDFT as well as a model to represent such strategies. Section V then evaluates the technique on two use case examples and compares the scalability of NDDFTs to standard DFTs. Finally, the paper concludes in Sec. VI 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. Although some of the following steps may be omitted in some cases, performing FDIR generally means applying the following procedural approach [3]:

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

To derive how faults relate to each other and eventually lead to a systemwide failure, failure analysis techniques such as the 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 (BEs). They correspond to a Boolean variable in which false represents the initial state of no failure. The variable is considered

Presented as Paper 2017-5163 at the AIAA Space and Astronautics Forum and Exposition, Orlando, FL, 12–14 September 2017; received 5 June 2018; revision received 24 October 2018; accepted for publication 24 October 2018; published online 19 November 2018.

*Research Scientist, Software for Space Systems and Interactive Visualization, Lilienthalplatz 7.
†Head of Department, Software for Space Systems and Interactive Visualization, Lilienthalplatz 7.
‡Associate Professor, Software Modeling and Verification Group, Ahornstrasse 55.

true in the case of a failure event. The branches of the trees are represented by the gates performing operations on the events. FTs are directed acyclic graphs starting from the BEs pointing over the gates toward the system failure event.

One of the very basic types of FTs are static fault trees (SFTs). They employ Boolean algebra to combine various different failure events by "AND" and "OR" operations, which are 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 their expressiveness and enhance their features. A particular extension is the notion of dynamic fault trees. 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 a 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 zero. By using a dormant failure rate of zero, 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 zero 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, nonnegative real values as well as immediate, nondeterministic 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, was discussed in Ref. [6].

Computing strategies for recovery purposes from a given fault model has also been researched. In Ref. [7], a similar approach was taken for repairable fault trees. Repairable fault trees are a FT formalism in which failed basic events can be repaired. In this case, this is realized with repair boxes each of them being equipped with a repair policy that 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, the failure analysis also becomes possible, not only for permanent failures but also for transient ones. The authors consider nondeterministic repair policies in which the repair order is not fixed. Optimal repair policies are then computed by converting the repairable fault tree to a Markov decision process, which is a time-discrete version of Markov automata. However, the authors do not consider DFT models.

In Ref. [8], dynamic decision networks (DDNs) were employed and their inference capabilities were exploited to create autonomous onboard FDIR systems for spacecraft that could select reactive and preventive recovery actions during run time. In Ref. [9], the authors proposed creating the DDN from an extension of the DFT model.

Also set in the domain of space systems, timed failure propagation graphs were used in Ref. [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

Figure 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 one-vote gate corresponds to an OR gate and a k-vote gate with k inputs corresponds to an AND gate. Implementationwise, 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 in which the TLE can go from having failed to working again as new failures occur. These fault trees are known as noncoherent fault trees and have been dismissed as being a sign for modeling errors [11].

Figure 2 depicts the notation to extend SFTs to DFTs introducing new gates "priority OR" (POR), "priority AND" (PAND), spare, and functional dependency (FDEP). The PAND gate propagates in case all inputs fail in sequence from left to right. It does not propagate the failure in the case in which the sequence is not obeyed. The POR gate propagates in the case in which the leftmost input occurs before all other inputs [12]. Priority gates may usually come in two flavors: exclusive or inclusive. The inclusive PAND gate also propagates if the inputs occur simultaneously. On the other hand, the exclusive PAND gate only propagates if all inputs occur strictly after each other. Similarly, the exclusive POR gate only propagates if the leftmost input occurs strictly before all other inputs. It was shown in Ref. [13] that exclusive POR gates were expressive enough to model all priority gates. In this work, priority gates are considered to be exclusive.

The spare gate propagates a failure if the primary input fails and all spares are either claimed or fail themselves. 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 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.

Fig. 1 Gates and events in a static fault tree.

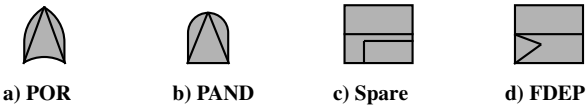


Fig. 2 Standard dynamic gates.

The FDEP has a trigger event on the left-hand side, and any number of dependent events is functionally dependent on the triggering event. When the trigger event occurs, the dependent events are set to fail as well. The output of an FDEP gate only indicates to which tree it belongs and has no further semantic meaning.

Generally, basic events will be denoted by $b_1, b_2, \ldots$, sets of basic events will be denoted by $B_1, B_2, \ldots$, and failure rates will be denoted 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.

To avoid semantic problems, a number of additional syntactical restrictions to the fault tree structure are imposed. We say that a fault tree is well formed if 1) it has exactly one root element, which is the top-level event; 2) spare subtrees do not have common child nodes with other subtrees (FDEPs, however, may have dependent events across different spare subtrees); and 3) FDEPs may have any event as a trigger event, but they may not induce loops through dependent events.

