Uncovering Dynamic Fault Trees

Sebastian Junges*, Dennis Guck†, Joost-Pieter Katoen*†, Mariëlle Stoelinga†
*Software Modeling and Verification, RWTH Aachen University, Germany
†Formal Methods and Tools, University of Twente, The Netherlands

***Abstract*—Fault tree analysis is a widespread industry standard for assessing system reliability. Standard (static) fault trees model the failure behaviour of systems in dependence of their component failures. To overcome their limited expressive power, common dependability patterns, such as spare management, functional dependencies, and sequencing are considered. A plethora of such dynamic fault trees (DFTs) have been defined in the literature. They differ in e.g., the types of gates (elements), their meaning, expressive power, the way in which failures propagate, how elements are claimed and activated, and how spare races are resolved. This paper systematically uncovers these differences and categorises existing DFT variants. As these differences may have huge impact on the reliability assessment, awareness of these impacts is important when using DFT modelling and analysis.**

I. Introduction

Current innovation in IT systems, like robots, the Internet-of-Things, autonomous cars, and 3D printing rapidly increase our dependence on computer-based systems. Reliability engineering is an important field that provides methods, tools and techniques to evaluate and mitigate the risks related to complex systems. Fault tree analysis (FTA) is one of the most important techniques in that field, and is commonly deployed in industry ranging from railway and aerospace system engineering to nuclear power plants. A fault tree (FT) describes how failures propagate through the system, and how component failures lead to system failures. An FT is a tree (or rather, a directed acyclic graph) whose leaves model component failures, and whose gates model failure propagation. Standard (or: static) FTs (SFTs) contain a few basic gates, like AND and OR, making them easy to use and analyse, but also limited in expressivity. To cater for more complex dependability patterns, like spare management and causal dependencies, a number of extensions to FTs have been proposed.

Dynamic Fault Trees (DFTs) [1], [2] are the most common extension and are widely used. Since the original proposal by Dugan *et al.* [1], a plethora of variants of DFTs have been proposed in the literature. An overview of extensions has recently been given in [3]. Table I summarises an overview of the various features of existing DFT dialects[1]. They differ in e.g., the types of supported gates (elements) and the way in which failures propagate. In addition to having *different expressive power*, the various DFT dialects provide *different interpretations* to syntactically identical DFT models. As the differences between these interpretations are rather subtle, they are mostly not recognised. This is highly undesirable, since it can easily lead to *different analysis results* and *wrong conclusions* about the system's reliability—if an FTA tool interprets a model differently from its user, then one may conclude that the system meets its dependability requirements, whereas in reality it does not.

[1]The first row indicates the different formalisms used for providing a DFT semantics, e.g., FTA = Fault Tree Automaton and DBN = Dynamic Bayesian Network. A detailed account is given in Sect. IV.

TABLE I. Comparing different semantics for DFTs.

	FTA [4]	CTBN [5]	DBN [6], [7]	SWN [8]	GSPN [9]	IMC [10], [11]	AE [12], [13]
Spare modules	BEs	BEs	BEs	BEs	BEs	subtrees	BEs
Dep. events	BE	BE	BE	BE	BE	BE+gates	BE
PAND	$\leq$	$\leq$	$\leq$	$<$	$<$	$<$	$\leq$
POR	no	no	no	no	no	$<$	$\leq$
SEQ	yes	no	cold spares	yes	no	cold spares	no
PDEP	no	no	yes	no	no	no	no
Replication	event	no	no	subtrees	no	no	no
Evidence	no	no	yes	no	no	no	yes[a]

Legend:
Spare modules = the type of spare modules supported of a SPARE gate.
Dep. events = the type of the events that an FDEP gate can trigger.
PAND = whether the interpretation of priority-AND is inclusive ($\leq$) or exclusive ($<$).
POR = whether priority-ORs are supported.
SEQ = whether sequence enforcers are supported or can be modelled via a cold spare.
PDEP = whether PDEPs are supported.
Replication = what subtrees are allowed to be replicated.
[a] Modelled directly within the framework.

Awareness of these semantic issues is thus important to achieve faithful and reliable DFT modelling and analysis. This paper systematically uncovers the semantic differences and subtleties of DFT variants. We extensively discuss and illustrate the possible pitfalls, and categorise existing DFT variants in the literature so as to enable their systematic comparison. Our study focusses on the following issues:

1) *Can we apply minimal cut sets to DFTs?* Minimal cut sets (MCSs) are sets of basic events whose failure cause the failure of the DFT. They are at the core of many analysis techniques for SFTs. We will show that they are insufficient to the analysis of DFTs.
2) *Is there a notion of causal or temporal ordering in failure propagation?* In addition to failure combination as present in SFTs (e.g., an AND-gate fails if all its children do fail), DFTs allow for failure forwarding in which failures may trigger other failures. This complicates the understanding of DFTs, and requires causality. The same applies for modelling the behaviour of spare elements.
3) *Expressive power.* In particular, we investigate whether sequence enforcers—that enforce the order of failed elements—add expressivity. Many DFT variants incorporate priority gates whose failure depends on the ordering of their children to fail. We study the subtle interplay between simultaneous failures and priority-gates. Finally, we study whether constantly failed elements can be expressed by existing gates.
4) *How should spare races be resolved?* DFTs incorporate spare gates that model the usage of spare components

representing redundant parts. As spare components may be shared by various spare gates, so-called spare races may occur in which simultaneously failed gates "claim" the same redundant part. We study this phenomenon in more depth, and provide solutions.

5) *What is the difference between activation and claiming?* All modules of a SPARE can be in active (i.e., in use) or dormant mode (not in use, being redundant, and having a lower failure rate). It is the responsibility of the SPARE gate to switch modules from dormant to active. This activation of redundant parts is related to "claiming". We give an in-depth study of the interplay between these mechanisms.

As pointed out in [4], [14], some subtleties in DFTs are due to undefined semantics; e.g. in [15]. We do not repeat issues that are due to missing semantics. Instead, we discuss the possible choices in the semantics and the issues that arise if these choices are not carefully accounted for. The presented intricacies are not independent of choices made in the semantics. However, great care was taken to present a broad range of possible choices for semantics. All intricacies originate from possibly undesired behaviour of, or claims made about, existing semantics. In this work, we omit the concept of repairs, which give rise to several other open questions [16].

Organisation of the paper. Sect. II introduces DFTs and their common interpretation. Sect. III is the main body of the paper and extensively discusses the aforementioned five issues with DFTs. Sect. IV discusses the existing DFT semantics. Sect. V presents a case study collection and indicates which intricacies occur in them.

II. DFTs: THE GENERAL RECIPE

A. Static Fault Trees

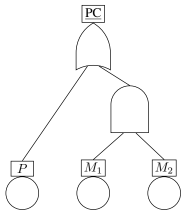


Fig. 1. Fault Tree

FTs [2] are trees that model how component failures propagate to system failures. Since subtrees can be shared, FTs are directed acyclic graphs (DAGs) rather than trees. The leaves of the tree (or rather sinks of the DAG) model component failures. They are called *basic events (BEs)* and are usually equipped with a *failure rate*, i.e., the parameter of an exponential probability distribution indicating the probability for the component to fail within a deadline T. Other probability distributions, like Weibull, are often supported as well.

Various names and symbols exist for sinks that either have already failed or never fail. We call them *constants* and denote them by CONST($\top$) and CONST($\perp$) respectively, see Figure 2(b)-(c). All non-sink nodes of the FT, i.e., nodes with one or more children (a.k.a. inputs of the node), are equipped with gates. Static fault trees feature three types of gates, depicted in Figure 2(d)-(f): AND, OR, and VOT(k). These fail if respectively all, one, or k of their children fail. The latter is also known as the k out of n, where n refers to the number of children. Clearly, the VOT(1) gate is equivalent to an OR-gate and a VOT(k) gate with k children is equivalent to an AND-gate. Sometimes other gates are considered, like the XOR (exclusive OR) and the NOT gate [2], [17]. Adding such gates makes fault trees *non-coherent*, i.e. an additional event causes the TLE to switch from failed to operational again. We exclude them here, but briefly discuss them in Sect. III-C. The root of the tree is called the *top level event* (TLE, denoted by an underlined identifier). It represents the failure condition of interest, such as the stranding of a train, or the unplanned unavailability of a satellite. The DFT fails if the TLE fails. A sample SFT is depicted in Fig. 1. The fault tree encodes that a computer either fails if both memory units M_1, M_2 fail or if the processor P fails.

