Scalable Analysis of Fault Trees with Dynamic Features

Jan Krčál
Saarland University – Computer Science,
Saarbrücken, Germany
krcal@cs.uni-saarland.de

Pavel Krčál
Lloyd’s Register Consulting, Stockholm, Sweden
Uppsala University, Uppsala, Sweden
pavelk@it.uu.se

Abstract—Fault trees constitute one of the essential formalisms for static safety analysis of various industrial systems. Dynamic fault trees (DFT) enrich the formalism by time-dependent behavior, e.g., repairs or functional dependencies. Analysis of DFT is so far limited to substantially smaller models than those required for, e.g., nuclear power plants. We propose a fault tree formalism that combines both static and dynamic features, called SD fault trees. It gives the user the freedom to express each equipment failure either statically, without modeling temporal information, or dynamically, allowing repairs and other timed interdependencies. We introduce an analysis algorithm for an important subclass of SD fault trees. The algorithm (1) scales similarly to static algorithms and (2) allows for a more realistic analysis compared to static algorithms as it takes into account temporal interdependencies. Finally, we demonstrate the applicability of the method by an experimental evaluation on fault trees of nuclear power plants.

I. INTRODUCTION

Static fault trees (SFT) is a formalism for inductive failure modeling which underlies many safety assessments in a wide variety of areas ranging from automotive and aerospace [23] over chemical and process industry [11] to nuclear power plants [29], [17]. The modeling procedure considers the failure of a complete system and iteratively decomposes it into failures of subcomponents until reaching individual equipment, atomic external events, operator errors, etc., called basic events. Their logical dependencies are captured by a structure of logical gates. The granularity of basic failures reaches the level on which it is possible to estimate failure frequencies by statistical methods from operation history or simulations or even by engineering computations.

A strong point of static fault trees is their simplicity allowing efficient solution techniques even for very large models. For example, this formalism is an essential part of safety analyses in nuclear power plants [29], [17]. Each nuclear power plant has to maintain, update and refine a fault tree model consisting of thousands of basic events and tens of thousands of logical gates. A characterization of all fault combinations leading to the complete failure of a static fault tree exists in the form of minimal fault combinations — minimal cutsets. Even though the number of minimal cutsets can be exponential in the amount of basic events, the important ones can be found on such models within only a few hours.

Acceptable analysis time for models of this magnitude comes for the price of disregarding progress of the accident as a succession of state changes in real time. In other words, one can only specify order of failures, recovery of safety functions, or standby mode for currently unused equipment in a very simplified way. Especially, for analyses with longer mission time, such as probabilistic Level 2 (and Level 3) studies in nuclear power plants, abstracting away from the accident progression may lead to an unrealistic conservatism.

As an example, consider a scenario of reaching a cooled and stable state in a situation with the core melt arrested in the reactor pressure vessel. One of the safety functions in the scenario is a cooling system with several redundant pumps and diesel generators. Probability of each individual pump failing in operation grows with growing mission time. However, we do not need each pump to operate for the whole time. After a failure of one pump, another one is started, having the mission time shorter by the time for which other pumps successfully operated. Also, the chance to repair a failed pump or diesel generator grows with growing mission time and should be considered. Unfortunately, a static analysis of a nuclear power station generally ignores this timed dependency and models the pumps and generators as if they were working all the time and no repairs were possible. Therefore, static analysis typically leads to overly conservative results for longer mission times while such longer scenarios receive a lot of attention after the Fukushima accident.

There are numerous dynamic extensions of the fault tree formalisms, such as dynamic fault trees (DFT) [4], [5] or Boolean Driven Markov Processes [8] (BDMP), which allow for modeling of state changes during an accident. Analysis methods essentially translate these models to (continuous-time) Markov chains and solve them by efficient methods available in a number of tools (for instance PRISM [19]). There are also highly sophisticated algorithms avoiding construction of the complete Markov chain by a compositional analysis [7] or other pruning methods [8] that scale to models with a several hundred of basic events and gates. This is still remarkable as the complete Markov chain for 100 basic events has at least $2^{100}$ states. Unfortunately, the models from nuclear probabilistic safety studies induce Markov chains of sizes like $2^{2500}$ that are way beyond the reach of available methods for dynamic analysis.

In this paper, we choose a modest approach to dynamic modeling and analysis in safety studies – we extend the formalism to allow for modeling of some dynamic features while still being able to utilize efficient solving techniques for static fault trees. The new features can capture (1) sequential application of elementary safety functions and (2) repairs...
of failed components. These features address well the needs of modeling highly dependable systems (such as emergency cooling) with multiple redundant trains which (1) are not needed at the same time and where (2) repair rates are several orders of magnitude higher than failure rates.

In our formalism, we allow basic events to be dynamic. These are then modeled using Markov chains expressing how components degrade over time and also possibly get repaired. Additionally, inspired by the Boolean Driven Markov Processes [8], we model all timing dependencies by a simple concept of triggers: whenever a safety function (a gate of the tree) fails, it may activate other safety functions modeled by dynamic basic events. Finally, we allow static basic events to appear along dynamic ones in a single fault tree. We thus call our formalism SD fault trees. The reason for allowing also static basic events in the formalism is threefold. This enables us to use standard static modeling and faster analysis for components that (a) need not be modeled dynamically (where timing is not crucial), (b) cannot be faithfully modeled dynamically (due to model restrictions), or (c) simply are not modeled dynamically (in legacy safety studies).

The proposed analysis procedure first finds minimal cut-sets of an SD fault tree using standard fault tree solving methods. This allows utilizing all optimizations and heuristics implemented in state-of-the-art fault tree solvers. Then we evaluate the impact of dynamic timing on each minimal cut set by using Markov chain analysis. We formulate two simple syntactic conditions on triggers that keep the complexity of such dynamic evaluation in reasonable bounds.

