Deliverable D2.2

Inventory and Specification: Extended Fault Trees

Deliverable

Number and title:	D2.2 – Inventory and Specification: Extended Fault Trees		
Work package:	WP2 (Reliability and Resilience)		
Lead author:	Thomas Noll (RWTH)		
Contributors:	Matthias Volk (UT), Joost-Pieter Katoen (RWTH)		
Reviewers:	Andrea Turrini (IISG), Leonardo Diamonte (INVAP)		
Due date:	M17 (2023-02-28)	*Dissemination level:*	Public
		Version:	1.0 (final)
Submission date:	2023-02-24	*Pages:*	33

Version history

Version	*Date*	*Notes*
1.0	2023-02-24	First official release

Project

Title:	Models in Space Systems: Integration, Operation, and Networking		
Acronym:	MISSION	*Start date:*	01-10-2021
GA no.:	101008233	*Duration:*	48 months
Call:	H2020-MSCA-RISE-2020	*Website:*	mission-project.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

1 Introduction

Assessing and ensuring Reliability, Availability, Maintainability and Safety (RAMS) is an important task 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 failures, wrong sensor readings, external interferences and many more. To raise trust in handling system failures, reliability engineering tries to embed Fault Detection, Isolation and Recovery (FDIR) concepts. These concepts are derived using various tools and methodologies such as Fault Tree Analysis (FTA) [18], which is the main formalism for failure modelling adopted in the MISSION project.

The purpose of this report, which is allocated to Work Package 2 (Reliability and Resilience), is to assemble a survey of Fault Tree extensions with repair and maintenance aspects, to investigate issues concerning semantics, and to define a modular formal semantics onto an appropriate automata model. It aims to serve as the basis for the integration in the tooling environment to be developed in Work Package 4 (Integration and Validation).

This document is organised as follows. Section 2 starts with an overview of (Static) Fault Trees and their extension with dynamic elements as well as repair and maintenance aspects. Our proposed semantic model, Generalised Stochastic Petri Nets (GSPNs), is introduced in Section 3. In Section 4 we develop our compositional GSPN semantics of Dynamic Fault Trees and then extend it by repair operations in Section 5. Finally, the report concludes in Section 6.

2 Extended Fault Trees: An Overview

Fault Tree Analysis (FTA) is a methodology commonly used in industry for performing state-of-the-art failure analysis [25]. The resulting Fault Trees describe how faults propagate through components and subsystems of a system and eventually lead to a top-level system failure. They enable various forms of both qualitative and quantitative analyses. The former aim to identify fault combinations that lead to system failure. The latter are based on assigning failure probabilities to components, which allows for the computation of important RAMS metrics such as overall reliability or mean time to failure (MTTF).

In the following we provide a survey of related work on (Static) Fault Trees and their extension with dynamic elements as well as repair and maintenance aspects, putting the emphasis on approaches with a solid formal foundation.

2.1 Static Fault Trees

The most basic case of a Fault Tree is known as a *Static Fault Tree (SFT)*, which models how failures of (sub-)components propagate through the system and eventually lead to a system failure. The creation of a Fault Tree follows a top-down approach. It starts with the *top-level event (TLE)*, which represents the failure of the complete system. The system failure is subdivided into failures of sub-components, which can then be further subdivided. This hierarchical approach is continued down to the desired level of detail. The leaves in the resulting tree represent components which are not further decomposed, so-called *basic events (BEs)*. A basic event fails according to its associated probability distribution, if given.

If a BE fails, the failure is propagated upwards through the Fault Tree. The intermediate nodes – so-called *gates* – fail themselves if their failure condition is satisfied. For instance, gates of type AND fail if all of their sub-components have failed whereas gates of type OR fail if at least one of their sub-components has failed. The semantics of Static Fault Trees is therefore clearly defined as a Boolean function over the BEs encoding the system's failure conditions. For a detailed overview of different analysis techniques for Fault Trees, we refer to [25].

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

2.2 Dynamic Fault Trees

While Static Fault Trees are widely used in practice, they are not equipped to faithfully model more complex – dynamic – behaviour as it is usually present in modern systems. For this reason, they have been extended to *Dynamic Fault Trees (DFTs)*, which incorporate temporal aspects and new features to analyse both spare management and temporal as well as functional dependencies [12]. More concretely, DFTs introduce several new types of gates such as

- priority-AND (PAND) and sequence enforcer (SEQ) for order-dependent failures,
- SPARE for spare management and activation, and
- FDEP for functional dependency.

These gates increase the expressiveness and allow for more faithful modelling compared to the basic (static) formalism. However, this advantage comes at the price of raising semantic problems – the dynamic nature of the additional gates makes a clear definition of the semantics more complicated [11]. While each dynamic gate on its own seems to have a concise definition, the interplay between different gates leads to intricate issues which are often overlooked and not properly defined in the literature [19]. An example is the unclear behaviour of a PAND-gate if multiple sub-components fail at the same time. Another example are spare races, which occur when several failed components simultaneously compete for a common set of spares.

These semantic issues are addressed in the work presented in [20, 26], which aims to capture and compare the existing DFT semantics. To this end, it introduces a thorough semantic of DFTs in terms of *Generalised Stochastic Petri Nets (GSPNs)* [1]. This is achieved in a modular way by defining the behaviour of each element in a given DFT independently in terms of a GSPN. Composing the individual GSPNs then yields a complete GSPN representing the behaviour of the overall DFT. Since this constitutes a very systematic and flexible approach, we also choose it as the basis for our semantic model (cf. Section 3). As we will see in Section 5, GSPNs are also appropriate for giving meaning to Fault Trees extended by repair and maintenance aspects.

2.3 Fault Trees with Repair

Another important aspect of (Dynamic) Fault Trees with practical relevance is repairability. In standard Fault Trees, fault events may only occur once and cannot be undone, i.e., faults are considered to be persistent. This makes it challenging to model transient failures and recovery by repair or replacement. *Repairable Fault Trees (RFTs)* aim to fill this gap. For a survey giving further insight into state-of-the-art techniques and model extensions for Fault Trees, also covering Repairable Fault Trees, we refer the interested reader to [25]. Most of our citations and explanations on RFT approaches are taken from this source.

Single-Component Repairs. The simplest way of incorporating repair aspects into DFTs is to equip BEs with additional *repair distributions*, similar to failure distributions. Repair times are often exponentially distributed and can thus be specified using a repair rate. In [8], Boudali et al. describe an approach to analyse DFTs with repair rates using Input/Output Interactive Markov Chains (I/O-IMCs). First, DFTs are converted into I/O-IMCs, which are then modified in order to model repairs of single components. However, details on the semantics for repairable elements are only given for AND gates and BEs that can only fail if they are activated. The repair delay is modelled by a Markovian transition with a rate corresponding to the repair rate. The impact of a repair operation on an AND gate is explained by means of a system composed of an AND gate with two BEs. However, sometimes this simple model that only considers repairs of single components is not sufficient.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

Another semantic model is employed by Mertens in [21], who presents compositional semantics for RFTs in terms of GSPNs by extending the framework proposed in [20]. Details will be considered in Section 5.

Multi-Component Repairs. In more advanced approaches, so-called *repair blocks* or *repair boxes (RBs)* are introduced which represent a repair involving multiple components. RBs can be linked to any element of a Fault Tree and are activated if the corresponding event occurs, i.e., if the element fails.

Bobbio et al. [7] introduce repair boxes which can be connected to a gate, and which begin repairs on the BEs of the sub-tree rooted at that gate when the gate fails. Codetta-Raiteri et al. [10] extend these repair boxes to allow different repair policies to be used in the model. In this formalism, each BE has a failure rate, which is the parameter of an exponential distribution that determines the time until the component fails. Each RB is connected to one or more BEs to repair, and one incoming BE or gate. When the incoming event occurs, the repair box is activated and initiates repairs on the outgoing components according to the repair policy. Every component also has a repair rate, which is the parameter of another exponential distribution modelling the time required to repair the component. Repair policies can be very simple, even equivalent to the simple repair rates model, or more complex, for example restricting the number of components that can be repaired simultaneously. The major advantage of this approach is that it allows modelling of more realistic systems, and analysis of what repair strategies are best. A disadvantage is that the resulting RFTs cannot be quantitatively analysed using combinatorial methods. Flammini et al. [14] add the possibility of giving priority to the repair of certain components, based on the repair rate, failure rate, or level of redundancy of the components. Other priority schemes can also be implemented within this system.

A different extension is provided by Beccuti et al. [4, 6] by adding non-determinism to the repair policies. This models cases where, for example, a mechanic individually decides which component to repair first. Conversion to Markov Decision Processes (MDPs) allows to automatically derive optimal repair policies from the Fault Tree when costs of unavailability, failures, and repairs are provided. A parametric version [5] of the formalism allows for more efficient modelling and analysis if the Fault Tree contains redundant sub-trees that differ only in the parameter values of the BEs.

In [15], Franceschinis et al. introduce RFTs which extend the Static Fault Tree model by adding nodes that correspond to RBs. They can be connected to elements in the RFT and are activated if the corresponding element fails, and can implement different repair policies. One repair strategy, the complete repair strategy, is explained in more detail. It includes the repair of the failed element as well as the repair of all its causes, i.e., the sub-components. Temporal behaviour is modelled by equipping the BEs with repair distributions that specify the time needed to eliminate the failure and by varying the number of timed transitions that represent a repair strategy to model parallel or sequential timing. As the underlying semantic model, RFTs are transformed into GSPNs by first generating the Fault Tree and then adding the repair blocks.

In [9], Codetta-Raiteri describes how several extensions for Fault Trees, including dynamic gates and repair boxes, can be integrated into Fault Trees. In this formalism, an RB is connected to a set of BEs and to a trigger whose failure leads to the activation of the RB. The RB solely eliminates the failures of the BEs it is directly assigned to. However, the repair of these BEs may lead to an indirect repair of other elements. Regarding the definition of dynamic gates with repairs, priority-AND, sequence-enforcer, and SPARE gates are considered. Repairs of BEs triggering functional dependencies are not supported. Later work by Monti et al. [22] introduces a compositional semantics for this model in terms of Input/Output Stochastic Automata (IOSA), which allows for the modelling of events occurring according to general continuous distributions.

Maintenance. While we only consider repair operations, a distinction between *preventive* and *corrective* maintenance is drawn by Guck et al. in [16, 24]. The former describes the process of inspecting

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

a component in order to put it in a better condition, whereas the latter refers to repairing it. In order to reflect these variants, additional types of BEs are introduced, namely those that are maintainable, repairable, or both. Moreover inspection modules, which determine the interval in which the health status of BEs is inspected, and repair units, which implement the corrective maintenance and which correspond to the RBs described before, are added. Altogether, this yields so-called *Fault Maintenance Trees*, which enable maintenance strategies to be directly defined on the level of the Fault Tree.

3 Generalised Stochastic Petri Nets

Petri Nets [23] are commonly used to model concurrent systems. Their extension in the form of *Generalised Stochastic Petri Nets (GSPNs)* was first introduced in [2]. This section gives a brief introduction to GSPNs as formalised in [13].

GSPNs are Petri nets with both timed and immediate transitions. In our setting, the former are used to model the occurrence of basic events in DFTs, while the latter represent the instantaneous failure propagation within DFT gates. Inhibitor arcs prevent transitions from firing repeatedly. We use them to model that components do not fail repeatedly. Transition weights allow to resolve possible non-determinism. Priorities will be (as explained later) the key to distinguish the different DFT semantics; they control the order of transition firings for, e.g., the failure propagation in DFTs. Finally, partitions of immediate transitions allow for a flexible treatment of non-determinism.

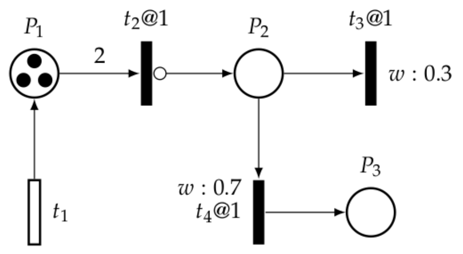
Definition 1 (GSPN)**.** *A* Generalised Stochastic Petri Net (GSPN) *extends a Petri net and is given as a 10-tuple* $\mathcal{G} = (P, T, I, O, H, m_0, W, \Pi Dom, \Pi, \mathcal{D})$ *where*

- *P is a finite set of places;*
- *$T = T_i \cup T_t$ is a finite set of transitions, where T_i is the set of immediate transitions and T_t is the set of timed transitions;*
- *$I, O, H : T \to M$ respectively define the input places, output places and inhibition places of the transitions where $M = P \to \mathbb{N}$ denotes the set of markings. For all places $p \in P$, we let $H(t)(p) = \infty$ if p is not an inhibitor place of t;*
- *$m_0 \in M$ is the initial marking;*
- *$W : T \to \mathbb{R}_{>0}$ defines the weights of the transitions;*
- *ΠDom is the priority domain;*
- *$\Pi : T \to \Pi Dom$ defines the transition priorities; and*
- *$\mathcal{D} \subseteq 2^{T_i}$ is a partition of the immediate transitions.*

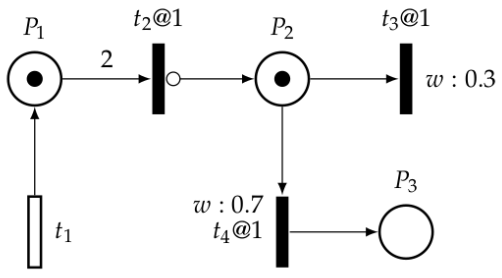
Following [2], we assume that timed transition have priority zero, i.e., $\Pi(t) = 0$ for all $t \in T_t$, whereas immediate transitions have a non-zero priority, i.e., $\Pi(t) > 0$ for all $t \in T_i$.

In the graphical representation of a GSPN, we use circles to depict places, and solid and empty bars to depict immediate and timed transitions, respectively. Tokens in a place – constituting a marking – are indicated by bullets in the corresponding circle. Input arcs are displayed with an arrow from the place to the transition whereas output arcs are displayed with an arrow from the transition to the place. Inhibitor arcs are depicted by an arc from the place to the transition with a small circle as arc-head. Arc-multiplicities greater than one are indicated by a small number near the arc. We display transition weights with the prefix w and transition priorities with the prefix $@$. We often omit the weight for a transition t if $W(t) = 1$. The partitioning is not depicted but given separately.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.



(a) GSPN with enabled transition t_2



(b) GSPN after firing of t_2

Figure 1: GSPN example

Example 2. *Consider the GSPN depicted in Figure 1a. The set of places corresponds to $\{P_1, P_2, P_3\}$, and the set of transitions equals $\{t_1, t_2, t_3, t_4\}$, where t_1 is the only timed transition.*

Since no arc is directed to t_1, we have $I(t_1)(p) = 0$ for all places p. The same applies to the inhibition function. The directed arc from t_1 to P_1 means $O(t_1)(P_1) = 1$. For all other places $p \neq P_1$, we have $O(t_1)(p) = 0$. Note that the arc between t_2 and P_2 is both an output arc from t_2 to P_2 (as indicated by the arrowhead) and an inhibitor arc from P_2 to t_2 (as indicated by the circle). The cardinality above the arc from place P_1 to transition t_2 induces that $I(t_2)(P_2) = 2$. The remaining cases work similarly and are therefore omitted.

