Reliability Engineering and System Safety 142 (2015) 184–191

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Towards a sound semantics for dynamic fault trees

Antoine Rauzy *, Chaire Blériot-Fabre

Laboratoire de Génie Industriel, Ecole Centrale Paris, Grande voie des vignes, Châtenay-Malabry 92295, France

ARTICLE INFO

Article history:
Received 1 November 2012
Received in revised form
30 March 2015
Accepted 26 April 2015
Available online 16 May 2015

Keywords:
Dynamic fault trees
Guarded transitions systems

ABSTRACT

In this article, we study the semantics of dynamic fault trees and related formalisms. We suggest that there are actually three mechanisms at work in dynamic fault trees: first, changes of states due to occurrences of events, second bottom-up propagations of values as in static fault trees, and third top-down propagations of demands of activations of components. We propose a direct translation of dynamic fault trees into guarded transitions systems, the underlying mathematical model of the AltaRica 3.0 modeling language. This encoding provides a good basis for our study. We discuss also assessment algorithms at hand in light of this translation.

1. Introduction

In the recent past years, dynamic fault trees and related formalisms have focused a large attention in the reliability engineering literature (see e.g. [7,13,3,11,4,5,8,15,17]). By adding some extra-logical constructs to regular/static fault trees, one aims to describe dynamic behaviors, i.e. to put constraints on order of occurrences of events, while maintaining the conceptual (and graphical) simplicity of fault trees. This increase in expressive power comes indeed with a price: models cannot be interpreted anymore in the Boolean algebra framework.

In this article, we claim that there are actually three mechanisms at work in dynamic fault trees and Boolean Driven Markov Processes: first, changes of states of due to occurrences of events, second bottom-up propagations of values as in static fault trees, and third top-down propagations of demands of activations of components. To define a sound semantics of dynamic fault trees, we encode them into guarded transitions systems. Guarded transitions systems have been introduced in reference [19]. They are at the core of the new version of the high level modeling language AltaRica 3.0 (see e.g. [20]). Guarded transitions systems are (finite or infinite) state automata with input and outputs. They generalize most of the formalisms used for probabilistic safety analyses, including static fault trees, reliability block diagrams and generalized stochastic Petri nets. The definition of a semantic for dynamic fault trees involves actually multi-state components, immediate and timed (stochastic) transitions as in generalized stochastic Petri nets [1] (these concepts are not available in block diagrams or static fault trees), as well as block-wise construction and remote value propagation as in reliability block diagrams (these concepts are not available in generalized stochastic Petri nets).

The encoding we propose here is based on some preprocessing and a one-to-one correspondence between dynamic fault tree constructs and their counterparts in terms of guarded transitions systems. In other words, we design a library of reusable modeling components, one per dynamic fault tree construct. The design of a dynamic fault tree model consists then simply in assembling these predefined components. Proceeding this way presents at least three important advantages compared to specific approaches. First, it clarifies the semantics of each and every construct. Second, it makes it easy to extend the library with new constructs. Third, all assessment tools designed for guarded transition systems are instantly applicable to dynamic fault trees. Regarding this last point, the key question is to determine whether assessment algorithms can take advantage of the specificity of dynamic fault tree constructs. We give arguments to show that this question should be studied in light of the models chosen for basic components and that the answer is probably negative.

The original contribution of this article is twofold. First, we show that guarded transitions systems provide a suitable framework to clarify the semantics of dynamic fault trees. Second, we relate this semantic and assessment algorithms at hand with models chosen for basic events.

The remainder of the article is organized as follows. Guarded transitions systems are introduced Section 2. The translation of dynamic fault tree constructs into guarded transition systems is studied Section 3. Finally, algebraic interpretations and assessments algorithms are discussed Section 4.

2. Guarded transitions systems

2.1. Definition

A Guarded Transitions System is a quadruple, $\langle V = S \uplus F, E, T, A \rangle$ where,

* Corresponding author.
E-mail address: Antoine.Rauzy@centralesupelec.fr (A. Rauzy).

http://dx.doi.org/10.1016/j.ress.2015.04.017
0951-8320/© 2015 Elsevier Ltd. All rights reserved.

- S and F are two disjoint (finite) sets of variables. S is the set of state variables, F the set of flow variables. Variables have a type (Boolean, Integer, an Enumeration of symbolic constants…) and a default/initial value.
- E is a (finite) set of events. Events are either immediate or stochastic.
- T is a set of transitions. Transitions are triple ⟨G, e, P⟩, where G is Boolean condition built over state and flow variables, e is an event, and P is an assignment of a state variables, i.e. a set of individual assignments of the form s:=K, where s is a state variable and K is an expression built over variables. G and P are called respectively the guard and the action of the transition. For the sake of clarity, the transition ⟨G, e, P⟩ is denoted e: G→P.
- A is an assertion, i.e. a set of assignments in the form f:=L, where f is a flow variable and L is an expression built over variables. A flow variable is assumed to appear only once as the left member of an equation.

A transition is fireable when its guard is satisfied by the current values of variables. The firing of a transition is a two steps process: first, the action is performed, i.e. the values of (some) state variables are changed; second, the values of flow variables are updated by means of the assertion. Immediate transitions take no time while timed (or stochastic) transitions are assumed to take some (possibly infinitely small) amount of time. The underlying model of time is similar to the one of generalized stochastic Petri nets [1] if we assume that delay of stochastic transitions are Markovian (i.e. obey negative exponential distributions).