In the following, only well-formed fault trees are considered. To illustrate the DFT notation, two example DFTs will be considered now. Figure 3 shows a system consisting of two memory components that are covered by two spare memories for failures. The two spares are shared among the two spare gates. According to DFT semantics, Memory3 will be used before Memory4 in the case of a failure of Memory1 or Memory2. In addition, the system has two hot redundant, always active power sources: Power1 and Power2. Both primaries Memory1 and Memory2 are powered by Power1, and the redundancies Memory3 and Memory4 are powered by the second power source: Power2. Using FDEPs, the failure of the power sources is propagated to the memory components. Note that it would be possible to emulate FDEP1 using OR gates. But, for Power2, it is necessary to use an FDEP gate because using an OR gate would make Power2 an actual part of the spare subtrees. In the figure, FDEP-dependent events are marked by an arrow and dashed lines indicate the parent of an FDEP.

Figure 4a shows a use case for the PAND gate. The system itself is depicted in Fig. 4b 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 nonfailing 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.

IV. Nondeterministic Dynamic Fault Trees

The novel contributions of this paper begin with this section. As described in the previous sections, DFTs require that spares are activated in a fixed and rigid order. This order cannot be adapted by depending on faults that have previously occurred. Additionally, in cases of spare races, it is

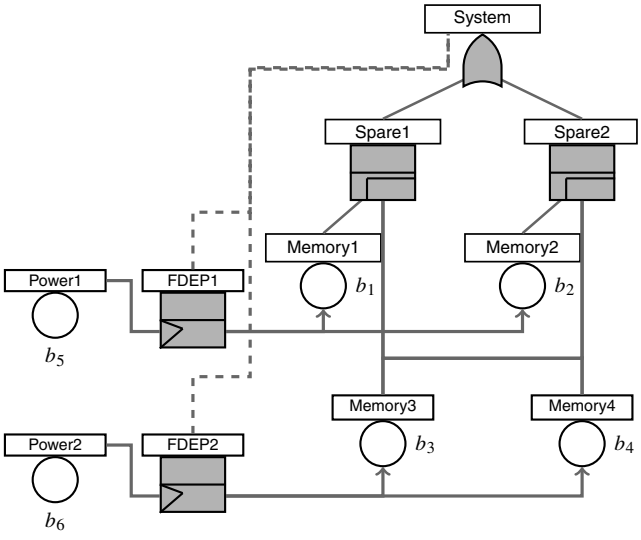
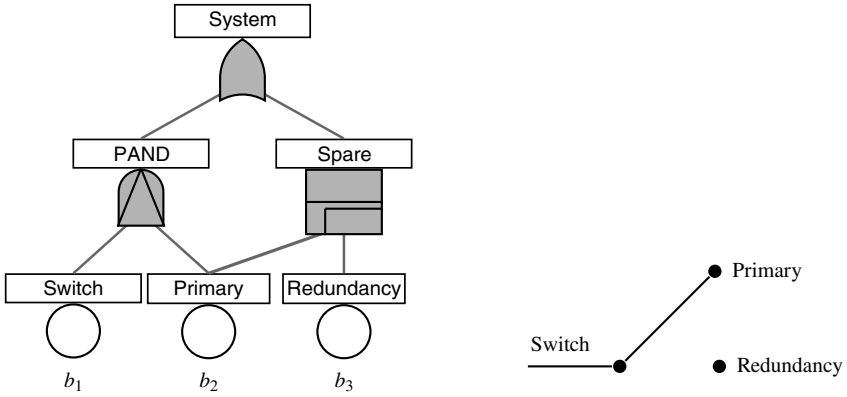


Fig. 3 Example DFT showing shared redundancies and FDEPs.



a) Spare with switch

b) A physical system with a cold spare and a switch

Fig. 4 Figure 4a shows an example DFT with a PAND gate, and Fig. 4b is the system represented by Fig. 4a.

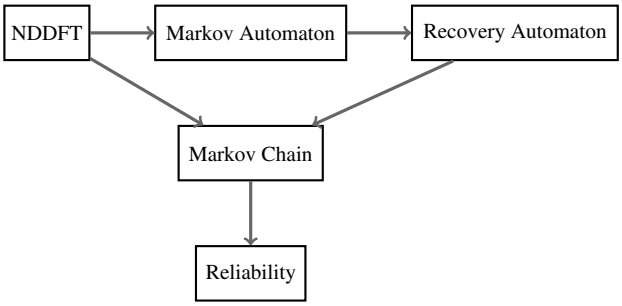


Fig. 5 Transformation road map.

not semantically clear which spare gate claims the actual redundancy. To relax on this semantic restriction of the DFT model, an inherently nondeterministic DFT model (following the naming in Ref. [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 us to derive a so-called recovery automaton. This recovery automaton provides the optimal strategy to react on failures in the NDDFT. Figure 5 visualizes the overall procedure.

