One Net Fits All
A Unifying Semantics of Dynamic Fault Trees Using GSPNs

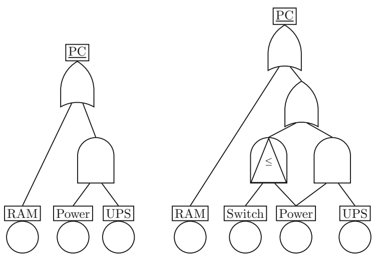
Sebastian Junges[1], Joost-Pieter Katoen[1,2], Mariëlle Stoelinga[2,3], and Matthias Volk[1](✉)

[1] Software Modeling and Verification, RWTH Aachen University, Aachen, Germany
matthias.volk@cs.rwth-aachen.de
[2] Formal Methods and Tools, University of Twente, Enschede, Netherlands
[3] Department of Software Science, Radboud University Nijmegen, Nijmegen, Netherlands

Abstract. Dynamic Fault Trees (DFTs) are a prominent model in reliability engineering. They are strictly more expressive than static fault trees, but this comes at a price: their interpretation is non-trivial and leaves quite some freedom. This paper presents a GSPN semantics for DFTs. This semantics is rather simple and compositional. The key feature is that this GSPN semantics unifies *all* existing DFT semantics from the literature. All semantic variants can be obtained by choosing appropriate priorities and treatment of non-determinism.

1 Introduction

Fault trees (FTs) [1] are a popular model in reliability engineering. They are used by engineers on a daily basis, are recommended by standards in e.g., the automotive, aerospace and nuclear power industry. Various commercial and academic tools support FTs; see [2] for a survey. FTs visualise how combinations of components faults (their leaves, called basic events) lead to a system failure. Inner tree nodes (called gates) are like logical gates in circuits such as AND and OR. The simple FT in Fig. 1(a) models that a PC fails if either the RAM, or both power and UPS fails.



(a) FT (b) Dynamic FT

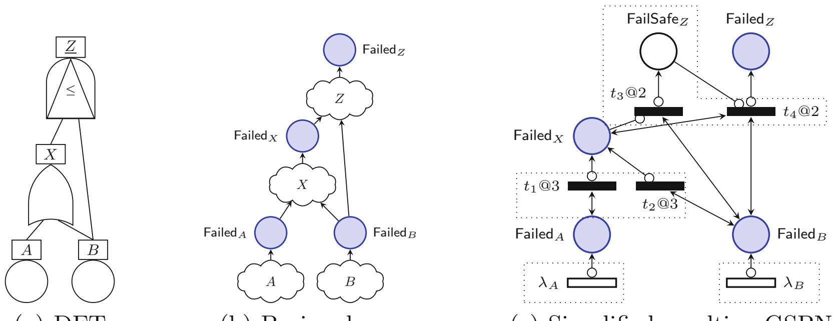
Fig. 1. Fault tree examples

Standard FTs appeal due to their simplicity. However, they lack expressive power to faithfully model many aspects of realistic systems such as spare components, redundancies, etc. This deficiency is remedied by *Dynamic Fault Trees*

This work is supported by the CDZ project CAP, the DFG RTG 2236 “UnRAVeL”, the STW project 154747 SEQUOIA, and the EU project SUCCESS.

© Springer International Publishing AG, part of Springer Nature 2018
V. Khomenko and O. H. Roux (Eds.): PETRI NETS 2018, LNCS 10877, pp. 272–293, 2018.
https://doi.org/10.1007/978-3-319-91268-4_14

(DFTs, for short) [3]. They involve a variety of new gates such as spares and functional dependencies. These gates are dynamic as their behaviour depends on the failure history. For instance, the DFT in Fig. 1(b) extends our sample FT. If the power fails while the switch is operational, the system can switch to the UPS. However, if the power fails after the switch failed, their parent PAND-gate causes the system to immediately fail[1]. The expressive power of DFTs allows for modelling complex failure combinations succinctly. This power comes at a price: the interpretation of DFTs leaves quite some freedom and the complex interplay between the gates easily leads to misinterpretations [4]. The DFT in Fig. 2(a) raises the question whether B's failure first causes X to fail which in turn causes Z to fail, or whether B's failure is first propagated to Z making it impossible for Z to fail any more? These issues are not just of theoretical interest. Slightly different interpretations may lead to significantly divergent reliability measures and give rise to distinct underlying stochastic (decision) processes.



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

Fig. 2. Compositional semantics of DFTs using GSPNs

This paper defines a *unifying* semantics of DFTs using generalised stochastic Petri nets (GSPNs) [5,6]. The use of GSPNs to give a meaning to DFTs is not new; GSPN semantics of (dynamic) fault trees have received quite some attention in the literature [7–10]. Many DFT features are naturally captured by GSPN concepts, e.g., the failure of a basic event can be modelled by a timed transition, the instantaneous failure of a gate by an immediate transition, and places can be exploited to pass on failures. This work builds upon the GSPN-based semantics in [7]. The appealing feature of our GSPN semantics is that it *unifies* various existing DFT semantics, in particular various state-space based meanings using Markov models [11–13], such as continuous-time Markov Chains (CTMC), Markov automata (MA) [14], a form of continuous-time Markov decision process, or I/O interactive Markov chain (IOIMC) [15]. The key is that we capture all these distinct interpretations by a *single* GSPN. The structure of the net is the same for all possible meanings. Only two net features vary: the transition priorities and the partitioning of immediate transitions. The former

[1] A PAND-gate fails if all its children fail in a left-to-right order.

steer the ordering of how failures propagate through a DFT, while the latter control the possible ways in which to resolve conflicts (and confusion) [16].

Table 1. Semantic differences between supported semantics

	Monolithic CTMC [11]	IOIMC [12]	Monolithic MA [13]	Orig. GSPN [7]	New GSPN
Tool support	Galileo [17]	DFTCalc [18]	Storm [19]	--	--
Underlying model	CTMC	IMC [15]	MA [14]	GSPN/CTMC [5,6]	GSPN/MA [16]
Priority gates	$\leq$	$<$	$\leq$	$<$	$\leq$ and $<$
Nested spares	Not supported	Late claiming	Early claiming	Not supported	Early claiming
Failure propagation	Bottom-up	Arbitrary	Bottom-up	Arbitrary	Bottom-up
FDEP forwarding	First	Interleaved	Last	Interleaved	First
Non-determinism	Uniform	True (everywhere)	True FDEP	Uniform	true (PAND, SPARE)

The benefits of a unifying GSPN are manifold. First and foremost, it gives insights in the choices that DFT semantics from the literature—and the tools realising these semantics—make. We show that already three DFT aspects distinguish them all: failure propagation, forwarding in functional dependencies, and non-determinism, see the last three rows in Table 1. Mature tool-support for GSPNs such as SHARPE [20], SMART [21], GreatSPN [22] and its editor [23] can be exploited for all covered DFT semantics. Thirdly, our *compositional* approach, with simple GPSNs for each DFT gate, is easy to extend with more gates. The compositional nature is illustrated in Fig. 2. The occurrence of an event like the failure of a DFT node is reflected by a dedicated (blue) place. The behaviour of a gate is represented by immediate transitions (solid bars) and auxiliary (white) places. Failing BEs are triggered by timed transitions (open bars).

Our framework allows for expressing different semantics by a mild variation of the GSPN; e.g., whether B's failure is first propagated to X or to Z can be accommodated by imposing different transition priorities. The paper supports a rich class of DFTs as indicated in Table 2. The first column refers to the framework, the next four columns to existing semantics from the literature, and the last column to a new instantiation with mild restrictions, but presumably more intuitive semantics. The meaning of the rows is clarified in Sect. 2.2.