When the first condition on triggers (static branching) is met, the dynamic evaluation is very efficient as it requires to consider only dynamic basic events from the minimal cut set. When only the second condition on triggers (static joins) is met, the dynamic evaluation needs to consider also all dynamic basic events that may trigger any event from the minimal cut set. When none of these conditions is met, the dynamic evaluation needs to consider all basic events that may trigger any event from the minimal cut set. In the second and especially in the third case, efficiency of dynamic evaluation depends on the structure of the concrete model (and can be predicted and indicated to the user).

**Contribution of the paper:** The main theoretical contribution is identifying these restrictions on the triggering structure so that a small Markov chain model for each minimal cut set is sufficient for a precise time-aware evaluation.

The practical contribution is twofold. First, our analysis has feasible time and memory requirements even on very large SD fault trees as it decomposes the analysis of the global state space (e.g., $2^{2500}$ states) into analyses of small Markov chains for individual failure scenarios (e.g., less than 100,000 Markov chains where each has mostly less than 100 states). Second, our formalism still allows for modeling of dynamic features relevant for nuclear safety studies. We demonstrate both these aspects by experiments on an example model as well as real life fault tree models of nuclear power plants.

**II. STATIC FAULT TREES**

This section presents the formalism of static fault trees upon which we build in the rest of the paper.

A (coherent) static fault tree FT is a finite DAG $(V, E)$ where its leaves, denoted by $B$, are called basic events, its inner nodes, denoted by $G$, are called gates, a distinguished root node $g_{top}$ is called the top gate, and direct successors of a gate are called its inputs. Furthermore, each gate is either of type AND or of type OR, and for each basic event $a$ we specify its probability of failing $p(a) \in [0, 1]$.

The top gate models a failure of the complete system. The logical structure under the top gate covers all ways in which this system might fail that are relevant for the analysis. Proceeding downward through the tree, we move from a failure of a larger part of the system to failures of its subparts. Finally, basic events model atomic failures such as failures of individual pieces of equipment, external events, physical phenomena, or operator errors. A modeler has to supply failure data for basic events in order to run a probabilistic analysis of a static fault tree.

A **scenario** $\Xi$ is a set of basic events that represents the situation where all basic events from $\Xi$ fail and the remaining basic events stay functional. These scenarios can be inductively extended to gates. We say that a gate $g$ is **failed** by a scenario $\Xi$ if and only if the type of the gate $g$ is

- AND and all inputs of this gate are failed by $\Xi$, or
- OR and at least one input of this gate is failed by $\Xi$.

Furthermore, we say that $\Xi$ is a failure scenario if it fails the top gate $g_{top}$. The semantics of a static fault tree is given by its probability of failure:

$$p(FT) = \sum_{\Xi \text{ failure scenario}} p(\Xi).$$

where $p(\Xi)$ is the probability of the scenario $\Xi$ defined by

$$p(\Xi) = \prod_{a \in \Xi} p(a) \cdot \prod_{a \in B \setminus \Xi} (1 - p(a)).$$

**Example 1.** As a running example we consider a simple (toy) model that captures possible failure scenarios of an emergency cooling system. The system consists of a water tank and two redundant pumps. One functional pump is sufficient to cool the reactor; the second pump is used only when the first one fails. The model encodes combinations of failures of basic equipment which lead to a complete failure.

Basic events are depicted by circles and gates are represented by the usual symbols:

![Static Fault Tree Example](image)
Let us assume that the probability of each pump failing to start is \( p(a) = p(c) = 3 \times 10^{-3} \), the probability of each pump failing in operation is \( p(b) = p(d) = 1 \times 10^{-3} \), and the probability of the water tank to fail is \( p(e) = 3 \times 10^{-6} \). The probability \( p(\Xi) \) of the failure scenario \( \Xi = \{a, d\} \) (i.e., pump 1 fails to start, pump 2 fails in operation and no other equipment fails) is 
\[
p(a) \cdot p(d) \cdot (1 - p(b)) \cdot (1 - p(c)) \cdot (1 - p(e)) \approx 2.988 \times 10^{-6}
\]

III. STATIC AND DYNAMIC FAULT TREES

A failure scenario in a static fault tree does not distinguish an order in which basic events occur, not mentioning timing of these occurrences. If a basic event is characterized only by a probability of its occurrence then there is indeed no timing information present in the model. Therefore, we allow a timing-aware characterization of basic events in SD fault trees, modeled by triggered continuous-time Markov chains. This will allow timing-aware scenarios of failures that occur within a fixed time horizon.

A. Triggered Continuous-time Markov Chains

Dynamic basic events are usually modeled using continuous-time Markov chains (CTMC). A CTMC over a finite state space \( S \) is specified by an initial distribution \( \nu \) over \( S \), a rate matrix \( R : S \times S \rightarrow \mathbb{R}_{\geq 0} \), and a set of failed states \( F \subseteq S \). It can be viewed as a (finite) transition system where each transition is decorated by a rate \( \lambda \in \mathbb{R}_{\geq 0} \). Each transition corresponds to some event (such as a failure or a repair of a machine) occurring randomly in continuous-time according to an exponential distribution; the rate expresses how many such events are expected to occur per unit time.

**Example 2.** We first illustrate CTMC and triggered CTMC on dynamic models of failure in operation of the pumps from Example 1. On the left hand side, there is a simple model of the first pump failing in operation. We specify its failure rate as 0.001, i.e., once per 1000 hours of operation and its repair rate as 0.05, i.e., once per 20 hours.

![Example 2 Diagram]

Since the second pump is actually a spare pump, it does not operate from the beginning of the emergency situation. Hence, we define the pump to start in the state off and stay there until the pump gets triggered (here, we neglect the possibility that the pump fails while it is not in operation). When triggered, the pump takes the dashed edge into the state on where it behaves like the first pump. When the first pump gets repaired, the second pump is not needed any more and gets untriggered – a dashed edge is taken again (here, we assume that a failed pump is being repaired even if it is not required at the moment).