A. Definition of Nondeterministic Dynamic Fault Trees

The NDDFT defined here 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 nondeterministic 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 an 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 1) [ ] (empty action); or 2) CLAIM(G, S), such that spare gate G claims spare S, where S is 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 $R(\mathcal{T})$. Furthermore, we define the set of recovery action sequences $RS(\mathcal{T}) := (R(\mathcal{T}) \setminus \{[\,]\})^*$. In the following, transitions of Markov automata will be labeled by single recovery actions and transitions of recovery automata will be labeled by recovery action sequences. Similarly, we denote the set of all nonempty subsets of basic events by $\mathrm{BES}(\mathcal{T})$. Sets of basic events instead of single events are used because FDEPs may cause several basic events to fail simultaneously. Given the observed faults (basic events), a recovery strategy is then a mapping that returns the recovery action sequence that should be taken accordingly. The NDDFT considers recovery strategies that only have recovery actions as defined in Definition 1 and is defined as follows:

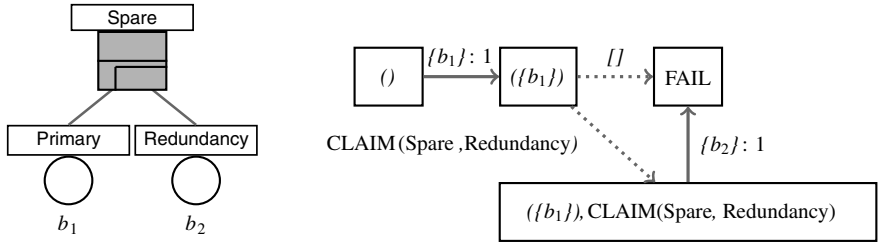
Definition 2 (Recovery Strategy): A recovery strategy for an NDDFT $\mathcal{T}$ is a mapping $\mathrm{Recovery}\colon \mathrm{BES}(\mathcal{T})^* \to RS(\mathcal{T})^*$ such that 1) $\mathrm{Recovery}(\varepsilon) = \varepsilon$; and 2) $\mathrm{Recovery}(B_1, \ldots, B_n) = \mathrm{Recovery}(B_1, \ldots, B_{n-1}), rs_n$ for some $rs_n \in RS(\mathcal{T})$.

As each basic event can occur (at most) once, the recovery strategy only needs to be defined for pairwise disjoint sets of basic events, i.e., $B_i \cap B_j = \emptyset$ for $i \neq j$. Later, this will ensure that each recovery strategy can be represented by a finite-state automaton that only accepts finite traces.

B. Transformation of NDDFTs into Markov Automata

In this setting, Markovian transitions are used to represent basic event transitions, with the transition rate being the failure rate of a basic event. The nondeterministic transitions are used to 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 the base algorithm, we use the one given in Ref. [4]. The adapted algorithm operates by memorizing two sets of data in every one of its states: the first is the history of the occurred basic event sets $(B_1, B_2, \ldots, B_n)$, and the second is 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 (i.e., those that are not associated to an unactivated spare) are used to compute successors for each of them while extending the history accordingly. The respective basic event set is obtained by taking the active basic event and computing all basic events that transitively fail due to FDEPs. The transitions are labeled with the respective failure rate of the basic event causing the transition. For further computation purposes, they are also labeled with the basic event set. A special state of "FAIL" is added as well. All transitions that would lead to a state that implies that the top-level event (system failure) has occurred are then connected to the FAIL state instead. After performing a Markovian transition to a next state, the algorithm generates successors using nondeterministic transitions. Each nondeterministic transition is labeled by a valid recovery action. The algorithm updates the mapping of claimed spares accordingly. This yields an MA with an alternating Markovian and sequences of nondeterministic transitions, which are, respectively, labeled by basic event sets with the failure rates of the failing basic events 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 [14]) state and reduces the state space accordingly.

Figure 6 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). The active failure rates of the basic events are $F_A(b_1) = F_A(b_2) = 1$, and the dormant failure rates are $F_D(b_1) = F_D(b_2) = 0$. Note that b_2 is initially dormant. Hence, the initial state ( ) only contains $\{b_1\}$: 1 as a successor transition. Dotted lines represent the nondeterministic transitions, and solid lines represent the Markovian transitions. B: λ denotes that the basic event set B occurs (actively or dormant) with rate λ. 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 $\mathrm{Recovery}(\{b_1\}) = \mathrm{Claim}(\mathrm{Spare}, \mathrm{Redundancy})$.



a) NDDFT **b) Markov automaton**

Fig. 6 Example transformation of NDDFT to MA.