The update of flow variables after each transition firing is performed thanks to a fixpoint mechanism [19], i.e. values of left members of equations are recalculated until the system stabilizes. This stabilization is obtained in at most two passes, i.e. it is linear in the size of the assertion in the worst case. It is atomic, i.e. it is assumed to take no time (as immediate transitions). The important point here is that this fixpoint mechanism makes it possible to model remote interactions between components.

To illustrate the above definitions, let us consider first a simple non-repairable component. The guarded transition system for this component is pictured Fig. 1. It is made of the following elements:

- A state variable s, which takes its value into the enumeration {working, failed}. The initial value of s (working) is indicated with a small entering arrow.
- A Boolean flow variable: out.
- A stochastic event: failure.
- A transition transitions:
- failure: s=working→s:=failed
- An assertion made of an unique assignment:
- out:= (s=failed)

Now consider a spare, non-repairable component in cold redundancy. The guarded transition system for that component is pictured Fig. 2. It is made of the following elements:

- Two state variables a and s, which take respectively their values into enumerations {standby, active} and {working, failed}.
- Two Boolean flow variables: demand and out.
- Four events: start, failureOnDemand, failure and dormantFailure. start and failureOnDemand are immediate (pictured as dashed arrows). Failure and dormantFailure are stochastic transitions (pictured as plain arrows).
- Four transitions:
- start: a=standby and demand→a:= active
- failureOnDemand: a=standby and s=working and demand→ a:= active, s:= failed
- failure: a=active and s=working→s:= failed
- dormantFailure: a=standby and s=working→s:= failed
- An assertion made of a unique assignment (“demand” is an input flow variable):
- out:=(s=failed)

This example is helpful to introduce a point we did not discussed so far. Three transitions leave the state “a=standby and s=working”. Two of them are immediate (“start” and “failureOnDemand”) and one is stochastic (“dormantFailure”). When the input flow “demand” gets true, these three transitions are in conflict. However, in the Markovian framework adopted in this article, the probability that the delay associated with the stochastic transition is null is zero. Therefore, immediate transitions have the priority. Still they are in conflict. The choice between “start” and “failureOnDemand” is non-deterministic. It is possible however to influence the probabilities with which transitions in conflict are fired by associating a weight (called expectation in AltaRica 3.0 jargon) with each of them. The probability that a particular transition is fired is then the weight of this transition divided by the sum of the weights of the transitions in conflict. By default, the weight of a transition is 1.

2.2. Composition

Guarded transition systems can be enclosed into blocks (as illustrated Figs. 1 and 2). Then blocks can be composed to create larger blocks, as illustrated Fig. 3 where the model for the simple component described Fig. 1 and the model for the spare component described Fig. 2 are composed with a block G representing an AND gate. The idea behind the encoding of dynamic fault trees into guarded transition systems is to design a library of generic blocks representing each type of basic events and gates and then to assemble instances of these blocks just as exemplified Fig. 3. This encoding provides a sound semantics for dynamic fault trees because the semantics of guarded transition systems is itself completely and formally defined [19]. Guarded transitions systems are actually richer than what we presented here. However, this presentation suffices for the purpose of the present article.

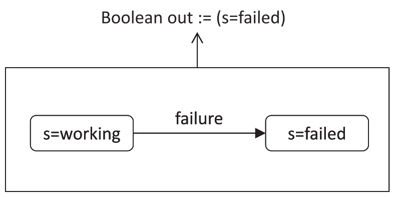

Fig. 1. The guarded transition system for a non-repairable component.

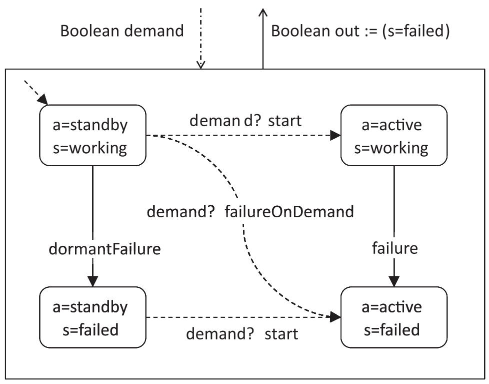

Fig. 2. The guarded transitions system for a spare non-repairable component.

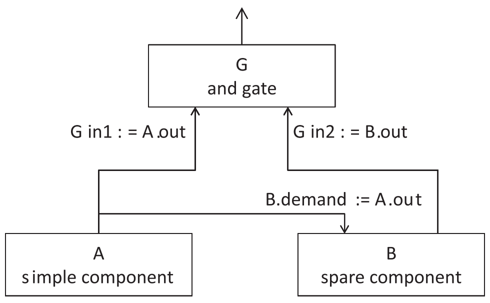

Fig. 3. Composition of three guarded transition systems.

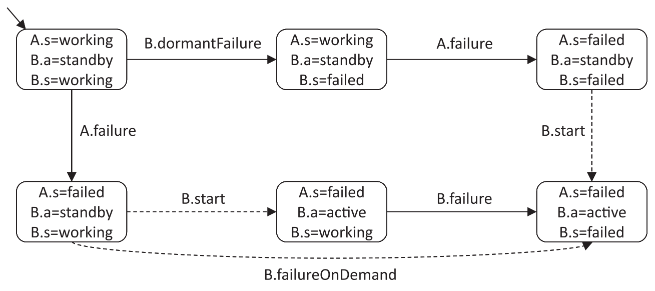

Fig. 4. Reachability graph for the guarded transition system pictured Fig. 3.