Related Work. The semantics of DFTs is naturally expressed by a state-transition diagram such as a Markov model [11–13]. Support of nested dynamic

Table 2. Syntax supported by different semantics

DFT feature	Framework	Monolithic CTMC	IOIMC	Monolithic MA	Orig. GSPN	New GSPN
Share SPAREs	✓	✓	✓	✓	✗	✓
SPARE w/subtree	✓	✗	✓	✓	✗	✓
Shared primary	✓	✗	✓	✗	✗	✓
Priority gates	PAND/POR	PAND	PAND	PAND/POR	PAND	PAND/POR
Downward FDEPs	✓	✗	✓	✓	✗	✗
SEQs on gates	✗	✓	✗	✓	✗	✗
PDEP	✓	✗	✗	✓	✗	✓

gates is an intricate issue, and the resulting Markov model is often complex. To overcome these drawbacks, semantics using higher-order formalisms such as Bayesian Networks [24,25], Boolean logic driven Markov processes [26,27] or GSPNs [7,9] have been proposed. DFT semantics without an underlying state-space have also been investigated, cf. e.g., [28,29]. These semantics often consider restricted classes of DFTs, but can circumvent the state-space explosion. Fault trees have been expressed or extracted from domain specific languages for reliability analysis such as Hip-HOPS, which internally may use Petri net semantics [30]. For a preliminary comparison, we refer to [1,4]. Semantics for DFTs with repairs [8], or maintenance [31] are more involved [32], and not considered in this paper.

Organisation of the Paper. Section 2 introduces the main concepts of GSPNs and DFTs. Section 3 presents our compositional translation from DFTs to GSPNs for the most common DFT gate types. It includes some elementary properties of the obtained GSPNs and reports on prototypical tool-support. Section 4 discusses DFT semantics from the literature based on the unifying GSPN semantics. Section 5 concludes and gives a short outlook into future work. An extended version containing proofs and translations for more DFT gates can be found in [33].

2 Preliminaries

2.1 Generalised Stochastic Petri Nets

This section summarises the semantics of GSPNs as given in [16]. The GSPNs are (as usual) Petri nets with timed and immediate transitions. The former model the failure of basic events in DFTs, while the latter represent the instantaneous behaviour of DFT gates. Inhibitor arcs ensure that transitions do not fire repeatedly, to naturally model that components do not fail repeatedly. Transition weights allow to resolve possible non-determinism. Priorities will (as explained later) be 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) *G is a tuple $(P, T, I, O, H, m_0, W, \Pi\mathsf{Dom}, \Pi, \mathcal{D})$ where*

- *P is a finite set of* places*.*
- *$T = T_i \cup T_t$ is a finite set of* transitions*, partitioned into the set T_i of* immediate transitions *and the set T_t of* timed transitions*.*
- *$I, O, H \colon T \to (P \to \mathbb{N})$, the* input-, output- *and* inhibition-multiplicities *of each transition, respectively.*
- *$m_0 \in M$ is the* initial marking *with $M = P \to \mathbb{N}$ the set of* markings*.*
- *$W \colon T \to \mathbb{R}_{>0}$ are the* transition-weights*.*
- *$\Pi\mathsf{Dom}$ is the* priority domain *and $\Pi \colon T \to \Pi\mathsf{Dom}$ the* transition-priorities*.*
- *$\mathcal{D} \in 2^{T_i}$, a* partition *of the immediate transitions.*

For convenience, we write $G = (\mathcal{N}, W, \Pi\mathsf{Dom}, \Pi, \mathcal{D})$ and $\mathcal{N} = (P, T, I, O, H, m_0)$. The definition is as in [16] extended by priorities and with a mildly restricted (i.e., marking-independent) notion of partitions. An example GSPN is given in Fig. 2(c). Places are depicted by circles, transitions by open (solid) bars for timed (immediate) transitions. If $I(t, p) > 0$, we draw a directed arc from place p to transition t. If $O(t, p) > 0$, we draw a directed arc from t to p. If $H(t, p) > 0$, we draw a directed arc from p to t with a small circle at the end. The arcs are labelled with the multiplicities. For all gates in the main text, all multiplicities are one (and are omitted). Some gates in [33] require a larger multiplicity. Transition weights are prefixed with a w, transition priorities with an @, and may be omitted to avoid clutter.

We describe the GSPN semantics for $\Pi\mathsf{Dom} = \mathbb{N}$, and assume in accordance with [6] that for all $t \in T_t : \Pi(t) = 0$ and for all $t \in T_i : \Pi(t) = c > 0$. Other priority domains are used in Sect. 4. The semantics of a GSPN are defined by its *marking graph* which constitutes the state space of a MA. In each marking, a set of transitions are enabled.

Definition 2 (Concession, enabled transitions, firing). *The set $\mathit{conc}(m)$ of* conceded transitions *in $m \in M$ is:*

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

The set $\mathit{enabled}(m)$ of enabled transitions in *m is:*

$$\mathit{enabled}(m) = \mathit{conc}(m) \cap \{t \in T \mid \Pi(t) = \max_{t \in \mathit{conc}(m)} \Pi(t)\}$$

The effect of firing *$t \in \mathit{enabled}(m)$* on *$m \in M$ is a marking $\mathit{fire}(m, t)$ such that:*

$$\forall p \in P : \mathit{fire}(m, t)(p) = m(p) - I(t)(p) + O(t)(p).$$

Example 1. Consider again the GSPN in Fig. 2(c). Let $m \in M$ be a marking with $m(\mathsf{Failed}_B) = 1$ and $m(p) = 0$ for all $p \in P \setminus \{\mathsf{Failed}_B\}$. Then the transitions t_2 and t_3 have concession, but only t_2 is enabled. Firing t_2 on m leads to the marking m' with $m'(\mathsf{Failed}_B) = 1 = m'(\mathsf{Failed}_X)$, and $m'(p) = 0$ for $p \in \{\mathsf{Failed}_A, \mathsf{Failed}_Z, \mathsf{FailSafe}_Z\}$.

If multiple transitions are enabled in a marking m, there is a *conflict* which transition fires next. For transitions in different partitions, this conflict is resolved non-deterministically (as in non-stochastic Petri nets). For transitions in the same partition the conflict is resolved probabilistically (as in the GSPN semantics of [6]). Let $C = \mathsf{enabled}(m) \cap D$ be the set of enabled transitions in $D \in \mathcal{D}$. Then transition $t \in C$ fires next with probability $\frac{W(t)}{\sum_{t' \in C} W(t')}$. If in a marking only timed transitions are enabled, in the corresponding state, the sojourn time is exponentially distributed with exit rate $\sum_{t' \in C} W(t')$. If a marking enables both timed and immediate transitions, the latter prevail as the probability to fire a timed transition immediately is zero.

A Petri net is *k-bounded* for $k \in \mathbb{N}$ if for every place $p \in P$ and for every reachable marking $m(p) \leq k$. Boundedness of a GSPN is a sufficient criterion for the finiteness of the marking graph. A k-bounded GSPN has a *time-trap* if its marking graph contains a cycle $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m$ such that for all $1 \leq i \leq n$, $t_i \in T_i$. The absence of time-traps is important for analysis purposes.

2.2 Dynamic Fault Trees

This section, based on [13], introduces DFTs and their nodes, and gives some formal definitions for concise notation in the remainder of the paper. The DFT semantics are clarified in depth in the main part of the paper.

Fault trees (FTs) are directed acyclic graphs with typed nodes. Nodes without successors (or: *children*), are *basic events* (BEs). All other nodes are *gates.* BEs represent system components that can fail. Initially, a BE is *operational*; it *fails* according to a negative exponential distribution. A gate fails if its *failure condition* over its children is fulfilled. The key gates for static fault trees (SFTs) are typed AND and OR, shown in Fig. 3(b) and (c). These gates fail if all (AND) or at least one (OR) children have failed, respectively. Typically, FTs express for which occurrences of BE failures, a specifically marked node (*top-event*) fails.

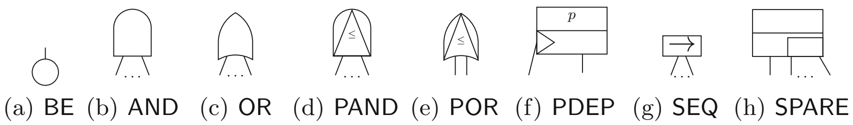


Fig. 3. Node types in ((a)–(c)) static and (all) dynamic fault trees.

SFTs lack an internal state—the failure condition is independent of the history. Therefore, SFTs lack expressiveness [2,4]. Several extensions commonly referred to as *Dynamic Fault Trees* (DFTs) have been introduced to increase the expressiveness. The extensions introduce new node types, shown in Fig. 3(d)–(h); 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 allow for order-dependent failure propagation. A *priority-and* (PAND) fails if all its children have failed in order from left to right. Figure 4(a) depicts a PAND with two children. It fails if A fails before B fails. The *priority-or* (POR) [29] only fails if the leftmost child fails before any of its siblings do. The semantics for simultaneous failures is discussed in Sect. 3.2. If a gate cannot fail any more, e.g., when B fails before A in Fig. 4(a), it is *fail-safe*.