We have $\Pi(t) = 1$ for all immediate transitions t. The priority of t_1 is omitted, but we know by definition that $\Pi(t_1) = 0$. Furthermore, the depicted weights for transitions t_3 and t_4 imply that $W(t_3) = 0.3$ and $W(t_4) = 0.7$.

A transition has concession if the tokens in each place satisfy the conditions imposed by the input-, output- and inhibition arcs. From all transitions with concessions, those with highest priority are enabled. Firing an enabled transition consumes tokens in the input places and places tokens in the output places.

Definition 3 (Concession, enabledness, firing)**.** *Let $\mathcal{G}$ be a GSPN as introduced in Definition 1.*

- *The set conc(m) of* transitions with concession *in marking $m \in M$ is defined by*

$$\mathit{conc}(m) = \{t \in T \mid \forall p \in P.\, m(p) \geq I(t)(p) \wedge m(p) < H(t)(p)\}\,.$$

- *The set enabled(m) of* enabled transitions *in marking $m \in M$ is given by*

$$\mathit{enabled}(m) = \mathit{conc}(m) \cap \left\{ t \in T \;\middle|\; \Pi(t) = \max_{t' \in \mathit{conc}(m)} \Pi(t') \right\}.$$

- *The effect of* firing *an enabled transition $t \in$ enabled(m) on marking $m \in M$ is the marking $m' \in M$ such that*

$$\forall p \in P.\, m'(p) = m(p) - I(t)(p) + O(t)(p).$$

If multiple transitions are enabled at the same time, there is a *conflict* between these transitions. A conflict is resolved in two steps. First, a *non-deterministic* choice is made over all partitions containing an enabled transition. Second, a *probabilistic* choice is made over all enabled transitions within the selected partition. The probability of selecting a transition is given by the weight of each enabled transition.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

Example 4. *Consider the GSPN in Figure 1a with $m \in M$ being the marking corresponding to the depicted number of tokens, i.e., $m(P_1) = 3$ and $m(P_2) = m(P_3) = 0$. We have $conc(m) = \{t_1, t_2\}$. We assume that the immediate transitions all have the same priority. Since immediate transitions have a higher priority than timed transitions, only t_2 is enabled, i.e., $enabled(m) = \{t_2\}$.*

Firing t_2 leads to the new marking m' as shown in Figure 1b. Now we have $conc(m') = \{t_3, t_4\}$ and both transitions t_3 and t_4 are enabled because they have the same priority. Thus, t_3 and t_4 are in conflict. Note that the firing of t_3 would lead to a different marking than the firing of t_4. Also note that t_2 is not enabled, for two reasons: The input place P_1 does not contain the required two (or more) tokens, and moreover t_2 is blocked by the incoming inhibitor arc from P_2.

Tool support for GSPNs is available through for example the GreatSPN Editor[1] [3] or Storm[2] [17]. The latter is a probabilistic model checker developed at RWTH Aachen University that handles a variety of input languages and modelling formalisms. In particular, it accepts GSPNs given in either of two formats: the Petri Net Markup Language (PNML)[3], and the pnpro XML format, which is used by the GreatSPN Editor. Later we will see that Storm also provides support for (Repairable) Fault Trees through a translation from (R)FTs to GSPNs.

4 Formal Semantics of Dynamic Fault Trees

The goal of this section is to formally introduce Dynamic Fault Trees (DFTs), and to define their semantics in terms of Generalised Stochastic Petri Nets (GSPNs). Our approach is based on the work described in [20, 26], which unifies various DFT semantics.

4.1 Dynamic Fault Trees

Fault Trees (FTs) are directed acyclic graphs with typed nodes. Nodes without successors (or: children) represent *basic events (BEs)*; all other nodes denote *gates*. We start with the formal definition of DFTs.

Definition 5 (DFT)**.** *A* Dynamic Fault Tree (DFT) *is a tuple $\mathcal{F} = (V, \sigma, t, top, FR_a, FR_p)$ where*

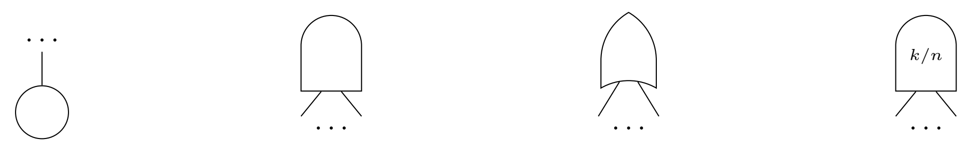
- *V is a finite set of nodes;*
- *$\sigma \colon V \to V^*$ yields the ordered sequence of children for each node;*
- *$t \colon V \to \{BE, AND, OR, \ldots\}$ defines the node type. We use $BE = \{v \in V \mid t(v) = BE\}$ to denote the set of all BEs;*
- *$top \in V$ is the unique top-level event (TLE); and*
- *$FR_a, FR_p \colon BE \to \mathbb{R}_{>0}$ associate an active and passive failure rate, respectively, to each BE.*

We require that DFTs are *well-formed*, that is, (1) the directed graph induced by V and σ is acyclic, and (2) the leaves are exactly the BEs.

For node $v \in V$, we also write $v \in \mathcal{F}$. If $t(v) = K$ for some $K \in \{\mathsf{BE}, \mathsf{AND}, \ldots\}$, we write $v \in \mathcal{F}_K$ (and thus $BE = \mathcal{F}_{\mathsf{BE}}$). We denote the i-th child of v by $\sigma(v)_i$ and use v_i as shorthand.

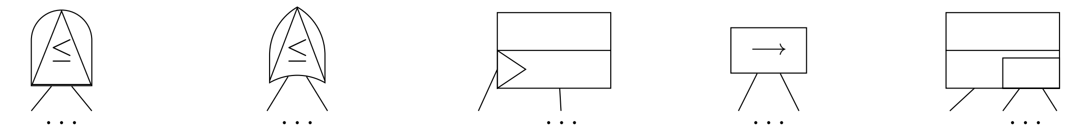
BEs represent system components that can fail. Initially, a BE is operational; it fails according to a negative exponential distribution $P(t) = 1 - e^{-\lambda t}$ with rate λ given by FR_a (or FR_p). A gate fails instantaneously, that is, without any delay when its failure condition over its children is fulfilled. The failure behaviour of a fault tree can be explained by traces of failures.

[1] https://github.com/greatspn/SOURCES
[2] https://www.stormchecker.org
[3] https://www.pnml.org



(a) BE (b) AND (c) OR (d) VOT$_{k/n}$

Figure 2: Basic event and static gates



(a) PAND (b) POR (c) FDEP (d) SEQ (e) SPARE

Figure 3: Dynamic gates

Definition 6 (Failure traces). *Let $\mathcal{F}$ be a DFT. The* failure *of a DFT node v is denoted by event f_v. If v is a BE, the failure is additionally highlighted by $\mathbf{f}_v$. Event $\oslash_v$ indicates that the status of v did not change, event $\lightning_v$ indicates that the failure of v is not valid, event cl_v^s indicates that SPARE gate s claimed its child v, and event $\uparrow_v$ indicates the activation of node v. A* DFT trace *is a sequence of events $\tau = e_1 e_2 \ldots$.*

The key gates for Static Fault Trees (SFTs) are typed AND and OR, shown in Figure 2. These gates fail if all of their children or at least one of them have failed, respectively. Both are instances of the VOT$_{k/n}$ gate, which is depicted in Figure 2d. It has n children and fails if at least k of those have failed. Typically, FTs express for which combinations of BE failures, the specifically marked *top-level event* fails.

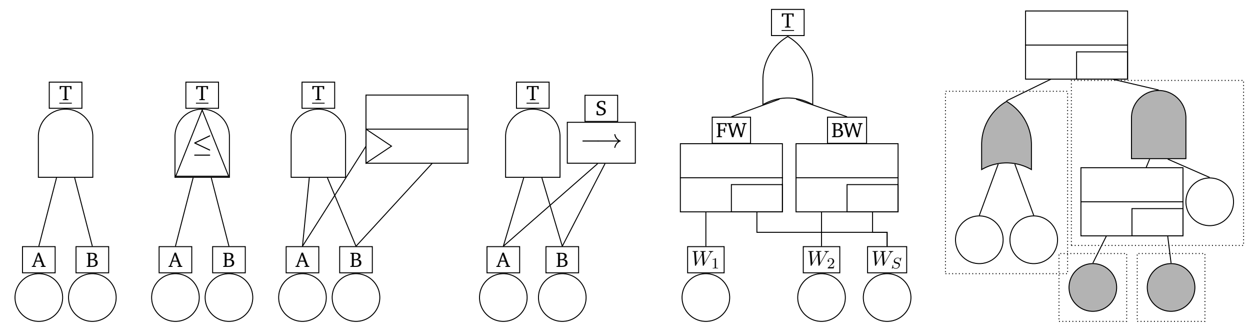
Example 7 (SFT). *The SFT in Figure 4a fails if both A and B have failed, i.e., $\tau_1 = \mathbf{f}_A \mathbf{f}_B \mathrm{f}_T$ and $\tau_2 = \mathbf{f}_B \mathbf{f}_A \mathrm{f}_T$ are both failure traces. Note that events $\mathbf{f}_A$ and $\mathbf{f}_B$ indicate that both BEs fail on their own. In contrast, f_T indicates that the AND gate T failed due to the failure of its children.*

SFTs do not have an internal state – the failure condition is independent of the history. Therefore, SFTs lack expressiveness [19, 25]. Several extensions commonly referred to as *Dynamic Fault Trees (DFTs)* have been proposed to mitigate this problem [12]. The extensions introduce new node types, shown in Figure 3; we categorise them as *priority gates*, *dependencies*, *restrictors*, and *spare gates*.

Priority Gates. These gates extend static gates by imposing a condition on the ordering of failing children and thus allow for order-dependent failure propagation. A priority-AND (PAND) fails if all its children have failed in order from left to right. Figure 4b depicts a PAND with two children A and B. It fails if A fails before B, i.e., $\mathbf{f}_A \mathbf{f}_B \mathrm{f}_T$. The priority-OR (POR) only fails if the leftmost child fails before any of its siblings do. If a gate cannot fail any more, e.g., when B fails before A in Figure 4b, it is *fail-safe*. The corresponding trace is $\mathbf{f}_B \mathbf{f}_A \oslash_T$ where event $\oslash_T$ indicates that the status of T did not change and in particular did not fail. For simplicity, we only consider the *inclusive* variants of both gates, i.e., those that admit simultaneous failures (sometimes denoted by PAND$_\leq$ and POR$_\leq$, respectively). Excluding simultaneous failures is possible as well [11, 20].

Dependencies. Dependencies do not propagate a failure to their parents. Instead, when their trigger (first child) fails, this failure is forwarded to all dependent events (remaining children) which are then rendered failed as well. For example, Figure 4c shows an FDEP gate where the failure of trigger A causes a failure of BE B (provided B has not failed before). The corresponding trace is $\mathbf{f}_A \mathrm{f}_B \mathrm{f}_T$. Note that we use f_B instead of $\mathbf{f}_B$ to highlight that B did not fail on its own but was triggered through A. As a generalisation, it is possible to introduce probabilistic dependencies (PDEP), which are equipped with a parameter p. PDEPs forward the failure only with probability p; with probability $1-p$ the dependent events stay unchanged. Details are discussed in [11, 20].

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.



(a) AND (b) PAND (c) FDEP (d) SEQ (e) SPARE (f) Spare modules

Figure 4: Simple examples of DFTs

Restrictors. Restrictors limit the possible failures. Sequence enforcers (SEQ) require that their children only fail from left to right. This differs from priority gates which do not prevent certain orderings, but only propagate if an ordering condition is met. For instance, the AND gate in Figure 4d fails if A and B have failed (in any order) but the SEQ enforces that A must fail prior to B. In contrast to Figure 4b, T is never fail-safe. Thus, $\mathbf{f}_A\mathbf{f}_B\mathrm{f}_T$ is a trace for both the SEQ and the PAND. However, while $\mathbf{f}_B\mathbf{f}_A\oslash_T$ is a trace for the PAND, the trace for the SEQ is $\mathbf{f}_B\mathbf{f}_A\lightning_S$. Here, event $\lightning_S$ indicates that the order of the sequence enforcer S is violated and the trace is therefore invalid. Another restrictor is the MUTEX gate, which ensures that at most one of its children fails. We will not further consider it here as it can be simulated by SEQ, cf. [19].

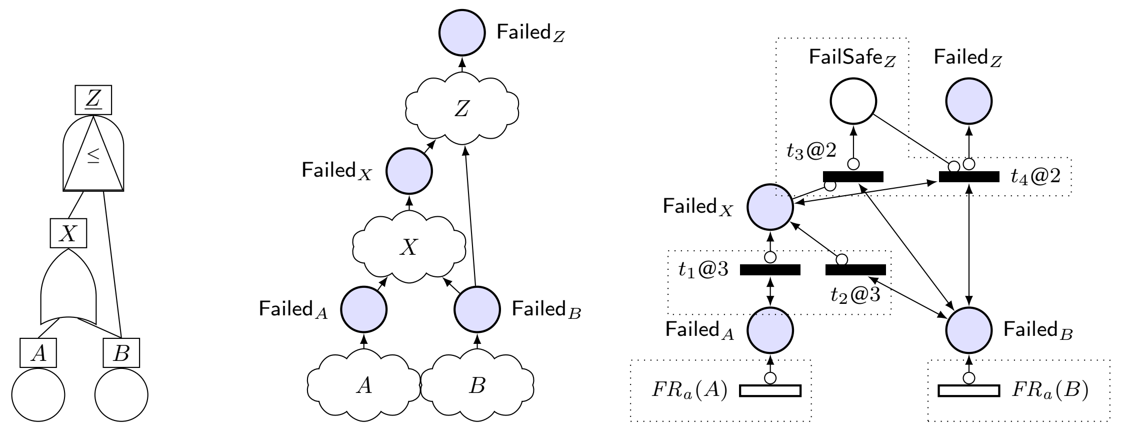
Spare Gates. Consider the DFT in Figure 4e modelling (part of) a motor bike with a spare wheel. A bike needs two wheels to be operational. Either wheel W_1 and W_2 can be replaced by the spare wheel W_S, but not both. The spare wheel is also less likely to fail until it is in use. Assume the front wheel W_1 fails. Then the spare wheel W_S is available and can be used (also called *claiming*). As W_S is in use now, it is also more likely to fail. If any other wheel fails next (e.g., W_2), no spare wheel is available any more, and the parent SPARE fails.