Any hierarchy of blocks is equivalent to a flat guarded transitions system. To obtain this guarded transitions system it suffices to prefix names of variables and events of each internal block by the name of the block, as illustrated on Fig. 3 in equations (assertion) linking flow variables of the different components. This transformation of a hierarchical model into a flat one is thus of linear complexity with respect to the size of the model. It is performed automatically by assessment tools prior to any calculation.

In our example, the flat guarded transition system is made of the following elements:

- Three state variables A.s, B.a and B.s, which take respectively their values into enumerations {working, failed}, {standby, active} and {working, failed}. The initial value of A.s and B.s is "working" while the initial value of B.a is "standby". The component G which represents a combinatorial gate has no internal state.
- Six Boolean flow variables: A.out, B.demand, B.out, G.in1, G.in2 and G.out.
- Four events: A.failure, B.start, B.failure and B.failureOnDemand. B.start and B.failureOnDemand are immediate. A.Failure and B. failure are stochastic.
- Five transitions:
- A.failure: A.s=working→A.s:= failed
- B.start: B.a=standby and B.demand→B.a:= active
- B.failureOnDemand: B.a=standby and B.s=working and B. demand→B.a:= active, B.s:= failed
- B.failure: B.a=active and B.s=working→B.s:= failed
- B.dormantFailure: B.a=standby and B.s=working→B. s:= failed
- An assertion made of six assignments:
- A.out:= (A.s=failed)
- B.out:= (B.s=failed)
- B.demand =A.out
- G.in1=A.out
- G.in2=B.out
- G.out=G.in1 and G.in2

There is a one-to-one correspondence between the mathematical formulation and the graphical representation of guarded transition systems. For this reason, we shall base our presentation on graphical representations, which are more intuitive. But it should be clear that the mathematical formulation in implicitly understood.

2.3. Reachability graph

A guarded transition system describes a reachability graph (more formally a Kripke structure) in an implicit way. The states of this graph are labeled with variable assignments and its transitions with events.

For instance, the reachability graph described by the guarded transition system pictured Fig. 3 is pictured Fig. 4. Flow variables are not represented on this figure for they depend functionally on state variables. Their value is calculated by means of the assertion.

The article [19] gives the precise structured operational semantics of guarded transition systems and their composition operations. The full exposition of this semantics goes beyond the scope of the present article. The interested reader can refer to the cited article.

The reachability graph is usually exponentially larger than the guarded transition system that describes it and gets very big even for medium size models. Therefore, assessment algorithms never built it in computer memory neither they explore it fully. This is reason why defining the semantics of dynamic fault trees in terms of guarded transition systems is the right way to proceed: it is both sound from a mathematical view point and efficient from an algorithmic viewpoint.

3. Encoding dynamic fault trees into guarded transition systems

In this section, we shall present the translation of dynamic fault tree constructs into guarded transition systems. We shall examine first basic events and combinatorial gates. Then we shall discuss the encoding of dynamic gates introduced by Dugan in her seminal article [14], i.e. PAND gates, FDEP gates, SEQ gates and Spare gates. Finally, we shall discuss the encoding of triggers of Boolean Driven Markov Processes [7].

3.1. Mechanisms at work

To start with, assume we want to encode a static fault tree into a guarded transition system. To do so, we need to design generic blocks for basic events and combinatorial gates (AND, OR,K-out-of-N…). Models for basic events will be typically as the one represented Fig. 1, possibly with a repair transition. Models for combinatorial gates will be typically similar to the AND gate implicitly described Fig. 3. Two fundamental mechanisms are at work in these models: first, changes of states (of basic components) under the occurrence of stochastic events (failures and repairs), and second bottom-up propagations of values starting from basic components and going to the top event through the intermediate gates. These two mechanisms are indeed still present in dynamic fault trees as well, but they are not sufficient to capture their semantics. As we shall see, immediate transitions are necessary to describe reconfigurations. Moreover, not only models for basic events have states, but also those for some of the dynamic gates. Most importantly, a third fundamental mechanism is at work: components can be activated and deactivated. Activation and deactivation demands are carried out top-down in the tree. This mechanism is illustrated Fig. 5 where activation flows are represented with dashed lines.

On this figure, the gate G has two parents gates P1 and P2 and two children gates (or basic components) C1 and C2. Then G is thus active when at least one of its parent gates is active. G

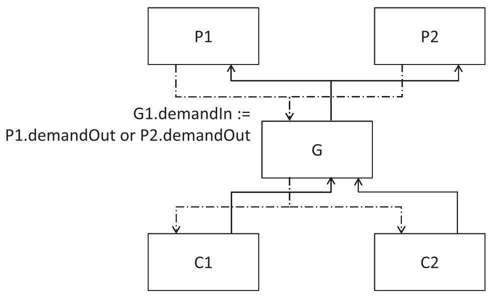

Fig. 5. Propagation of activation demands.

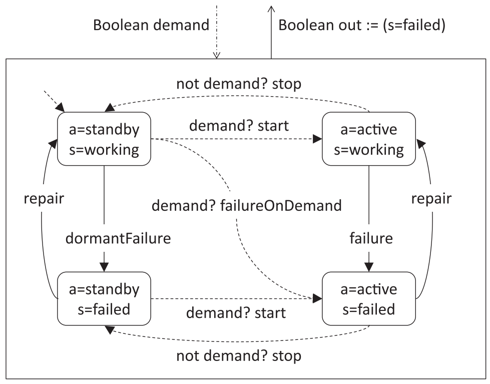

Fig. 6. Generic model for basic events.

propagates activation demands to its children gates. The propagation of activation demands is similar to the propagation of values excepted it is carried out top-down instead of bottom-up. A Boolean flow variable “demandIn” is thus associated with each block (encoding a basic component or a gate). Blocks encoding gates define one or more Boolean flow variables “demandOut”. The top blocks of a dynamic fault tree (there may have several) are always active, i.e. the assertion sets their input demand to “true”.