As the failure of the TLE is determined by the (unordered) set of failed BEs, these FTs are *static*. SFTs are appealing as they are a relatively simple, useful modelling tool. SFTs however cannot model several important failure patterns in safety-critical systems, such as:

- *Order-dependent failures*, i.e. failures that only occur if BEs occur in a particular order. For example, a water leakage in a pump only leads to a short-circuit if the power supply has not failed before.
- Simple support for *feedback loops*. Although systems with feedback loops in the error behaviour and cascaded errors can be modelled with SFTs, doing so requires verbose work-arounds which makes modelling error prone, cf. [2].
- *Sequencing* expressing that an event can only happen after another one, e.g. a short circuit in a pump can only occur after a leakage.
- *Spare management and spare parts.* A spare part is an interchangeable part that is used for the replacement of failed elements, e.g. spare car wheels. *Cold* spare parts can only fail while used. *Warm* spare parts can always fail but when being spare they fail with a reduced failure rate. *Hot* spare parts always have the same failure rate.
- *Mutual exclusion*, as e.g. a valve can either fail by being stuck open or stuck closed, but not both.

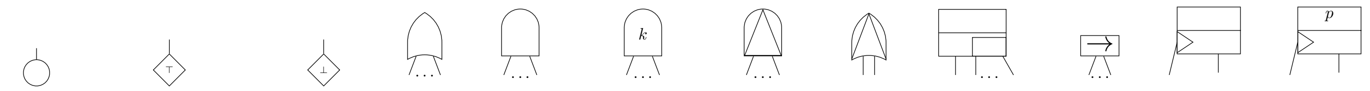
B. Dynamic Fault Trees

We discuss six additional element types (see Fig. 2(g)-(l)). The priority-AND gate, SPARE, functional dependency and SEQ are commonly included. Priority-ANDs are also discussed as single extension to static fault trees in, among others, [18], [19] or in combination with other gates, e.g. in [20]. As DFTs support passive standby, e.g. in spare wheels, each BE has now two failure rates: *active* and *passive*. Also in the presence of these gates, DFTs are coherent.

a) A *priority-and gate* (pand-gate, PAND) is an AND-gate which fails only if its children fail from left to right. The literature does not agree what happens in case two children fail simultaneously, see Sect. III-C.

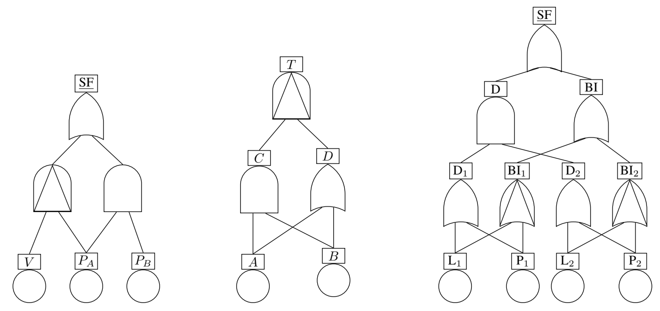
Example 1: We model two pumps and a valve, see the DFT in Fig. 3a. Consider a pump (P_A) with a backup pump (P_B). Pump P_B can only replace P_A after valve V has opened the pipe to P_B. Starting from the top, two conditions lead to the system failure SF. Either (left subtree) V fails before P_A fails. Upon failure of P_A, it is then impossible to switch to P_B. Or (right subtree) both P_A and P_B fail (in arbitrary order).

Extending SFTs with PAND gates makes the CONST($\perp$) element syntactic sugar. In Fig. 3b, using that two BEs do not fail simultaneously, either A or B fail before both A and B have failed, i.e. D always fails before C, thus T never fails.



(a) BE (b) CONST(⊤) (c) CONST(⊥) (d) OR (e) AND (f) VOT(k) (g) PAND (h) POR (i) SPARE (j) SEQ (k) FDEP (l) PDEP

Fig. 2. Element types in Static ((a)-(f)) and Dynamic Fault Trees (all)



(a) PAND example (b) Model CONST(⊥) (c) POR example

Fig. 3. Priority-gate examples

b) The priority-OR gate (POR) is considered in temporal fault trees[2] by Walker *et al.* [21], [20], [22]. A POR fails if the first child fails before any other child does. The binary POR is dual to the binary PAND, as a POR with children A and B fails iff a PAND with children B and A is infallible.

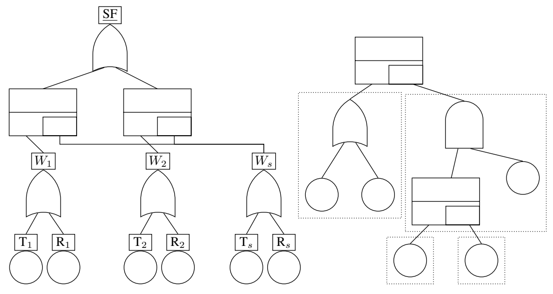
Example 2: Consider two devices connected via a data link. The system is operational as long as one device is operational and no device blocks the data link, e.g. by turning into a "babbling idiot"[3]. A processor failure causes a device failure. A data link failure turns a device with an operational processor into a babbling idiot. A babbling idiot leads to a system failure. The DFT is given in Fig. 3c. The TLE fails if either both devices (D) fail or a babbling idiot (BI) occurs. D if both (D_1) or (D_2) fails. Device i fails if either a processor (P_i) or the data link (L_i) fails. It starts babbling (BI_i) if (L_i) fails and (P_i) has not occurred before.

c) Spare gates (SPARE) manage model usage of—potentially shared—spare components. Children of the SPARE gate represent *modules*; which model groups of redundant parts. The first child of the SPARE gate is the *primary*; the other inputs are *spares*. Initially, the spare uses the primary. Whenever the currently used child fails, the SPARE gate attempts to switch to the next (left to right) available spare module, i.e. a child which has not yet failed and is not used by another SPARE. If no such child exists, the SPARE fails. Thus, modules can be in active (i.e., in use) or dormant mode; BEs in dormant mode fail with the passive failure rate. It is the responsibility of the SPARE module to switch modules from dormant to active, i.e. all BEs in the spare module change their failure rates from the passive to the active failure rate.

Example 3: Consider a motor bike (or car) with a spare wheel. Each wheel either breaks due to a broken rim or due to a flat tire. The car fails if one of its wheels fail and cannot be replaced, see Fig. 4a. As soon as the first tire or rim fails, the corresponding wheel W_i ($1 \leq i \leq 2$) fails. The SPARE with primary W_i then claims the spare wheel W_s, thereby activating W_s—and thus its tire and its rim. For a subsequent failure of any other wheel W_j, $j \neq i$, its corresponding SPARE cannot claim the spare wheel W_s anymore. Thus, the SPARE (and the system) fail. Note that assuming a passive standby for W_s adds a—less likely—scenario where W_s fails before any of the primary wheels. In that case, nothing happens after the failure of W_s, but any failure on the primary wheels immediately causes the system to fail.

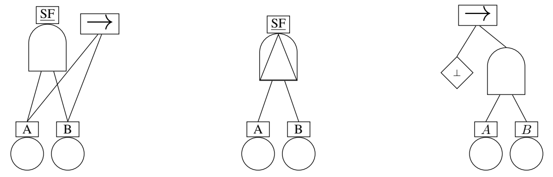
[2]Temporal fault trees ≈ static fault trees with priority gates.

[3]Babbling idiots are devices that constantly send messages over a data link, blocking communication of other devices.



(a) Motor bike with a spare wheel (b) Order-enforcing

Fig. 4. SPARE gate examples



(a) Order-enforcing (b) Order-dependence (c) Modelling Mutex

Fig. 5. Sequence enforcers

Originally, spare modules were limited to BEs, but these restrictions have been relaxed in e.g. [11]. In the example above, everything connected to a child of a SPARE is part of the spare module. In Fig. 4b, we depict the DFT, with four spare modules, which we indicate by the dotted boxes. Indeed, a child of a SPARE represents a spare module. Children of a gate v are in the same spare module as v, unless v is a SPARE. We discuss the extent of spare modules in Sect. III-E.

d) Sequence enforcers (SEQ) guarantee that their children only fail from left to right. In [15], the SEQ is presented as an alternative to the PAND, which might be misleading. Whereas the PAND is a special AND-gate which only fails if the order-restriction is met, the SEQ prevents certain orders from occurring. Unlike common gates, SEQs do not propagate failures, therefore SEQs have no parents.

Example 4: Consider a water pump which fails if a leakage (A) occurs and the motor has a short circuit (B), see Fig. 5a. This short circuit can only occur after the leakage. Fig. 5b shows a DFT that fails after the BEs have failed in the order AB. Contrary to the DFT in Fig. 5a, the possibility that B fails before A does is not excluded, however, the DFT does not fail if first B and then A occurs.

SEQs are powerful elements and can be used to express e.g. mutual exclusion, as shown for events A and B in Fig. 5c.