SPARE gates involve two mechanisms: *claiming* and *activation*. Claiming works as follows. SPAREs use one of their children. If this child fails, the SPARE tries to claim another child (from left to right). Only operational children that have not been claimed by another SPARE can be claimed. If claiming fails – because all spare components are either failed or already used by other SPARE gates – the SPARE fails. As example, consider the following trace

$$\tau = \mathrm{cl}_{W_1}^{FW}\,\mathrm{cl}_{W_2}^{BW}\,\mathbf{f}_{W_1}\,\mathrm{cl}_{W_S}^{FW}\,\mathbf{f}_{W_2}\,\mathrm{f}_{BW}\,\mathrm{f}_T.$$

Initially, both SPAREs FW and BW claim their first child. This is represented by events such as $\mathrm{cl}_{W_1}^{FW}$ which indicates that SPARE FW claimed child W_1. After the failure of the claimed child W_1, SPARE FW needs to claim a new child. Child W_S is available, because it is both operational and still unused. FW therefore claims W_S, represented by event $\mathrm{cl}_{W_S}^{FW}$. When W_2 fails, SPARE BW cannot claim any other child anymore and it becomes failed, i.e., event f_{BW} happens.

Let us now consider activation. SPAREs may have independent, i.e., disjoint, sub-DFTs as children. This can also include nested SPAREs, i.e., SPAREs having SPAREs as children. A *spare module* is a sub-tree with one of the SPARE's children as root node. This child is called the *module representative*. Figure 4f gives an example of spare modules (depicted by boxes) and their representatives (shaded nodes). Here, a spare module contains all nodes which have a path to the spare representative without going through an intermediate SPARE. Thus, every leaf of a spare module is either a BE or again a SPARE. Nodes outside of spare modules are always active. For each active SPARE and used child v, the nodes in v's spare module are activated. Active BEs fail with their active failure rate (given by FR_a); all other BEs fail with their passive failure rate (given by FR_p). We extend the trace τ from before to also include



(a) DFT (b) Basic scheme (c) Simplified resulting GSPN

Figure 5: Compositional semantics of DFTs using GSPNs [20]

activation:

$$\tau' = \uparrow_T \uparrow_{FW} \uparrow_{BW}\ \mathrm{cl}^{FW}_{W_1} \uparrow_{W_1} \mathrm{cl}^{BW}_{W_2} \uparrow_{W_2} \mathrm{f}_{W_1}\ \mathrm{cl}^{FW}_{W_S} \uparrow_{W_S} \mathrm{f}_{W_2}\ \mathrm{f}_{BW}\ \mathrm{f}_T.$$

Initially, gates T, FW and BW are active, because they are not part of a spare module. Children are activated through claiming. For example, event $\mathrm{cl}^{FW}_{W_1}$ activates W_1. After the initial claiming and activation, all elements except for the spare child W_S are active. W_S becomes active when it is claimed. For presentation purposes, we usually omit the activation events in the following as they can be inferred from the claiming events.

For presentation purposes we restrict DFTs to conventional DFTs in the remainder of this document.

Definition 8 (Conventional DFT)**.** *A DFT is* conventional *if*

1. *Spare modules are only shared via their (unique) representative. In particular, spare modules are disjoint.*
2. *All children of a SEQ are BEs.*
3. *All children of an FDEP are BEs.*

Restriction 1 ensures that spare modules can be seen as a single entity with respect to claiming and activation. Lifting this requirement to allow for non-disjoint spare modules raises new semantic issues [19]. Restriction 2 ensures that the failing BEs are immediately deducible. Restriction 3 simplifies the presentation. Allowing gates as trigger event leads to additional semantic questions, which will be addressed in Section 5.

4.2 Generic Translation of DFTs to GSPNs

This section describes the general idea for defining the semantics of a DFT $\mathcal{F}$ by a GSPN $\mathcal{T}_{\mathcal{F}}$ based on the elaboration in [20]. We only provide an intuitive idea of the individual steps in this section since we will describe the detailed semantics for repairable DFTs in Section 5.

A high-level overview of the translation process is depicted in Figure 5. It shows a DFT on the left, which is translated into the GSPN on the right. The general idea is that each DFT element is translated to a GSPN according to some template as indicated by the intermediate step depicted in Figure 5b. Then these templates are combined using the *interface places* as shown in Figure 5c. The interface places are depicted as blue circles and are used as connections between the GSPN templates. A token in an interface place reflects the failure of the corresponding DFT element.

Now consider the areas framed by dotted lines in Figure 5c. Each area indicates a part (i.e., BE or gate) of the DFT on the left. The two timed transitions at the bottom trigger the failure of the respective BEs A and B. They fire with their respective active failure rates $FR_a(A)$ and $FR_a(B)$. The remaining white places, referred to as *auxiliary places*, and the immediate transitions simulate the behaviour of the corresponding gate.

As mentioned above, the places colored in blue are interface places. Besides places that indicate whether an element has failed, there are more places that act as interface places.

Definition 9 (Interface places). *The set of* interface places *of a DFT $\mathcal{F}$ is given by*

$$\mathcal{I}_{\mathcal{F}} = \{\mathsf{Failed}_v, \mathsf{Unavail}_v, \mathsf{Active}_v \mid v \in \mathcal{F}\} \cup \{\mathsf{Disabled}_v \mid v \in \mathcal{F}_{\mathsf{BE}}\}.$$

The interface places $\mathcal{I}_{\mathcal{F}}$ serve the purpose of communication between elements of the DFT $\mathcal{F}$: the failure of gate v is reflected by putting a token in Failed_v. $\mathsf{Unavail}_v$ and Active_v places are used for the claiming and activation mechanisms of SPARE gates, respectively. $\mathsf{Disabled}_v$ is used for SEQ gates.

The DFT elements are composed of auxiliary places, transitions and arcs. In the following, we introduce *GSPN templates* that specify the composition of different DFT gate types. Before we formally define those, we consider *priority variables*. The priority variables $\pi = \{\pi_v \mid v \in \mathcal{F}\}$ determine the order of firing transitions and are used, e.g., to specify the order of failure propagation in DFTs. The priorities of the transitions are functions over the priority variables π. Thus, the priority function is of type

$$\Pi\colon T \to \mathbb{N}^n$$

where $n = |\pi|$ denotes the number of variables, and the instantiation in Definition 10 replaces the priority variables by their concrete values.

Definition 10 (GSPN Template). *The GSPN $\mathcal{T} = (P, T, I, O, H, m_0, W, \mathbb{N}^n, \Pi, \mathcal{D})$ is a (π-parametrised)* template *over interface places $\mathcal{I} \subseteq P$. The instantiation of $\mathcal{T}$ with $\mathbf{c} \in \mathbb{N}^n$ is defined as the GSPN*

$$\mathcal{T}[\mathbf{c}] = \big(P, T, I, O, H, m_0, W, \mathbb{N}, \Pi', \mathcal{D}\big)$$

with $\Pi'(t) = \Pi(t)(\mathbf{c})$ for all $t \in \mathcal{T}$.

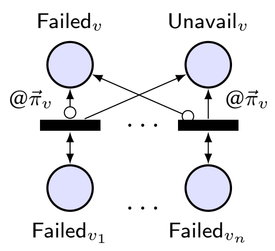
As already mentioned, the places in P are either interface or auxiliary places. Timed transitions simulate the failure of BEs, whereas immediate transitions are used to model the behaviour of gates. The weights of the transitions realise randomisation mechanisms such as the failure rates of BEs. Consider the simplified GSPNs corresponding to the BEs in Figure 5c (indicated by the two boxes at the bottom). The firing rates of the immediate transitions correspond to the active failure rates $FR_a(A)$ and $FR_a(B)$ from Definition 5. The partition $\mathcal{D}$ of the transitions specifies how non-determinism is treated, e.g., whether conflicts are resolved non-deterministically or probabilistically.

We will not go further into detail on weights, priorities and partitioning in this section since these concepts are explained in detail in [20] and moreover will be described with respect to repairs in the DFT elements in Section 5. For simplicity, we assume for now that each immediate transition has its own partition, i.e., each conflict is solved by non-determinism.

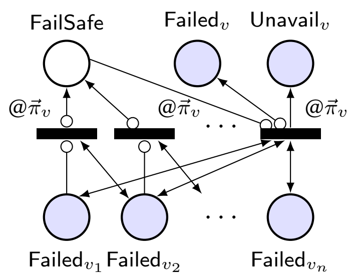
We continue to elaborate the main concept of combining templates to obtain a template for an entire DFT by means of the following example.

Example 11 (Translation of a DFT into a GSPN). *In the following we explain how the DFT from Figure 5a is translated into the GSPN shown in Figure 5c. Figure 6 depicts two GSPN templates. The one on the left simulates the behaviour of an OR gate, and the one on the right simulates a PAND. In both cases, the places at the bottom each represent an interface place of a child whereas the interface places at the top represent the status of the respective gate. Now consider the intermediate step depicted in Figure 5b. The blue places*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.



(a) GSPN template for OR



(b) GSPN template for PAND

Figure 6: GSPN templates for OR and PAND

correspond to the interface places of the GSPN templates. We omit the Unavail$_v$ places for a more intuitive understanding. Adding the remaining transitions and auxiliary places of both templates ultimately leads to the entire GSPN depicted in Figure 5c. In there, we replaced X by the GSPN template for an OR, Z by the template for a PAND, and both A and B by the template for a BE (not depicted).

The compositional approach allows to translate DFT to GSPNs by simply translating each gate and BE individually. New types of gates can therefore be introduced by simply defining the corresponding GSPN template which then specifies their semantics.

4.3 Semantic Issues

The literature on DFTs introduces various kinds of semantics. These semantics agree on the behaviour of single gates. The behaviour of each gate can therefore be clearly defined by providing the corresponding GSPN template [20]. However, the DFT semantics differ with regard to the handling of failure propagation, forwarding in functional dependencies, and non-determinism. This can be seen on the level of the GSPN by the fact that multiple orderings are possible in which immediate transitions fire.

Example 12 (Failure propagation in the GSPN)**.** *Consider again the DFT from Figure 5a. The trace $\mathbf{f}_A \mathrm{f}_X \mathbf{f}_B \mathrm{f}_Z$ leads to a failure of the DFT. The trace $\mathbf{f}_B \mathrm{f}_X \mathrm{f}_Z$ also leads to a failure. Note that here X and B have failed simultaneously (without any progress of time).*

The first trace can be replayed in the GSPN from Figure 5c by first firing the timed transition with rate $FR_a(A)$ and then firing the immediate transition t_1. Next, the timed transition with rate $FR_a(B)$ fires and lastly, immediate transition t_4 fires and places a token in Failed$_Z$ which indicates that DFT gate Z has failed.

The second trace can be replayed by first firing the timed transition for B which places a token in Failed$_B$. However, now two transitions t_2 and t_4 are enabled; we thus have a conflict. Firing t_2 first allows to fire t_4 afterwards and a token is placed in Failed$_Z$. This token indicates that Z has failed which also agrees with the DFT trace. However, if transition t_4 fires first, a token is placed in FailSafe$_Z$. This indicates that PAND Z has become fail-safe and can never fail. This corresponds to the trace $\mathbf{f}_B \oslash_Z \mathrm{f}_X$.

The order in which the conflict between transitions t_2 and t_4 is resolved, therefore leads to vastly different outcomes.

The semantic differences are surveyed in [19], and a comprehensive overview of existing DFT semantics is given in [20, Table 1]. Moreover, the latter publication also shows that all those variants can be covered by varying two parameters in the definition of GSPNs: the transition priorities Π and the partitioning $\mathcal{D}$ of the immediate transitions. The priorities constrain the ordering of transitions, while the latter controls the treatment of non-determinism. In [20, Table 4] it is shown that all different DFT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

semantics from the literature can be captured by only adapting the transition priorities and partitioning. In particular, the net structure itself remains the same for all semantics.

4.4 Tool Support

The probabilistic model checker Storm [17], mentioned in Section 3, does not only accept GSPNs in two different formats but also features a translation of DFTs into GSPNs. This functionality was implemented in Storm version 1.2.1 and is further described in [20]. It expects DFTs to be given in the Galileo format[4]. The generated GSPNs are exported as GreatSPN Editor projects in the pnpro format[5] and can be further analysed with the GreatSPN Editor [3]. A collection of DFT examples and corresponding GSPNs is available online[6].

5 Formal Semantics of Fault Trees with Repair

So far we postulated that failures of components are persistent. In this section, we no longer make this assumption and consider repairs in Dynamic Fault Trees. In *Repairable DFTs (RFTs)*, failed components can be repaired after the failure. Thus, a failed system may turn operational again after the repair of a component. We again begin with formally introducing RFTs, and then define their semantics in terms of GSPNs. Furthermore, we identify semantic issues that need to be considered when integrating repairs in the formalism of DFTs and propose potential solutions. The contents of this section are based on the techniques described in [21].

5.1 Repairable Fault Trees

The most important syntactic extension that is required is the introduction of *repair rates* for BEs, which complement the active and passive failure rates as defined in Definition 5. Similarly, a repair rate is the parameter of an exponential distribution. While failure rates model the time to component failure, a repair rate models the time until the component is repaired. It is defined as follows.

Definition 13 (Repair rate). *Let $v \in BE$. The repair rate $RR(v)$ of v is given by the function*

$$RR : BE \to \mathbb{R}_{\geq 0}.$$

A BE v with repair rate zero, i.e., $RR(v) = 0$, cannot be repaired.

Another syntactic extension is concerned with dependency gates (FDEP), which are obviously affected by repair operations as they can have an impact on failure propagation. We assume that the repair of the trigger event stops failure propagation while a repaired dependent child fails again by cause of the failed trigger. Thus, FDEP gates do not propagate repairs. In order to support the propagation of repairs affecting the triggering event, we additionally introduce RDEP gates. The semantic details are provided in Section 5.2.1.

We extend the failure traces of Definition 6 to repairs by introducing the repair event r_v (and $\mathbf{r}_v$ for BEs). Note that repairs can only take place if the corresponding element has failed before, i.e., a trace such as $\mathbf{f}_A\mathbf{r}_A\mathbf{r}_A$ is invalid.