Dependencies. Dependencies do not propagate a failure to their parents, instead, when their *trigger* (first child) fails, they update their *dependent events* (remaining children). We consider *probabilistic dependencies* (PDEPs) [24]. Once the trigger of a PDEP fails, its dependent events fail with probability p. Figure 4(b) shows a PDEP where the failure of trigger A causes a failure of BE B with probability 0.8 (provided it has not failed before). *Functional dependencies* (FDEPs) are PDEP with probability one (we omit the p then).

Restrictors. Restrictors limit possible failure propagations. *Sequence enforcers* (SEQ s) enforce 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 is met. The AND SF in Fig. 4(c) fails if A and B have failed (in any order), but the SEQ enforces that A fails prior to B. In contrast to Fig. 4(a), SF is never fail-safe. Another restrictor is the MUTEX (not depicted) which ensures that exactly one of its children fails.

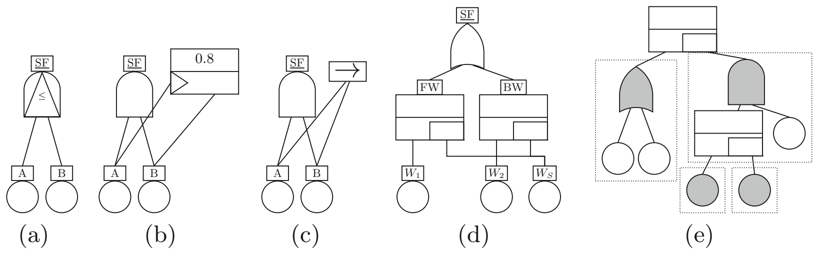


Fig. 4. Simple examples of dynamic nodes [13].

Spare Gates. Consider the DFT in Fig. 4(d) modelling (part of) a motor bike with a spare wheel. A bike needs two wheels to be operational. Either wheel can be replaced by the spare wheel, but not both. The spare wheel is less likely to fail until it is in use. Assume the front wheel fails. The spare wheel is available and used, but from now on, it is more likely to fail. If any other wheel fails, no spare wheel is available any more, and the parent SPARE fails.

SPAREs 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—modelling that all spare components have failed—the SPARE fails. Let us now consider activation. SPAREs may have (independent, i.e., disjoint) sub-DFTs as children. This includes nested SPAREs, SPAREs having SPAREs as children. A *spare module* is a set of nodes linked to each child of the SPARE. This child is the *module representative.* Figure 4(e) gives an example of spare modules (depicted by boxes) and the representatives (shaded nodes). Here, a spare module contains all nodes which have a path to the representative without an intermediate SPARE. Every leaf of a spare module is either a BE or a SPARE. Nodes outside of spare modules are *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, all other BEs with their passive failure rate.

DFTs Formally. We now give the formal definition of DFTs.

Definition 3 (DFT). *A* Dynamic Fault Tree *$\mathcal{F}$ (DFT) is a tuple (V, σ, Tp, top):*

- *V is a finite set of* nodes.
- *$\sigma \colon V \to V^*$ defines the (ordered)* children *of a node.*
- *$Tp \colon V \to \{\mathsf{BE}\} \cup \{\mathsf{AND}, \mathsf{OR}, \mathsf{PAND}, \dots\}$ defines the* node-type.
- *$top \in V$ is the* top event.

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

We assume (as all known literature) that DFTs are *well-formed*, i.e., (1) The directed graph induced by V and σ is acyclic, i.e., the transitive closure of the parent-child order is irreflexive, and (2) Only the leaves have no children.

For presentation purposes, for the main body we restrict the DFTs to *conventional DFTs*, and discuss how to lift the restrictions in [33].

Definition 4 (Conventional DFT). *A DFT is* conventional *if*

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

Restriction 1 restricts the DFTs syntactically and in particular ensures that spare modules can be seen as a single entity w.r.t. claiming and activation. Lifting this restriction to allow for non-disjoint spare modules raises new semantic issues [4]. Restriction 2 ensures that the fallible BEs are immediately deducible. Restriction 3 simplifies the presentation, in Sect. 4.4 we relax this restriction.

3 Generic Translation of DFTs to GSPNs

The goal of this section is to define the semantics of a DFT $\mathcal{F}$ as a GSPN $\mathcal{T}_{\mathcal{F}}$. We first introduce the notion of GSPN templates, and present templates for the common DFT node types such as BE, AND, OR, PAND, SPARE, and FDEP in Sect. 3.2. (Other node types such as PDEP, SEQ, POR, and so forth are treated in [33].) Sect. 3.3 presents how to combine the templates so as to obtain a template for an entire DFT. Some properties of the resulting GSPNs are described in Sect. 3.4 while tool-support is shortly presented in Sect. 3.5.

3.1 GSPN Templates and Interface Places

Recall the idea of the translation as outlined in Fig. 2. We start by introducing the set $\mathcal{I}_{\mathcal{F}}$ of *interface places*:

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

The places $\mathcal{I}_{\mathcal{F}}$ manage the communication for the different mechanisms in a DFT. A token is placed in Failed_v once the corresponding DFT gate v fails. On the failure of a gate, the tokens in the failed places of its children are not removed as a child may have multiple parents. Inhibitor arcs connected to Failed_v prevent the repeated failure of an already failed gate. The $\mathsf{Unavail}_v$ places are used for the claiming mechanism of SPAREs, Active_v manages the activation of spare components, while $\mathsf{Disabled}_v$ is used for SEQs.

Every DFT node is translated into some *auxiliary places*, transitions, and arcs. The arcs either connect interface or auxiliary places with the transitions. For each node-type, we define a template that describes how a node of this type is translated into a GSPN (fragment).

To translate contextual behaviour of the node, we use *priority variables* $\boldsymbol{\pi} = \{\boldsymbol{\pi}_v \mid v \in \mathcal{F}\}$. Transition priorities are functions over the priority variables $\boldsymbol{\pi}$, i.e., $\Pi\colon T \to \mathbb{N}[\boldsymbol{\pi}]$. These variables are instantiated with concrete values in Sect. 4, yielding priorities in $\mathbb{N}$. This section does not exploit the partitioning of the immediate transitions; the usage of this GSPN ingredient is deferred to Sect. 4. Put differently, for the moment it suffices to let each immediate transition constitute its (singleton) partition.

Definition 5 (GSPN-Template). *The GSPN* $\mathcal{T} = (\mathcal{N}, W, \mathbb{N}[\boldsymbol{\pi}], \Pi, \mathcal{D})$ *is a* ($\boldsymbol{\pi}$-parameterised) template over $\mathcal{I} \subseteq P$. *The* instantiation *of* $\mathcal{T}$ *with* $\boldsymbol{c} \in \mathbb{N}^n$ *is the GSPN* $\mathcal{T}[\boldsymbol{c}] = (\mathcal{N}, W, \mathbb{N}, \Pi', \mathcal{D})$ *with* $\Pi'(t) = \Pi(t)(\boldsymbol{c})$ *for all* $t \in T$.

The instantiation replaces the n priority variables by their concrete values.

3.2 Templates for Common Gate Types

We use the following notational conventions. Gates have n children. Interface places $\mathcal{I}$ are depicted using a blue shade; their initial marking is defined by the initialisation template, cf. Sect. 3.3. Other places have an initial token if it is drawn in the template. Transition priorities are indicated by @ and the priority function, e.g., @$\boldsymbol{\pi}_v$. The role of the priorities is discussed in detail in Sect. 4.

Basic Events. Figure 5(a) depicts the template $\mathsf{templ}_{\mathsf{BE}}(v)$ of BE v. It consists of two timed transitions, one for active failure and one for passive failure. Place Failed_v contains a token if v has failed. The inhibitor arcs emanating Failed_v prevent both transitions to fire once the BE has failed. A token in $\mathsf{Unavail}_v$ indicates that v is unavailable for claiming by a SPARE. If Active_v holds a token, the node fails with the active failure rate λ, otherwise it fails with the passive failure rate μ which typically is $c{\cdot}\lambda$ with $0 < c \leq 1$. The place $\mathsf{Disabled}_v$ contains a token if the BE is not supposed to fail. It is used in the description of the semantics of, e.g., SEQ in [33].