C. Synthesizing Recovery Automata Based on Markov Automata

Using existing techniques for optimizing the scheduling of a Markov automaton, it is possible to determine which state should be chosen to maximize the reliability of the system for each state with outgoing nondeterministic transitions. To extract a recovery strategy from a scheduler for a Markov automaton, a formal way to represent 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 event sets and outputs its next state as well as the recovery action sequence that should be taken. In this sense, it is basically a Mealy automaton, with the input alphabet being the power set of basic events $\mathrm{BES}(\mathcal{T})$ and the output alphabet the set of recovery action sequences $RS(\mathcal{T})$. The notion of these automata models is formalized with the following definition:

Definition 3 (recovery automaton): A recovery automaton $\mathcal{R}_\mathcal{T} = (Q, \delta, q_0)$ of an NDDFT $\mathcal{T}$ is an automaton in which 1) Q is a finite set of states, 2) $q_0 \in Q$ is an initial state, and 3) $\delta\colon Q \times \mathrm{BES}(\mathcal{T}) \to Q \times RS(\mathcal{T})$ 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.

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 sequences of transitions for states $s_0, s_1, \ldots, s_n$ of the form

$$(s_0, B\colon \lambda, s_1), (s_1, r_1, s_2), \ldots, (s_{n-1}, r_n, s_n)$$

where B is a basic event set, λ is a failure rate, and $r_1, \ldots, r_n$ are recovery actions, by the transition $\delta(s_0, B) = (s_n, r_1, \ldots, r_n)$, where empty recovery actions are ignored. This applies to all transitions in which $s_1, \ldots, s_n$ are the successors computed by the optimized schedule of the Markov automaton. All other nondeterministic transitions are then discarded. Observe that multiple recovery actions from the Markov automaton

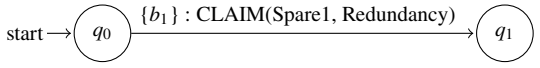


Fig. 7 Example of a simple recovery automaton.

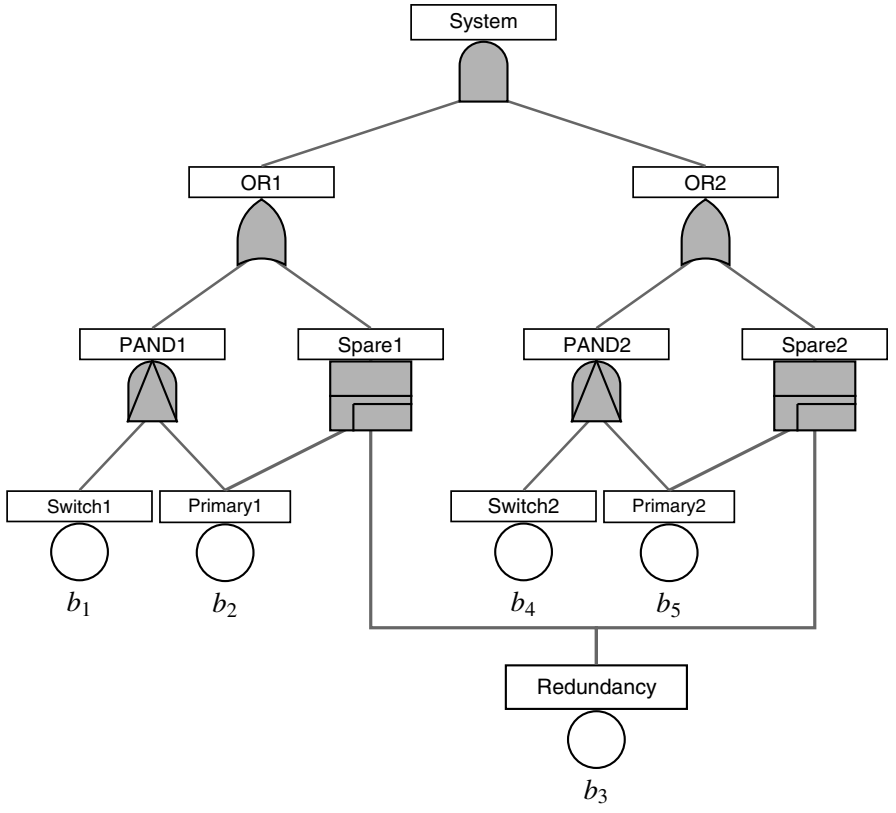


Fig. 8 DFT with a shared redundancy and two switches.