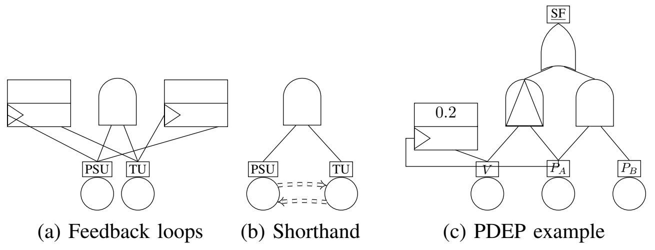


Fig. 6. FDEP examples

Here, the left child never fails, thus AND(A, B) may not fail.

e) A *functional dependency* (FDEP) is an element type which forces other elements to fail. As SEQs, FDEPs have no parents. The first child of an FDEP is called *trigger*, all other children are *dependent events*. If the trigger fails, then its dependent events also fail. FDEPs are helpful to model common cause failures and feedback loops. We consider the example given in [2], where a failing thermal unit (TU) causes the power supply unit (PSU) to overheat, and a failure of the PSU causes a failing TU. We give the corresponding DFT in Fig. 6a. Besides offering a convenient modelling principle, FDEPs are often used in combination with SPAREs to work around the limitations for (indirect) children of a SPARE, cf. Sect. III. To simplify the depicted DFTs, we use a double dotted arrow to denote FDEPs. The origin of the arrow corresponds to the trigger, while the target corresponds to the dependent event. We depict the identical DFT with the alternative notation in Fig. 6b.

f) The probabilistic dependency (PDEP) of Montani *et al.* [6], [7], [23] extends an FDEP with a probability p. When the trigger of the PDEP fails, the dependent events fail with probability p. Thus, a PDEP with p=1 is equivalent to an FDEP. If p=0, the dependent events are almost surely not triggered and the PDEP is superfluous.

Example 5: Consider the pumps from Ex. 1. Besides the failure rate, we assume that as soon as the valve is actually opened, it fails with a given (discrete) probability. We have modelled this by the DFT in Fig. 6c.

C. Mechanisms in DFTs

Before describing potential pitfalls in the usage and interpretations of DFTs, we review the different mechanisms which are exhibited by DFT elements.

Failure propagation. Failure propagation in DFTs is similar as in SFTs. However, DFTs exhibit two types of failure propagation: the usual upwards propagation (hereafter: *failure combination*), and the propagation via FDEPs (hereafter: *failure forwarding*). While failure combination is never cyclic, failure forwarding may (indirectly) cause the failure to propagate to the original element. These two mechanisms result in two different reasons why an element can fail.

Claiming. SPAREs require exclusive use of spare modules. Recall the car with the spare wheel from Ex. 3, where the spare wheel can only be used once. This requires a mechanism to ensure that a spare module is not replacing any other failed modules. We call this mechanism *claiming*. As soon as a spare module fails, the SPARE which claimed the module claims another module (represented by the child of the SPARE) which is not yet claimed. This newly claimed module communicates to its parents that it is no longer available to other SPAREs.

Activation propagation. To realise the support of reduced failure rates in case a component is in standby, DFTs introduce an *activation mechanism*. Spare modules are initially considered inactive, and thus all their elements are called inactive. The other elements are active. Active SPAREs propagate the activation signal to their used spare module. Inactive SPAREs do not emit any activation signal to any of the spare modules. Thus, as soon as an already active SPARE starts a child, all BEs in the spare module are activated. Active BEs fail with failure It is important to notice that FDEPs do not propagate the activation signal.

Event prevention. With SEQs, certain failure combinations can be explicitly excluded from the analysis. This is not limited to ruling out specific orders of BEs, but can be generalised to restrict certain claiming resolutions, although, in many cases, it requires ingenious fault trees.

III. DFTs: THE FLAVOURS

After having discussed the basic ingredients of DFTs, we are now in a position to discuss the various semantic issues raised in the introduction: lifting of cut sets to DFTs, the expressive power of including several fates, ordering in failure propagation, spare races, and activation versus claiming. This section will treat each of these matters in more detail.

A. Lifting minimal cut sets

Many analysis techniques for SFTs are based on *minimal cut sets* (MCSs) [3]. A cut set is a set of BEs causing the TLE to fail. A cut set is minimal if no true subset is a cut set. An SFT fails if one of its MCSs does so. This is rooted in AND gates to distribute over ORs. Since the order of failures matter in DFTs, (minimal) cut sets are extended to ordered tuples of BEs, called *(minimal) cut sequences* [24], [25]. There are however several caveats.

Minimality. First, one may think that a DFT fails whenever one of its MCSs does so. This is not true, because failure of the TLE may require BEs outside the cut sequence not to fail. Indeed, inserting extra failing BEs in a cut sequence may not yield a cut sequence. This is shown in Fig. 7(a), where AB and AC are MCSs, but BAC is not a cut sequence.

$$\boxed{\text{Cut sequences require the non-failure of BEs outside the cut sequence.}}$$

Characterisation. Whereas MCSs completely characterise the failure behaviour of the SFT, they do not do so for DFTs. We illustrate the issue in Fig. 7. Note that the right DFT is obtained from the left one by distributing the PAND over the OR. The MCSs for both DFTs in Fig. 7 are AB and AC. However, BAC is a cut sequence of 7(b) but not of 7(a).

$$\boxed{\text{Minimal cut sequences are insufficient to characterise the failure behaviour of a DFT.}}$$

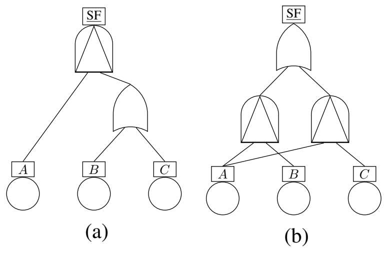


Fig. 7. Invalid distribution of ORs over PANDs.

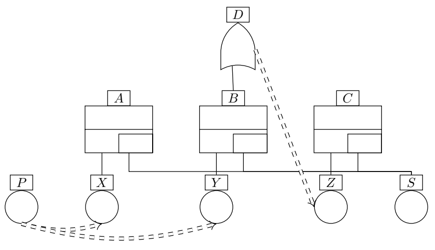


Fig. 8. Part of a DFT to justify the need for causality.

B. *Temporal and causal order*

A temporal order indicates which events happen before, after, or at the same time as another event. A causal order indicates for simultaneous events which events caused the failure of an event.

The need for simultaneity. The most common interpretation of DTFs assumes that both failure combination and forwarding is instantaneous. That is, as soon as a BE fails, all ancestors whose fail condition is fulfilled fail simultaneously. In particular, all dependent events that are triggered fail simultaneously, possibly giving rise to further simultaneous failures. Since the behaviour of the priority gates crucially depends on the fact whether or not their inputs fail simultaneously, it is important to cover this semantically. In particular, simultaneity should be included in cut sequences.

Simultaneity is needed to understand the behaviour of a DFT.

The need for a causal order. Multiple SPAREs can fail at once due to instantaneous failure propagation (cf. Fig. 14a). The claiming of spare modules is then ill-defined, as an ordering is required for claiming. This situation is called a *spare race*. Spare races can be resolved in numerous ways (as discussed later), however, it is important to respect causality.

Example 6: Consider the DFT in Fig. 8 containing three SPAREs (A, B, C) sharing a spare module S. The failure of P causes a spare race between A and B as they both want to claim S. Assume that A wins the race and claims S. SPARE B fails as it has no available children left. By failure combination, D fails too. The failure of D triggers the failure of Z, yielding C joining the spare race. Assuming that claiming and failure propagation are instantaneous, this means that C is racing with A and B so as to claim S. C however cannot win the race, as A claimed S in the argument before. C can thus *not* claim S once P fails.

In addition to temporality, causality is needed to understand DFT behaviour.

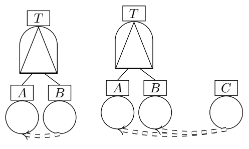


Fig. 9. Causal order and temporal ordering combined.

Embedding causal order in a temporal order. DFTs exhibit a notion of ordering as priority gates fail depending on the ordering of their children' failures. Usually, this is interpreted as a temporal order, which leads to two different notions of ordering in DFTs. To circumvent this, a possible realisation of causal order is by assuming (infinitesimal) time steps for applying cause-effect relations. Thereby, only one order relation (temporal) is present in the DFT. In many situations, assuming infinitesimal time steps in failure forwarding seems adequate, while for failure combination it often is not:

- In Fig. 6a+b, the failure PSU triggers TU to fail. One can interpret this as PSU causing the failure of TU. This is failure forwarding.
- In Ex. 3, the replacement of the spare wheel can be interpreted as claiming the spare wheel by a SPARE causing the wheel to be unavailable for any subsequent replacements.
- In Ex. 1, the failure of both motors does not happen after the last motor has failed, but exactly with the failure of the last motor.