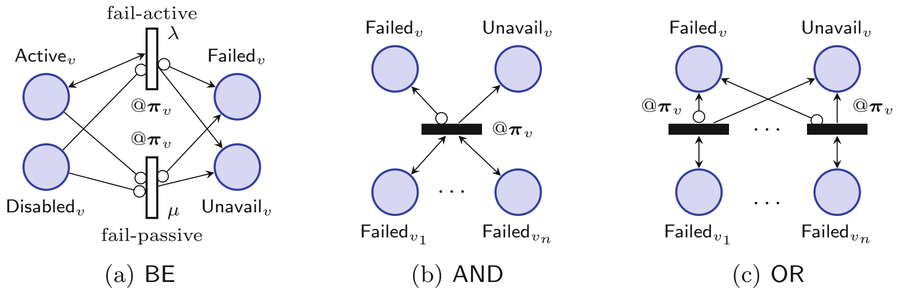


Fig. 5. GSPN templates for basic events and static gates

AND and OR. Figure 5(b) shows the template $\mathsf{templ}_{\mathsf{AND}}(v)$ for the AND gate v. A token is put in Failed_v as soon as the places Failed_{v_i} for all children v_i contain a token. Place Failed_v is thus marked if v has failed. Firing the (only) immediate transition puts tokens in Failed_v and $\mathsf{Unavail}_v$, and returns the tokens taken from Failed_{v_i}. Similar to the BE template, an inhibitor arc prevents the multiple execution of the failed-transition once v failed. The template for an OR gate is constructed analogously, see Fig. 5(c). The failure of one child suffices for v to fail; thus each child has a transition to propagate its failure to Failed_v.

PAND. We distinguish two versions [11] of the priority gate PAND: *inclusive* (denoted $\leq$) and *exclusive* (denoted $<$).

The *inclusive* $\mathsf{PAND}_{\leq}$ v fails if all its children failed in order from left to right while *including simultaneous failures* of children. Figure 6(a) depicts its template. If child v_i failed but its left sibling v_{i-1} is still operational, the $\mathsf{PAND}_{\leq}$ becomes fail-safe, as reflected by placing a token in FailSafe. The inhibitor arc of FailSafe now prevents the rightmost transition to fire, so no token can be put in Failed_v any more. If all children failed from left-to-right and $\mathsf{PAND}_{\leq}$ is not fail-safe, the rightmost transition can fire modelling the failure of the $\mathsf{PAND}_{\leq}$.

The *exclusive* $\mathsf{PAND}_{<}$ v is similar but *excludes the simultaneous failure* of children. Its template is shown in Fig. 6(b) and uses the auxiliary places $X_1, \ldots, X_{n-1}$ which indicate if the previous child failures agree with the strict failure order. A token is placed in X_i if a token is in X_{i-1} and the child v_i has

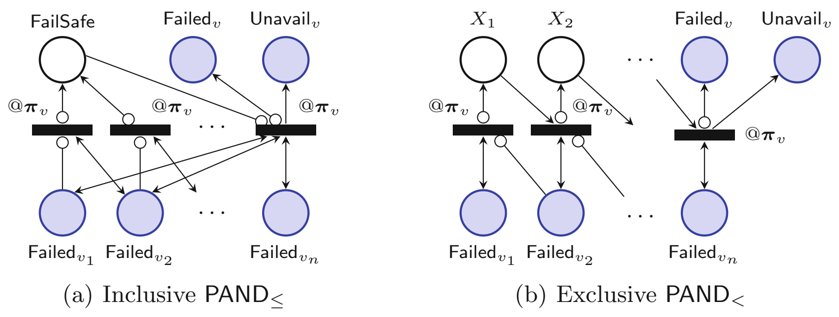


Fig. 6. GSPN templates for inclusive and exclusive PAND

just failed but its right sibling v_{i+1} is still operational. A token can only be put in Failed$_v$ if the rightmost child fails and X_{n-1} contains a token. If the child v_i violates the order, the inhibitor arc from its corresponding transition prevents to put a token in X_{i-1}. This models that PAND$_<$ becomes fail-safe.

The behaviour of both PAND variants crucially depends on whether children fail simultaneously or strictly ordered. The moment children fail depends on the order in which failures propagate, and is discussed in detail in Sect. 4.1.

SPARE. We depict the template templ$_{\mathsf{SPARE}}(v)$ for SPARE in two parts: Claiming[2] is depicted in Fig. 7, activation is shown in Fig. 8.

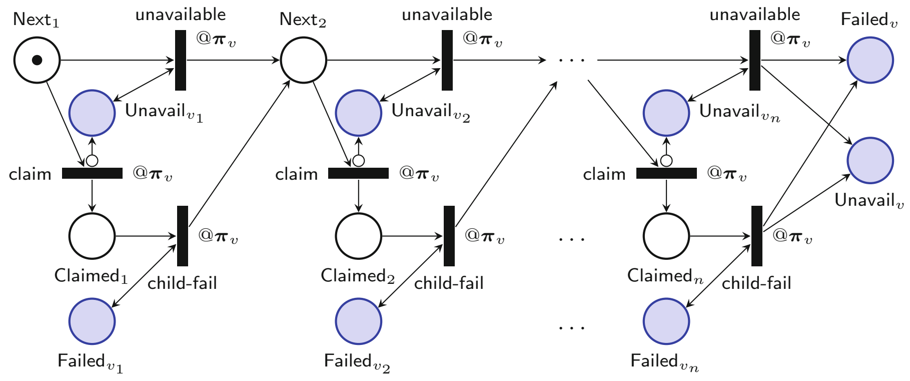


Fig. 7. GSPN template for SPARE, the claiming mechanism

Claiming. templ$_{\mathsf{SPARE}}(v)$ has two sorts of auxiliary places for each child i: Next$_i$ and Claimed$_i$. A token in Next$_i$ indicates that the spare component v_i is the next in line to be considered for claiming. Initially, only Next$_1$ is marked as the

[2] We consider *early* claiming; the concept of *late* claiming is described in [33].

primary child is to be claimed first. A token in $\mathsf{Claimed}_i$ indicates that SPARE v has currently claimed the spare component v_i. This token moves (possibly via $\mathsf{Claimed}_i$) through places Next_i and ends in Failed_v if all children are unavailable or already claimed. The claiming mechanism considers the Unavail places of the children. If $\mathsf{Unavail}_i$ is marked, the i-th spare component cannot be claimed as either the i-th child has failed or it has been claimed by another SPARE. In this case, the transition unavailable fires and the token is moved to Next_{i+1}. Then, spare component $i+1$ has to be considered next.

An empty place $\mathsf{Unavail}_i$ indicates that the i-th spare component is available. The SPARE can claim it by firing the claim transition. This results in tokens in $\mathsf{Claimed}_i$ and $\mathsf{Unavail}_i$, marking the spare component unavailable for other SPAREs. If a spare component is claimed (token in $\mathsf{Claimed}_i$) and it fails, the transition child-fail fires, and the next child is considered for claiming.

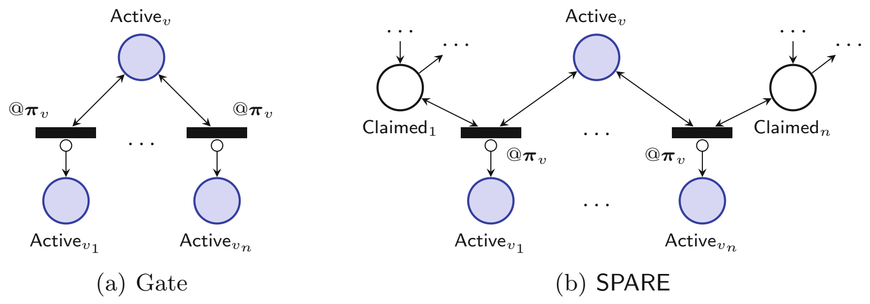


Fig. 8. GSPN template extensions for the activation mechanism of DFT elements

Activation. When an active SPARE claims a spare component c, all nodes in the spare module (the subtree) M_c become active, i.e., BEs in M_c now fail with their active (rather than passive) failure rate, and SPAREs in M_c propagate the activation downwards. The GSPN extensions for the activation mechanism are given in Fig. 8. The activation in SPAREs is depicted in Fig. 8(b). If a token is in $\mathsf{Claimed}_i$ indicating that the SPARE claimed the ith-child, and the SPARE itself is active, the transition can fire and places a token in Active_{v_i} indicating that the ith-child has become active. Other gates simply propagate the activation to their children as depicted in Fig. 8(a).