Formally, a triggered continuous-time Markov chain (triggered CTMC) is a CTMC over a finite state space \( S \) partitioned into states \( S^{\text{off}} \) where the equipment is switched off and states \( S^{\text{on}} \) where the equipment is switched on such that the device is failed only if it is switched on, \( F \subseteq S^{\text{on}} \), and the initial distribution supports only states from \( S^{\text{off}} \). Furthermore, we specify a total triggering function on \( : S^{\text{off}} \to S^{\text{on}} \) and a total untriggering function \( : S^{\text{on}} \to S^{\text{off}} \) describing how these modes are switched.

B. SD Fault Trees

In the following, we incorporate dynamic basic events into a fault tree where all other basic events are static. Our way is inspired by the (purely dynamic) formalism of Boolean Driven Markov Processes (BDMP) [8].

Analogously to SFT, a static and dynamic (SD) fault tree is a finite DAG \((V, E)\) where its leaves are partitioned into sets \( B_s \), called static basic events, and \( B_d \), called dynamic basic events, and its inner nodes \( G \) are called gates. Like before, each gate is either of type AND or of type OR and there is a distinguished root node \( g_{\text{top}} \). Each static basic event \( a \) is specified by its probability of failing \( p(a) \) and each dynamic basic event \( a \) is specified by a CTMC \( T(a) \). A failure of any gate \( g \) may trigger one or more dynamic basic events given in the set \( \text{trig}(g) \); we say that each event \( a \in \text{trig}(g) \) is triggered by \( g \). If a dynamic basic event \( a \) is triggered then \( T(a) \) has to be a triggered CTMC.

Without loss of generality, we assume that each dynamic basic event is triggered by at most one gate. Formally, for any pair of gates \( g \neq g' \) we have \( \text{trig}(g) \cap \text{trig}(g') = \emptyset \). We also require that there are no cyclic dependencies in the triggering structure. Formally, the fault tree graph with edges directed from root to leaves enriched by reversed trigger edges

\[
(V, E \cup \{(a, g) \mid g \in G, a \in B_d, a \in \text{trig}(g)\})
\]

is acyclic. Scenarios excluded by this requirement are exactly “deadlocks” situations where none from a group of several dynamic events can fail before all other have failed. Finally, for a CTMC \( T(a) \) of a dynamic basic event \( a \in B_d \), we use \( S_a \) to denote its set of states, \( R_a \) to denote its rate matrix, etc.

**Example 3.** We can refine the system from Example 1 using the SD fault tree formalism as follows. Dynamic basic events \( b \) and \( d \) are denoted by double circles and their triggered Markov chains are given in Example 2. Failure of the pump 1 triggers the dynamic event from the pump 2, depicted by the dashed edge.

![Example 3 Diagram]
C. Semantics

In case of SFT, failure probability of the whole tree was defined by means of failure scenarios. In case of an SD fault tree \( FT \), we define it by failure runs of a product Markov chain \( C_{FT} \) which captures the states of all basic events.

To this end, we express every static basic event using a Markov chain as well. We define \( T(a) \) for each static basic event \( a \in B_a \) to be a CTMC over states \( S_a = \{ \text{ok}, \text{fail} \} \) where only fail is a failed state. Furthermore, we set \( \nu_a(\text{fail}) = p(a) \) and \( \nu_a(\text{ok}) = 1 - p(a) \) and define \( R_a \) to be a zero matrix. Intuitively, a failure is determined by the random choice of the initial state as there are no transitions that would allow to change the state later. Although static basic events can be in this way easily expressed by dynamic basic events, the distinction between static and dynamic basic events is exploited heavily in our algorithm for analyzing SD fault trees.

1) Product Markov chain \( C_{FT} \): Each state of the Markov chain \( C_{FT} \) captures the current state of every basic event. The state space \( S \) of \( C_{FT} \) is a product of state spaces of Markov chains for all basic events, \( S = \prod_{a \in B} S_a \). A product state might fail a gate \( g \) in \( FT \) by failing the boolean structure under \( g \) as if \( FT \) was a static fault tree, i.e., discarding triggers and considering all basic events as static. Formally, we assign a scenario \( \Xi \) to a product state \( s \) by \( \Xi = \{ a \in B \mid s(a) \in F_a \} \). We say that a product state \( s \) fails a gate \( g \) if and only if the scenario \( \Xi \) fails this gate. We define the set of failed states \( F \) of the Markov chain \( C_{FT} \) as the set of all product states \( s \) which fail the top gate \( g_{top} \).

Example 4. Returning to the SD fault tree from Example 3, let us first list all the (triggered) CTMC for the basic events:

\[
\begin{align*}
T(a): & \quad T(b): & \quad T(c): & \quad T(d): & \quad T(e): \\
\begin{array}{cccc}
\text{ok} & \quad \text{ok} & \quad \text{ok} & \quad \text{ok} \\
1 - p(a) & 0.05 & 0.05 & 0.05 \\
0 & 0.001 & 0.001 & 0.001 \\
\text{fail} & \quad \text{fail} & \quad \text{fail} & \quad \text{fail} \\
\end{array}
\end{align*}
\]

The state space \( S \) is here \( S_1 \times S_2 \times S_3 \times S_4 \times S_5 \). The set \( F \) of failed states contains, e.g., the following three states:

- \((\text{ok, ok, ok, off, fail}) - \text{failure of the water tank,}\)
- \((\text{fail, ok, ok, fail, ok}) - \text{failure of both pumps, and}\)
- \((\text{fail, fail, fail, fail, fail}) - \text{a complete failure.}\)