[4]https://www.cse.msu.edu/~cse870/Materials/FaultTolerant/manual-galileo.htm
[5]https://www.stormchecker.org/publications/gspn-semantics-for-dfts.html
[6]https://github.com/moves-rwth/dft-gspn-examples

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

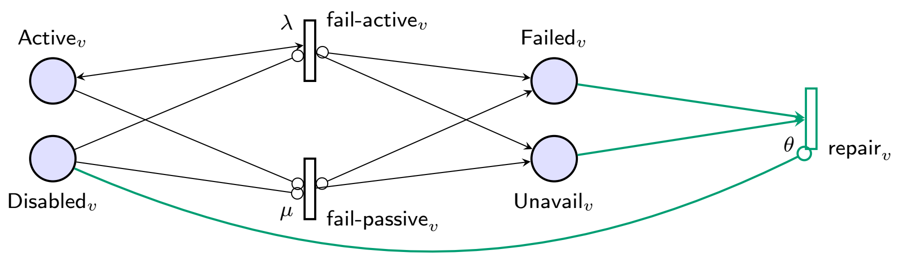


Figure 7: GSPN template for repairable BE

5.2 Generic Translation of RFTs to GSPNs

This section explains the compositional translation introduced in [21]. We begin with GSPN templates for basic events and gate types of repairable DFTs and then present their combination.

5.2.1 Templates for RFT Elements

The repair extension of templates is introduced in [21], which is again based on the GSPN templates as presented in [20]. Therefore, we can re-use the notion of GSPN templates from Section 4. We highlight the changes due to repairs in green color. The interface places $\mathcal{I}$ from Definition 9 remain highlighted in blue, and their initial marking is defined by the initialisation template to be defined later. The other places are initially marked with tokens as shown in the template.

If not otherwise depicted, gates have n children and the i-th child of gate v is denoted by v_i. Children are represented by some of the interface places that are relevant for mechanisms in the respective template. Transition priorities are indicated by @ and the priority function π where $\vec{\pi}_v$ is the priority variable for node v. We will elaborate on the priorities later. In order to provide a straightforward representation of templates, we omit transition priorities in the majority of cases if they are equal to $@\vec{\pi}_v$.

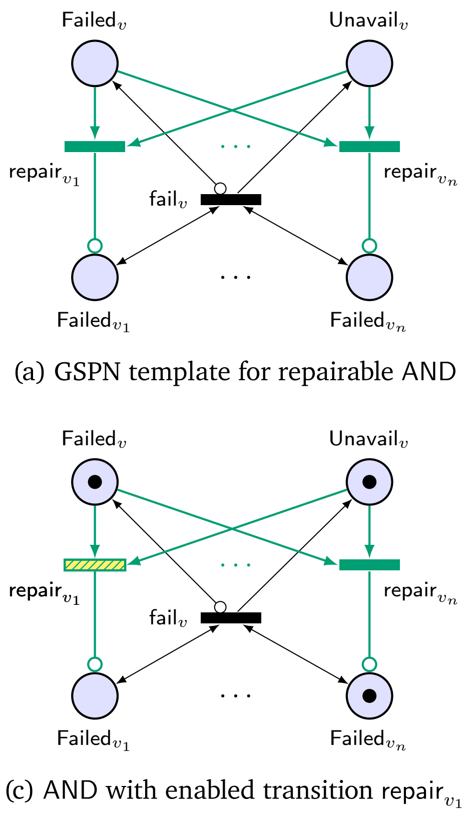
Basic Events. Figure 7 shows the template $\mathsf{templ}_{\mathsf{BE}}(v)$ for a repairable BE v. Node v is in a failed state if place Failed$_v$ contains a token. An empty Unavail$_v$ indicates that v is available for claiming by a SPARE, otherwise it is unavailable.

The failing of v is represented by firing one of the timed transitions fail-active$_v$ or fail-passive$_v$, placing a token into each of Failed$_v$ and Unavail$_v$. If the place Active$_v$ holds a token, fail-active$_v$ is enabled and the node fails with the active failure rate $\lambda = FR_a(v)$. The transition fail-passive$_v$ can fire if Active$_v$ does not contain a token. Thus, v fails with the passive failure rate $\mu = FR_p(v)$. A token in Disabled$_v$ prevents both the active and passive failure of v. BE v is disabled if firing it would violate a SEQ. The inhibitor arcs emanating Failed$_v$ prevent v to fail again if v is already in a failed state.

The timed transition repair$_v$ indicates the repair of a failed v with the repair rate $\theta = RR(v)$ according to Definition 13. The tokens in Failed$_v$ and Unavail$_v$ are removed by repair$_v$. Again, an inhibitor arc prevents repair$_v$ to fire if Disabled$_v$ holds a token. Once v has been repaired, the failure of v can occur again, because the place Failed$_v$ is empty again.

Example 14 (Failure of BE). *Consider the simple trace $\mathbf{f}_v\mathbf{r}_v\mathbf{f}_v$. This trace can be mimicked in the GSPN by first firing the timed transitions fail-active$_v$, then firing repair$_v$ and lastly firing fail-active$_v$ again. Note that trace $\mathbf{f}_v\mathbf{r}_v\mathbf{f}_v$ therefore yields the same outcome as $\mathbf{f}_v$. This is intended behaviour as a repair can be thought of as "undoing" the failure.*

AND/OR Gates. The template $\mathsf{templ}_{\mathsf{AND}}(v)$ of the AND gate v is shown in Figure 8a. A failure of all children leads to the failure of AND. This is represented by transition fail$_v$ that puts a token in the



(a) GSPN template for repairable AND

(b) Failed AND

(c) AND with enabled transition repair$_{v_1}$

(d) AND after firing of transition repair$_{v_1}$

Figure 8: GSPN template and repair procedure for AND

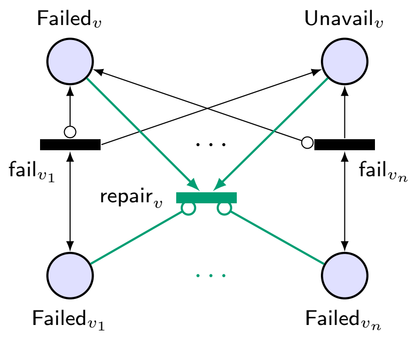


Figure 9: GSPN template for repairable OR

places Failed$_v$ and Unavail$_v$. Thus, v is marked as failed as shown in Figure 8b. The marking therefore corresponds to the trace $\tau_1 = \mathbf{f}_{v_1} \dots \mathbf{f}_{v_n} \mathrm{f}_v$.

Once at least one child is repaired, AND is repaired as well. For each child v_i of v, there exists an immediate transition repair$_{v_i}$ with an inhibitor arc coming from the place Failed$_{v_i}$ of child v_i. The inhibitor arc ensures that repair$_{v_i}$ can only fire if no token is in Failed$_{v_i}$, i.e., v_i is operational. In Figure 8c, the transition repair$_{v_1}$ highlighted in green is about to fire because v_1 was repaired. It corresponds to the trace $\tau_2 = \tau_1 \mathbf{r}_{v_1}$. Once a repair transition fires, the tokens from Failed$_v$ and Unavail$_v$ are removed, resulting in v becoming operational again as depicted in Figure 8d. This corresponds to trace $\tau_3 = \tau_2 \mathrm{r}_v$.

As shown in Figure 9, the template templ$_{\mathsf{OR}}(v)$ of OR v is similarly structured.

Voting Gates. The template for repairs in a VOT$_{k/n}$ gate v is pictured in Figure 10. The failure of v occurs if k of the n children of v have failed. The failure of child v_i enables transition fail$_{v_i}$, which puts a token in place Collect$_v$. Furthermore, to ensure that fail$_{v_i}$ can only fire once, fail$_{v_i}$ removes the token

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

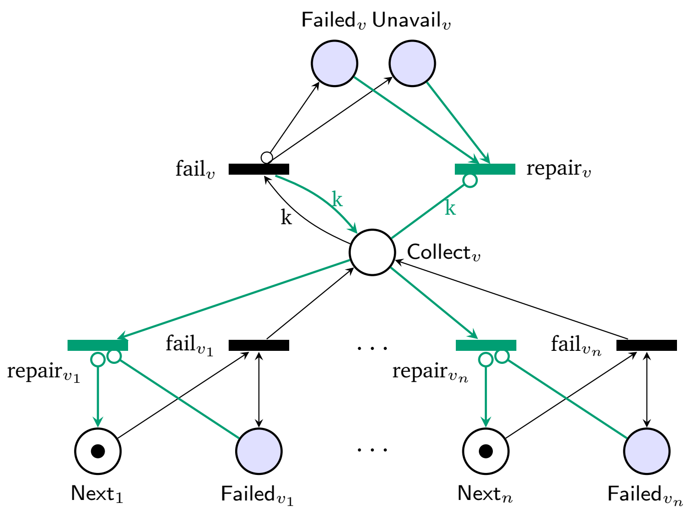


Figure 10: GSPN template for repairable VOT$_{k/n}$

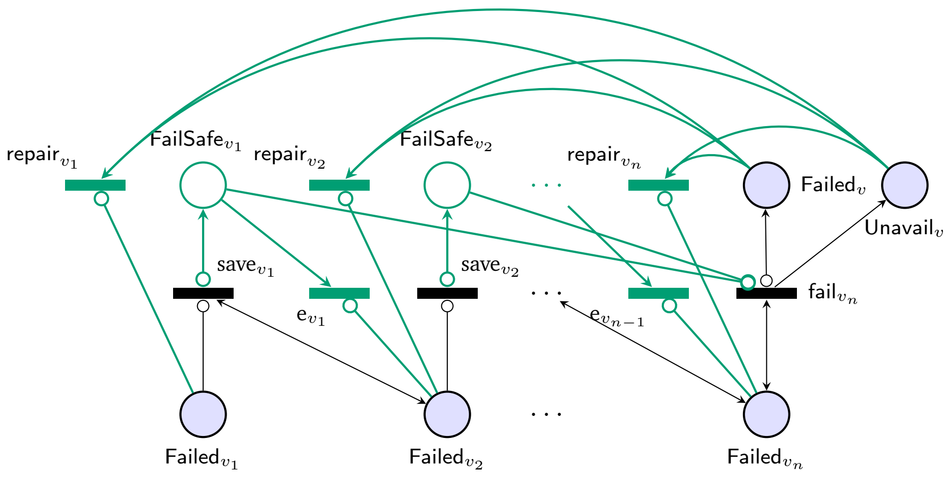


Figure 11: GSPN template for repairable PAND

from place Next$_i$. Once the collector place contains k tokens, transition fail$_v$ fires, consumes k tokens from Collect$_v$ and places a token in Failed$_v$ and Unavail$_v$. In order to perform the repair procedure, it is necessary to hold record of the number of failed children. Therefore, fail$_v$ also returns the k tokens back into Collect$_v$.

If child v_i is repaired, the transition repair$_{v_i}$ fires, removes one token from Collect$_v$ and puts a token in Next$_{v_i}$. A token in Next$_{v_i}$ prevents the removal of multiple tokens from Collect$_v$ and allows once again the firing of fail$_{v_i}$. If Collect$_v$ contains less than k tokens, the transition repair$_{v_i}$ can fire, causing the repair of v.

Priority Gates. Priority gates fail if their children fail in the given order (left to right), and they turn fail-safe if a failure violates the failing order. When the children are repairable, only the most recent failure of every child is taken into account. Note that a repair can turn a fail-safe gate to non-fail-safe and vice versa.

The PAND gate fails if the children failed in order from left to right including simultaneous failures. The corresponding template templ$_{\text{PAND}}(v)$ with respect to repairs, is shown in Figure 11. For each child v_i except for the last one, there exists a place FailSafe$_{v_i}$. If the right child v_{i+1} fails while child

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

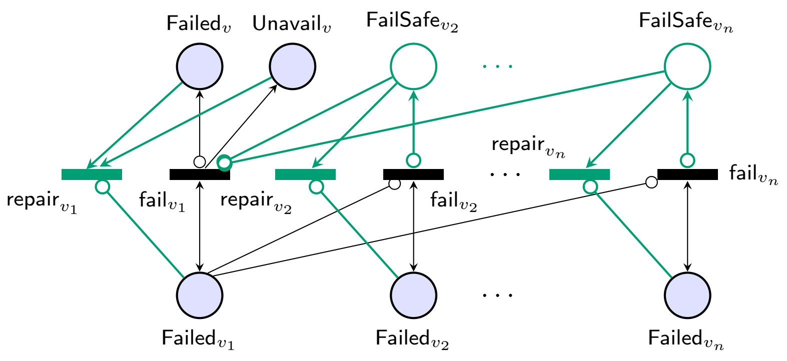


Figure 12: GSPN template for repairable POR

v_i is still operational, the transition save$_{v_i}$ places a token in FailSafe$_{v_i}$. This indicates that the PAND is fail-safe due to v_i (and v_{i+1}) violating the order. Once at least one FailSafe$_{v_i}$ place contains a token, it prevents the transition fail$_{v_n}$ to fire. As a consequence, v is fail-safe because fail$_{v_n}$ is the only transition that could place a token in Failed$_v$. In case the children fail in the correct failure order, every FailSafe$_{v_i}$ place stays empty and Failed$_{v_n}$ eventually contains a token. Thus, fail$_{v_n}$ fires and marks the failure of v.

If v is in a failed state and child v_i gets repaired, the transition repair$_{v_i}$ fires and removes the tokens from Failed$_v$ and Unavail$_v$. Thus, v is repaired. The repair of child v_i while child v_{i+1} is still failed results in v becoming fail-save, because repair$_{v_i}$ fires first followed by save$_{v_i}$. If the place FailSafe$_{v_i}$ holds a token and child v_{i+1} is repaired, the transition e$_{v_i}$ fires and empties FailSafe$_{v_i}$. The correct failure order is now restored through the repair. A similar template exists for the exclusive version of PAND [21], in which simultaneous failures lead to fail-safe.

Example 15 (Failure and repair of PAND)**.** *We consider a PAND T with three BEs A, B and C as children. Consider the trace $\tau_1 = \mathbf{f}_A\mathbf{f}_C\mathbf{f}_B$. BE A fails in the right order, but C failed before B. Thus, the PAND is fail-safe. If now C gets repaired, the trace is $\tau_2 = \mathbf{f}_A\mathbf{f}_C\mathbf{f}_B\mathbf{r}_C$. As the failure of C is "undone" through its repair, we can also think of it as never happening in the first place. An equivalent trace is therefore $\tau_2' = \mathbf{f}_A\mathbf{f}_B$. Using the GSPN we can show that both traces τ_2 and τ_2' yield the same resulting marking, i.e., both traces result in the same state of the DFT.*

A new failure of C, i.e., trace $\tau_3 = \mathbf{f}_A\mathbf{f}_C\mathbf{f}_B\mathbf{r}_C\mathbf{f}_C\mathrm{f}_T$ (or $\tau_3' = \mathbf{f}_A\mathbf{f}_B\mathbf{f}_C\mathrm{f}_T$) results in a failure of the PAND. Repairing the children out of order again leads to a fail-safe PAND, e.g. $\tau_4 = \mathbf{f}_A\mathbf{f}_B\mathbf{f}_C\mathrm{f}_T\mathbf{r}_B\mathrm{r}_T$ (or $\tau_4' = \mathbf{f}_A\mathbf{f}_C$).