Each block describes, together with its descendants in the tree, a physical or a functional part of the system under study. The part is active if the demand input flow is true. It is in standby otherwise. The part is failed if the output value of the block is true and working otherwise. The output value of a block depends on the state of its descendants in the tree.

This additional mechanism presented, we can now consider each and every construct of dynamic fault trees and give them a sound semantics in terms of guarded transition systems.

3.2. *Basic events*

Several models can be associated with basic events. For instance, the model pictured Fig. 1 is perfectly acceptable, assuming that an input flow “demand” with no effect is be added for the sake of completeness. A generic model for basic events is pictured Fig. 6. This model is an extension of the one pictured Fig. 2. It encompasses failures, dormant failures, failures on demand, repairs, activations and deactivations of the component.

We shall discuss Section 4 the impacts of the choice of models for basic events on assessment algorithms.

3.3. *Combinatorial gates*

Combinatorial gates (AND, OR, K-out-of-N…) encoded by blocks with no internal state, event and transition, as illustrated by the encoding of an OR gate pictured Fig. 7. They propagate activation demands as explained in Section 3.1.

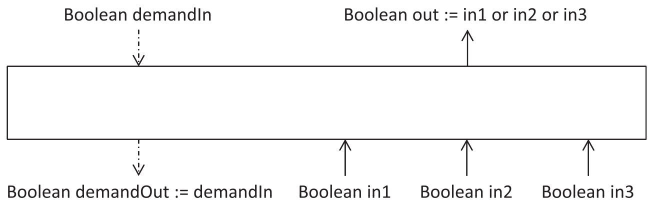
Boolean demandIn
Boolean out := in1 or in2 or in3
Boolean demandOut := demandIn
Boolean in1
Boolean in2
Boolean in3

Fig. 7. Guarded transition system encoding an OR gate.

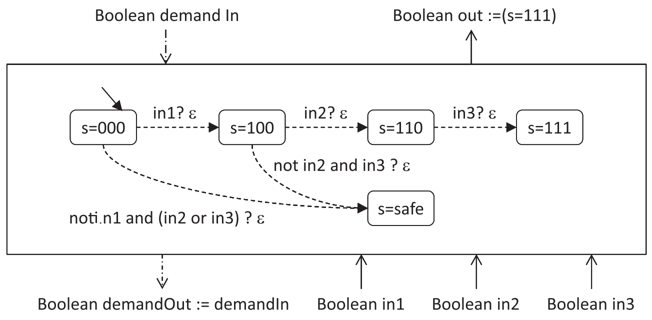

Fig. 8. Guarded Transitions System for a PAND Gate.

There is an interesting point here. Consider an OR gate G with two child gates A and B. If, for some reasons, the output value of A gets true, then the output value of the gate will be true, whatever the output value of B. In other words, events occurring in B do not influence the output value of G (except for repair events of components that are shared with A). It may worth, for the sake of the efficiency of assessment algorithms, to deactivate these components, so not to pollute the calculations with the firing of inconsequential events. This can be easily done by modifying the assertion as follows.

$$\text{demandOut} := \text{demandIn and not out}.$$

Such a deactivation is possible because the output values of basic components and gates depend only on the internal states.

3.4. *Priority AND gates*

The output value of a Priority AND Gate (PAND) over blocks A, B, C… is true if and only if the output values of A, B, C… become true in this order. The guarded transitions system associated with this gate should thus memorize the order in which its fanins become true. The guarded transitions system for a PAND gate with three child blocks is pictured Fig. 8. All transitions are immediate. They are guarded by the output values of its child blocks. They are labeled with the same internal and silent event “ε”. This model can be easily extended to any number of inputs.

The model pictured on Fig. 8 raise immediately at least two issues that are not discussed in the literature (but partly in Merle's work), because authors assume that PAND gates take only basic events as inputs (there is however no significant reason to limit ourselves this way):

– First, what does happen if both in1 and in2 (or in3) gets simultaneously true in state s = 000? The model presented here assumes that this situation does prevent the failure of the PAND gate. But it is acceptable to make the opposite choice, and even to let the case undetermined (as explained Section 2.1)
– Second, what does happen if in1 cease to be true in state s = 100? Does the gate go back to state s = 000 or does it stay in state s = 100 (as in our model)? Here again, there is a priori no

reason to make a choice rather than another. The same question applies indeed to inputs in1 and in2 in state s = 110 and the three inputs together in state s = 111.

In both cases, it possible and very easy to modify the guarded transition model so to obtain the chosen semantics. However, the best is probably to introduce different PAND gates to the library so to let the analyst choose by himself which gate is suitable for its own purpose. This is one of the big advantages of encoding dynamic fault trees as a library of guarded transition systems.