The definition of the initial distribution \( \nu \) and the rate matrix \( R \) is a bit more involved and requires us to understand two concepts: (a) how the Markov chains of dynamic basic events evolve independently in parallel and (b) how dynamic basic events influence each other by triggers.

a) Parallel interleaving of Markov chains: The rates of transitions in each basic event (component of \( S \)) stay as before, independent of other components. Each transition of \( C_{FT} \) corresponds to a transition of a single basic event. Thus, we say that a state \( s \) evolves into a state \( s' \) with rate \( \lambda \) if

- the rate \( \lambda \) is the rate of this transition in \( T(a) \), i.e., \( R_a(s(a), s'(a)) = \lambda \).

b) Effect of triggers: Such evolution into a state \( s' \) may newly fail a gate triggering a basic event \( a \) which is currently switched off, i.e., \( s'(a) \in S^\text{off}_a \). In this case, the state \( s' \) has to be left immediately (similarly to a vanishing marking in a stochastic Petri net) by switching this basic event on. Dually, we immediately leave the states where some basic events have to be switched off because their triggering gates have got repaired. In both cases, this is achieved by taking the on/off transitions. Formally, \( s' \) is updated into \( s'' \) if for every dynamic basic event \( a \) triggered by some gate \( g \) we have

\[
s''(a) = \begin{cases} 
\text{on}(s'(a)) & \text{if } s' \text{ fails and } s''(a) \in S^\text{on}_a, \\
\text{off}(s'(a)) & \text{if } s' \text{ does not fail and } s''(a) \in S^\text{off}_a, \\
\text{otherwise.} & \end{cases}
\]

We say that a state \( s' \) is consistent if no basic event has to be switched on or off, i.e., \( s' \) fails a triggering gate of a basic event \( a \) if and only if \( a \) is switched on in \( s' \) (consistent states correspond to tangible markings in a stochastic Petri net).

Example 5. In our running example, we have, e.g.,

\[
s_1 = (\text{ok, ok, ok, off, fail}) \quad \text{updated} \quad (\text{ok, ok, ok, on, ok})
\]

\[
\text{evolves with rate } 0.001 \quad \text{evolves with rate } 0.05
\]

\[
(\text{ok, fail, ok, off, fail}) \quad \text{updated} \quad (\text{ok, fail, ok, on, ok}) = s_2
\]

\[
(\text{ok, fail, fail, fail, fail}) = s_3
\]

Note, that several updates might be necessary for reaching a consistent state. On the other hand, due to the acyclicity of the triggering edges, one always reaches a consistent state after finitely many updates. Transitions in the product Markov chain \( C_{FT} \) lead from consistent states to consistent states by an evolution step possibly followed by a sequence of updates. Each transition is labeled by the rate of the evolution step.

Formally, we define the rate matrix as follows. For a pair of consistent states \( s \neq \tilde{s} \), we define \( R(s, \tilde{s}) = \lambda \) if there is a state \( s' \) such that \( s \) evolves into \( s' \) with rate \( \lambda \) and \( s' \) is (iteratively) updated into \( \tilde{s} \). Otherwise, we define \( R(s, \tilde{s}) = 0 \). By this, we merge two steps, a probabilistic evolution of a component state followed by a sequence of deterministic updates due to triggers, into a single transition of the CTMC \( C_{FT} \).

Finally, the initial distribution of Markov chains for basic events extends to the product Markov chain \( C_{FT} \) in the following way. Intuitively, the system should start in any state \( s \) with the product probability \( \nu(s) := \prod_{a \in B} \nu_a(s(a)) \). However, we need to reflect updates of triggers also here. Thus, an inconsistent product state has the initial probability equal to 0. The initial probability \( \nu(\tilde{s}) \) of a consistent product state \( \tilde{s} \) is equal to its own probability \( \nu(\tilde{s}) \) summed up with probabilities \( \nu(s) \) of all inconsistent product states \( s \) which get (possibly iteratively) updated to \( \tilde{s} \).

\footnote{The definition is correct as there cannot be two different such states \( s'_1, s'_2 \) that would be (iteratively) updated into the same \( \tilde{s} \) (again due to acyclicity of triggering edges).}
Example 6. As regards states from Example 5, we have, \( R(s_1, s_2) = 0.001 \), \( R(s_2, s_1) = 0.05 \), and \( R(s_2, s_3) = 0.001 \). The chain is initiated, e.g., in \( s_1 \) with \( \nu(s_1) := \xi(s_1) = (1-p(a)) \cdot (1-(1-p(c)) \cdot (1-p(e))) \). However, for the state \( s_2 \) we have \( \nu(s_2) = \xi(s_2) + \xi(s) \) where \( s = (\text{ok}, \text{fail}, \text{ok}, \text{off}, \text{ok}) \).

2) The probability of failure: Behavior of a Markov chain is usually captured by a run – an infinite sequence of consistent product states interleaved by times spent waiting in individual states. By standard definitions (see, e.g., [21]), one obtains a probability measure \( \Pr \) over the set of all runs of \( FT \), also denoted \( \Pr_{FT} \) if the Markov chain is not clear from the context. The probability of failure of an SD fault tree \( FT \) within a fixed time horizon \( t \) is defined as

\[
\Pr(FT) := \Pr \left[ \text{Reach}^{\leq t}(F) \right]
\]

where \( \text{Reach}^{\leq t}(F) \) is the set of runs that visit a state from \( F \) before the time horizon \( t \).

IV. ANALYSIS OF STATIC FAULT TREES

As the analysis algorithm for SD fault trees is based on the algorithm for static fault trees, we briefly explain it in this section. Recall, that for a static fault tree, each failure scenario is a set that contains exactly the basic events that fail. In the following, we show how all failure scenarios can be characterized using a standard concept of minimal cutsets (MCS), how the probability of failure can be computed from MCSs and how all relevant MCSs can be found algorithmically.