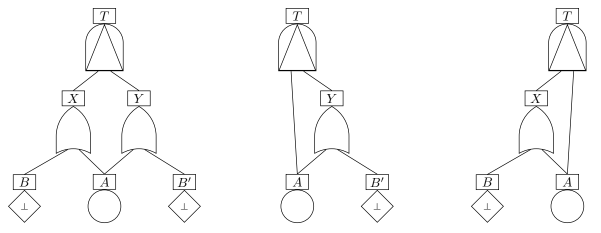
Thus, one might argue that failure forwarding can be naturally embedded in a temporal ordering, thereby resolving the issues in Ex. 6. The distinct applications of failure combination and failure forwarding often go along the lines sketched above. Later on, we will encounter another usage of failure forwarding where the usage of FDEPs is used inject failures into spare modules, see Ex. 12. In such scenarios, the natural embedding in a temporal ordering is often lost, and reliability is impacted by this, as in e.g. Fig. 9a, where then failure of B prevents the failure of the PAND.

The standard interpretation of BEs uses a total order, i.e. the standard interpretation of gates is based on this total order. If the causal order of failure forwarding is resolved by a temporal argument, the semantics of a temporal gate are affected by the resolution of the causal order. This suggests that causality for failure forwarding is a *partial* ordering.

Example 7: Consider the DFT depicted in Fig. 9b and assume a causal ordering for failure forwarding. A failure of C causes subsequent failures of A and B. Depending on the order of A and B, T fails. Assuming that the causal order is resolved by a (total) temporal order, then A and B fail in some order, but not simultaneously. This causes a nondeterministic failure behaviour of T.

Causality needs to be a partial order.

Combining causal and temporal ordering thus raises new issues. One other issue we point at is whether in Fig.9b, A and B should fail simultaneously after C. It could be argued that the infinitesimal time steps are almost surely not identical and so A and B do not fail simultaneously. On



(a) Reference (b) Deleted X (c) Deleted Y

Fig. 10. Illustrating the effect of ordered failure combination.

the other hand, it seems rather natural to assume that A and B may fail simultaneously. This should not be excluded, as any assumption ruling out simultaneous failures of A and B ignores potential interesting corner cases.

The embedding of causality in a temporal order can rule out interesting corner cases.

Ordered failures of gates. The combination of the—often unexpected—causal relation due to failure combination and the temporal conditions of the gates yields DFTs that are hard to comprehend and analyse. In particular, gates do not only fail when their failure condition is fulfilled, but they also influence the possible orderings of gate-failures. In the next example, failure combination is assumed to be totally ordered, i.e. gates do not fail simultaneously. If this ordering is assumed to be temporal, seemingly equivalent DFTs have different interpretations.

Example 8: Consider the DFT in Fig. 10a. Note that X fails iff A fails. By symmetry, the same applies to Y. Thus one expects that replacing X (or Y) by A in the DFT would result in an equivalent DFT, see Fig. 10b (and Fig. 10c). In the original DFT, after a failure of A, either X fails before Y or Y before X. The failure of PAND T depends on the ordering. The two "equivalent" DFTs differ fundamentally. In Fig. 10b, T fails, as A fails before Y—as failure combination is totally ordered. In Fig. 10c, T does not fail, as A fails before X violating the order of the PAND T.

Embedding causality in claiming yields also room for different interpretations, which we do not discuss here. To summarise, while merging a causal with a temporal ordering seems appropriate in many situations, it leads to delicate issues for DFTs. In particular, it seems more natural to assume a partial-order semantics for failure combination.

Embedding causality into a total, temporal order raises issues. A partial-order temporal order would be more adequate.

C. Expressiveness

Priority gates. PAND and POR gates require their inputs to fail from left to right. Various papers [2], [15] do not specify the behaviour of such gates in case inputs fail simultaneously. This may lead to problems [4]. The main issue is whether a strict (denoted $<$) or a weak ordering (denoted $\leq$) of the inputs is required. The former case is known as *exclusive*; the latter as *inclusive*. Table I on page 1 shows that both variants occur. Let us discuss some consequences choosing $<$ or $\leq$. In addition to OR, AND, PAND and POR gates, we consider simultaneous-AND (SAND, for short) gates. They only fail if their inputs fail simultaneously. Table II indicates the behaviour of the various gates in case inputs A and B fail. There are three possible scenarios: A occurs before B, A and B occur simultaneously, or A occurs after B. These three situations are listed in the table as $t(A) < t(B)$, $t(A) = t(B)$, and $t(A) > t(B)$, respectively. For every gate, the shaded area depicts the situations in which it is considered failed.

TABLE II. BEHAVIOUR OF SEVERAL BINARY GATES WITH GIVEN OCCURRENCE OF THEIR INPUTS.

	$t(A) < t(B)$	$t(A) = t(B)$	$t(A) > t(B)$
	A ─┼─ B ─┼─→	A, B ─┼─→	B ─┼─ A ─┼─→
AND(A, B)	──┼──┼▨▨→	──┼▨▨→	──┼──┼▨▨→
OR(A, B)	──┼▨▨┼▨▨→	──┼▨▨→	──┼▨▨┼▨▨→
SAND(A,B)	──┼──┼──→	──┼▨▨→	──┼──┼──→
PAND$_{\leq}$(A, B)	──┼──┼▨▨→	──┼▨▨→	──┼──┼──→
PAND$_{<}$(A, B)	──┼──┼▨▨→	──┼──→	──┼──┼──→
POR$_{\leq}$(A, B)	──┼▨▨┼▨▨→	──┼▨▨→	──┼──┼──→
POR$_{<}$(A, B)	──┼▨▨┼▨▨→	──┼──→	──┼──┼──→

Assuming instantaneous failure combination it follows:

SAND and inclusive POR (PAND) can be expressed using inclusive PAND (POR). Exclusive priority gates cannot be expressed using inclusive priority gates. All priority gates are expressible using exclusive POR. No other priority gate can be expressed using exclusive PAND.

Using ORs to express XOR. As mentioned before, some DFT dialects include XOR gates. These gates may yield non-coherent fault-trees [26]: an additional failed event may turn the system into an operational state again. As this seldomly occurs [17], a common approach (for SFTs) is to represent XOR elements by OR gates. Whereas this approach is guaranteed to under-approximate the system performance for SFTs, it may yield an *over-approximation* for DFTs. Let us illustrate this. Consider the DFT in Fig. 11, where the OR gate X is used to model an XOR. Consider failing sequence BCD. The system (with a truly XOR behaviour) fails after the additional occurrence of sequence AE, as Y does not fail. The DFT however cannot fail as X failed, causing the failure of Y and Z, thereby violating the ordering requirement of the PAND TLE. (Note that the precise behaviour of the PAND in case one of its children is non-coherent is not specified. Therefore, we use a somewhat verbose DFT in which we circumvent any unspecified behaviour.) The above phenomenon is caused by priority gates, where failed children may prevent the failure propagation of their parents. Observing that SFTs are DFTs too, it follows:

OR under- or over-approximates XOR, depending on the context.

Emulating SEQs. Sequence enforcers (SEQ) guarantee that elements only fail from left to right. Some papers [11], [23]

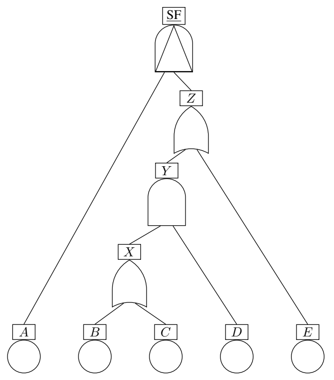


Fig. 11. Using an OR for XOR over-approximates the system's reliability.

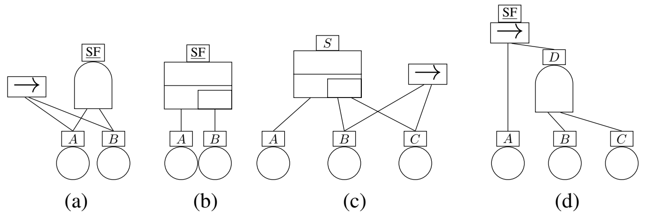


Fig. 12. Replacing a SEQ by a cold SPARE, for (c+d) this is invalid.

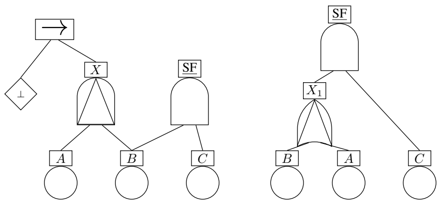
exclude SEQs and express them using cold SPAREs. The underlying idea of using cold SPAREs for SEQs is that the activation mechanism of the SPARE is able to enforce a sequence of events, by assuming cold standby. The DFT in Fig. 12a fails after A and B have failed, and B can only fail after A did. The DFT in Fig. 12b is equivalent. The (cold) SPARE ensures that B cannot fail if A did not fail, and the SPARE only fails once A and B both failed.