3.5. Functional dependencies

A Functional Dependency (FDEP) asserts a dependency between a triggering event T and several triggered events $E_1,\ldots,E_k$: the failure of T causes the immediate and simultaneous failure of E_i's. FDEP gates are special in the sense that they have no output. It is assumed in the literature that the E_i's are basic events. It is of course of interest to release this assumption. To do so, it would be possible to encode the triggering process by introducing new flows and events to our base model or to use synchronization, another feature of guarded transition systems (not presented in this article). However, this would make our semantics notably more complex and difficult to master. As noted by Stamatelatos and Vesely in [24], the Ei's can occur either by themselves or because they have been triggered by T, no matter the order in which these actions occur. Their idea, which we follow here, is therefore to introduce a preprocessing phase, in which an OR gate GEi is substituted for each Ei, i.e. all parent gates of Ei are made pointing to GEi instead. Inputs of GEi are Ei on the one hand and T on the other hand. This process is illustrated Fig. 9 for two child gates A and B.

3.6. Sequence enforcer

A Sequence Enforcer (SEQ) asserts that events can occur only in a given order. This gate is special in the sense that it has no output. Moreover, it is assumed in the literature that it applies only to basic events, a condition that we would like to release, as previously. There are actually two ways to consider the problem. Let a sequence enforcer with child gates A, B, and C as illustrated Fig. 10. The first idea consists in considering only the effect of the realization of A, B and C but to let output values of get true in any possible order: the output value of B has an effect only if it gets true after the output value of A, the output value of C has an effect only if it gets true after the output value of B and so on if there are more child gates. This idea can be implemented by means of a preprocessing, as for FDEP gates. A PAND gate GE_i is substituted for each E_i. GE_i takes GE_{i-1} and E_i as inputs, in this order. This process is illustrated Fig. 10.

The second idea consists in using the activation mechanism: the gate B is activated only if the gate A is activated and its output value, the gate C is activated only if the gate B is activated and its output value is true and so on if there are more child gates To implement this idea, we need to proceed in two steps.

The first step consists in introducing a trigger block whose role is to activate their child when a given condition is satisfied. The semantics of triggers is given Fig. 11.

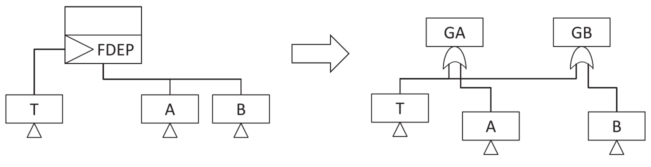

Fig. 9. Preprocessing of FDEP gates.

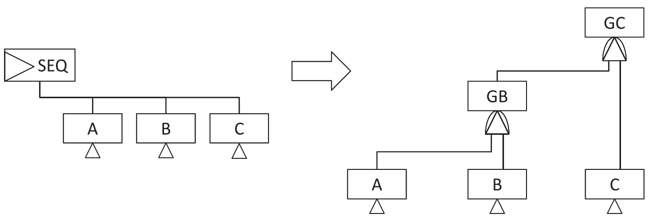

Fig. 10. Preprocessing of a Sequence Enforcer using PAND gates.

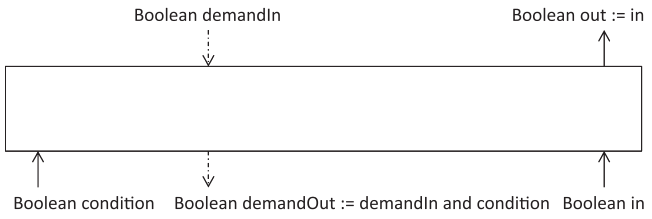

Fig. 11. The guarded transition system for a trigger.

The second step consists in substituting a trigger for each child gate and to plug, thanks to the assertion, this trigger onto the previous child gate and the child gate, as illustrated (Fig. 12).

If the output values of child gates cannot get true while the child gate is in standby, this encoding ensures that they always get true in order.

3.7. Spare Gates

In reference [12] Cold, Warm, Hot and Spare Gates (CSP, WSP, HSP) are defined as follows. "When the primary input fails, available spare inputs are used in order until none is left, at which time the gate fails. Spare inputs can be shared among spare gates, in which case the first spare gate to utilize the spare input makes it inaccessible to the other spare gates. The "temperature" of a spare gate indicates whether unused spare components cannot fail (cold), fail at a rate attenuated by the dormancy factor of the spare (warm), or fail at their full rates (hot)."

Two mechanisms are therefore at work in spare gates: first, an activation mechanism, which is the same as we saw previously. The spare part is activated when the primary part fails. Second, a resource sharing mechanism: when the spare part is activated to replace the primary part, it is locked so to prevent any other part to use it. This mechanism is to some extent independent from the notion of spare gate and deserves an extra-logical construct on its own. We call this extra-logical construct a switch for it is actually its role. Its semantics for two parts sharing a spare part is pictured Fig. 13. The extension to more sharing parts and more spare parts is easy. The idea is then to apply a preprocessing dedicated to spare gates. If a spare part is referenced by only one spare gate, then a trigger can be substituted for the spare gate (as we did for sequence enforcers). Otherwise, a switch is substituted the different spare gate referencing the same spare part, as illustrated (Fig. 14).