FDEP. Figure 9 depicts the template $\mathsf{templ}_{\,\mathsf{FDEP}}(v)$ for FDEP v; the generalized PDEP is discussed in [33]. If the first child of the FDEP fails, the dependent children fail too. Thus, if Failed_{v_1} is marked, then all transitions can fire and place tokens in the Failed places of the children indicating the failure propagation to dependent nodes. There is no arc to Failed_v as the FDEP itself cannot fail.

FDEPs introduce several semantic problems for DFTs, cf. [4]. This leads to different semantic interpretations which can be captured in our GSPN translation by different values for the priority variables $\boldsymbol{\pi}_v$; as elaborated in Sect. 4.

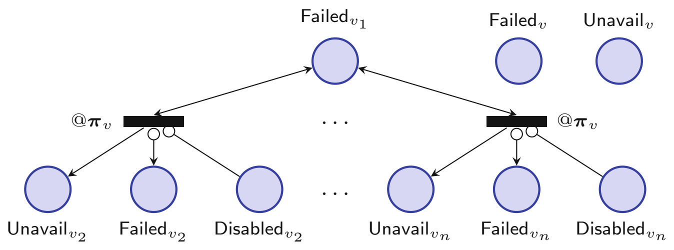


Fig. 9. GSPN template for FDEP

3.3 Gluing Templates

It remains to describe how the GSPN templates for the DFT elements are combined. We define the merging of templates. A more general setting is provided via graph-rewriting, cf. [7].

Definition 6 (Merging Templates). *Let $\mathcal{T}_i = (\mathcal{N}_i, W_i, \mathbb{N}[\boldsymbol{\pi}], \Pi_i, \mathcal{D}_i)$ for $i = 1, 2$ be $\boldsymbol{\pi}$-parameterised templates over $P_1 \cap P_2 = \mathcal{I}$. The* merge *of $\mathcal{T}_1$ and $\mathcal{T}_2$ is the $\boldsymbol{\pi}$-parameterised template over $\mathcal{I}$, $\mathsf{merge}(\mathcal{T}_1, \mathcal{T}_2) = (\mathcal{N}, W, \mathbb{N}[\boldsymbol{\pi}], \Pi, \mathcal{D})$ with*

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

An n-ary merge of templates over $\mathcal{I}_\mathcal{F}$ is obtained by concatenation of the binary merge. As the (disjoint) union on sets is associative and commutative, so is the merging of templates. Let $\mathsf{merge}(\mathbb{T} \cup \mathcal{T})$, where $\mathbb{T}$ is a finite non-empty set of templates over some $\mathcal{I}$ and $\mathcal{T}$ is a template over $\mathcal{I}$, denote $\mathsf{merge}(\mathcal{T}, \mathsf{merge}(\mathbb{T}))$.

The GSPN translation converts each DFT node v into the corresponding GSPN using its type-dependent template $\mathsf{templ}_{Tp(v)}$.

Definition 7 (Template for a DFT). *Let DFT $\mathcal{F} = (V, \sigma, Tp, top)$ and $\{\mathsf{templ}_{Tp(v)}(v) \mid v \in \mathcal{F}\}$ be the set of templates over $\mathcal{I}_\mathcal{F}$ each with priority-variable $\boldsymbol{\pi}_v$. The* GSPN template *$\mathcal{T}_\mathcal{F}$ for DFT $\mathcal{F}$ with places $P \supset \mathcal{I}_\mathcal{F}$ is defined by $\mathcal{T}_\mathcal{F} = \mathsf{merge}\left(\{\mathsf{templ}_{Tp(v)}(v) \mid v \in \mathcal{F}\} \cup \{\mathsf{templ}_{\mathsf{init}}\}\right)$.*

Initialisation Template. The initialisation template $\mathsf{templ}_{\mathsf{init}}$, see Fig. 10, is ensured to fire once and first, and allows to change the initial marking, e.g., already initially failed DFT nodes. This construct allows to fit the initial marking to the requested semantics without modifying the overall translation. The leftmost transition fires initially, and places a token in Active_{top}. The transition models starting the top-down activation propagation from the top-level node. Furthermore, a token is placed in the place $\mathsf{Evidence}$, enabling the setting of *evidence*, i.e., already failed DFT nodes. If $\{e_1, \ldots, e_n\} \subseteq \mathcal{F}_{\mathsf{BE}}$ is the set of already failed BEs, firing the rightmost transition puts a token in each Failed_{e_i} for all already failed BE e_i.

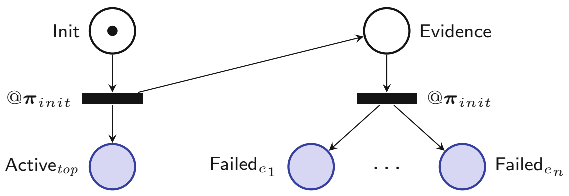


Fig. 10. GSPN template for initialisation

3.4 Properties

We discuss some properties of the obtained GSPN $\mathcal{T}_\mathcal{F}$ for a DFT $\mathcal{F}$. Details can be found in [33].

The size of $\mathcal{T}_\mathcal{F}$ is linear in the size of $\mathcal{F}$. Let $\sigma_{\mathsf{max}} = \max_{v \in \mathcal{F}} |\sigma(v)|$ be the maximal number of children in $\mathcal{F}$. The GSPN $\mathcal{T}_\mathcal{F}$ has no more than $6 \cdot |V| \cdot \sigma_{\mathsf{max}} + 2$ places and immediate transitions, and $2 \cdot |\mathcal{F}_{\mathsf{BE}}|$ timed transitions.

Transitions in $\mathcal{T}_\mathcal{F}$ fire at most once. Therefore, $\mathcal{T}_\mathcal{F}$ does not contain time-traps. Tokens in the interface places Failed_v, Active_v and $\mathsf{Unavail}_v$ are never removed. For such a place p and any transition t, $O(p)(t) \leq I(p)(t)$. Typically, the inhibitor arcs of interface places prevent a re-firing of a transition. In $\mathsf{templ}_{\mathsf{PAND}_<}(v)$, $\mathsf{templ}_{\mathsf{SPARE}}(v)$ and $\mathsf{templ}_{\mathsf{init}}$ tokens move from left to right, and no transition is ever enabled after it has fired.

The GSPN $\mathcal{T}_\mathcal{F}$ is two-bounded, all places except $\mathsf{Unavail}_v$ are one-bounded. Typically, either the inhibitor arcs prevent adding tokens to places that contain a token, or a token moves throughout the (cycle-free) template. However, two tokens can be placed in $\mathsf{Unavail}_v$: One token is placed in $\mathsf{Unavail}_v$ if v is claimed by a SPARE. Another token is placed in $\mathsf{Unavail}_v$ if v failed. The GSPN templates can be easily extended to ensure 1-boundedness of $\mathsf{Unavail}_v$ as well, cf. [33].

3.5 Tool Support

We realised the GSPN translation of DFTs within the model checker Storm [19], version 1.2.1[3]. Storm can export the obtained GSPNs as, among others, GreatSPN Editor projects [23]. Table 3 gives some indications of the obtained sizes of

Table 3. Experimental evaluation of GSPN translations

Benchmark	DFT				GSPN		
	#BE	#Dyn	#Nodes	σ_{max}	#Places	#Timed Trans	#Immed. Trans
HECS 5_5_2_np	61	10	107	16	273	122	181
MCS 3_3_3_dp_x	46	21	80	7	246	92	163
RC 15_15_hc	69	33	103	34	376	138	240

[3] http://www.stormchecker.org/publications/gspn-semantics-for-dfts.html.

the GSPNs for some DFT benchmarks from [13]. All GSPN translations could be computed within a second. As observed before, the GSPN size is linear in the size of the DFT.