A. Minimal cutsets (MCSs)

Each set \( C \subseteq B \) of basic events represents a whole class of scenarios where all events from \( C \) are failed; these scenarios differ only in failures of additional basic events outside of \( C \). Formally, we say that \( C \) represents \( \Xi \) if \( C \subseteq \Xi \). Furthermore, we say that \( C \) is a cutset if all represented scenarios are failure scenarios, i.e., the top gate is failed in scenarios described by \( C \). Note, that since there are no negated gates in our fault trees, if the scenario \( \Xi = C \) is a failure scenario then all scenarios represented by \( C \) are also failure scenarios. Finally, \( C \) is minimal if there is no smaller cutset \( C' \subseteq C \).

Example 7. In the SFT from Example 1, the set \( \{a, b, c\} \) is a cutset because if all these basic events are failed, then also the top gate (the whole cooling system) is failed. However, it is not a MCS because if we remove, e.g., the failure of \( b \), the resulting set \( \{a, c\} \) is still a cutset. The MCSs in this model are \( \{e\} \), \( \{a, c\} \), \( \{a, d\} \), \( \{b, c\} \), and \( \{b, d\} \).

Minimal cutsets have three crucial properties that make them appropriate for expressing the failure probability (we search for similar properties in the algorithm for SD fault trees):

i) Minimal cutsets represent exactly all failure scenarios. Namely, every failure scenario is a superset of some MCS and every superset of a MCS is a failure scenario.

ii) For a minimal cutset \( C \), one can easily compute the total probability \( p(C) \) of all the scenarios it represents:

\[
p(C) := \sum_{\Xi \supseteq C \text{ MCS}} p(\Xi) = \prod_{a \in C} p(a).
\]

iii) A reasonably tight over-approximation, called rare event approximation, of the failure probability of the whole list of relevant minimal cutsets can be calculated by

\[
p_{rea}(FT) = \sum_{\text{MCS }C:p(C) > c^*} p(C),
\]

where we consider a cutset relevant only if its probability is above a fixed cutoff constant \( c^* \), such as \( 10^{-15} \).

Because of i), we usually obtain by \( p_{rea}(FT) \) a higher number than \( p(FT) \) as many failure scenarios are represented by multiple MCSs and their probability is thus counted several times (e.g. the failure scenario \( \{a, b, c\} \) is represented by MCSs \( \{a, c\} \) and \( \{b, c\} \)) in Example 7.\footnote{One can theoretically obtain \( p(FT) \) exactly by inclusion-exclusion principle and by additional quantification of intersections of sets of failure scenarios represented by tuples of distinct MCSs. This is however infeasible in practice for larger fault tree models.} In reliability studies, basic events typically model rare failures and hence their probabilities are small, typically smaller than \( 10^{-2} \). Under this assumption, the difference \( p_{rea}(FT) - p(FT) \) is acceptably small. Considering only relevant minimal cutsets leads to an under-approximation but it is acceptable in practice [29].

B. Generating relevant minimal cutsets

The algorithm for generation of minimal cutsets in many commercial fault tree solvers (e.g., Saphire or RiskSpectrum [20]) is based on the MOCUS algorithm [14].

The algorithm systematically refines partial cutsets into cutsets and returns the minimal ones. A partial cutset is a set of basic events that are already chosen to be failed and gates that need to be failed in order to turn it into a cutset. The initial partial cutset is \( \{g_{top}\} \).

The efficiency of MOCUS is based on discarding partial cutsets as soon as it is clear that they will yield only irrelevant cutsets, i.e., when the product of probabilities of basic events from the partial cutset is below the cutoff constant \( c^* \).

Example 8. In the SFT from Example 1, the search starts with the partial cutset \{“cooling”\}, branching in the first step into \{e\} and \{“pumps”\}. The former is a MCS, the latter gets processed to \{“pumps 1”, “pump 2”\} and further to \{a, “pump 2”\} and \{b, “pump 2”\}. From these two elements, we finally obtain MCSs \{a, c\}, \{a, d\}, \{b, c\}, and \{b, d\}.
i) Minimal cutsets represent exactly the failed runs. Namely, every failed run is in the set $\text{Reach}^{\leq t}(\text{Failed}(C))$ for some MCS $C$ and for any MCS $C$, every run of $\text{Reach}^{\leq t}(\text{Failed}(C))$ is failed.

ii) For a minimal cutset $C$, one can (under some restrictions on the tree structure) easily compute the probability of runs it represents, denoted by $\tilde{p}(C) = \Pr[\text{Reach}^{\leq t}(\text{Failed}(C))]$. Discussed in Section V-C.

iii) Similarly, we calculate a reasonably tight over-approximation (rare event approximation) of the failure probability of the whole list of relevant minimal cutsets by

$$p_{\text{rea}}(FT) = \sum_{\text{MCS } C: \tilde{p}(C) > \epsilon^*} \tilde{p}(C).$$

Obtaining such a list is discussed in Section V-B.

Before explaining these two steps of the algorithm, we discuss restrictions on the triggering structure which are essential for efficiency of this method in Section V-A.

### A. Limitations

In order to get correct and efficient quantification of minimal cutsets, we need to restrict dynamic modeling in sub-trees of triggering gates. As a consequence, some dynamic features cannot be handled by our framework, but we illustrate on several examples below that it is possible to model important timing dependencies relevant in the nuclear safety domain.

We say that a gate is **dynamic** if there is a dynamic basic event in its subtree. There are two types of subtrees of triggering gates that we can deal with efficiently:

- We say that a gate has **static branching** if for every OR gate $g$ in its subtree, at most one child of $g$ is dynamic.
- We say that a gate has **static joins** if for every AND gate $g$ in its subtree, no child of $g$ is dynamic.