The POR gate fails if its leftmost child fails first (also allowing concurrent failing of other siblings). The corresponding template templ$_{\mathsf{POR}}(v)$ is depicted in Figure 12. If the leftmost child fails before or simultaneously with its siblings, transition fail$_{v_i}$ fires, which leads to the failure of v. The inhibitor arcs emanating from Failed$_{v_1}$ prevent the other fail$_{v_i}$ transitions to fire. If a child v_i right of v_1 fails while v_1 is operational, the transition fail$_{v_i}$ is enabled and puts a token in FailSafe$_{v_i}$. The inhibitor arc emanating from FailSafe$_{v_i}$ then prevents the transition fail$_{v_1}$ from firing. Thus, once at least one FailSafe$_{v_i}$ place contains a token, v is fail-safe.

For each child v_i, there exists a transition repair$_{v_i}$. By repairing the leftmost child v_1, the transition repair$_{v_1}$ fires and removes the tokens from Failed$_v$ and Unavail$_v$ making v operational again. If a child v_i right of v_1 gets repaired and FailSafe$_{v_i}$ contains a token, repair$_{v_i}$ fires and removes the token from FailSafe$_{v_i}$. If no FailSafe$_{v_i}$ contains a token, v is not fail-safe and v can fail again. Again, an exclusive variants exists as well, see [21].

Dependency Gates. Recall that we restricted DFTs to conventional DFTs as in Definition 8, and thus all children of an FDEP gate are BEs.

Figure 13 shows the template templ$_{\mathsf{FDEP}}(v)$ for an FDEP v with repairs. The FDEP itself can not fail. Hence, the place Failed$_v$ is not connected to any transition. As long as the trigger v_1 is failed, the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

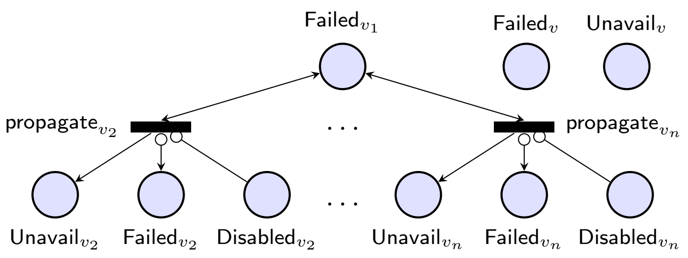


Figure 13: GSPN template for FDEP

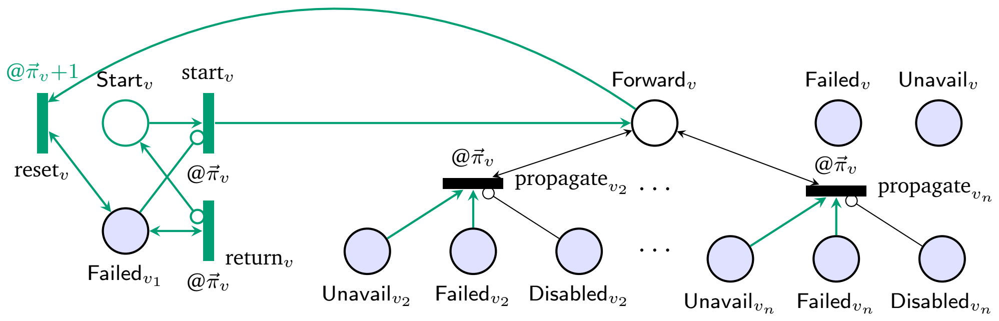


Figure 14: GSPN template for RDEP

transition $\mathsf{propagate}_{v_i}$ can fire if the child v_i is operational and enabled. Firing $\mathsf{propagate}_{v_i}$ marks child v_i as failed. Note that all propagation transitions are enabled at the same time, i.e., they are in conflict. The order in which dependent children are rendered failed thus depends on the priorities, partitioning and weights. A possible trace is for example $\mathbf{f}_{v_1}\mathrm{f}_{v_2}\mathrm{f}_{v_3}$.

Note that dependent children which are repaired can be immediately rendered failed again by a (still triggered) dependency. An example is the trace $\mathbf{f}_{v_1}\mathrm{f}_{v_2}\mathrm{f}_{v_3}\mathbf{r}_{v_2}\mathrm{f}_{v_2}$ where v_2 fails again due to the dependency. Once v_1 is repaired, the failure propagation stops because Failed_{v_1} is empty. However, the repair is not forwarded to the dependent children and they can still be failed.

In order to forward repairs, we introduce a new dependency gate: the RDEP gate. While an FDEP forwards failures, an RDEP forwards repairs. Thus, if the trigger of a RDEP gets repaired, the children are repaired as well. An example trace is $\mathbf{f}_{v_2}\mathbf{f}_{v_1}\mathbf{f}_{v_3}\mathbf{r}_{v_1}\mathrm{r}_{v_2}\mathrm{r}_{v_3}$ in which the repair of v_1 also triggers the repairs of v_2 and v_3. The template $\mathsf{templ}_{\mathsf{RDEP}}(v)$ for RDEP v is depicted in Figure 14.

Because the trigger v_1 of RDEP v has to fail first before v_1 can be repaired, Start_v is initially left empty. Only if v_1 fails, transition return_v can fire and puts a token in place Start_v. If v_1 is then repaired, the transition start_v fires and moves the token from Start_v to $\mathsf{Forward}_v$. As long as $\mathsf{Forward}_v$ holds a token, every failed (and not disabled) dependent child v_i is repaired by the transition $\mathsf{propagate}_{v_i}$. Once v_1 fails again, reset_v removes the token from $\mathsf{Forward}_v$ and the repair propagation stops. Note that the higher priority of the transition reset_v in comparison to $\mathsf{propagate}_{v_i}$ prevents the propagation to carry on, once v_1 and dependent child v_i fail simultaneously.

Note that in an RDEP – as for the FDEP – dependent children which are failed can be immediately repaired again by a repaired trigger. An example is the trace $\mathbf{f}_{v_1}\mathbf{f}_{v_2}\mathbf{r}_{v_1}\mathrm{r}_{v_2}\mathbf{f}_{v_2}\mathrm{r}_{v_2}$ where v_2 is immediately repaired after its failure due to the repair dependency.

Restrictor Gates. As described in Definition 8, we assume that children of restrictors such as SEQ (and MUTEX) are solely BEs. Children of a SEQ are only allowed to fail in strict order from left to right. Failures out of order are not possible. To ensure this order, repairs can only happen in the opposite direction from right to left. Hence, if multiple children have failed, the rightmost child needs

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

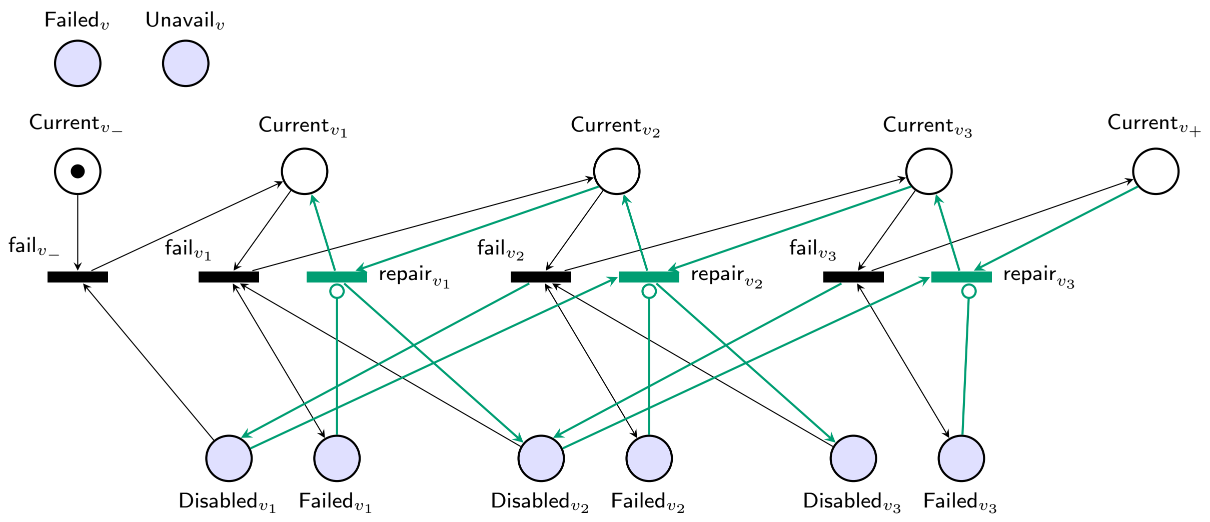


Figure 15: GSPN template for a SEQ with repair restrictions

to be repaired before the other ones can be repaired.

Figure 15 shows the template $\mathsf{templ}_{\mathsf{SEQ}}(v)$ for the SEQ gate v with three exemplary children. Cases with a different number of children work similarly. For each SEQ which affects the behaviour of BE v_i, the Disabled$_{v_i}$ place initially contains a corresponding token. Disabled$_{v_i}$ is therefore one of the only places in the overall GSPN which can contain more than one token.

A token in Current$_{v_i}$ indicates that the child v_i is currently the only enabled and operational child. Furthermore, the child v_{i-1} is failed and enabled for repair. All children except for v_i and v_{i-1} are disabled.

Because v_1 has no left sibling, Current$_{v_-}$ initially holds a token. Thus, the transition fail$_{v_-}$ fires and enables v_1 by removing a token from Disabled$_{v_1}$. In addition, the transition moves the token from Current$_{v_-}$ to Current$_{v_1}$ and marks that v_1 is now enabled and operational.

In general, if Current$_{v_i}$ holds a token and child v_i fails, the fail$_{v_i}$ transition fires. The firing of fail$_{v_i}$ enables child v_{i+1} (by removing a token from Disabled$_{v_{i+1}}$) and disables child v_{i-1} (by again placing a token in Disabled$_{v_{i-1}}$). Additionally, transition fail$_{v_i}$ moves the token from Current$_{v_i}$ to the successor place Current$_{v_{i+1}}$ (provided it exists).

As an example, consider the trace $\mathbf{f}_{v_1}\mathbf{f}_{v_2}$. After the failure of v_1, place Current$_{v_2}$ contains a token, and places Disabled$_{v_1}$ and Disabled$_{v_2}$ contain no token any more. The failure of v_2 allows to fire transition fail$_{v_2}$, which moves the token to Current$_{v_3}$ indicating that v_3 could fail next. Moreover, v_3 is enabled by removing the token from the corresponding place Disabled$_{v_3}$. Child v_1 becomes disabled again through a token in Disabled$_{v_1}$, because v_1 cannot be repaired before v_2 is repaired.

The repairs can take place from left to right. If Current$_{v_i}$ holds a token and child v_{i-1} is repaired, repair$_{v_{i-1}}$ fires. This enables child v_{i-2} and disables child v_i. Furthermore, repair$_{v_{i-1}}$ moves the token from Current$_{v_i}$ to Current$_{v_{i-1}}$ indicating that v_{i-1} is now enabled and operational. If the rightmost child v_3 has failed, place Current$_{v_+}$ contains a token, indicating that all children are currently failed.

As an example for repairs in SEQ, consider the trace $\mathbf{f}_{v_1}\mathbf{f}_{v_2}\mathbf{r}_{v_2}$, which extends the previous trace. Repairing v_2 fires transition repair$_{v_2}$, which moves the token from Current$_{v_3}$ back to Current$_{v_2}$. Furthermore, v_3 is disabled while v_1 is enabled.

SPARE Gates. A SPARE can claim one of its children. If the claimed child fails, the SPARE attempts to claim another child that is currently operational and available, i.e., not already claimed by another SPARE. If no child can be claimed any more, the SPARE fails. The order in which the children are claimed by a SPARE can be specified in the design of the SPARE. By default, children are claimed

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

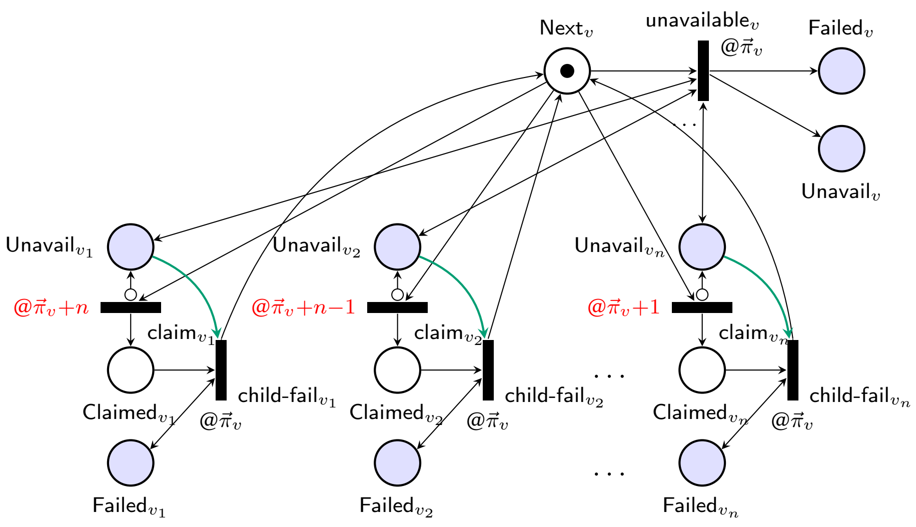


Figure 16: GSPN template segment for SPARE with left-to-right claiming order

from left to right. In RFTs, children can also be repaired. Thus, we need to specify the impact of a child-repair on the SPARE.

Another aspect in SPAREs is activation. Claiming a child activates it – and the elements in the corresponding subtree. Active elements fail with their active failure rate, while inactive elements fail with their passive failure rate. Repairs in RFTs therefore also necessitate a deactivation mechanism.

As a consequence, the template for SPARE$_v$ is divided into four parts: Claiming, Child Repair, Activation and Deactivation. We consider only SPAREs with early claiming based on the template from [20]. Early claiming means that a (nested) SPARE claims children even if it itself is not yet activated. We refer to [19] for details on early claiming and other claiming behaviour.

(1) Claiming: There are multiple possibilities in which order the children are claimed by a SPARE. We will consider the left-to-right claiming order and the arbitrary claiming order. The part of the template for SPARE v which sets the claiming order of the children from left to right is depicted in Figure 16. We consider the trace $\mathrm{cl}^v_{v_i} \mathrm{f}_{v_i} \mathrm{cl}^v_{v_{i+1}}$. In the template, Next$_v$ initially contains a token. Whenever a token is in Next$_v$, a new child has to be claimed. If the child v_i is available for claiming, meaning that Unavail$_{v_i}$ is empty, the transition claim$_{v_i}$ can fire and removes the token from Next$_v$. This corresponds to event $\mathrm{cl}^v_{v_i}$. After firing the transition, Next$_v$ is empty and the claiming of another child is prevented. Additionally, claim$_{v_i}$ puts a token in Claimed$_{v_i}$ and Unavail$_{v_i}$, thus making the claimed child unavailable for other SPAREs. If multiple children are available for claiming while Next$_v$ holds a token, multiple claim$_{v_i}$ have concession. In this case, the leftmost available child is claimed because the priorities of the claim$_{v_i}$ transitions decrease from left to right and thus, only the left-most transition is enabled.