3.8. Boolean driven Markov processes

Boolean Driven Markov Processes have been introduced by Bon and Bouissou in Reference [7]. They do not introduce dynamic gates but a unique construct, called trigger as illustrated Fig. 15. A trigger is an arrow (represented with a dashed line on the figure). When the gate at its source (G) fails, then the gate at its end (H) is awaken. Our notion of trigger is almost the same as the one of Boolean Driven Markov Processes, except that we make it a dynamic gate. To get a guarded transition system from the Boolean

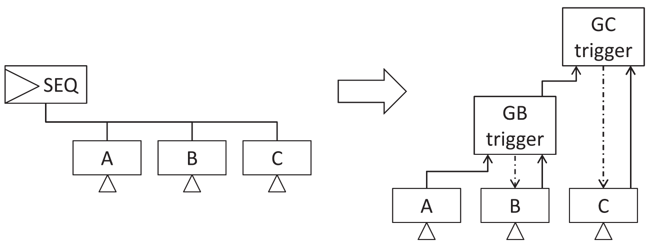

Fig. 12. Preprocessing of a Sequence Enforcer using triggers.

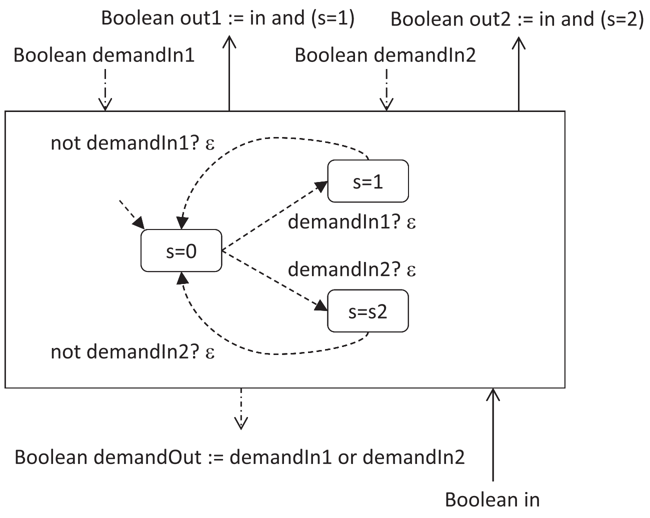

Fig. 13. Guarded transition system for a switch.

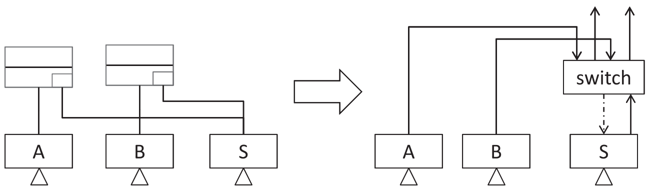

Fig. 14. Preprocessing of spare gates.

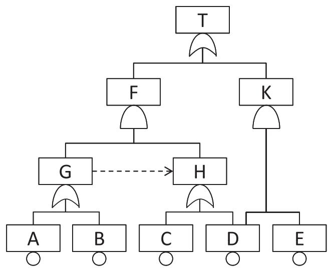

Fig. 15. A Boolean Driven Markov Process.

Driven Markov Process pictured Fig. 15 it suffices to introduce a trigger gate on top of the H gate and to plug the output value of the gate G as the input condition of this trigger gate.

Note that the basic component D is active even if the gate H is not, because it is activated through the gate K. Note also that if the basic component A fails, then G and more importantly B remain active. However, B cannot contribute anymore to the realization of the top event because the output value of G is already true. Therefore descendants of G can be deactivated as explained Section 3.3.

3.9. Discussion

As pointed out by one of the referees, some combinations of dynamic gates may lead to a non-deterministic behavior. Consider for instance a functional dependency between the triggering event T and two triggered events A and B. Assume moreover that A and B are the two children of a PAND gates. In that case, depending on A is triggered first or B is triggered first, the output value of the PAND gate will be true or false. This problem exists independently of any interpretation of dynamic fault trees. The above model should probably be considered as incorrect. Guarded transition systems (and AltaRica 3.0 assessment tools) can however handle this problem in the same way the conflict “start” versus “failureOnDemand” is handled, as discussed Section 2.1, i.e. by accepting both orders, possibly weighting them differently.

4. Algebraic interpretation and assessment Algorithms

4.1. Generic algorithms

Several algorithms have been proposed to assess Dynamic Event Trees. Dugan et al., in their tool Galileo [15], isolate modules containing dynamic gates, compile these modules into Markov chains, calculate probability distributions for these processes and finally re-inject these distributions into the tree by substituting basic events for the dynamic modules. This approach is very pragmatic. The compilation of into a complete Markov chain is however possible only if modules are small enough. The Markov chain is actually usually exponentially larger than its implicit representation.

Bouissou et al. [7] developed an algorithm to generate (most probable) failure sequences from a Boolean Driven Markov Process. This algorithm can be applied to any implicit representation of Markov chains, including of course guarded transitions systems. It avoids, at least to some extent, the combinatorial explosion of the state space for it explores only a (hopefully) limited portion of it.

The generation of partial Markov chain as proposed in [9,10] is another approach to handle large Markov chains without generating them explicitly.

None of the above mentioned algorithms, neither of course Monte-Carlo simulation, takes advantage of the fact that the model is a dynamic fault tree rather than a general guarded transition system. The author strongly believe that no such advantage can be obtained unless the formalism is further constrained, typically by considering specific models for basic components.

Two such restrictions seem worth to consider: first, the case where basic components are always active and non-repairable. Second, the case where basic components are non-repairable, but may be initially in standby. In the next two sections we shall discuss these two restrictions in turn.