Furthermore, if the model contains sequences of components with static joins that trigger each other then we need these components to satisfy an additional property for efficient analysis: A gate has **uniform triggering** if all dynamic basic events under this gate are triggered and all share the same triggering gate.

Apart from these restrictions on the triggering structure, the definitions in Section III also require certain properties of SD fault trees. We summarize all limitations in the following table.

<table>
<thead>
<tr>
<th>Limitation</th>
<th>Restriction on modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic basic events initiated in $S^{\text{off}}$, $F \subseteq S^{\text{on}}$</td>
<td>Section III-A. A basic event can fail only when its triggering gate is failed. We cannot model a functional dependency that remains failed after a repair of the triggering gate.</td>
</tr>
<tr>
<td>Each dynamic basic event is triggered by at most one gate</td>
<td>Section III-B. This limitation does not restrict the modeling. If we want two gates to trigger a dynamic basic event then we have to connect them by a new gate and let this gate be the triggering one.</td>
</tr>
<tr>
<td>Triggering is acyclic</td>
<td>Section III-B. This limitation does not reduce expressiveness of SD fault trees. Loops in the triggering structure can only be a modeling error.</td>
</tr>
<tr>
<td>Either:</td>
<td>If the triggering system consists of several components which all have to operate at the same time then only one can be modeled dynamically. We have to choose the one where dynamic modeling makes the greatest impact.</td>
</tr>
<tr>
<td>Static branching</td>
<td>The triggering system cannot have redundant components modeled dynamically. (The whole triggering system has to start at the same moment of time – being triggered by the same gate.)</td>
</tr>
<tr>
<td>Or:</td>
<td></td>
</tr>
<tr>
<td>Static joins (+ uniform triggering)</td>
<td></td>
</tr>
</tbody>
</table>

Even with the limitations mentioned above, it is possible to model various types of dynamic behavior relevant for nuclear safety analyses by SD fault trees. Important examples of usage of static branching are illustrated in Figure 1 on the left:

1) The simplest triggering gate where a dynamic basic event directly triggers another one. This is useful for modeling of simple spares. Actually, spare gates in dynamic fault trees have been considered only with basic events as inputs for a long time [25].

2) Component with a static and a dynamic failure. Typically, a (static) failure to start and a (dynamic) failure in operation. This way of modeling is common in the nuclear domain, e.g., for pumps, diesel generators, motors, or regulating valves.

3) Combination of two such redundant components in a single system. This models, for instance, the Emergency Core Cooling system with two redundant pumps both started from the beginning, which is the case in the first hours after the accident initiating event.

4) Combination with an arbitrarily complex structure of
static failures and hierarchical triggering within the system. For example, a cooling system with two redundant pump trains and statically modeled isolation valves shared by both trains, such as the Main Feed Water or Emergency Feed Water systems. Thanks to the hierarchical triggering, a failure of the Main Feed Water system (both trains fail one after another) can trigger the Emergency Feed Water system.

Static joins are illustrated in Figure 1 on the right. They lift the restriction of choosing only one out of several components that need to operate at the same time for dynamic modeling:

1) The simplest gate with static joins allows dynamic modeling of two components in one system. The triggering pattern of this gate will be a result of failures and repairs of both dynamic components.

2) A fault tree for one train of a cooling system with both the pump and the power generator failing in operation modeled dynamically. Cooling systems (Main Feed Water, Emergency Core Cooling or Emergency Feed Water systems) have a backup power generator (a diesel generator or a gas turbine) started in case of a loss of offsite power. A failure of a power generator can be again modeled as a failure to start or a failure in operation. Also, common cause failures [22] can be modeled by static joins.

3) Chaining of systems modeled by static joins. Note, that this is also an example of uniform triggering. One can also add an arbitrarily complex static structure under the triggering OR gates. This allows us to model sequential failures of pumps and power generators within a cooling system and also sequential failures of whole systems (a failure of the Main Feed Water system triggers the Emergency Core Cooling system).

Apart from exhibiting various types of dynamic behavior, models from the nuclear safety domain contain large amount of such dynamic behavior. A standard higher level formalism, event trees, usually captures the order in which safety functions are demanded (static analysis cannot make any use of this information). This simplifies adding of triggers into a safety study, because timed dependencies are already present in the model. Event trees can span over tens of safety function, offering a possibility for long triggering chains.

B. Generating relevant minimal cutsets

We apply the standard MOCUS algorithm for generating minimal cutsets. To this end, we translate the SD fault tree $FT$ into a static fault tree $\overline{FT}$ that has the same minimal cutsets.

1) Structure of the static fault tree $\overline{FT}$: We need to remove two dynamic features of SD fault trees, dynamic basic events and trigger edges. Dynamic basic events are replaced by static basic events (their probability of failure is discussed below). The trigger edges are translated to AND gates. Precisely, we replace each dynamic basic event $b$ triggered by a gate $g$ by a new AND gate with $b$ and $g$ as children. Then we remove all the trigger edges.

By the construction, we have that a set of basic events is an MCS in an SD fault tree $FT$ if and only if it is an MCS in the induced SFT $\overline{FT}$. Hence, the MCSs of $\overline{FT}$ can be obtained by running the MOCUS algorithm on $\overline{FT}$.

Example 9. Consider the following SD fault tree, where the gates triggering basic events $j$, $g$, and $e$ have static branching, static joins (without uniform triggering), and none of these conditions, respectively.

![Static fault tree](image)

The static fault tree $\overline{FT}$ with equivalent minimal cutsets that is produced by the algorithm is below.

![Static fault tree](image)

2) Failure probabilities in $\overline{FT}$. We also need to set the failure probability of each static basic event originating from a dynamic one.