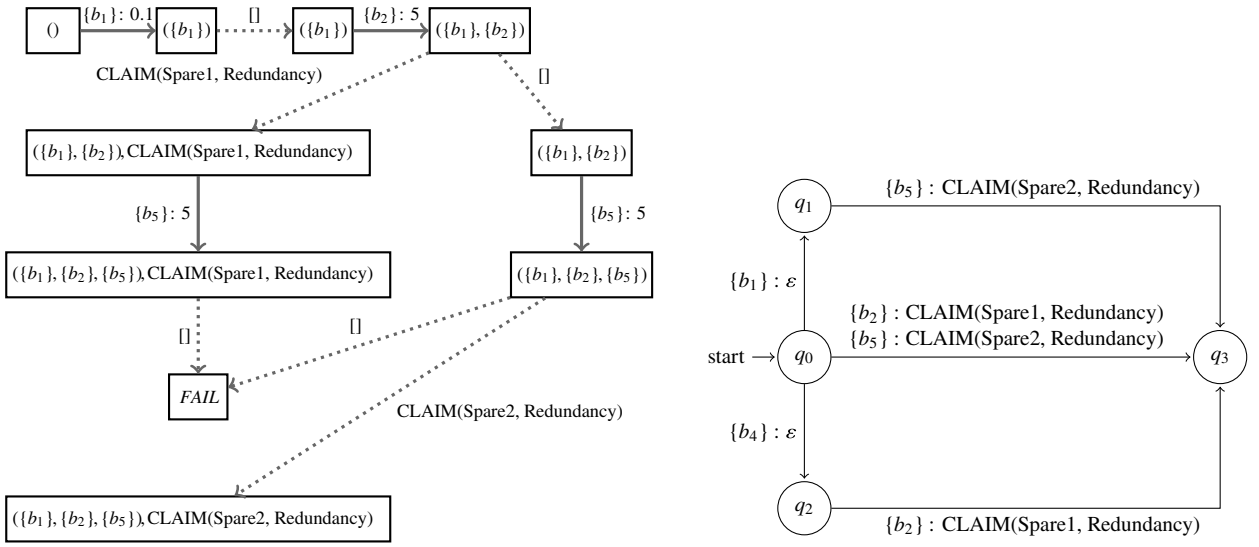
are combined into one recovery action sequence in the recovery automaton. Finally, the algorithm discards all unreachable states and minimizes the resulting automaton for a more compact representation. A simple example of a recovery automaton for the NDDFT in Fig. 6 inducing the previously described strategy Recovery is shown in Fig. 7.

V. Results

A prototype of the described synthesis algorithm has been implemented within the Virtual Satellite 4.0 framework [15], which is a tool intended for performing model-based systems engineering for the whole life cycle of space systems. Together with the synthesis algorithm, the implementation also features modeling of NDDFTs and recovery automata and, finally, computing metrics such as reliability and the mean time to failure (MTTF). As input language, we support the Galileo file format [16], which can also be used to describe NDDFTs because they are syntactically the same as DFTs. All reliability datasets and synthesized recovery automata showcased in the examples in the following sections are generated using this prototypical implementation.

A. Example Construction of an Adaptable Recovery Strategy

To illustrate the construction, in the following, we consider an example NDDFT and the resulting recovery automaton. Then, we compare the reliability of the DFT version with the NDDFT with recovery automaton version. To demonstrate the state space construction, a small excerpt of the constructed Markov automaton is also shown. As an example NDDFT considers the model shown in Fig. 8, which is similar to the one depicted in Fig. 4a, except that, now, two pieces of equipment are dependent on the respective switches activating their spares. The spare redundancy is shared among the two spare gates: SPARE1 and SPARE2. Should both subsystems fail, then the entire system fails.



a) Excerpt from the Markov automaton for the NDDFT in Fig. 8 for the fault sequence ($\{b_1\}$,$\{b_2\}$,$\{b_5\}$)

b) Synthesized recovery automaton for the switch system

Fig. 9 Example extraction of recovery automaton.

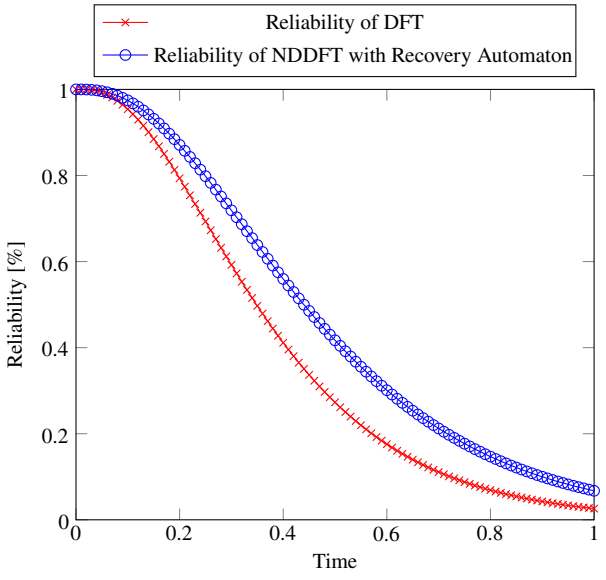


Fig. 10 Reliability of DFT vs reliability of NDDFT with recovery automaton.

Table 1 DFT vs NDDFT with recovery automaton

Metric	DFT	NDDFT	Factor
MTTF	0.38	0.47	1.24
No. of states	109	149	1.43
No. of transitions	146	226	1.55

Figure 9b depicts a recovery automaton that we have computed for the NDDFT of Fig. 8. Because 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 the basic event set B that occurs, and the recovery action r should be taken. For the failure rates, we consider the (unitless) values 1) $F(b_2) = F(b_5) = F_A(b_3) = 5$ (equal active failure rates for all equipments), 2) $F_D(b_3) = 0$ (the spare is a cold redundancy), and, 3) $F(b_1) = F(b_4) = 0.1$ (low switch failure rate).