This replacement of SEQs by SPARES is not applicable in general, though. We consider two examples. Let us first consider Fig. 12c, assuming that B and C are warm spare modules. The SEQ expresses that C can only fail after B. Note that as B and C are in warm standby, the failure of A is not required for B or C to fail. Replacing the SEQ by a cold SPARE Y, say, yields—apart from some questionable syntax—that claiming B by Y prevents SPARE S from claiming B. In addition, C requires two distinct dormant failure distributions, as it is both in cold and warm standby. Our second example illustrates that application of the replacement in Fig. 12 yields different results in case one of the SEQ children is a gate (and not just a basic event). Consider Fig. 12d where B and C are initially both enabled. The failure sequence BAC respects the SEQ. For B to fail first, D needs to be active, and so B and C need to be both active. Once B fails, C needs to be disabled, as otherwise D could fail before A does, violating the SEQ. But C needs (again) to be activated once A occurs. Since SPAREs only allow spare modules to be activated once, this behaviour cannot be expressed using SPAREs.

$\boxed{\text{SEQs cannot always be expressed using cold SPAREs.}}$

Consider Fig. 13a in which X is not allowed to fail (by the SEQ). This enforces that B may only fail before A. Such usages of SEQs can be more naturally expressed using the POR, as shown in Fig. 13a.

SPAREs as dependent gates of FDEPs. FDEPs have a trigger and a dependent basic event. Some DFT dialects allow



(a) With a SEQ (b) With a POR

Fig. 13. Order-dependent event prevention.

FDEPs to have dependent gates (rather than events) [10]. Such gates can fail themselves by their type-dependent condition on their children, or when triggered. The dependent gate then propagates it failure upwards; this can also be explicitly modelled by an additional OR gate. Consider an FDEP with a dependent SPARE. Assume the SPARE fails by triggering. In this situation, it seems natural to expect the SPARE to neither claim nor activate any of its spare modules. However, many semantics do not support such a deactivation of a SPARE.

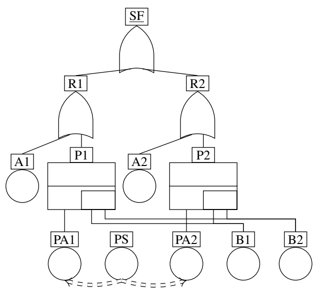
$\boxed{\text{FDEPs with dependent gates require special treatment.}}$

Evidence. A popular use of DFTs is to derive specific scenarios in which certain states (or failures)—referred to as evidences—are assumed in the system. This is particularly interesting when considering the system's survivability, i.e. how well can a system recover from a specific bad state? Evidences can be naturally modelled by using constant fault elements. This however requires a well-defined semantic treatment of "simultaneous" failures at initialisation, a situation that is not conform the common assumption that BEs never fail simultaneously. In particular the treatment of SPAREs requires attention, as the initial failure of spare modules may yield an underspecified state of the DFT.

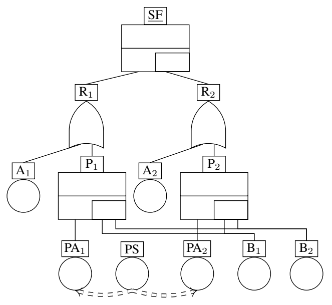
$\boxed{\text{Setting evidence may yield underspecified initial states.}}$

D. The resolution of spare races

Recall that spare races occur when multiple SPAREs fail simultaneously, and some of the failed SPAREs claim a shared spare module at the same time, see also Ex. 6. How can spare races be resolved? One possible strategy is to resolve spare races *nondeterministically*, i.e., consider all potential resolutions by leaving open the exact claiming order. Some authors have argued against nondeterminism in fault trees, e.g. Merle *et al.* [12] claim that critical infrastructures should be deterministic. However, especially in systems where human actions are involved, policies might not be as precise (or precisely followed) and system behaviour may not always be deterministic. Moreover, FTA is not only applied to existing systems, but often also during design time [2]. In early design phases, not all concrete information about the system is available, and typically deliberately left open so as to allow for several system implementations. In these setting, nondeterminism is a convenient means to model underspecification. Nevertheless, we agree that it is valuable to have additional support for deterministic claiming policies in DFTs.



(a) Simple communication system.



(b) With a spare radio.

Fig. 14. DFT of a communication system.

Example 9: Consider the DFT in 14a. The DFT describes a communication system consisting of two radios R_1 and R_2, which both have to be operational. Each radio consists of an antenna (A_1 and A_2, respectively) and a power unit (P_1 and P_2, respectively). Both power units have their own power adaptor (PA_1 and PA_2, respectively). The power adaptors are connected to a common power supply (PS). Every power unit can use one of the spare batteries (B_1, B_2) which have identical failure distributions. The failure of PS leads to a spare race. However, as there are spares for both P_1 and P_2, the actual order of claiming has no influence. This is even true if there would be only a single (available) spare module, as the second power unit which tries to claim a battery would fail, and therefore, the whole system would fail. Thus in this case the resolution of the spare race does not influence the DFT's reliability. However, if the system only fails once both radios fail, i.e., if SF would be an AND-gate, then a single spare module may lead to different outcomes, if either one of the antennas fails prior to the spare race, or it the antennas have different failure distributions. Note that with two available spare modules (and without the assumption that the failure distributions of the spare modules are identical) the outcome may also depend on the way the spare race is resolved.

Spare races can alternatively be resolved randomly, e.g. by imposing a uniform distribution over all alternatives [4]. This choice is justified by the assumption that the spare modules (in [4] these are BEs) typically have equal properties. This is however not enforced. The practical downside of a random resolution is that the DFT's reliability is influenced by the used distributions, a fact that is often ignored when presenting the reliability results. This bias is not present when solving races nondeterministically.

Spare races are preferably resolved deterministically.
If impossible, nondeterministic resolution is preferred.

E. Claiming versus activation

Conceptually, there are two important differences between claiming a spare module (e.g., a BE) and activation of a BE: (a) the moment a module is being used somewhere is not necessarily the moment it is activated, and (b) components not subject to exclusive claiming may be inactive or active.

Early and late claiming. In practice, various scenarios may occur in which spare modules itself are complex trees which may make use of spare parts. This can be modelled by nested SPAREs, i.e. SPAREs that have one or more SPAREs as (indirect) child. Consider a SPARE that acts as spare module. Can this SPARE claim one of its spare modules if it is not itself activated yet? If yes, we call this *early claiming*; if not, it is called *late claiming*. Under early claiming, a SPARE can claim regardless of being activated or not. Inactive SPAREs cannot claim under late claiming. Under early claiming it is clear when a SPARE fails, as it can claim whenever it wants. But for late claiming there are two possibilities: *early* or *late failing*. With early failing, a late claiming SPARE fails if all its primary and spare modules failed. Under this regime, an inactive SPARE can thus fail (as a kind of "look ahead" feature.) With late failing, this cannot happen; in this case claiming only takes place upon activation. The following example illustrates the differences.

Example 10: Consider the DFT in Fig. 14b, originating from a communication system as in Ex. 9. Assume the radio R_2 is in passive standby. Consider the failure of PA_2. Under early claiming, the power unit P_2 directly claims some battery which then cannot be claimed anymore by P_1. Under late claiming, P_2 does not claim any of the batteries yet. Instead, it will only claim a battery once R_1 failed and R_2 has subsequently been activated. Using early failing, P_2 fails—regardless of R_2 being activated or not—whenever either both B_1 and B_2 failed, or one of them failed and the other was claimed by P_1. Using late failing, P_2 fails only if it fails to claim something upon activation of R_2.

When to claim and when to fail do matter.

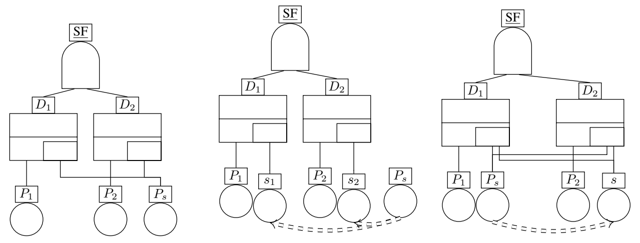
Which behaviour fits best depends on the use case and cannot be fixed a priori in general. The different possible semantic interpretations have more effects, however, the differences only apply to DFTs with nested spares.

On one hand, late claiming introduces failure due to claiming or activation, respectively. With early claiming, a SPARE only claims after a child has failed. It does not fail to claim without a child failing at the same moment. With late claiming and early failure, SPAREs may fail due to children being claimed by other SPAREs. Moreover, claiming may cause spare races, resulting in event propagation and claiming becoming interdependent. Analogously, for late claiming with late failure, SPAREs may fail upon activation. Thereby, activation may cause spare races; event propagation and activation become interdependent. On the other hand, using late claiming allows a uniform treatment of activation and claiming.