Ideally, one would set $p(a)$ to the probability that the dynamic basic event $a$ in $FT$ fails at least once up to the horizon $t$. If $a$ is not triggered, this number can be easily computed from its Markov chain. When triggers are present, this value may depend on arbitrarily many other basic events and its computation may thus be infeasible.

Instead, we define $p(a)$ as the worst-case probability over all potential ways how $a$ may get triggered. Formally,

$$p(a) = \sup_{\overline{FT}} \Pr_{\overline{FT}}[\text{Reach}^{\leq t}(\text{Failed}(a))]$$

where $\overline{FT}$ ranges over the infinite class of all SD fault trees that contain the basic event $a$. Such a worst-case analysis is correct as we range over a set of trees containing $FT$ and interestingly also easier to compute than the precise value in our tree $FT$. Especially, if we know that the worst case above is attained when $a$ is triggered at time 0 and never untriggered then we can compute $p(a)$ by standard methods on the Markov chain of $a$ (by “shifting” the initial distribution by the function on and ignoring (un)triggering edges afterwards). If we do not know that this is the case, one can efficiently approximate this number by a slightly more elaborate algorithm [10], polynomial in the size of the Markov chain $T(a)$ and pseudo-polynomial in $t$ and in an approximation error $\varepsilon$.

Furthermore, the cutoff rule with threshold $c^*$ in $\overline{FT}$ is conservative with respect to $FT$. Namely, it does not lead to a loss of any MCS in $FT$ with its probability above the threshold.
 Indeed, by the definition in (1), for any partial cutset with a set of basic events $C' \subseteq C$ we obtain
\[
\prod_{a \in C'} p(a) \geq \Pr \left[ \text{Reach}^t(\text{Failed}(C)) \right].
\]

C. Quantification of failure probability of a minimal cutset

For a fixed MCS $C$, we express the probability $\tilde{p}(C)$ using the failure probability of a small SD fault tree $FT_C$:
\[
\tilde{p}(C) = \Pr_{FT_C} \left[ \text{Reach}^t(F) \right] \cdot \prod_{\text{static } a \in C} p(a).
\]

This value can be efficiently computed by building the (also reasonably small) semantical CTMC of the fault tree $FT_C$ and applying a numerical algorithm for the transient analysis of CMTC [18].

Example 10. We first show example fault tree models for minimal cutsets from Example 9. On the left, there is the SD fault tree $FT_C$, for the MCS $C_1 = \{a,c,f,h,i,k\}$ that only checks that all the dynamic basic events from $C_1$ are failed at the same time. The MCS $C_2 = \{a,c,e,g,j\}$ contains triggered basic events and the fault tree on the right needs to include the triggering logic as well.

The SD fault tree $FT_C$ is created in three steps:

1) The top gate of $FT_C$ is an AND gate with all dynamic basic events from $C$ as inputs.
2) For each triggered basic event $a \in C$ we model the logic of its triggering gate. Let $\text{Dyn}a$ and $\text{Sta}a$ denote the sets of all dynamic and static basic events in the subtree of the gate triggering $a$. Based on the assumption which the triggering gate satisfies, we fix a subset $\text{Rel}_a \subseteq \text{Dyn}a \cup \text{Sta}a$ of basic events relevant for the timing of the triggering.

Static branching The set $\text{Rel}_a := \text{Dyn}a \cap C$ contains only the dynamic basic events from the subtree that are in the minimal cutset $C$.

Example: Assuming the MCS $C_2$ above, $\text{Rel}_j = \{g\}$.

Static joins The set $\text{Rel}_a := \text{Dyn}a$ contains all dynamic basic events in the subtree.

Example: $\text{Rel}_e = \{e,f\}$.

The general case The set $X := Y \cup (Z \setminus C)$ contains all basic events from the subtree except for the static ones from $C$ (for which we already assume a failure).

Example: $\text{Rel}_e = \{a,b,c,d\}$ (as $d \notin C_2$).

Having the set $\text{Rel}_a$, we calculate minimal subsets $A_1, \ldots, A_k$ of $\text{Rel}_a$ such that each $A_i$ together with static basic events from $C$ fail the gate triggering $a$.

Example: For the triggered basic event $e$ above, we have $A_1 = \{a,c\}, A_2 = \{a,d\}, A_3 = \{c,b\}$, and $A_4 = \{d,b\}$.

We model the triggering by a fault tree whose minimal cutsets are exactly the sets $A_1, \ldots, A_k$ and let its top gate trigger $a$. This can be added to $FT_C$ in three steps:

a) For any basic event from $A_1, \ldots, A_k$ that is not in $FT_C$, yet, we add it as a leaf (e.g. $d$ and $b$ above).

b) We add all events from each $A_i$ under an AND gate.

3) If we add new triggered dynamic basic events in Step 2 then we first check if their triggering gates are already a part of $FT_C$. If yes, then we finish the construction by adding the corresponding triggering edges. Otherwise, we proceed with Step 2, but we model the triggering logic using the general case, irrespective of the assumptions which the newly modeled triggering gates satisfy.

Example 11. Let us sketch on the example, why additional basic events are needed for the triggering at all.

- Static joins (trigger of $g$). In order to quantify the cutset $C_2$ correctly, we need to add the dynamic basic event $f$. Otherwise, the quantification would exclude runs from $\text{Reach}^t(\text{Failed}(C))$ that have, e.g., the following form: $f$ fails first which causes $g$ and later also $j$ to fail, then $f$ gets repaired making $g$ and $j$ inactive (but still broken), only after that $b$ and $e$ fail causing the top gate to fail.

- The general case (trigger of $e$). We need to add the static basic event $d$ as it “guards” the dynamic basic event $b$. The influence of $b$ on triggering is similar as that of $f$ above.

Triggering gates with static branching or with static joins and uniform triggering never force the construction to repeat Step 2 using the less efficient general case.\(^3\) This guarantees that we need to solve only small Markov chains to quantify cutsets.