Figure 9a showcases a small excerpt of the constructed Markov automaton. In the recovery automaton, we can find the corresponding fragment in states q_0 and q_1. In essence, the synthesized recovery automaton states that the spare gates SPARE1 and SPARE2 should not allocate the redundancy if the respective switch has already failed.

Figure 10 shows the reliability when considering the fault tree in Fig. 8 as a standard DFT, as well as when considering it as an NDDFT and equipping it with the recovery automaton from Fig. 9b. For the timeframe, a unitless mission time of one is chosen. We can see that both fault tree reliabilities converge toward zero. However, employing an adaptive strategy for activating spares yields a reliability curve that is consistently better than the fixed-order strategy of the standard DFT. Further data on the difference between the two semantics are given in Table 1. The increase in reliability leads to a total increase in the mean time to failure of about 24%. On the other hand, applying the NDDFT semantics brings about an increase of state space and transition count by 43 and 55%, respectively.

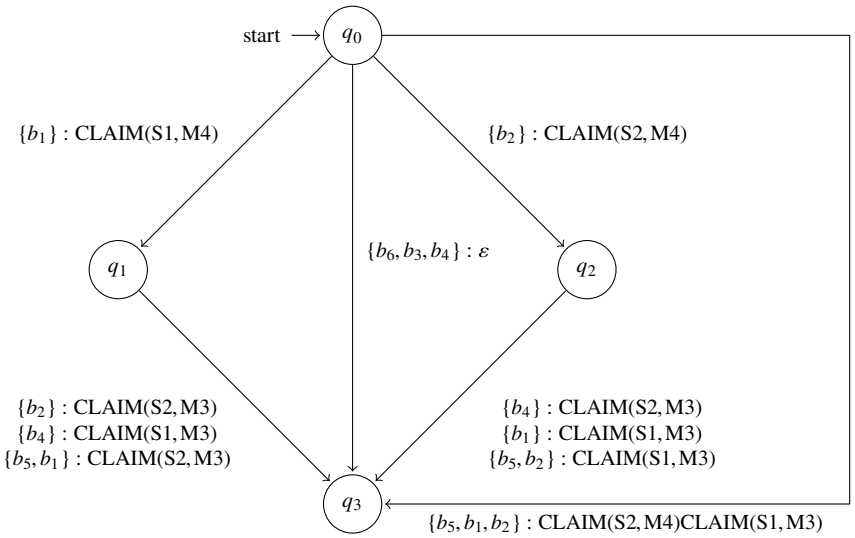


Fig. 11 Synthesized recovery automaton for memory system.

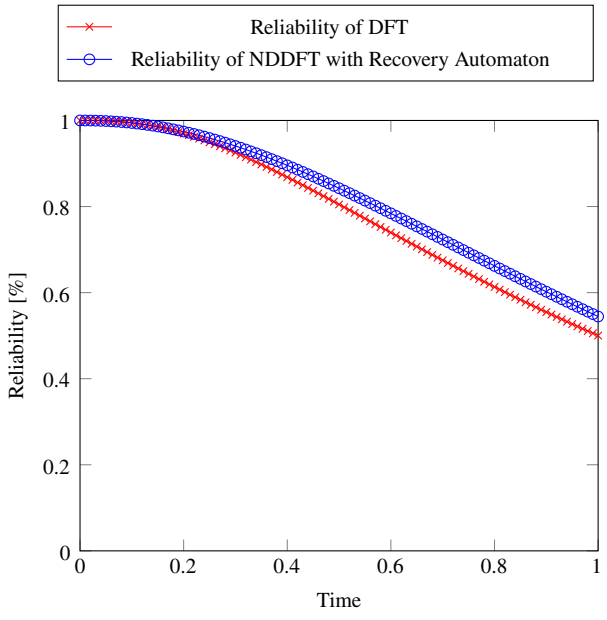
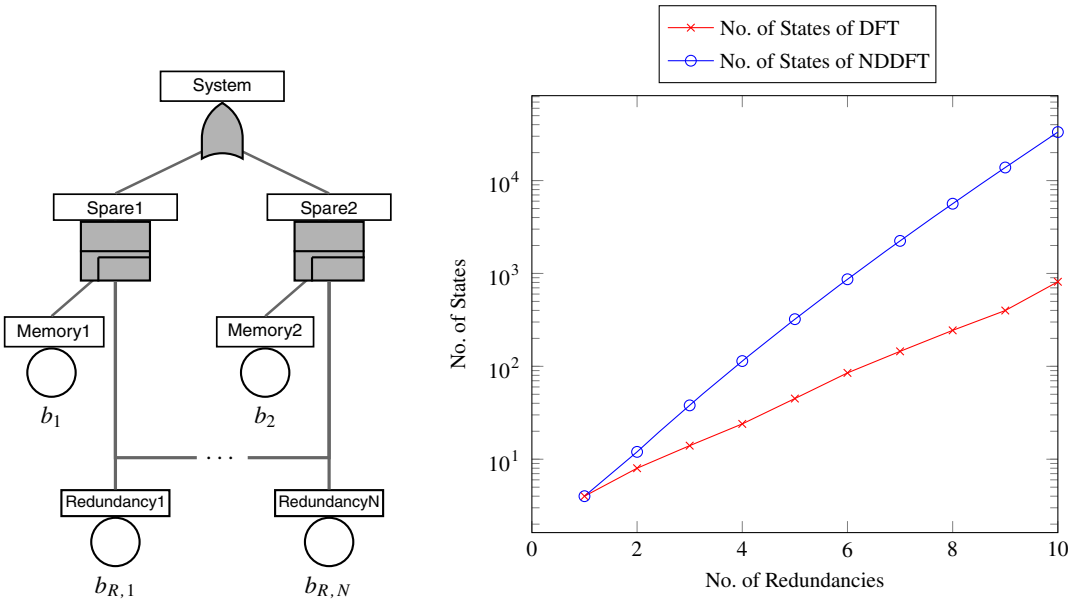