Example 11: Consider again Fig. 14b where PA_2 failed. Let subsequently be PS B_1 B_2 the failure order. Under late claiming, P_2 does not claim any battery (as it is not yet active) when PS fails. P_1 thus claims B_1, and once B_1 fails, it claims B_2. Under early failing, P_2 now fails as none of its children is available anymore. It thus fails before B_2 does. Under late failing, P_1 fails once B_2 fails, and R_2 is activated. Now P_2 is activated. As it cannot claim any child, P_2 fails after B_2.

Early and late failing (claiming) are incomparable.

Spare modules revisited. We now focus on the precise extent of spare modules and the consequences thereof w.r.t. claiming and activation. If we restrict spare modules to BEs, activation



(a) Exclusive power generator. (b) Permanently active power generator. (c) Work-around

Fig. 15. Different attempts of modelling a shared power generator.

and claiming coincide. To activate a component, it has to be claimed. As claiming is exclusive, this leads to work-arounds to model system behaviour with DFTs.

Example 12: Consider two medical devices, which fail once their power supply fails. The power supply can be replaced by a power generator, which is powerful enough to drive both devices. The failure rate of the power generator rises on using the generator, i.e. once a primary power supply fails. Fig. 15 shows different DFTs for the system. In Fig. 15a, once the primary power supply (P_1 and P_2, respectively) fails, the corresponding device (D_1, D_2) claims the power generator P_s. If the primary supply of the other unit fails, the DFT assumes the system fails, as the usage of the power generator is exclusive. Thus, the depicted DFT fails to model that the power generator is able to power both devices. Consider now Fig. 15b. Once one of the primary power supplies fails, the device claims their "connection" to the power generator (s_1 or s_2). However, only the connection is activated. The power generator P_s thus either stays active or passive, independent of the failure of the primary power supplies. In Fig. 15c, we give a work-around of these problems. The first device with a failed power supply claims and activates the power generator. If the power supply of the other unit fails as well, it claims and activates the connection, which fails with the power generator. Depending on the exact failure propagation behaviour, a failure of the power generator before the second primary power supply fails might cause the power unit which used the power generator before to claim the connection (s). However, this connection would then directly fail.

A straightforward adaption would be given by propagating activation in reverse direction through FDEPs. Upon activation of the dependent event, the trigger of such FDEP is then activated as well. Notice that reverse throughput is not always appropriate, e.g. when modelling feedback loops. Without using subtrees as spare modules, scenarios in which a module is activated with a spare module but does not add to its failure distribution are possible.

Until now, we have been rather imprecise about the exact interpretation of spare modules. We recall the accurate description depicted in Fig. 4b. There, the spare modules were independent, i.e. unconnected graphs. The independence criterium leads to work-arounds which do not agree with the hierarchical way DFTs should be created. However, dropping it yields multiple open questions. We do not cover further details here, as any extension towards this requires dedicated treatment and is not present in any existing interpretation of DFTs.

Claiming and activation coincide only for simple cases. Their precise semantics is intricate and open.

IV. DFTs: THE COOKS

In this section, we consider and compare several existing formalisations of DFTs. The origin of DFTs, e.g. in [27] did not provide formal semantics. This has led to an unclear meaning of specific fault trees, as outlined in [4]. Since this initial formalisation, several others have been introduced which are not fully compatible to each other. We discuss eight different formalisations in greater depth. A tabular comparison of specific features is given in Table I. We do neither include the formalisations used in the Monte-Carlo approaches in [28], [29] nor the definition in [3]. Attempts for using MCSs as presented in [24] and in [25] are excluded as these are not suitable to describe the behaviour of DFTs, as discussed in Sect. III-A. Many techniques [30], [31], cf. also [3] are known to speed-up the analysis. The correctness of such approaches depends on the chosen semantics.

A. Fault tree automaton construction

A DFT semantics in terms of a *fault tree automaton* (FTA) is given in [4]. It defines an operational semantics-style axiomatisation of DFTs, formalised in Z [32]. The semantics builds upon the notion of a state of the DFT containing information about the order in which elements have failed as well as usage information for the SPAREs. Given a DFT in state s, an occurrence e and the resulting child s', the semantics formalises whether s' is a valid resulting state. Based on this, a FTA can be constructed which describes the DFT by a non-deterministic automaton. For analysis, the underlying CTMC (Continuous-Time Markov Chain) is constructed by resolving non-determinism by a uniform distribution, see Sect. III-D. The reliability a DFT is then computed on the underlying CTMC. The formalisation of PANDs is inclusive. FDEPs cause immediate failure propagation to the dependent events. Triggers are allowed to be subtrees, while the dependent events are BEs. SPAREs require that their children are BEs. All such BEs are required to have only FDEPs, SPAREs or SEQs as parents. SEQs are included in the most general form. Spare races are resolved nondeterministically. Note that no notion of causality is included. Tool support for FTAs in Galileo is presented in [33]; underlying algorithms are described in [34].

B. Reduction to Bayesian networks

A popular method to support quantitative analysis of (dynamic) FTs is based on Bayesian Networks (BN) [35]. We consider the reduction to Discrete Time Bayesian Networks (DTBN) in [36], Continuous Time Bayesian Networks (CTBN) in [5], and Dynamic Bayesian Networks (DBN, [37]) in [6], [23], [7]. The underlying idea is to introduce random variables for each DFT event. Random variables representing gates are conditionally dependent on the random variables representing the children. Cycles introduced by FDEPs are disallowed, as this would yield a cyclic BN. BEs are represented by multi-valued variables, which encode not only whether the BE failed, but also whether it is active. This enables the integration of warm-standby. Tools for BN analysis are widely available. The reduction to BNs allows several additional analyses on fault trees, e.g. the *most likely explanation* analysis which is not presented in other semantics.

Reduction to Discrete or Continuous Time BN. The reduction of a DFT to a DTBN as described in [36] is based on discretising a time interval $]0, T]$, where T denotes the mission time, by slicing it into n possibly equidistant intervals. A failure event occurs during an interval $i \leq n$, instead of a time point $t \leq T$. Note that for $\lim_{n\to\infty}$ the DTBN is equivalent to the CTBN described in [5]. Obviously, the discretisation introduces some inaccuracies. However, on the global level presented here, we can ignore these inaccuracy and regard the two formalisms as equivalent. Each gate is represented by a random variable and conditional dependencies with all children are given. For static gates, the conditional rules are directly derived from the truth table of the gates. PANDs are assumed to be inclusive. FDEPs directly cause the failure of their dependent events. The triggers may be subtrees, while the dependent events are assumed to be BEs. SPAREs only have BEs as children. Sharing of spares is not considered. Moreover, common cause failures for spare modules are not handled.

Reduction to Dynamic BN. The encoding to a DBN is also based on discretising time. However, instead of slicing a time interval, DBNs assume discrete time points for each event. For each element in the DFT, a DBN is introduced, which are merged into a single DBN afterwards. During the merging process, the conditional probability tables are merged. For this process, it is assumed that the conditional failure probability equals the maximum conditional failure probability in the two merged nodes, given any condition. This introduces an error but yields smaller tables. For the PAND, an additional variable is introduced which keeps track of the ordering. PANDs are inclusive. FDEPs are extended to PDEPs. Failures are instantaneously propagated. SPAREs assume BEs as children. The behaviour in case of a spare race is not specified. SEQs are not included. Tool support is provided by Radyban[7].

C. Reduction to Stochastic Well-formed Petri Nets

Stochastic Well-formed Coloured Nets (SWN) [38] are an extension to Petri nets with exponential timing. The work [39] reduces parametric fault trees to SWNs. Parametric fault trees are SFTs with subtree replication aimed at yielding a smaller state space by exploiting symmetry. This approach was expanded to DFTs in [8]. Although the work also considers repairs, this is outside the scope of this work. For each DFT element, a small Petri net is given, which has input places and an output place, i.e. each DFT element operates based on the presence of tokens in its input places. A failure of the gate causes a transition to fire which then places tokens in any predecessor nets of the DFT. To compose a SWN for a DFT with multiple elements, the inputs and outputs are merged according to the structure of the DFT.

Remark: The semantics of a SWN as described in the references cited in [8] allow only one transition to fire at a time. Therefore, we assume that only one transition fires at once. SWNs allow for priority assignments to select which transition fires when multiple transitions are enabled. The presented semantics for DFTs however do not mention this.

Gate failures are ordered, as synchronisation between the elements is done via placing tokens, the effect of any element failing is not simultaneously processed by the parents. PANDs are inclusive, and FDEPs do not distinguish different dependent events. Both triggers and dependent events are assumed to be BEs. SPAREs assume BEs as children. Any sharing is implicitly assumed to be amongst symmetric SPAREs using identical BEs as spare elements. Thus, non-determinism during claiming is hidden. Warm and cold standby are discussed, but their interpretation remains unclear. SEQs are presented in a general fashion, but their interpretation when putting restrictions on gates is different. Instead of invalidating a sequence, the failures are delayed until the more-to-the-left children of the SEQ have failed.