4.2. Active, non-repairable basic components

Let is consider first the case where all components are active and non-repairable, i.e. correspond to the model pictured Fig. 1. In this framework, dynamic fault trees describe sequences of events $E1{\ast}E2{\ast}\ldots{\ast}Ek$ such that $Ei \neq Ej$ for $i \neq j$ (there is no repeated event). Its algebraic foundations have been clarified by Lesage & al. [17,16]. The sequence operator $\ast$ lies somehow between the AND of the Boolean algebra (which is commutative, associative and idempotent) and the concatenation operator of the free monoid (which is just associative). This framework is indeed quite restrictive: sequence enforcers, spare gates and triggers can only be translated into PAND gate.

In reference [17], Lesage et al. introduced, aside logic operators, a comparator of dates of occurrence of events '$A < B$'. They propose

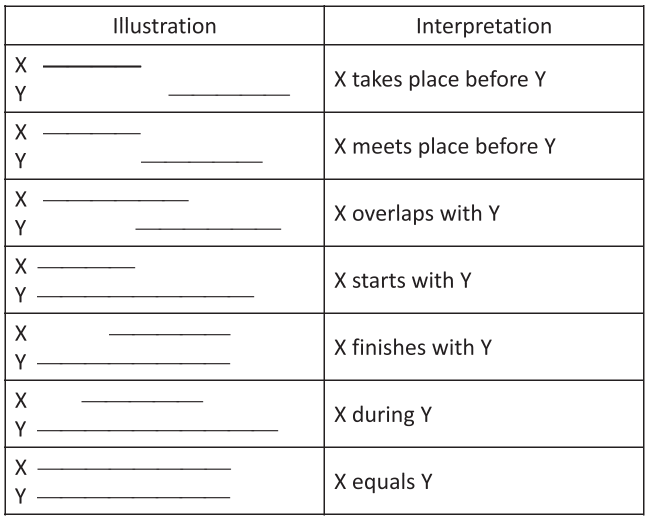

Illustration	Interpretation
X Y	X takes place before Y
X Y	X meets place before Y
X Y	X overlaps with Y
X Y	X starts with Y
X Y	X finishes with Y
X Y	X during Y
X Y	X equals Y

Fig. 16. Relations of the Allen algebra.

an algorithm to rewrite dynamic fault trees into a normal form which consists in minimal cutsets with an additional part (using the comparator) to put constraints on order of events.

In reference [23], it is proposed to generate these sequences by means of symbolic operations themselves implemented thanks to Sequence Decision Diagrams, a data structure derived from Minato's Zero-Suppressed Binary Decision Diagrams [18].

Both approaches remain to test on large real life models. The interest of the framework we considered in this section is questionable: compare to static fault trees, it makes it possible to put constraints on the order of events. However, this increment in the expressive power seems to come with a very price in terms of cost of calculations.

4.3. Non-repairable components

We shall now consider the case where some of the components are in standby at time 0 and are awaken when some other part of the system fails. This framework extends the one of the previous section. Here events can be seen as tasks with a beginning (the activation of the component) and an end (the failure of the component). Sequences are made of beginning and end of tasks with two constraints: first, the beginning of a task occurs before its end; second a task begins at most once in a sequence. Allen's algebra [2] provides such an algebraic framework, which has been extensively studied in the Artificial Intelligence literature. In his seminal article, Allen pointed out that two tasks can be positioned relatively one another in thirteen different ways, therefore introducing as many constraints (or operators), as illustrated Fig. 16 (on the figure, inverse relations such as “Y takes place before X” are not mentioned).

In order to be used to interpret dynamic fault frees, operators of the Allen algebra have to be lifted up to apply to sequences of tasks, and not only to individual tasks. In our article [23], we started this generalization for the case where all the tasks start at time 0, i.e. for the algebraic framework we considered in the previous section.

It is worth noticing that Allen algebra may also help us to imagine new dynamic gates of interest for some practical purposes. Also, Lesage et al. algorithm to rewrite dynamic fault trees into a normal form can probably be extended to handle Allen's relations. Similarly, Sequence decision diagrams can probably be used to encode and to manipulate Allen algebra expressions. Such developments go beyond the scope of the present article, but are certainly worth to pursue.

4.4. Discussion

The introduction of repairable components requires only minor changes to the models described in the previous section. However, if we look at them from an algorithmic perspective, things change dramatically. With repairable components, sequences may go through behavioral loops. Such loops cannot be discarded by minimality arguments, for probabilistic reasons. Consider for instance a system made of unique component whose mean time to failure is 1000 h, whose repair is (almost) instantaneous, for a mission time of 10,000 h. Then, the sequence “failure, repair, failure, repair, failure” has a much higher probability than the sequence made of a single “failure”. Discarding the former because it is not minimal would therefore be a big mistake. For this reason, the reduction to a normal form, or the use of Binary Decision Diagrams like data structure, is certainly not possible if repairable components have to be considered.

5. Conclusion

In this article, we gave a sound semantics for Dynamic Fault Trees and Boolean Driven Markov Processes. To do so, we translate their logical and extra-logical constructs into Guarded Transitions Systems. This translation is efficient: each construct is translated into a small guarded transitions system and the resulting blocks are then assembled at no cost.

This study shows that guarded transitions systems provide a suitable framework to unify Dynamic Fault Trees and Boolean Driven Markov Processes, to make their semantics precise and to discuss assessment algorithms. One of the most remarkable points is that we made no Markovian hypothesis and more generally no assumption about probability distributions associated with events. As a consequence, models for basic components can be changed without changing the semantics of gates. The only constraint to respect stands in the interface, i.e. to have the demand as input and an indicator of failure as output. It could be possible for instance to introduce dependencies amongst basic components (e.g. to model common cause failures, limited number of repair crews…), without perturbing the superstructure of the tree.