Fig. 12 Reliability of DFT vs reliability of NDDFT with recovery automaton for memory system.

Table 2 DFT vs NDDFT with recovery automaton for memory system

Metric	DFT	NDDFT	Factor
MTTF	1.09	1.19	1.09
No. of states	18	24	1.33
No. of transitions	54	97	1.8



a) Memory system with N redundancies **b) State-space growth**

Fig. 13 State-space growth of a memory system with N redundancies.

B. Example of Optimized Spare Ordering

As a second example use case, we reconsider the redundant memory system shown in Fig. 3. 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 1) $F(b_1) = F(b_2) = 1$, 2) $F_A(b_3) = 5$ (modeling a low-quality spare), 3) $F_A(b_4) = 0.5$ (modeling a high-quality spare), 4) $F_D(b_3) = F_D(b_4) = 0$ (the spares are cold redundancies), and 5) $F(b_5) = F(b_6) = 0.1$ (always modeling active power sources with a low failure rate).

Hence, according to DFT semantics, the low-quality spare will always be activated first.

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. 11. It basically states that the system should always first activate Memory4 (that is, the high-quality spare) before the low-quality spare Memory3.

To evaluate the recovery strategy induced by the recovery automaton, the reliability curves of the fault tree models are plotted in Fig. 12. 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. Although, 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. Additional data on the details are given in Table 2. The improvement of the reliability curve yields a slight increase in the mean time to failure of about 9%. The state space increases slightly by 33%. The transition count suffers a significant increase by about 80%.

C. Impact on Scalability

To assess how the NDDFT semantics impact the state-space growth as compared to the standard DFT semantics, we reconsider a family of DFTs based on the prior memory system use case. The model family is depicted in Fig. 13a. As before, the system consists of two main memory units: Memory1 and Memory2. They possess a shared pool of redundancies of size N. The failure rates are simply chosen as $F(b_1) = F(b_2) = F_A(b_{R,i}) = 1$ and $F_D(b_{R,i}) = 0$ for any $1 \leq i \leq N$.

Figure 13b shows how the state-space size increases with varying N for both the DFT and the NDDFT semantics, respectively. Note that the y axis is scaled logarithmically. It can be seen that both semantics suffer from exponential state-space explosion as the number of available redundancies goes up. The NDDFT semantics, however, pay a price for achieving better reliability results by producing a state space that is up to 40 times larger than the DFT counterpart.

VI. Conclusions

In this paper, the problem of synthesizing recovery strategies was considered by focusing on the aspect of scheduling spare activations and resolving spare races. For this purpose, the formalism of nondeterministic Dynamic fault trees was introduced and motivated. An algorithm for converting nondeterministic dynamic fault trees (NDDFTs) to Markov automata was given, and it was discussed how recovery strategies could be extracted from the optimized scheduler of the generated Markov automaton.

Additionally, a representation for recovery strategies in the form of the also newly introduced model of the recovery automata was given. Furthermore, two examples were considered on how equipping an NDDFT with a recovery automaton could noticeably improve the reliability

curve. One use case was adapted dynamically to events, whereas the other provided an optimized ordering for spares. Complex combinations of these use cases were, of course, also possible; and they could be handled in the NDDFT model. Finally, an assessment was made of how the state-space growth was affected by the NDDFT semantics. It is hoped that the burden imposed by the NDDFT semantics could be lessened in future work.

In the future, consideration should be made about how not all basic events may be observable. Instead, only events for which there are monitors that can indeed observe them should be reacted to. Also, the set of possible recovery actions should be expanded: for example, by including notions of repair, such as restarts; or mode changes, such as deciding the point in time when a satellite should switch to a safe mode.

Another interesting direction for research is the consideration of 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 ϵ margin and without breaking potential additional failure detection, isolation, and recovery requirements, such as having a three-fault-tolerant system remain three-fault tolerant.

References

[1] "Fault Tree Analysis (FTA)," International Electrotechnical Commission, International STD IEC 61025, Geneva, 2006.