D. Reduction to GSPN

A reduction of DFTs to Generalised Stochastic Petri Nets (GSPN) [40] is given in [9]. The overall idea is to use a graph transformation for an element-wise reduction to a GSPN. This GSPN can then be reduced to a CTMC using existing algorithms [41]. In a first step, a place for each vertex is added, where a marking on such a place means that the element has failed. Each gate is then replaced by a subnet which places a token in its output place depending on markings in the input places, thus, gates fail ordered. The static gates are trivially defined. For PANDs, an extra place checks whether the ordering is respected. The ordering is assumed to be inclusive. FDEPs mark all dependent events failed, in a non-deterministic ordering. It remains unclear how this non-determinism is resolved for the reduction to a CTMC (most probably by a uniform distribution). Warm spares are supported, but neither spare pool sharing nor non-singleton spare pools are handled. The SEQ requires all children to be BEs.

E. Reduction to a set of IOIMCs

Input/Output-Interactive Markov Chains (IOIMCs) extend IMCs [42] by distinguishing inputs and outputs. This model is amenable to a compositional design of Markovian processes. The overall idea is to define small IOIMCs for each DFT element and to compose these IOIMCs to obtain a representation for the entire DFT [14], [10], [11]. Gate inputs are encoded as input-transitions while failure propagations are modelled as output-transitions. The behaviour of the entire DFT is given by the parallel composition of the individual IOIMCs. Moreover, SPAREs distribute claiming and activation information via extra transitions. It is important to notice that only one transition at a time can fire in IOIMCs. Therefore, all elements fail in some order, where the order is given by non-deterministic choices. The encoding of the static gates is straightforward. PANDs are non-inclusive (simultaneous failures do not occur), FDEPs propagate their failures of the triggers to the dependent elements. Both triggers as well as dependent elements can be subtrees. Dependent gates are resolved by extra internal transitions which are alike extra BEs connected to an OR-gate, as discussed in III-C. SPAREs have independent subtrees as spare modules. Nested SPAREs are allowed and follow the late claiming regime with early failure mechanism. SEQs are not included. Tool-support is given by DFTCalc [43], which includes support for evidence by replacing BEs with constants.

F. Algebraic encoding

This section is dedicated to a formalisation of DFTs by an algebraic description, as described in [12], [44], [13].

TABLE III. List of case studies and potential issues.

Name	Source	Spare races	OFC	FSG	SEQs	Evidence
AHRS	[47]					
B3	[48]				✓	
CAS	[49], [10]					✓
FTTP	[1]	✓			✓	
FDS	[22]		✓			
HECS	[2]					✓
MAS	[49], [1]				✓	✓
MCS	[9], [43], [6]	✓		✓		✓
NDWP	[10]	✓				✓
RC	[50]					
SAP	[51]		✓			✓
SF	[52]				(✓)	
SSS	[20]					

Similar efforts can be found in [45], [20] and [46]. However, those formalisations do not include SPAREs and are therefore excluded here. SFTs are trivially embedded into Boolean algebra. In [12], the authors extend Boolean algebra with temporal operators for *before*, *inclusive before* and *simultaneous* to formalise "priority DFTs with repeated events", in fact SFTs with PANDs and FDEPs. "Repeated events" are used to emphasise that the underlying graph is not necessarily a tree. Although the formalisation method supports both inclusive and exclusive PANDs, the authors choose the inclusive variant as it "seems more coherent with the designers' expectations". Failure propagation is immediate. Furthermore, each DFT is given a canonical representation in the algebra which extends MCSs with ordering information over all BEs, as well as a scheme for deducing the top-level failure distribution given fault distributions of the BEs. In [44], the authors use the same algebra for SPAREs. The considered SPAREs only allow BEs as children. It is assumed that the BEs do not occur simultaneously, which excludes common cause failures in SPAREs. Activation is realised by considering two events, one with a warm and one with a hot failure rate thereby explicitly excluding the occurrence of both failures. SEQs are not included. Constant failures are not presented, although present in the algebra.

V. DFTs: Benchmarks

So far we have discussed the several issues of DFTs on the basis of small illustrative examples. The question rises whether these phenomena also occur in realistic DFTs. To answer this question (affirmatively), we studied several DFT benchmarks from the literature, see Table III. The spare race column marks the benchmarks in which spare races occur and influence the reliability of the DFT. In some other benchmarks, spare races only occur after the DFT has failed. The OFC column lists benchmarks in which ordered failure combination yields possibly wrong results—this also relates to inclusive vs. exclusive priority gates. FSG indicates that only in the MCS benchmark, the behaviour of failed spare gates is relevant. SEQs occur in MAS and FTTP, and should have been included in SF (see below). The column evidence indicates that including a constant failure yields ambiguities. Evidences then give rise to more issue in presence spare modules and (including vs. excluding) priority gates.

A few remarks are in order. Most benchmarks, except MAS and FTPP, were presented in the literature to show the feasibility of some particular approach. These DFTs are therefore often compact and have only a small static fragment. However, with the current state-of-the-art, much larger DFTs could be analysed. Our experience with industrial partners indicate that most DFTs are indeed largely static, i.e. the vast majority of gates is static, presumably even the vast majority of subtrees. This is backed also by [46], [50], [24]. Most DFTs accompanying DFT analysis tools do not match the guidelines [2] for hierarchically constructed FTs—it seems that many constructs are crafted to match the system, but not following a hierarchical approach. Furthermore, SEQs were only used in older versions of some DFTs to model the spare management. With existing tool support for the warm SPARE, SEQs are not present anymore. In the SF benchmarks, the PANDs are (wrongly) used as SEQs. We finally observe that (correct usage) of priority gates is mostly used to model reconfiguration. This applies to SSS, FDS, as it gives more freedom than SPAREs. Without SPAREs, warm spares modules can only be modelled via a combination of extra basic events and SEQs, as e.g. in MAS.

In all benchmarks, the spare modules are BEs—none of the benchmarks contains subtrees, or spares, as spare modules. This is justified by the fact that the analysis support for this has received scant attention so far. More succinct and comprehensible DFTs for the HECS and AHRS benchmarks could be obtained by relaxing the restriction on spare modules being BEs. SPARE gates in general yield more compact DFTs, e.g. the DFTs to FTTP and MAS were significantly simplified by exploiting SPARE gates.

VI. Conclusion

A detailed discussion of existing DFT features and their possible (mis-)interpretations was presented. Problems of the various DFT dialects were described, approaches for solving these problems were suggested, and distinctive features with respect to the variants' syntactical and semantic aspects were elaborated. The interplay between temporal and causal dependencies makes DFTs complex and intricate to understand. Classical concepts like cut sets are difficult to generalise to DFTs. The bottom line of this thorough investigation is that despite their seeming simplicity, *DFTs are complex objects*. Engineers should be aware of the various subtleties and nuances of interpretation so as to interpret the DFT analysis results in an appropriate way.

Acknowledgement

This work has been partially supported by the STW-ProRail partnership program ExploRail under the project ArRangeer (12238), CDZ project CAP (GZ 1023), and the EU FP7 grant agreement no. 318003 (TREsPASS). We thank Enno Ruijters for the inspiring discussions and the anonymous reviewers for their extensive feedback.

References

[1] J. B. Dugan, S. J. Bavuso, and M. A. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems," *IEEE Trans. Rel.*, vol. 41, no. 3, pp. 363–377, 1992.

[2] W. Vesely and M. Stamatelatos, "Fault tree handbook with aerospace applications," NASA Headquarters, USA, Tech. Rep., 2002.

[3] E. Ruijters and M. Stoelinga, "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools," *Computer Science Review*, vol. 15, pp. 29–62, 2015.

[4] D. Coppit, K. J. Sullivan, and J. B. Dugan, "Formal semantics of models for computational engineering: a case study on Dynamic Fault Trees," in *Proc. of ISSRE*. IEEE Computer Society, 2000, pp. 270–282.

[5] H. Boudali and J. B. Dugan, "A continuous-time Bayesian network reliability modeling and analysis framework," *IEEE Trans. Rel.*, vol. 55, no. 1, pp. 86–97, 2006.

[6] S. Montani, L. Portinale, A. Bobbio, and D. Codetta-Raiteri, "Automatically translating dynamic fault trees into dynamic Bayesian networks by means of a software tool," in *Proc. of ARES*, 2006, pp. 804–810.

[7] ——, "Radyban: A tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks," *Rel. Eng. & Sys. Safety*, vol. 93, no. 7, pp. 922–932, 2008.

[8] A. Bobbio and D. Codetta-Raiteri, "Parametric fault trees with dynamic gates and repair boxes," in *Proc. of RAMS*, 2004, pp. 459–465.