4 A Unifying DFT Semantics

The interpretation of DFTs is subject to various subtleties, as surveyed in [4]. Varying interpretations have given rise to various DFT semantics in the literature. The key aspects are summarised in Table 1. In the following, we focus on three key aspects—failure propagation, FDEP forwarding, and non-determinism—and show that these suffice to differentiate *all* five DFT semantics, see Fig. 11. Note that we consider the interleaving semantics of nets.

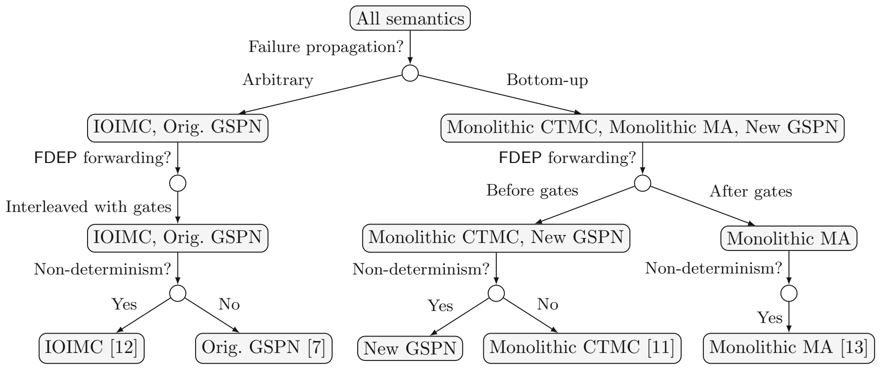


Fig. 11. Decision tree to compare five different DFT semantics

We expose the subtle semantic differences by considering the three aspects using the translated GSPNs of some simple DFTs. The simple DFTs contain structures which occur in industrial case-studies [4]. We vary two ingredients in our net semantics: *instantiations of the priority variables* $\boldsymbol{\pi}$, and the *partitioning* $\mathcal{D}$ *of immediate transitions*. The former constrain the ordering of transitions, while the latter control the treatment of non-determinism. This highlights a key advantage of our net translation: all different DFT semantics from the literature can be captured by small changes in the GSPN. In particular, the net structure itself stays the same for all semantics. Each of the following subsections is devoted to one of the aspects: failure propagation, FDEP forwarding, and non-determinism. Afterwards, we summarise the differences in Table 4.

4.1 Failure Propagation

This aspect is concerned with the order in which failures propagate through the DFT. Consider (a) the DFT $\mathcal{F}_1$ and (b) its GSPN $\mathcal{T}_{\mathcal{F}_1}$ in Fig. 12 and suppose B has failed, as indicated in red and the token in place Failed_B (the same example

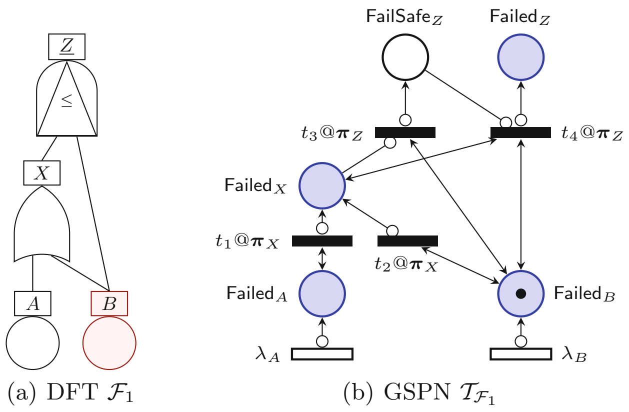


Fig. 12. Example for failure propagation

was used in the introduction). The question is how B's failure propagates through the DFT. Considering a total ordering on failure propagations, there are two scenarios. *Is B's failure first propagated to gate X, causing* PAND *Z to fail, or is B's failure first propagated to gate Z, turning Z fail-safe?*

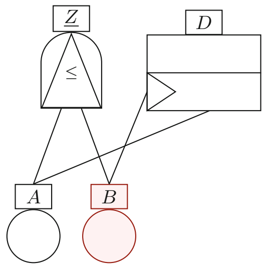
The question reflects in net $\mathcal{T}_{\mathcal{F}_1}$: Consider the enabled transitions t_2 and t_3. Firing t_2 places a token in Failed$_X$ (and in Failed$_B$) and models that B's failure first propagates to X. Next, firing t_4 places a token in Failed$_Z$ and models that the failures of B and X propagate to Z. Now consider first propagating B's failure to Z. This corresponds to firing t_3 and a token in FailSafe$_Z$ modelling that Z is fail-safe. (B's failure can still be propagated to X, but Z remains fail-safe as transition t_4 is disabled due to the token in FailSafe$_Z$.)

The order of failure propagation is thus crucial as it may cause a gate to either fail or to be fail-safe. Existing ways to treat failure propagation are: (1) allow for all possible orders, or (2) propagate failures in a bottom-up manner through the DFT. The former is adopted in the IOIMC and the original GSPN semantics. This amounts in $\mathcal{T}_{\mathcal{F}_1}$ to give all transitions the same priority, e.g., $\boldsymbol{\pi}_v = 1$ for all $v \in \mathcal{F}$. Case (2) forces failures to propagate in a bottom-up manner, i.e., a gate is not evaluated before all its children have been evaluated. This principle is used by the other three semantics. To model this, the priority of a gate v must be lower than the priorities of its children, i.e., $\boldsymbol{\pi}_v < \boldsymbol{\pi}_{v_i}, \forall i \in \{1, \dots, |\sigma(v)|\}$. In $\mathcal{T}_{\mathcal{F}_1}$, this yields $\boldsymbol{\pi}_Z < \boldsymbol{\pi}_X$, forcing firing t_2 before t_3, see Table 4.

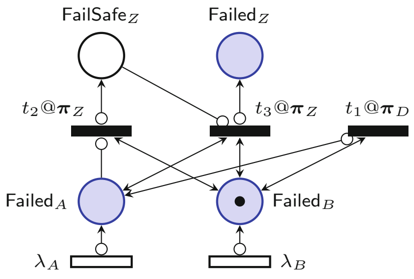
4.2 FDEP Forwarding

The second aspect concerns how FDEPs forward failures in the DFT. Consider (a) the DFT $\mathcal{F}_2$ and (b) its GSPN $\mathcal{T}_{\mathcal{F}_2}$ in Fig. 13. Suppose B fails. The crucial question is—similar to failure propagation—when to propagate B's failure via FDEP D to A. *Is B's failure first propagated via D, causing A and Z to fail, or does B's failure first cause Z to become fail-safe before A fails?* The first scenario is possible as Z is inclusive and A and B are interpreted to fail simultaneously.

In $\mathcal{T}_{\mathcal{F}_2}$, the scenarios are reflected by letting either of the enabled transitions t_1 and t_2 fire first. A similar scenario can be constructed with a $\mathsf{PAND}_<$ and an FDEP from A to B.



(a) DFT $\mathcal{F}_2$ [4]



(b) GSPN $\mathcal{T}_{\mathcal{F}_2}$

Fig. 13. Example for FDEP forwarding

The order of evaluating FDEPs is thus crucial (as above). We distinguish three options: evaluating FDEPs (1) before, (2) after, or (3) interleaved with failure propagation in gates. The first two options evaluate FDEPs either before or after all other gates, respectively. In $\mathcal{T}_{\mathcal{F}_2}$, these options require that all transitions of an FDEP template get the (1) highest (or (2) lowest, respectively) priority, i.e.,

$$\forall f \in \mathcal{F}_{\mathsf{FDEP}} : \boldsymbol{\pi}_f > \boldsymbol{\pi}_v, \forall v \in \mathcal{F} \setminus \mathcal{F}_{\mathsf{FDEP}} \quad \text{(or, } \boldsymbol{\pi}_f < \boldsymbol{\pi}_v \text{ respectively).}$$

The monolithic CTMC and the new GSPN semantics[4] evaluate FDEPs before gates, whereas the monolithic MA semantics evaluate them after gates. In option (3), FDEPs are evaluated interleaved with the other gates. This option is used by the IOIMC and the original GSPN semantics. In $\mathcal{T}_{\mathcal{F}_2}$, interleaving corresponds to giving all transitions the same priority, e.g. $\boldsymbol{\pi}_v = 1, \forall v \in \mathcal{F}$, see Table 4.