If the claimed child v_i fails, i.e., event f_{v_i} happens, then child-fail$_{v_i}$ fires. Firing the transition removes a token from Claimed$_{v_i}$, and places a token in Next$_v$ indicating v needs to claim a new child. Since a failed child can be repaired and claimed by a SPARE again, the transition child-fail$_{v_i}$ additionally removes one token from Unavail$_{v_i}$, indicating that the child is not claimed by v any more. Notice that one token was placed there during the claiming process and another during the failure of child v_i. Thus, after firing of child-fail$_{v_i}$, Unavail$_{v_i}$ still holds a token. Next, as place Next$_v$ contains a token again, SPARE v tries to claim a new child, for example $\mathrm{cl}^v_{v_{i+1}}$. If every child of v is unavailable for claiming, unavailable$_v$ can fire. As a consequence, unavailable$_v$ places tokens in both Failed$_v$ and Unavail$_v$ and thus marks the failure of SPARE v.

Note that the order in which SPARE children are claimed can easily be changed by simply adapting the transition priorities of transitions claim$_{v_i}$. Other claiming orders can therefore be implemented

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

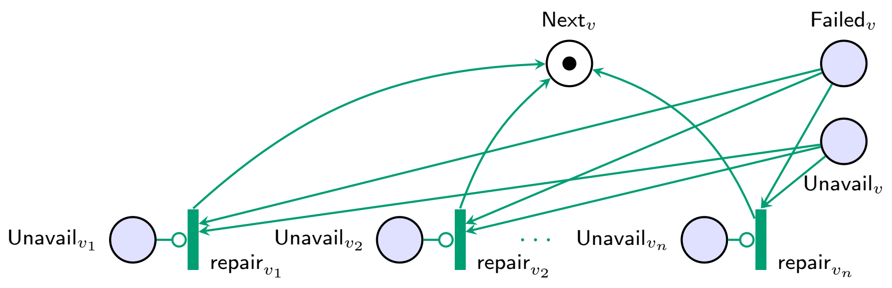
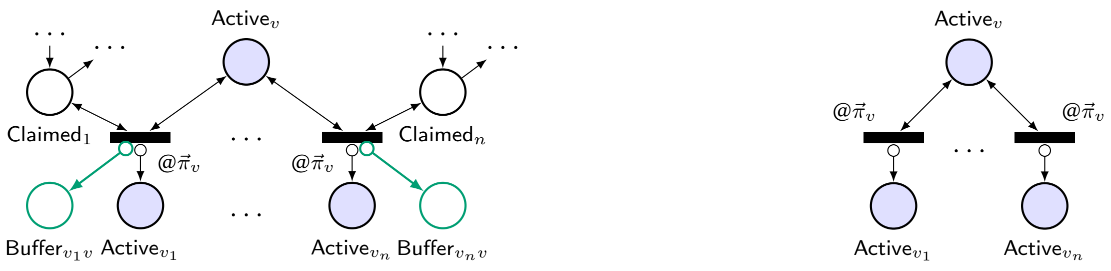


Figure 17: GSPN template segment for SPARE with unchanged claiming after child repair



(a) Activation in SPARE

(b) Activation in gates

Figure 18: GSPN template segment for activation mechanism of DFT elements

individually per SPARE without changing the general structure of the template. In particular, the arbitrary claiming order can be defined by setting all priorities to the same value.

(2) Child repair: There are (at least) two different ways in which a repair of a child impacts a SPARE that has claimed this or another child. The first version, which we will formalise here, specifies that a claimed child stays claimed by the SPARE as long the child is operational. Thus, only if the currently claimed child fails, another child can be claimed. An example for this behaviour is the following trace

$$\tau = \mathrm{cl}^v_{v_1} \mathbf{f}_{v_1} \mathrm{cl}^v_{v_2} \mathbf{r}_{v_1} \mathbf{f}_{v_2} \mathrm{cl}^v_{v_1}.$$

Here, the repaired child v_1 is only claimed again after the currently used child v_2 has failed. The second version, which is described in [21], defines that every time a child is repaired, the currently claimed child is released and a new claiming takes place. Applied on the previous trace τ, this behaviour can yield the following trace

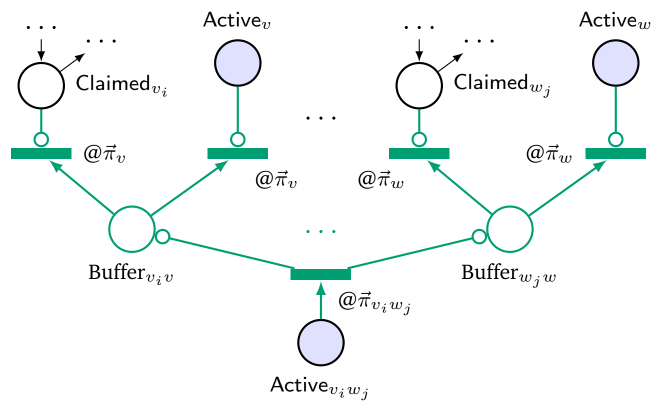
$$\tau' = \mathrm{cl}^v_{v_1} \mathbf{f}_{v_1} \mathrm{cl}^v_{v_2} \mathbf{r}_{v_1} \mathrm{cl}^v_{v_1}.$$

Here, the repaired child v_1 is immediately claimed again after its repair – even though child v_2 is still operational. In both cases, if a child is repaired while the SPARE is failed, the SPARE is repaired and it is possible for the SPARE to claim a child.

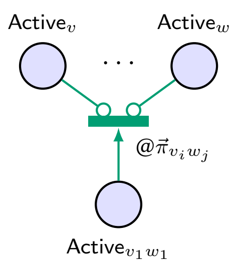
The template segment of SPARE v depicted in Figure 17 specifies that a child stays claimed by v as long as it is operational. Transition repair$_{v_i}$ can fire if v is failed and child v_i is available again. Note that v_i can become available either because it is repaired or it becomes unclaimed by another SPARE. Recall from Figure 16 that Next$_v$ is empty when v has failed. Thus, repair$_{v_i}$ places a token in Next$_v$ and v can try to claim a child again.

(3) Activation: If a child is claimed by an active SPARE, the nodes in the spare module belonging to this child are activated. An activated BE fails with the corresponding active failure rate. If a SPARE is activated, it propagates the activation to its claimed child. When other gates are activated, they activate their children. Thus, the activation propagates downwards until all nodes are activated.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.



(a) Deactivation mechanism in SPARE children



(b) Deactivation in gates

Figure 19: GSPN template segment for deactivation mechanism of DFT elements

The part of the template depicted in Figure 18a shows the activation mechanism in SPAREs. Consider the trace $\uparrow_v \mathrm{cl}^v_{v_i} \uparrow_{v_i}$. Here, SPARE v is active and has claimed child v_i. Both places Active$_v$ and Claimed$_{v_i}$ therefore contain a token. As a consequence, the transition activate$_{v_i}$ can fire if the child v_i has not been activated already. If activate$_{v_i}$ fires (corresponding to event $\uparrow_{v_i}$), it puts a token in Active$_{v_i}$ indicating that child v_i is activated. Note that the place Buffer$_{v_i v}$ will become relevant for the deactivation.

Other gates simply propagate the activation to their children as depicted in Figure 18b. All children of v are activated if Active$_v$ contains a token, meaning v is active.

(4) Deactivation: With the possibility of repairing a node, deactivation of components becomes possible. For instance, if a child that is claimed by a SPARE fails, the claiming could be withdrawn. In this case, the child and all nodes in the spare module should be deactivated. Deactivation of a node v is indicated by event $\downarrow_v$.

Similarly to activation, deactivation propagates downwards. If a SPARE gate releases the claim on a child, all nodes in the spare module of that child are deactivated. Deactivated BEs fail with their passive failure rate. The deactivation of a SPARE also leads to the deactivation of the claimed child. As for activation, other gates propagate the deactivation downwards to the leaves of the spare module. As mentioned in [19], dependencies such as FDEPs do not propagate activation signals. Thus, they do not propagate deactivation signals either.

The deactivation mechanism of SPARE children is depicted in Figure 19a. A node v_i that was activated by SPARE v is deactivated either if v withdraws the claim of the node or if v itself is deactivated. This is handled by removing a token from Buffer$_{v_i v}$. Since a node can have multiple parents $v, w, \dots$ that keep the child activated, every parent node has to be considered in the deactivation process.

Figure 19b shows the deactivation mechanism in gates. A node gets deactivated if all its parent gates are not active.

Extensions. The compositionality of the GSPN framework allows to easily support additional types of gates. For a new gate, we simply need to specify the corresponding GSPN template and connect it with the existing interface places.

For example, one could add a GSPN template for repair boxes and, thus, allow more complex repair operations. Supporting repair boxes would require adding new interface places to BEs capturing whether a repair of the BE is currently under way. The repair box would then place a token into the corresponding place for a BE v to enable the timed transition repair$_v$. This signals that the repair process of BE v has started. In contrast to the existing BE template, repairs would only be possible if initiated by a repair box. Repairing multiple inputs simultaneously could be modelled similar to the GSPN template for an RDEP.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

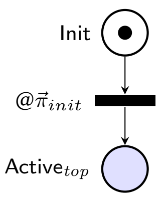


Figure 20: GSPN template for initialisation

5.2.2 Combining Templates

In this section, we define how the GSPN templates for individual DFT elements are combined to obtain a GSPN template $\mathcal{T}_{\mathcal{F}}$ for an entire DFT $\mathcal{F}$ (cf. Figure 5). Because the GSPN templates for the DFT elements in repairable DFTs provide a similar structure and in particular the same interface places as in Section 4.2, the mechanisms of combining templates are similar to the process described there.

Initialisation Template. Before we can translate a DFT to a GSPN, we need to define the GSPN template that reflects the initialisation of a DFT. It is denoted by $\mathsf{templ}_{\mathsf{init}}$, and is depicted in Figure 20. In order to start the top-down activation mechanism, the immediate transition is the first transition that fires. It places a token in the Active$_{top}$ place which corresponds to the active place of the top event top of the DFT. This template therefore correspond to event $\uparrow_{top}$.

Merging Templates. Lastly, it remains to describe how different GSPN templates are combined. We define the merge of two GSPN templates as follows.

Definition 16 (Merging Templates)**.** *Let $\mathcal{T}_i = (P_i, T_i, I_i, O_i, H_i, m_{0,i}, W_i, \mathbb{N}[\pi], \Pi_i, \mathcal{D}_i)$ for $i = 1, 2$ be π-parametrised templates over the interface places $\mathcal{I} = P_1 \cap P_2$. The merge of $\mathcal{T}_1$ and $\mathcal{T}_2$ is the π-parametrised template merge$(\mathcal{T}_1, \mathcal{T}_2) = (P, T, I, O, H, m_0, W, \mathbb{N}^n, \Pi, \mathcal{D})$ with:*

- $P = P_1 \cup P_2$,
- $T = T_1 \uplus T_2$, $O = O_1 \uplus O_2$, $I = I_1 \uplus I_2$, $H = H_1 \uplus H_2$,
- $m_0 = m_{0,1} + m_{0,2}$,
- $W = W_1 \uplus W_2$, $\Pi = \Pi_1 \uplus \Pi_2$, $\mathcal{D} = \mathcal{D}_1 \uplus \mathcal{D}_2$.

Since a DFT can consist of more than two components, we define the merge of multiple templates over $\mathcal{I}_{\mathcal{F}}$ by concatenating the binary merge. Let $\mathbb{T}$ denote a non-empty set of templates over some $\mathcal{I}$ and let $\mathcal{T}$ be a template over $\mathcal{I}$. The merge operator is associative and commutative since the union of sets satisfies these properties. Hence, we let

$$\mathsf{merge}(\mathbb{T} \cup \{\mathcal{T}\}) = \begin{cases} \mathsf{merge}(\mathcal{T}, \mathsf{merge}(\mathbb{T})) & |\mathbb{T}| > 1 \\ \mathsf{merge}(\mathcal{T}, \mathcal{T}') & \mathbb{T} = \{\mathcal{T}'\}\,. \end{cases}$$

To obtain the GSPN for an entire repairable DFT, each RFT element v with type $t(v)$ is transformed into a GSPN via the templates $\mathsf{templ}_{t(v)}(v)$ as described before. Additionally, the initialisation template $\mathsf{templ}_{\mathsf{init}}$ is required. Thus, we define the GSPN template of an entire RFT as follows.

Definition 17 (Template for an RFT)**.** *Let $\mathcal{F} = (V, \sigma, t, top, FR_a, FR_p, RR)$ be an RFT, and moreover let $\big\{\mathsf{templ}_{t(v)}(v) \mid v \in \mathcal{F}\big\}$ be the set of templates over $\mathcal{I}_{\mathcal{F}}$, each with priority variable π_v. The GSPN $\mathcal{T}_{\mathcal{F}}$ for $\mathcal{F}$ with places $P \supset \mathcal{I}_{\mathcal{F}}$ is defined by*

$$\mathcal{T}_{\mathcal{F}} = \mathsf{merge}\left(\big\{\mathsf{templ}_{t(v)}(v) \mid v \in \mathcal{F}\big\} \cup \{\mathsf{templ}_{init}\}\right).$$

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

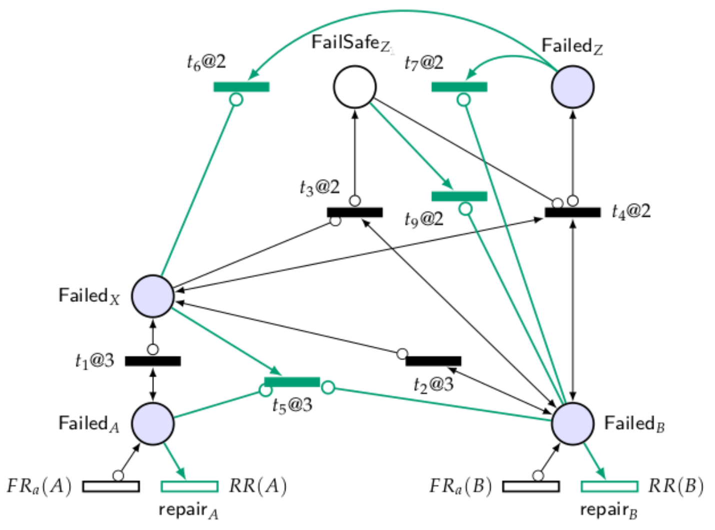


Figure 21: Simplified resulting GSPN for a repairable DFT

Recall that we already converted the DFT depicted in Figure 5a to the GSPN shown in Figure 5c. When doing this, we assumed that the DFT consisted of non-repairable elements. We now consider a repairable variant.