[9] D. Codetta-Raiteri, "The conversion of dynamic fault trees to stochastic Petri nets, as a case of graph transformation," in *Proc. of PNGT*, vol. 127, no. 2, 2005, pp. 45 – 60.

[10] H. Boudali, P. Crouzen, and M. Stoelinga, "A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains," in *Proc. of ATVA*, ser. LNCS, vol. 4762. Springer, 2007, pp. 441–456.

[11] ——, "A rigorous, compositional, and extensible framework for dynamic fault tree analysis," *IEEE Trans. Dependable Secure Computing*, vol. 7, no. 2, pp. 128–143, 2010.

[12] G. Merle, J.-M. Roussel, J.-J. Lesage, and A. Bobbio, "Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events," *IEEE Trans. Rel.*, vol. 59, no. 1, pp. 250–261, 2010.

[13] G. Merle, J.-M. Roussel, and J.-J. Lesage, "Quantitative analysis of dynamic fault trees based on the structure function," *Quality and Reliability Engineering International*, vol. 30, no. 1, pp. 143–156, 2014.

[14] H. Boudali, P. Crouzen, and M. Stoelinga, "Dynamic fault tree analysis using input/output interactive Markov chains," in *Proc. of DSN*. IEEE Computer Society, 2007, pp. 708–717.

[15] "Fault tree analysis (FTA)," Norm IEC 60050:2006, 2007.

[16] D. Codetta-Raiteri, "Integrating several formalisms in order to increase fault trees' modeling power," *Rel. Eng. & Sys. Safety*, vol. 96, no. 5, pp. 534 – 544, 2011.

[17] S. Contini, G. Cojazzi, and G. Renda, "On the use of non-coherent fault trees in safety and security studies," in *Proc. of ESREL*, vol. 93, no. 12. Elsevier, 2008, pp. 1886 – 1895.

[18] T. Yuge and S. Yanagi, "Quantitative analysis of a fault tree with priority AND gates," *Rel. Eng. & Sys. Safety*, vol. 93, no. 11, pp. 1577–83, 2008.

[19] J. Xiang, F. Machida, K. Tadano, K. Yanoo, W. Sun, and Y. Maeno, "A static analysis of dynamic fault trees with priority-and gates," in *Proc. of LADC*. IEEE Computer Society, 2013, pp. 58–67.

[20] M. Walker and Y. Papadopoulos, "A hierarchical method for the reduction of temporal expressions in pandora," in *Proc. of DYADEM-FTS*. ACM Press, 2010, pp. 7–12.

[21] ——, "Qualitative temporal analysis: Towards a full implementation of the fault tree handbook," *Control Engineering Practice*, vol. 17, no. 10, pp. 1115 – 1125, 2009.

[22] E. Edifor, M. Walker, and N. Gordon, "Quantification of priority-or gates in temporal fault trees," in *Proc. of SAFECOMP*, ser. LNCS. Springer, 2012, vol. 7612, pp. 99–110.

[23] S. Montani, L. Portinale, A. Bobbio, M. Varesio, and D. Codetta-Raiteri, "A tool for automatically translating dynamic fault trees into dynamic Bayesian networks," in *Proc. of RAMS*, 2006, pp. 434–441.

[24] Z. Tang and J. B. Dugan, "Minimal cut set/sequence generation for dynamic fault trees," in *Proc. of RAMS*, Jan 2004, pp. 207–213.

[25] D. Liu, W. Xing, C. Zhang, R. Li, and H. Li, "Cut sequence set generation for fault tree analysis," in *Embedded Software and Systems*, ser. LNCS. Springer, 2007, vol. 4523, pp. 592–603.

[26] T. Chu and G. Apostolakis, "Methods for probabilistic analysis of noncoherent fault trees," *IEEE Trans. Rel.*, vol. R-29, no. 5, pp. 354–360, 1980.

[27] J. B. Dugan, S. J. Bavuso, and M. Boyd, "Fault trees and sequence dependencies," in *Proc. of RAMS*, 1990, pp. 286–293.

[28] H. Boudali, A. Nijmeijer, and M. Stoelinga, "DFTSim: A Simulation Tool for Extended Dynamic Fault Trees," in *Proc. of ANSS*, 2009, p. 31.

[29] F. Chiacchio, L. Compagno, D. D'Urso, G. Manno, and N. Trapani, "An open-source application to model and solve dynamic fault tree of real industrial systems," in *Proc. of SKIMA*. IEEE, 2011, pp. 1–8.

[30] R. Gulati and J. B. Dugan, "A modular approach for analyzing static and dynamic fault trees," in *Reliability and Maintainability Symposium. 1997 Proceedings, Annual*. IEEE, 1997, pp. 57–63.

[31] S. Junges, D. Guck, J.-P. Katoen, A. Rensink, and M. Stoelinga, "Fault trees on a diet - automated reduction by graph rewriting," ser. LNCS, vol. 9409. Springer, 2015, pp. 3–18.

[32] J. Spivey, *The Z Notation: A Reference Manual*. Prentice Hall, 1992.

[33] K. Sullivan, J. B. Dugan, and D. Coppit, "The Galileo fault tree analysis tool," in *Proc. of FTCS*, 1999, pp. 232–235.

[34] R. Manian, D. W. Coppit, K. J. Sullivan, and J. B. Dugan, "Bridging the gap between systems and dynamic fault tree models," in *Proc. of RAMS*, 1999, pp. 105–111.

[35] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., 1988.

[36] H. Boudali and J. B. Dugan, "A discrete-time Bayesian network reliability modeling and analysis framework," *Rel. Eng. & Sys. Safety*, vol. 87, pp. 337–349, 2005.

[37] Z. Ghahramani, "Learning dynamic Bayesian networks," in *Adaptive Proc. of Sequences and Data Structures*. Springer, 1998, pp. 168–197.

[38] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, "Stochastic well-formed colored nets and symmetric modeling applications," *Trans. Computing*, vol. 42, no. 11, pp. 1343–1360, 1993.

[39] A. Bobbio, G. Franceschinis, R. Gaeta, and L. Portinale, "Parametric fault tree for the dependability analysis of redundant systems and its high-level Petri net semantics," *IEEE TSE*, vol. 29, no. 3, pp. 270–287, 2003.

[40] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*, 1st ed. Wiley, 1994.

[41] G. Balbo, "Introduction to generalized stochastic Petri nets," in *Formal Methods for Perf. Eval.*, ser. LNCS. Springer, 2007, vol. 4486, pp. 83–131.

[42] H. Hermanns, "Interactive Markov Chains: And the Quest for Quantified Quality," *LNCS*, 2002.

[43] F. Arnold, A. Belinfante, F. van der Berg, D. Guck, and M. Stoelinga, "DFTCalc: A tool for efficient fault tree analysis," in *Proc. of SAFECOMP*, ser. LNCS. Springer, 2013, vol. 8153, pp. 293–301.

[44] G. Merle, J.-M. Roussel, J.-J. Lesage, and N. Vayatis, "Analytical calculation of failure probabilities in dynamic fault trees including spare gates," in *Proc. of ESREL*, 2010, pp. 794–801.

[45] M. D. Walker, "Pandora: a logic for the qualitative analysis of temporal fault trees," Ph.D. dissertation, University of Hull, 2009.

[46] S. Schilling, "Beitrag zur dynamischen Fehlerbaumanalyse ohne Modulbildung und zustandsbasierte Erweiterungen," Ph.D. dissertation, Universität Wuppertal, 2009.

[47] H. Boudali and J. B. Dugan, "A new Bayesian network approach to solve dynamic fault trees," in *Proc. of RAMS*, 2005, pp. 451–456.

[48] H. Zhu, S. Zhou, J. B. Dugan, and K. Sullivan, "A benchmark for quantitative fault tree reliability analysis," in *Proc. of RAMS*, 2001, pp. 86–93.

[49] K. Vemuri, J. B. Dugan, and K. Sullivan, "Automatic synthesis of fault trees for computer-based systems," *IEEE Trans. Rel.*, vol. 48, no. 4, pp. 394–402, 1999.

[50] D. Guck, J.-P. Katoen, M. Stoelinga, T. Luiten, and J. Romijn, "Smart railroad maintenance engineering with stochastic model checking," in *Proc. of RAILWAYS*, ser. Civil-Comp Proceedings, vol. 104. Civil-Comp Press, 2014, p. 299.

[51] F. Chiacchio, L. Compagno, D. D'Urso, G. Manno, and N. Trapani, "Dynamic fault trees resolution: A conscious trade-off between analytical and simulative approaches," *Rel. Eng. & Sys. Safety*, vol. 96, no. 11, pp. 1515 – 1526, 2011.

[52] M. Bozzano, R. Cavada, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and X. Olive, "Formal verification and validation of AADL models," in *Proc. of ERTS*, 2010.