4.3 Non-determinism

The third aspect is how to resolve non-determinism in DFTs. Consider DFT $\mathcal{F}_3$ in Fig. 14 where BE X has failed and FDEP D forwards the failure to BEs A and B. This renders A and B unavailable for SPAREs $S1$ and $S2$. *The question is which one of the failed SPAREs ($S1$ or $S2$) claims the spare component C?* This phenomenon is known as a *spare race*. How the spare race is resolved is important: the outcome determines whether PAND Z fails or becomes fail-safe.

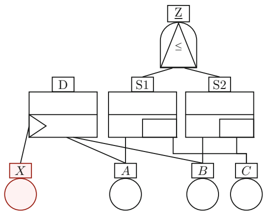


Fig. 14. Example for non-determinism (DFT $\mathcal{F}_3$)

[4] The new GSPN semantics needs further adaptions for downward FDEPs, cf. Sect. 4.4.

The spare race is represented in $\mathcal{T}_{\mathcal{F}_3}$ (depicted in [33]) by a conflict between the claiming transitions of the nets of $S1$ and $S2$. Depending on the previous semantic choices, the race is resolved in different ways. For the monolithic MA semantics, the race is resolved by the order of the FDEP forwarding. For the new GSPN semantics, the race is resolved by the order in which the claim-transitions originating from $\mathsf{templ}_{\mathsf{SPARE}}(S_1)$ and $\mathsf{templ}_{\mathsf{SPARE}}(S_2)$ are handled. In the IOIMC semantics, the winner of the race is determined by the order of interleaving.

For any semantics, the race is represented by a conflict between immediate transitions (with the same priority). We resolve a conflict either by (1) *randomisation*, or (2) *non-determinism*. We realise the randomisation by using weights, i.e., by equipping every immediate transition with the same weight like $W(t) = 1, \forall t \in T$ and letting $\mathcal{D} = T_i$ contain all immediate transitions. A conflict between enabled transitions is then resolved by means of a uniform distribution: each enabled transition is equally probable. This approach reflects the monolithic CTMC and the original GSPN semantics for DFTs.

Case (2) takes non-determinism *as is* and reflects the other three DFT semantics. In this case, in $\mathcal{T}_{\mathcal{F}_3}$ each immediate transition is a separate partition: $\mathcal{D} = \{\{t\} \mid t \in T_i\}$. In many DFTs, the non-determinism is *spurious* and its resolution does not affect standard measures such as reliability and availability. The example $\mathcal{F}_3$ however yields significantly different analysis results depending on how non-determinism is resolved.

Table 4. GSPN differences between supported semantics

DFT semantics	GSPN priority variables		GSPN partitioning
Monolithic CTMC	$\boldsymbol{\pi}_v < \boldsymbol{\pi}_{v_i}$	$\forall v \in \mathcal{F}$, $\forall i \in \{1, \ldots, \lvert\sigma(v)\rvert\}$	$\{T_i\}$
	$\boldsymbol{\pi}_f > \boldsymbol{\pi}_v$	$\forall f \in \mathcal{F}_{\mathsf{FDEP}}, \forall v \notin \mathcal{F}_{\mathsf{FDEP}}$	
IOIMC	$\boldsymbol{\pi}_v = \boldsymbol{\pi}_{v'}$	$\forall v, v' \in \mathcal{F}$	$\{\{t\} \mid t \in T_i\}$
Monolithic MA	$\boldsymbol{\pi}_v < \boldsymbol{\pi}_{v_i}$	$\forall v \in \mathcal{F}$, $\forall i \in \{1, \ldots, \lvert\sigma(v)\rvert\}$	$\{\{t\} \mid t \in T_i\}$
	$\boldsymbol{\pi}_f < \boldsymbol{\pi}_v$	$\forall f \in \mathcal{F}_{\mathsf{FDEP}}, \forall v \notin \mathcal{F}_{\mathsf{FDEP}}$	
Original GSPN	$\boldsymbol{\pi}_v = \boldsymbol{\pi}_{v'}$	$\forall v, v' \in \mathcal{F}$	$\{T_i\}$
New GSPN	$\boldsymbol{\pi}_v \leq \boldsymbol{\pi}_{v_i}$	$\forall v \in \mathcal{F}_{\mathsf{AND}} \cup \mathcal{F}_{\mathsf{OR}}$, $\forall i \in \{1, \ldots, \lvert\sigma(v)\rvert\}$	$\{\{t\} \mid t \in T_i\}$
	$\boldsymbol{\pi}_v < \boldsymbol{\pi}_{v_i}$	$\forall v \notin \mathcal{F}_{\mathsf{AND}} \cup \mathcal{F}_{\mathsf{OR}}$, $\forall i \in \{1, \ldots, \lvert\sigma(v)\rvert\}$	
	$\boldsymbol{\pi}_f \geq \boldsymbol{\pi}_{f_i}$	$\forall f \in \mathcal{F}_{\mathsf{FDEP}}$, $\forall i \in \{2, \ldots, \lvert\sigma(v)\rvert\}$	
	$\boldsymbol{\pi}_f \leq \boldsymbol{\pi}_{f_1}$	$\forall f \in \mathcal{F}_{\mathsf{FDEP}}$	

Remark 1. The semantics of GSPNs [5,6] assigns a weight to every immediate transition. These weights induce a probabilistic choice between conflicting immediate transitions. If several immediate transitions are enabled, the probability of selecting one is determined by its weight relative to the sum of the weights of all enabled transitions, see Sect. 2.1. Under this interpretation, the stochastic process underlying a confusion-free GSPNs is a CTMC. In order to capture the possibility of non-deterministically resolving, e.g., spare races, we use a GSPN semantics [16] where immediate transitions are partitioned. Transitions resolved in a random manner (by using weights) are in a single partition, transitions resolved non-deterministically constitute their own partition—their weights are irrelevant. For confusion-free GSPNs, our interpretation corresponds to [5,6] and yields a CTMC. In general, however, the underlying process is an MA.

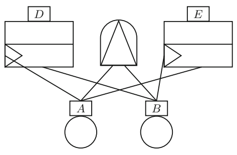
The GSPN adaptations for the different DFT semantics are summarised in Table 4. The last two rows of the table concern FDEPs that are triggered by gates (rather than BEs) and are discussed in detail below.

4.4 Allow FDEPs Triggered by Gates

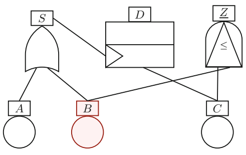
So far we assumed that FDEP triggers are BEs. We now lift this restriction simplifying the presentation and discuss the options when FDEPs can be triggered by a gate, see Fig. 15(b) and (c). The row "downward" FDEPs in Table 2 reflects this notion. The challenge is to treat *cyclic dependencies*. Cyclic dependencies already occur at the level of BEs, see Fig. 15(a). According to the monolithic CTMC and new GSPN semantics, FDEPs forward failures immediately: All BEs that fail are marked failed before any gate is evaluated, naturally matching bottom-up propagation. The effect is as-if the BEs A and B failed simultaneously. For the new GSPN semantics, we generalise this propagation, and support FDEPs triggered by gates. Consider $\mathcal{F}_5$ in Fig. 15(b): The failure of B indirectly (via S and D) forwards to C. If Z is evaluated after the failure is forwarded to C, the interpretation is that B and C failed simultaneously and the PAND fails, as intended. To guarantee that C is marked failed before Z is evaluated, *S and D require higher priorities than Z* in the net. Consequently, all children of Z are evaluated before Z is evaluated.

Concretely, we generalise bottom-up propagation by refining the priorities: First, we observe that only for dynamic gates, where the order in which children fail matters, the children need to be evaluated strictly before the parents. For other gates, we may weaken the constraints on the priorities. A non-strict ordering suffices: $\forall v \in \mathcal{F}_{\mathsf{AND}} \cup \mathcal{F}_{\mathsf{OR}} : \boldsymbol{\pi}_v \leq \boldsymbol{\pi}_{v_i}, \forall i \in \{1, \ldots, |\sigma(v)|\}$. Second, we mimic bottom-up propagation in FDEP forwarding, meaning that dependent events require a priority not larger than their triggers. Thus, we ensure for each FDEP f, $\boldsymbol{\pi}_f \leq \boldsymbol{\pi}_{f_1}$, and $\boldsymbol{\pi}_f \geq \boldsymbol{\pi}_{f_i}$ for all children $i \neq 1$. Equal priorities are admitted. For FDEPs, like for static gates, the status change is order-independent.