The translation of spare gates required introducing the notion of switch. This notion seems to add a real expressive to the formalism, namely the ability to model that a given part is the exclusive user of a spare part. There is certainly more to discover about this notion.

Another interesting perspective is the use of Allen's relation in conjunction with Dynamic Fault Trees, both from modeling and assessment algorithm viewpoints.

References

[1] AjmoneMarsan M, Balbo G, Conte G, Donatelli S, Franceschinis G. Modeling with generalized stochastic petri nets. Wiley Series in Parallel Computing. John Wiley and Sons; 1994.
[2] Allen JF.. Maintaining knowledge about temporal intervals. In: Communications of the ACM. 26/11/1983. ACM Press. p. 832–843, issn:0001-0782.
[3] Bobbio Andrea, Codetta Raiteri Daniele. Parametric fault trees with dynamic gates and repair boxes. In: Proceedings of the annual reliability and maintainability symposium (RAMS 2004). Los Angeles, CA, USA; 2004. p. 459–65.
[4] Boudali H, Crouzen P, Stoelinga M.. Dynamic fault tree analysis through input/output interactive Markov chains. In: Proceedings of international conference on dependable systems and networks (DSN 2007), IEEE Computer Society; 2007. p. 25–38.
[5] Boudali H, Crouzen P, Stoelinga M. A compositional semantics for dynamic fault trees in terms of interactive Markov chains. In: Proceedings of international symposium on automated technology for verification and analysis (ATVA'07). Springer Verlag – LNCS vol no. 4762; 2007. p. 441–56.
[7] Bouissou Marc, Bon Jean-Louis. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic-driven Markov processes. Reliab Eng Syst Saf 2003;82:149–63.

[8] Marc Bouissou. A generalization of dynamic fault trees through boolean logic driven markov processes (BDMP). In: Terje Aven, Jan Erik Vinnem editors. Proceedings of European safety and reliability conference, ESREL'2007. Taylor and Francis Ltd, Stavanger, Norway; June, 2007, 10: 0415447860 13: 978-0415447867.

[9] Brameret Pierre-Antoine, Rauzy Antoine, Roussel Jean-Marc. Preliminary system safety analysis with limited markov chain generation. In: Proceedings of 4th IFAC workshop on dependable control of discrete systems, DCDS 2013, York (Great Britain); 2013. p. 13–8. 978-3-902823-49-6, 1474-6670.

[10] Brameret Pierre-Antoine, Rauzy Antoine, Roussel Jean-Marc. Automated generation of partial Markov chain from high level descriptions. Reliab Eng Syst Saf 2015;139:179–87.

[11] Codetta Raiteri Daniele. The conversion of dynamic fault trees to stochastic petri nets, as a case of graph transformation. Electron Notes Theor Comput Sci 2005;127(2):45–60.

[12] Coppit David, Sullivan Kevin J., Bechta Dugan Joan. Formal Semantics for computational engineering: a case study on dynamic fault trees. In: Proceedings of 11th international symposium on software reliability engineering (ISSRE'00); 2000. p. 270.

[13] David Coppit, Sullivan Kevin J. Sound methods and effective tools for engineering modeling and analysis. In: Proceedings of 25th conference on software engineering ICSE'03, IEEE Computer Society, Portland, OR, USA; 2003. p. 198–07.

[14] Bechta Dugan Joanne, Bavuso Salvatore J, Boyd Marks A. Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans Reliab 1992;41 (3):363–77.

[15] Joshua Dehlinger, Dugan Joanne Bechta. Dynamic event/fault tree analysis of multi-agent systems using Galileo. In: Proceedings Workshop on Integration of Software Engineering and Agent Technologies, 2008: 429–434.

[16] Merle Guillaume. Algebraic modeling of Dynamic Fault Trees, contribution to qualitative and quantitative analysis. PhD thesis. LURPA, ENS de Cachan; 2010. p. 218.

[17] Merle Guillaume, Roussel Jean-Marc, Lesage Jean-Jacques, Bobbio Andrea. Probabilistic algebraic analysis of fault trees with priority gates and repeated events. IEEE Trans Reliab 2010;59(1):250–61.

[18] Minato S.. Zero-suppressed BDDs for set manipulation in combinatorial problems. In: Proceedings of the 30th ACM/IEEE design automation conference, DAC'93; 1993. p. 272–7.

[19] Rauzy Antoine. Guarded transitions systems: a new states/events formalism for reliability studies. J Risk Reliab Professional Eng Publishing 2008;222 (4):495–505.

[20] Prosvirnova Tatiana, Batteux Michel, Brameret Pierre-Antoine, Cherfi Abraham, Friedlhuber Thomas, Roussel Jean-Marc et al. The AltaRica 3.0 project for model-based safety assessment. In: Proceedings of 4th IFAC workshop on dependable control of discrete systems, DCDS'2013. International Federation of Automatic Control, York, Great Britain; September 2013. p. 127–32. 978-3-902823-49-6, 1474-6670.

[23] Rauzy Antoine. Dynamic fault trees, sequence algebra and sequence diagrams. Reliab Eng Syst Saf 2011;96(7):785–92.

[24] Vesely W, Stamatelatos M, Dugan J, Fragola J, Minarick III J, Railsback J. Fault trees handbook with aerospace applications. NASA Office of Safety and Mission Assurance Version 1.1; 2002. p. 1–205.