Example 18 (Translation of an RFT to a GSPN)**.** *Consider the DFT in Figure 5a again. We assume that all BEs are repairable. The corresponding GSPN for the RFT is depicted in Figure 21. Notice that we use a simplified version of the GSPN as we want to focus on the repair mechanisms. The parts that were added to model repairs are highlighted in green.*

The BEs A and B are now repairable as reflected by the corresponding repair$_A$ and repair$_B$ transitions. Thus, in addition to the failure rate $FR_a(v)$, each BE v has a repair rate $RR(v)$.

Consider the trace $\mathbf{f}_A\mathbf{f}_B\mathrm{f}_X\mathrm{f}_Z\mathbf{r}_A\mathbf{r}_B\mathrm{r}_X\mathrm{r}_Z$. The failed OR X is repaired if both its children are operational. In this case the immediate transition t_5 removes the token from Failed$_X$, indicating the repair of X. A failed PAND Z is repaired if at least one of its children X or B is operational and either t_6 or t_7 fires and removes the token from Failed$_Z$. Since Z is the top-level event, this reflects the repair of the entire DFT. In our case, Z is failed because A failed before B.

Consider the trace $\mathbf{f}_B\oslash_Z\mathbf{f}_A\mathrm{f}_X\mathbf{r}_B$. In this trace, Z is fail-safe because B failed before A. After the repair of B, the PAND Z returns to non-fail-safe. In this case, t_9 fires and removes the token from FailSafe$_Z$.

Notice that we already assigned priorities to the transitions in this example. In the following we explain how priorities affect the behaviour of the RFT and how conflicts between GSPN transitions can be resolved.

5.3 Semantic Issues

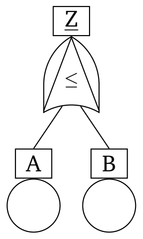
In this section, we analyse semantic aspects that need to be considered when including repairs in DFTs.

5.3.1 Failures Caused by Repairs

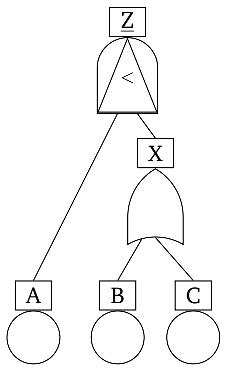
Intuitively, a successful repair operation should decrease the number of failed elements. However, this does not apply to all situations. For example, consider the DFT in Figure 22a and the trace

$$\tau = \mathbf{f}_B\mathbf{f}_A\oslash_Z.$$

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.



(a) Failures caused by repairs



(b) Differences in failure behaviour

Figure 22: Aspects of priority gates

The two children A and B of the POR gate Z failed but Z itself remains operational since B failed prior to A. In this situation, a repair of B would now lead to a failure of Z:

$$\tau' = \mathbf{f}_B\mathbf{f}_A\oslash_Z\mathbf{r}_B\mathrm{f}_Z.$$

This can easily be seen by looking at the corresponding "reduced" failure trace $\tau'' = \mathbf{f}_A\mathrm{f}_Z$ which omits failures that are "cancelled out" by repairs.

Repairs can increase the number of failed elements.

5.3.2 Remarks on Priority Gates

In this section, we elaborate on the difficulties that arise with priority gates with regard to repairs. Consider the DFT in Figure 22b and the following trace

$$\tau = \mathbf{f}_B\mathrm{f}_X\oslash_Z\mathbf{f}_A\mathbf{f}_C.$$

Since the failure of B led to a failure of X before A failed, the PAND gate Z is fail-safe. The problem arises when the child B of X is repaired, i.e., trace

$$\tau' = \mathbf{f}_B\mathrm{f}_X\oslash_Z\mathbf{f}_A\mathbf{f}_C\mathbf{r}_B.$$

According to the GSPN template for the OR gate, X is not repaired since its other child C is still failed. As a consequence, the failure order of Z is still violated and Z remains fail-safe. However, we obtain a different result if we consider the reduced failure trace

$$\tau'' = \mathbf{f}_A\mathbf{f}_C\mathrm{f}_X\mathrm{f}_Z.$$

Here, A failed before C (and X) and therefore Z fails as well.

Failures and repairs do not necessarily cancel each other out.

5.3.3 Remarks on Repair Dependencies

In this section, we focus on some aspects for repair dependencies that need further consideration. The first is that the proposed encoding of FDEPs only propagates failures but not repairs. However, we can easily extend our reasoning to repairs by combining the FDEP with an RDEP as exemplarily depicted in Figure 24. Here, BE C is a trigger for both the FDEP gate $D1$ and the RDEP gate $D2$. Thus, if the trigger C fails, the dependent element E fails. If C is repaired, E is repaired as well.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

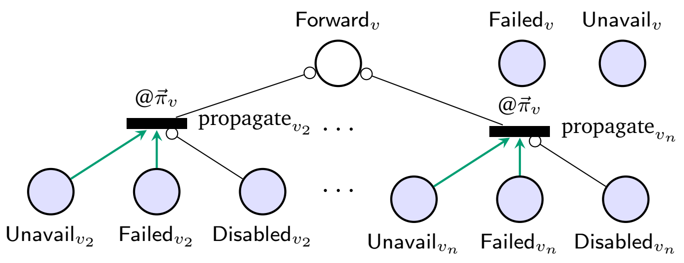


Figure 23: Alternative GSPN template for RDEP

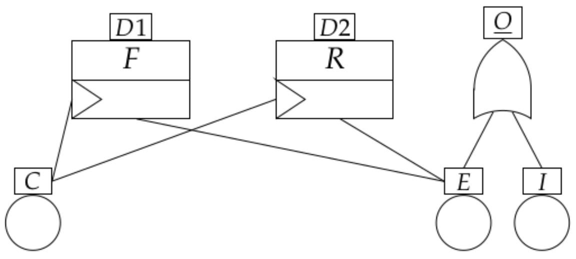


Figure 24: Modelling of external and internal causes of failures

Lastly, we demonstrate that it is not required to distinguish between an *internal* and an *external* failure of a BE. The former is modelled by the timed transition of the BE whereas the latter is caused by the propagation of an FDEP gate. We can model the distinction by using an additional BE and an OR gate. We explain the underlying idea by the following example.

Example 19. *Consider the DFT in Figure 24 that models an element O which can fail due to internal or external causes. The BE I reflects the internal part and it can only fail and be repaired by itself. An example of an internal failure and repair is the trace $\mathbf{f}_I \mathrm{f}_O \mathbf{r}_I \mathrm{r}_O$.*

In order to model the external part, we use the BE E that can neither fail nor get repaired by itself but is the dependent child of both an FDEP and an RDEP. The trigger C models the external cause of failure. As a result, E can only fail due to the failure of C and is only repaired if C is repaired. An example of an external failure and repair is the trace $\mathbf{f}_C \mathrm{f}_E \mathrm{f}_O \mathbf{r}_C \mathrm{r}_E \mathrm{r}_O$.

The whole element O is operational only if both the internal part I and the external part E are operational.

FDEP and RDEP allow to explicitly model forwarding of (1) only failures, (2) only repairs or (3) both. (Repair) dependencies can be used to distinguish between internal and external failure causes.

5.3.4 Failure and Repair Propagation

As mentioned earlier, the priority assignment of the transitions in Figure 21 has an impact on the behaviour of the DFT. Consider again the DFT in Figure 5a and the corresponding GSPN in Figure 21. The transitions that correspond to the GSPN part of the OR gate X have a higher priority than those of the PAND gate Z. Assume that every transition has the same priority instead. We consider the failure $\mathbf{f}_B$ which means that both transitions t_2 and t_4 are enabled. Firing t_2 leads to a failure of Z and corresponds to trace $\tau_1 = \mathbf{f}_B \mathrm{f}_X \mathrm{f}_Z$. Firing t_4 lead to Z becoming fail-safe and corresponds to trace $\tau_2 = \mathbf{f}_B \oslash_Z \mathrm{f}_X \oslash_Z$. Thus, the result of a failure also depends on the priority assignment within the GSPN.

As specified in [20], we can control the propagation of failures by assigning certain priorities to the transitions: if we assign the same priority to every transition, then the propagations can occur in every

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

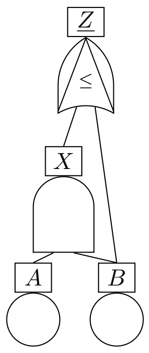


Figure 25: Example for repair propagation

possible order. But if we want to fix that the propagation proceeds in a bottom-up fashion, each gate must be assigned a lower priority than its children. Since all transitions belonging to the same DFT element v are affected in both cases, the repair propagation is modified correspondingly. In the following we show that the order in which repairs are propagated influences the behaviour as well.

Consider the DFT in Figure 25 and the trace

$$\tau = \mathbf{f}_B \oslash_Z \mathbf{f}_A \mathrm{f}_X \oslash_Z.$$

The POR gate Z is fail-safe as the failure of B occurred before the failure of X. Now assume that B is repaired, i.e., $\mathbf{r}_B$ happens. The status of Z depends on the evaluation order. If X is evaluated first, we obtain the following trace

$$\tau_1 = \mathbf{f}_B \oslash_Z \mathbf{f}_A \mathrm{f}_X \oslash_Z \mathbf{r}_B \mathrm{r}_X \oslash_Z.$$

Here, X is repaired first. Afterwards, Z is considered and it stays operational, because both its children are operational. However, if Z is evaluated before X, the corresponding trace is

$$\tau_1 = \mathbf{f}_B \oslash_Z \mathbf{f}_A \mathrm{f}_X \oslash_Z \mathbf{r}_B \mathrm{f}_F \mathrm{r}_X \mathrm{r}_Z.$$

The repair of B would lead to a failure of Z, because X is still failed at the moment of evaluation. After evaluation of X, X is repaired and therefore Z will be repaired as well. However, since the failure could be propagated by another gate in the meantime, it may have a permanent effect. If this is considered unintended behaviour, it can again be prevented by assigning a higher priority to the children in comparison to the gate.

The status of DFT elements depends on the order in which failures and repairs are propagated. The propagation order can be captured by the priority assignment in the GSPN.

5.3.5 FDEP and RDEP Forwarding

As discussed in [20], the order of evaluating FDEP gates can influence the behaviour of DFTs. Consider the DFT in Figure 26a and the failure of BE B. If the FDEP D forwards the failure to A before Z is evaluated, we obtain the following trace

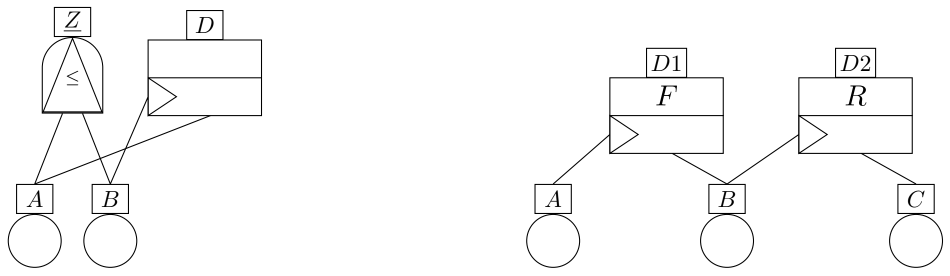
$$\tau_1 = \mathbf{f}_B \mathrm{f}_A \mathrm{f}_Z.$$

Z fails since the effect is that the failures of A and B happen simultaneously. However, if Z is evaluated before D, we obtain a different trace

$$\tau_2 = \mathbf{f}_B \oslash_Z \mathrm{f}_A \oslash_Z.$$

The failure of B is interpreted to happen strictly before the failure of A. Thus, Z becomes fail-safe. We distinguish between evaluating FDEPs (1) before, (2) after or (3) interleaved with failure propagation in gates.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.



(a) FDEP forwarding [20] (b) FDEP and RDEP forwarding

Figure 26: Examples for dependency forwarding

Similar to FDEPs, we also need to specify when repairs are forwarded through RDEPs. Consider the DFT in Figure 26b with two dependency gates and the trace

$$\tau = \mathbf{f}_A \mathrm{f}_B \mathbf{f}_C.$$

If B is repaired, both dependencies can act. Either RDEP $D2$ first forwards the repair from B to C or FDEP $D1$ first forwards the failure from A to B. The former results in a trace where C is repaired:

$$\tau_1 = \mathbf{f}_A \mathrm{f}_B \mathbf{f}_C \mathbf{r}_B \mathrm{r}_C \mathrm{f}_B.$$

The latter results in a trace which prevents the repair of C:

$$\tau_1 = \mathbf{f}_A \mathrm{f}_B \mathbf{f}_C \mathbf{r}_B \mathrm{f}_B.$$

Thus, it is very important when RDEP gates are evaluated, also in comparison with FDEP gates.

We distinguish between the evaluation of RDEPs (1) before, (2) after or (3) interleaved with FDEPs. Depending on the order of evaluating FDEPs and gates, we can specify if gates are evaluated before or after RDEPs by assigning the respective priorities. Thus, we can specify that for example FDEPs are evaluated before RDEPs and RDEPs before the remaining gates by assigning the highest priorities to transitions belonging to an FDEP, and the second highest ones to RDEPs. Accordingly, the lowest priorities are assigned to the remaining gates. Thus, for all $f \in \mathcal{F}_{\text{FDEP}}$, $r \in \mathcal{F}_{\text{RDEP}}$ and $g \in \mathcal{F} \setminus (\mathcal{F}_{\text{FDEP}} \cup \mathcal{F}_{\text{RDEP}})$ we have $\pi_f > \pi_r$ and $\pi_r > \pi_g$. Interleaved evaluation of FDEPs and RDEPs is realised by assigning the same priority to each transition.

FDEPs can be evaluated (1) before, (2) after or (3) interleaved with gates. RDEPs can be evaluated (a) before, (b) after or (c) interleaved with FDEPs. Priorities in the GSPN specify the order of evaluation.

5.3.6 Spare Races

In this section, we elaborate on the handling of so called *spare races*. As described in [20], races may occur if two or more SPARE gates share a child, and if their triggering events fail simultaneously. In this situation, multiple SPAREs try to claim the same child. The simultaneous failure of the children is possible if, e.g., an FDEP immediately propagates the failure to the dependent children once the trigger fails.