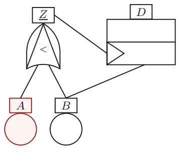
Some DFTs (with FDEPs triggered by gates and cyclic forwarding) do not admit a valid priority-assignment. We argue that the absence of a suitable priority assignment is natural; DFTs without valid priority assignment can model



(a) DFT $\mathcal{F}_4$



(b) DFT $\mathcal{F}_5$



(c) DFT $\mathcal{F}_6$

Fig. 15. Examples for downward FDEP forwarding

a paradox. The DFT $\mathcal{F}_6$ in Fig. 15(c) illustrates this. The new GSPN semantics induce the following constraints:

$$\boldsymbol{\pi}_A < \boldsymbol{\pi}_Z, \quad \boldsymbol{\pi}_B < \boldsymbol{\pi}_Z, \quad \boldsymbol{\pi}_Z \leq \boldsymbol{\pi}_D, \quad \text{and} \quad \boldsymbol{\pi}_D \leq \boldsymbol{\pi}_B.$$

The constraints imply $\boldsymbol{\pi}_B < \boldsymbol{\pi}_B$, which is unsatisfiable. BE A has failed and the exclusive POR Z fails too. (A detailed account of POR-gates is given in [33].) But then B fails because of FDEP D. If we now assume A and B to fail simultaneously, the exclusive POR cannot fail, as its left child A did not fail strictly before B. Then, D's trigger would have never failed. Thus, it is reasonable to exclude such DFTs and consider them ill-formed.

The IOIMC and the monolithic MA semantics support FDEPs triggered by gates, but have different interpretations of simultaneity. The monolithic CTMC semantics is in line with our interpretation, but the algorithm [34] claimed to match this semantics produces deviating results for the DFTs in this sub-section.

5 Conclusions and Future Work

This paper presents a unifying GSPN semantics for Dynamic Fault Trees (DFTs). The semantics is compositional, the GSPN for each gate is rather simple. The most appealing aspect of the semantics is that design choices for DFT interpretations are concisely captured by changing only transition priorities and the partitioning of transitions. Our semantics thus provides a framework for comparing DFT interpretations. Future work consists of extending the framework to DFTs with repairs [8,31] and to study unfoldings [35] of the underlying nets.

References

1. Trivedi, K.S., Bobbio, A.: Reliability and Availability Engineering: Modeling, Analysis, and Applications. Cambridge University Press, Cambridge (2017)
2. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15–16**, 29–62 (2015)
3. Dugan, J.B., Bavuso, S.J., Boyd, M.: Fault trees and sequence dependencies. In: Proceedings of RAMS, pp. 286–293. IEEE (1990)
4. Junges, S., Guck, D., Katoen, J.P., Stoelinga, M.: Uncovering dynamic fault trees. In: Proceedings of DSN, pp. 299–310 (2016)

5. Marsan, M.A., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM TOCS **2**(2), 93–122 (1984)
6. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley, Hoboken (1995)
7. Raiteri, D.C.: The conversion of dynamic fault trees to stochastic Petri nets, as a case of graph transformation. ENTCS **127**(2), 45–60 (2005)
8. Bobbio, A., Raiteri, D.C.: Parametric fault trees with dynamic gates and repair boxes. In: Proceedings of RAMS, pp. 459–465. IEEE (2004)
9. Bobbio, A., Franceschinis, G., Gaeta, R., Portinale, L.: Parametric fault tree for the dependability analysis of redundant systems and its high-level Petri net semantics. IEEE Trans. Softw. Eng. **29**(3), 270–287 (2003)
10. Kabir, S., Walker, M., Papadopoulos, Y.: Quantitative evaluation of Pandora temporal fault trees via Petri nets. IFAC-PapersOnLine **48**(21), 458–463 (2015)
11. Coppit, D., Sullivan, K.J., Dugan, J.B.: Formal semantics of models for computational engineering: a case study on dynamic fault trees. In: Proceedings of ISSRE, pp. 270–282 (2000)
12. Boudali, H., Crouzen, P., Stoelinga, M.: A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE TDSC **7**(2), 128–143 (2010)
13. Volk, M., Junges, S., Katoen, J.P.: Fast dynamic fault tree analysis by model checking techniques. IEEE Trans. Ind. Inform. **14**(1), 370–379 (2018)
14. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proceedings of LICS, pp. 342–351. IEEE Computer Society (2010)
15. Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality. LNCS, vol. 2428. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45804-2_3
16. Eisentraut, C., Hermanns, H., Katoen, J.-P., Zhang, L.: A semantics for every GSPN. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 90–109. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_6
17. Sullivan, K., Dugan, J.B., Coppit, D.: The Galileo fault tree analysis tool. In: Proceedings of FTCS, pp. 232–235 (1999)
18. Arnold, F., Belinfante, A., Van der Berg, F., Guck, D., Stoelinga, M.: DFTCalc: a tool for efficient fault tree analysis. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFECOMP 2013. LNCS, vol. 8153, pp. 293–301. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40793-2_27
19. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A Storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
20. Trivedi, K.S., Sahner, R.A.: SHARPE at the age of twenty two. SIGMETRICS Perform. Eval. Rev. **36**(4), 52–57 (2009)
21. Ciardo, G., Miner, A.S., Wan, M.: Advanced features in SMART: the stochastic model checking analyzer for reliability and timing. SIGMETRICS Perform. Eval. Rev. **36**(4), 58–63 (2009)
22. Baarir, S., Beccuti, M., Cerotti, D., Pierro, M.D., Donatelli, S., Franceschinis, G.: The GreatSPN tool: recent enhancements. SIGMETRICS Perform. Eval. Rev. **36**(4), 4–9 (2009)
23. Amparore, E.G.: A new GreatSPN GUI for GSPN editing and CSL^{TA} model checking. In: Norman, G., Sanders, W. (eds.) QEST 2014. LNCS, vol. 8657, pp. 170–173. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10696-0_13

24. Montani, S., Portinale, L., Bobbio, A., Raiteri, D.C.: Radyban: a tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks. Reliab. Eng. Syst. Saf. **93**(7), 922–932 (2008)
25. Boudali, H., Dugan, J.B.: A continuous-time Bayesian network reliability modeling, and analysis framework. IEEE Trans. Reliab. **55**(1), 86–97 (2006)
26. Bouissou, M., Bon, J.L.: A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. Reliab. Eng. Syst. Saf. **82**(2), 149–163 (2003)
27. Rauzy, A., Blériot-Fabre, C.: Towards a sound semantics for dynamic fault trees. Reliab. Eng. Syst. Saf. **142**, 184–191 (2015)
28. Merle, G., Roussel, J.M., Lesage, J.J.: Quantitative analysis of dynamic fault trees based on the structure function. Qual. Reliab. Eng. Int. **30**(1), 143–156 (2014)
29. Walker, M., Papadopoulos, Y.: Qualitative temporal analysis: towards a full implementation of the fault tree handbook. Control Eng. Pract. **17**(10), 1115–1125 (2009)
30. Chen, D., Mahmud, N., Walker, M., Feng, L., Lönn, H., Papadopoulos, Y.: Systems modeling with EAST-ADL for fault tree analysis through HiP-HOPS. IFAC Proc. Vol. **46**(22), 91–96 (2013)
31. Guck, D., Spel, J., Stoelinga, M.: DFTCalc: reliability centered maintenance via fault tree analysis (tool paper). In: Butler, M., Conchon, S., Zaïdi, F. (eds.) ICFEM 2015. LNCS, vol. 9407, pp. 304–311. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25423-4_19
32. Raiteri, D.C.: Integrating several formalisms in order to increase fault trees' modeling power. Reliab. Eng. Syst. Saf. **96**(5), 534–544 (2011)
33. Junges, S., Katoen, J.P., Stoelinga, M., Volk, M.: One net fits all: a unifying semantics of dynamic fault trees using GSPNs. CoRR abs/1803.05376 (2018)
34. Manian, R., Coppit, D.W., Sullivan, K.J., Dugan, J.B.: Bridging the gap between systems and dynamic fault tree models. In: Proceedings of RAMS, pp. 105–111 (1999)
35. Engelfriet, J.: Branching processes of Petri nets. Acta Inform. **28**(6), 575–591 (1991)