[2] Ruijters, E., and Stoelinga, M., "Fault Tree Analysis: a Survey of the State-of-the-Art in Modeling, Analysis and Tools," *Computer Science Review*, Vols. 15–16, Feb.–March 2015, pp. 29–62. doi:10.1016/j.cosrev.2015.03.001

[3] Wander, A., and Förstner, R., "Innovative Fault Detection, Isolation and Recovery Strategies on-Board Spacecraft: State of the Art and Research Challenges," *Deutscher Luft- und Raumfahrtkongress 2012*, German Soc. for Aeronautics and Astronautics—Lilienthal-Oberth e.V., Bonn, Germany, 2013, https://www.dglr.de/publikationen/2013/281268.pdf.

[4] Dugan, J. B., Bavuso, S. J., and Boyd, M. A., "Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems," *IEEE Transactions on Reliability*, Vol. 41, No. 3, 1992, pp. 363–377. doi:10.1109/24.159800

[5] Eisentraut, C., Hermanns, H., and Zhang, L., "On Probabilistic Automata in Continuous Time," *Logic in Computer Science*, IEEE Publ., Piscataway, NJ, 2010, pp. 342–351. doi:10.1109/LICS.2010.41

[6] Guck, D., Hatefi, H., Hermanns, H., Katoen, J.-P., and Timmer, M., "Modelling, Reduction and Analysis of Markov Automata," *Quantitative Evaluation of Systems*, Lecture Notes in Computer Science Series, Vol. 8054, Springer, New York, 2013, pp. 55–71. doi:10.1007/978-3-642-40196-1_5

[7] Beccuti, M., Franceschinis, G., Codetta-Raiteri, D., and Haddad, S., "Computing Optimal Repair Strategies by Means of NdRFT Modeling and Analysis," *Computer Journal*, Vol. 57, No. 12, 2014, pp. 1870–1892. doi:10.1093/comjnl/bxt134

[8] Codetta-Raiteri, D., and Portinale, L., "Dynamic Bayesian Networks for Fault Detection, Identification, and Recovery in Autonomous Spacecraft," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 45, No. 1, 2015, pp. 13–24. doi:10.1109/TSMC.2014.2323212

[9] Codetta-Raiteri, D., and Portinale, L., "Arpha: an FDIR Architecture for Autonomous Spacecrafts Based on Dynamic Probabilistic Graphical Models," *Proceedings of ESA Workshop on AI in Space@IJCAI*, Computer Science Inst., Università del Piemonte Orientale, TR-INF-2010-12-04-UNIPMN, Vercelli, Italy, Dec. 2010, http://www.di.unipmn.it/TechnicalReports/TR-INF-2010-12-04-UNIPMN.pdf.

[10] Bittner, B., Bozzano, M., Cimatti, A., De Ferluc, R., Gario, M., Guiotto, A., and Yushtein, Y., "An Integrated Process for FDIR Design in Aerospace," *Model-Based Safety and Assessment*, Lecture Notes in Computer Science Series, Vol. 8822, Springer, New York, 2014, pp. 82–95. doi:10.1007/978-3-319-12214-4_7

[11] Vesely, W. E., Goldberg, F. F., Roberts, N. H., and Haasl, D. F., "Fault Tree Handbook," Nuclear Regulatory Commission TR TRN: 82-003645, Washington, D.C., 1981, https://www.osti.gov/biblio/5762464-fault-tree-handbook.

[12] Edifor, E., Walker, M., and Gordon, N., "Quantification of Priority-OR Gates in Temporal Fault Trees," *International Conference on Computer Safety, Reliability, and Security*, Lecture Notes in Computer Science Series, Vol. 7612, Springer, New York, 2012, pp. 99–110. doi:10.1007/978-3-642-33678-2_9

[13] Junges, S., Guck, D., Katoen, J.-P., and Stoelinga, M., "Uncovering Dynamic Fault Trees," *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE Publ., Piscataway, NJ, 2016, pp. 299–310. doi:10.1109/DSN.2016.35

[14] Baier, C., and Katoen, J.-P., *Principles of Model Checking*, MIT Press, Cambridge, MA, 2008, pp. 808–816.

[15] Lange, C., Grundmann, J. T., Kretzenbacher, M., and Fischer, P. M., "Systematic Reuse and Platforming: Application Examples for Enhancing Reuse with Model-Based Systems Engineering Methods in Space Systems Development," *Concurrent Engineering*, Vol. 26, No. 1, Nov. 2017, pp. 77–92. doi:10.1177/1063293X17736358

[16] Dugan, J. B., Sullivan, K. J., and Coppit, D., "Developing a Low-Cost High-Quality Software Tool for Dynamic Fault-Tree Analysis," *IEEE Transactions on Reliability*, Vol. 49, No. 1, 2000, pp. 49–59. doi:10.1109/24.855536

M. J. Kochenderfer
Associate Editor