With the possibility of repairs, spare races can even emerge in scenarios that do not require multiple children to fail at once. For instance, consider the DFT in Figure 27 and the following trace

$$\tau = \mathrm{cl}_A^{S1} \mathrm{cl}_B^{S2} \mathbf{f}_B \mathrm{cl}_C^{S2} \mathbf{f}_A \mathrm{f}_{S1} \mathrm{f}_P \mathbf{f}_C \mathrm{f}_{S2}.$$

POR gate P failed because the SPARE gate $S1$ failed before $S2$.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

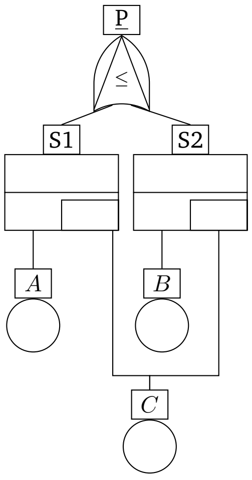


Figure 27: Example of a spare race

If child C is repaired, a spare race may arise, since C can be claimed by both $S1$ or $S2$. If $S1$ "wins", then P becomes fail-safe as can be seen by the trace

$$\tau' = \mathrm{cl}_A^{S1}\mathrm{cl}_B^{S2}\mathbf{f}_B\mathrm{cl}_C^{S2}\mathbf{f}_A\mathrm{f}_{S1}\mathrm{f}_P\mathbf{f}_C\mathrm{f}_{S2}\mathbf{r}_C\mathrm{cl}_C^{S1}\mathrm{r}_{S1}\mathrm{r}_P.$$

If $S2$ "wins", then P remains failed:

$$\tau' = \mathrm{cl}_A^{S1}\mathrm{cl}_B^{S2}\mathbf{f}_B\mathrm{cl}_C^{S2}\mathbf{f}_A\mathrm{f}_{S1}\mathrm{f}_P\mathbf{f}_C\mathrm{f}_{S2}\mathbf{r}_C\mathrm{cl}_C^{S2}\mathrm{r}_{S2}\emptyset_P.$$

In the GSPN the spare race is represented by a conflict between the two claim transitions.

The solutions proposed in [20] for DFTs can be applied to the GSPN semantics for RFTs as well: the conflict can be resolved either by randomisation or non-determinism. This choice can be specified in the GSPN by adapting the partition of the immediate transitions. Details are provided in [21, Sect. 4.6].

Repairs can lead to spare races which must be resolved either by randomisation or by non-determinism. The desired behaviour can be specified by setting the partition in the GSPN.

5.3.7 Allowing Dependencies Triggered by Repairable Gates

In Definition 8 we required that all children of an FDEP are BEs. If we drop this restriction (for the first child) and allow dependencies that are triggered by gates, further specifications regarding the propagation of failures and repairs by dependencies are necessary. According to the GSPN semantics proposed in [20], FDEPs forward failures immediately in order to reflect bottom-up failure propagation. This is realised by evaluating children of dynamic gates strictly before their parents, which is enabled by providing the following priority assignments:

$$\pi_v < \pi_{v_i} \qquad \forall v \in \mathcal{F}_{\mathsf{PAND}} \cup \mathcal{F}_{\mathsf{POR}} \cup \mathcal{F}_{\mathsf{SPARE}} \cup \mathcal{F}_{\mathsf{SEQ}}, \forall\, i \in \{1, \ldots, |\sigma(v)|\}\,.$$

In static gates, the order of failures is irrelevant and the priorities are specified in a non-strict way:

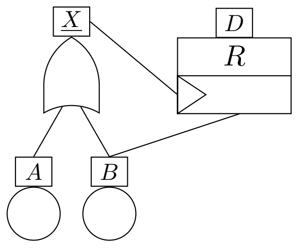
$$\pi_v \leq \pi_{v_i} \qquad \forall v \in \mathcal{F}_{\mathsf{AND}} \cup \mathcal{F}_{\mathsf{OR}} \cup \mathcal{F}_{\mathsf{VOT}k/n}, \forall\, i \in \{1, \ldots, |\sigma(v)|\}\,.$$

Triggers of FDEPs are assigned a priority not smaller than the priorities of the dependent children:

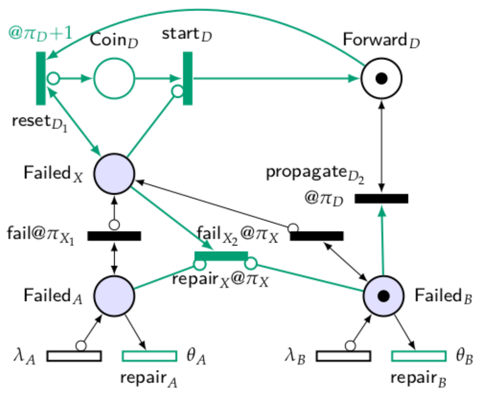
$$\pi_{f_1} \geq \pi_f \geq \pi_{f_i} \qquad \forall f \in \mathcal{F}_{\mathsf{FDEP}}, \forall\, i \in \{2, \ldots, |\sigma(f)|\}\,.$$

In the following, we will focus on RDEP gates but similar problems arise by considering only FDEP gates triggered by repairable gates. We apply the bottom-up failure propagation to repairable DFTs

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.



(a) RDEP triggered by gate



(b) GSPN for RDEP triggered by gate

Figure 28: Example for cyclic repair propagation

and assume that repairs are propagated in a bottom-up manner as well. Thus, we extend the priority assignment as follows:

$$\pi_{f_1} \geq \pi_f \geq \pi_{f_i} \qquad \forall f \in \mathcal{F}_{\mathsf{RDEP}}, \forall\, i \in \{2, \ldots, |\sigma(f)|\}\,.$$

Furthermore, a conflict between enabled transitions is resolved by non-determinism since every immediate transition is a separate partition. As the following example will show, non-determinism can occur in DFTs with dependencies triggered by gates that are repairable.

Consider the DFT in Figure 28a and the following trace

$$\tau = \mathbf{f}_B \mathrm{f}_Z \mathbf{r}_B \mathrm{r}_Z \mathbf{f}_B.$$

Figure 28b depicts the corresponding GSPN after the trace τ. Place Forward$_D$ contains a token indicating that RDEP D forwards the repair of X. Additionally, a token is in Failed$_B$ indicating that B has failed. In the GSPN, two transitions are now in conflict: fail$_{X_2}$ and propagate$_{D_2}$. If fail$_{X_2}$ fires first, a token is placed in Failed$_X$, and afterwards the token in Forward$_D$ is removed. Note that transition reset$_{D_1}$ always fires before propagate$_{D_2}$ because it has higher priority. This behaviour corresponds to trace

$$\tau_1 = \mathbf{f}_B \mathrm{f}_Z \mathbf{r}_B \mathrm{r}_Z \mathbf{f}_B \mathrm{f}_X.$$

In this trace, the repair propagation is stopped and B remains failed.

However, if propagate$_{D_2}$ fires first, the token is removed from Failed$_B$ and B becomes operational. This corresponds to trace

$$\tau_2 = \mathbf{f}_B \mathrm{f}_Z \mathbf{r}_B \mathrm{r}_Z \mathbf{f}_B \mathrm{r}_B.$$

Thus, the order of failures propagation decides whether B is failed or operational.

Intuitively, the conflict can be avoided by assigning appropriate priorities π_X and π_D. However, the priorities as described above induce the following restrictions:

$$\pi_A \geq \pi_X, \quad \pi_B \geq \pi_X, \quad \pi_X \geq \pi_D, \quad \text{and} \quad \pi_D \geq \pi_B.$$

Thus, the only valid priority assignment induces that B, X and D get the same priority since every other priority assignment would violate the constraints. Consequently, the conflict between the transitions fail$_{X_2}$ and propagate$_{D_2}$ can only be solved by non-determinism.

According to the monolithic semantics elaborated in [20] that allows dependencies triggered by gates, gates are evaluated strictly after their children and dependencies are evaluated strictly after all gates.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

If we apply this principle to repair dependencies, it implies that $\pi_X > \pi_D$, $\pi_A > \pi_X$ and $\pi_B > \pi_X$. If we apply these priorities to the GSPN depicted in Figure 28, only transition repair_X is enabled and only trace τ_1 is possible.

Allowing dependencies which are triggered by gates can lead to multiple possible orders of failure propagation. Priorities need to be specified to ensure that failures and repairs are propagated in the desired order.

5.4 Tool Support

As described in Section 4, the translation from DFTs to GSPNs is implemented in the probabilistic model checker STORM [17], more concretely its library STORM-DFT which provides tool support for DFT analysis. STORM-DFT reads DFTs from a custom JSON format or from the Galileo text format. The resulting GSPN can be exported into formats such as the GreatSPN Editor format or pnpro for further analysis [20].

In [21], this functionality was extended to Repairable DFTs. First of all, parsing support was extended to incorporate new types of gates such as RDEP. Moreover, for SPARE gates the claiming strategy (either left-to-right or non-deterministic choice) and the behaviour after a child-repair (keep currently claimed child or reclaim new child) can be specified. For BEs, the repair rate can be given via an optional argument.

After parsing an RFT, STORM-DFT automatically identifies the repairable DFT elements. RDEPs and BEs with non-zero repair rate are always repairable. All other gates are set to repairable if at least one child is repairable.

Following Definition 17, the transformation from an RFT to a GSPN is implemented compositionally. Each DFT element is translated individually according to the corresponding GSPN template as described before. The priorities for the immediate transitions are calculated such that they satisfy the constraints from [20] and ensure a bottom-up failure and repair propagation in the RFT. However, if needed, the priorities in the GSPN can also be changed later on to model a different order of propagations in the RFT.

6 Conclusion

In this report, we gave an overview of Fault Tree extensions with repair and maintenance aspects. Moreover, we investigated some issues concerning their semantics, and introduced a compositional approach to specifying the latter by defining a modular translation of Repairable Fault Trees into Generalised Stochastic Petri Nets (GSPNs). Finally, we provided links to existing implementations of parts of a corresponding tooling environment to be developed in the further course of the MISSION project.

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[2] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *Transactions on Computer Systems*, 2(2):93–122, 1984.

[3] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, and G. Franceschinis. 30 years of GreatSPN. In *Principles of Performance and Reliability Modeling and Evaluation*, pages 227–254. Springer, 2016. doi:10.1007/978-3-319-30599-8_9.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

[4] M. Beccuti, D. Codetta-Raiteri, G. Franceschinis, and S. Haddad. Non deterministic repairable fault trees for computing optimal repair strategy. In *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools*. ICST, 2008. doi:10.4108/ICST.VALUETOOLS2008.4411.

[5] M. Beccuti, G. Franceschinis, D. Codetta-Raiteri, and S. Haddad. Parametric NdRFT for the derivation of optimal repair strategies. In *IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 399–408. IEEE, 2009. doi:10.1109/DSN.2009.5270312.

[6] M. Beccuti, G. Franceschinis, D. Codetta-Raiteri, and S. Haddad. Computing optimal repair strategies by means of NdRFT modeling and analysis. *The Computer Journal*, 57(12):1870–1892, 2014. doi:10.1093/comjnl/bxt134.

[7] A. Bobbio and D. Codetta-Raiteri. Parametric fault trees with dynamic gates and repair boxes. In *Annual Symposium Reliability and Maintainability*, pages 459–465, 2004. doi:10.1109/RAMS.2004.1285491.

[8] H. Boudali, P. Crouzen, and M. Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 708–717. IEEE, 2007. doi:10.1109/DSN.2007.37.

[9] D. Codetta-Raiteri. Integrating several formalisms in order to increase fault trees' modeling power. *Reliability Engineering & System Safety*, 96(5):534–544, 2011. doi:https://doi.org/10.1016/j.ress.2010.12.027.

[10] D. Codetta-Raiteri, G. Franceschinis, M. Iacono, and V. Vittorini. Repairable fault tree for the automatic evaluation of repair policies. In *International Conference on Dependable Systems and Networks*, pages 659–668. IEEE, 2004. doi:10.1109/DSN.2004.1311936.

[11] D. Coppit, K. Sullivan, and J. Dugan. Formal semantics of models for computational engineering: a case study on dynamic fault trees. In *11th International Symposium on Software Reliability Engineering*, pages 270–282. IEEE, 2000. doi:10.1109/ISSRE.2000.885878.

[12] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Fault trees and sequence dependencies. In *RAMS*, pages 286–293, 1990. doi:10.1109/ARMS.1990.67971.

[13] C. Eisentraut, H. Hermanns, J.-P. Katoen, and L. Zhang. A semantics for every GSPN. In *Application and Theory of Petri Nets and Concurrency*, pages 90–109. Springer, 2013. doi:10.1007/978-3-642-38697-8_6.

[14] F. Flammini, N. Mazzocca, M. Iacono, and S. Marrone. Using repairable fault trees for the evaluation of design choices for critical repairable systems. In *Ninth IEEE International Symposium on High-Assurance Systems Engineering*, pages 163–172. IEEE, 2005. doi:10.1109/HASE.2005.26.

[15] G. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, and V. Vittorini. Towards an object based multi-formalism multi-solution modeling approach. In *Proceedings of the Second Workshop on Modelling of Objects, Components and Agents*, volume 561 of *DAIMI PB*, pages 47–65, 2002. URL: https://tidsskrift.dk/daimipb/issue/view/1399.

[16] D. Guck. *Reliable systems: fault tree analysis via Markov reward automata*. PhD thesis, University of Twente, Enschede, Netherlands, 2017. doi:10.3990/1.9789036542913.

[17] C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk. The probabilistic model checker Storm. *Int. J. Softw. Tools Technol. Transf.*, 24(4):589–610, 2022. doi:10.1007/s10009-021-00633-z.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

[18] International Electrotechnical Commission, Geneva, Switzerland. *Fault Tree Analysis (FTA)*, 2006.

[19] S. Junges, D. Guck, J.-P. Katoen, and M. Stoelinga. Uncovering dynamic fault trees. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 299–310. IEEE, 2016. doi:10.1109/DSN.2016.35.

[20] S. Junges, J.-P. Katoen, M. Stoelinga, and M. Volk. One net fits all. In *Application and Theory of Petri Nets and Concurrency*, pages 272–293. Springer, 2018. doi:10.1007/978-3-319-91268-4_14.

[21] H. Mertens. Repairs in dynamic fault trees: a Petri net semantics. Bachelor's thesis, RWTH Aachen University, Aachen, Germany, 2019.

[22] R. E. Monti, P. R. D'Argenio, and C. E. Budde. A compositional semantics for repairable fault trees with general distributions, 2019. doi:10.48550/ARXIV.1910.10507.

[23] W. Reisig. *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013. doi:10.1007/978-3-642-33278-4.

[24] E. Ruijters, D. Guck, P. Drolenga, and M. Stoelinga. Fault maintenance trees: Reliability centered maintenance via statistical model checking. In *Annual Reliability and Maintainability Symposium*. IEEE, 2016. doi:10.1109/RAMS.2016.7447986.

[25] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15–16:29– 62, 2015. doi:10.1016/j.cosrev.2015.03.001.

[26] M. Volk. *Dynamic Fault Trees: Semantics, Analysis and Applications*. PhD thesis, RWTH Aachen University, Aachen, Germany, 2022.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.