As a summary, for an efficient analysis, one can use arbitrary sequences of triggering where each trigger has either static branching or static joins with uniform triggering. Static joins without uniform triggering and general gates may be used at the beginning of each triggering sequence (i.e. to be started at the beginning of the accident) and only sparingly at other places.

VI. EXPERIMENTAL EVALUATION

As a proof-of-concept, we implemented our method using two (slightly adapted) tools, RiskSpectrum [20], a state-of-the-art commercial tool for static event/fault tree analysis, and PRISM [19], the leading academic tool for probabilistic verification. We connected these two tools by python scripts and run all the experiments on a regular laptop with an Intel i3 processor and 4GB RAM.

As the performance highly depends on the structure of the models, we restricted our experimental evaluation only to models with a reasonably realistic structure. We start with real SFT models that we enrich by repairs and triggers. While

\(^3\)Modeling triggering gates with static branching does not add any new dynamic basic events into $FT_C$. If modeling $g$ adds a new dynamic basic event then another dynamic event from its subtree is already in $C$ (static joins) which means that the triggering gate common to all dynamic events in $g$ is already in the model (uniform triggering).
potential triggering caused directly by basic events (the top left case with static branching in Figure 1) can be sufficiently well identified automatically, more complicated triggering logic (having only static joins with uniform triggering) needs to be chosen manually.

For triggers with static joins, we experimented on small-sized fault trees manually annotated by repairs and triggers. For triggers with static branching, we then scale to large-size real nuclear safety studies where repairs are chosen randomly and potential triggers are identified automatically by importance analysis (see Section VI-B). We use the static cutoff in all experiments.

A. Small-size fault tree with static joins and uniform triggering

For this section we consider an example safety study of a fictive boiling water reactor. The aim of these experiments is to illustrate effects of a complex triggering logic on the calculation time and on the resulting computed frequency of core damage. For this, we need a reasonably sized model where the effort of annotating the model by realistic triggering is feasible.

Model: We are interested in the following safety systems of the example model related to cooling:

- ECC - Emergency Core Cooling,
- EFW - Emergency Feed Water, and
- RHR - Residual Heat Removal.

Moreover, there are two additional subsystems appropriate for dynamic treatment:

- CCW - Component Cooling Water System, necessary for the correct functioning of both ECC and EFW, and
- SWS - Service Water System, necessary for the correct functioning of CCW.

All of these five systems have two redundant trains with a pump in each train. This pump can fail to start or it can fail in operation. We chose to model these failures of pumps in operation by repairable dynamic basic events. If the first train in ECC, EFW or RHR fails then it triggers the second train in the respective system, including subsystems. If both trains in RHR fail then an operator action FEED&BLEED is required as a recovery measure. Following the logic described above, we manually selected which basic events to replace by dynamic basic events. Additionally, we manually enriched the static model by realistic trigger dependencies.

Dynamic basic events: We replaced a static basic event (with failure rate $\lambda$) by the following simple dynamic basic event. We generalized the triggered CTMCs from Example 2 by splitting the failure to $k$ phases. If not triggered, the chain starts in phase 0, with rate $k\lambda$ moves from phase $i$ to phase $i + 1$, and is failed in phase $k$. For $k = 1$ this corresponds to exponentially distributed failures as in Example 2, for $k > 1$ this corresponds to Erlang distributed failures. In both cases we preserve the mean time to failure. If a basic event is triggered by some gate, we add corresponding passive states as in Example 2. We assume the failure rates in passive states to be 100 times lower than in active states, see the figure below.

Furthermore, we assume the repair to bring the equipment into the same state as being new. Hence, from the failed phase $k$, the chain jumps back to 0 with its repair rate. For simplicity, we assume that the equipment cannot be repaired before it gets triggered (as nobody knows it is failed) and that all components have the same repair rate $\mu$. This rate $\mu$ together with the number of phases $k$ are experiment parameters.

Experiments: We analyze the core damage frequency which gives us a fault tree with 68 basic events and 122 gates resulting in 11142 minimal cutsets above the cutoff $10^{-15}$. Generating this list of minimal cutsets takes less than a second. The rare event approximation yields the core damage frequency $4.09 \cdot 10^{-9}$.

The following table shows the impact of adding dynamic features. We assume $k = 1$ and analysis horizon equal to 24 hours. For the first part we vary the repair rate. For the second part we add triggers one by one (for each subsequent line, we consider triggers from all previous lines).

<table>
<thead>
<tr>
<th>setting</th>
<th>failure freq.</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>no timing</td>
<td>$4.09 \cdot 10^{-9}$</td>
<td>–</td>
</tr>
<tr>
<td>repair rate 1/100h</td>
<td>$4.09 \cdot 10^{-9}$</td>
<td>7.9s</td>
</tr>
<tr>
<td>repair rate 1/100h</td>
<td>$4.05 \cdot 10^{-5}$</td>
<td>7.5s</td>
</tr>
<tr>
<td>repair rate 1/100h</td>
<td>$3.80 \cdot 10^{-5}$</td>
<td>8.6s</td>
</tr>
<tr>
<td>+FEED&amp;BLEED trigger</td>
<td>$3.78 \cdot 10^{-8}$</td>
<td>8.6s</td>
</tr>
<tr>
<td>+RHR trigger</td>
<td>$3.35 \cdot 10^{-5}$</td>
<td>7.8s</td>
</tr>
<tr>
<td>+EFW trigger</td>
<td>$3.22 \cdot 10^{-5}$</td>
<td>9.5s</td>
</tr>
<tr>
<td>+ECC trigger</td>
<td>$3.13 \cdot 10^{-5}$</td>
<td>10.0s</td>
</tr>
<tr>
<td>+SWS trigger</td>
<td>$2.91 \cdot 10^{-5}$</td>
<td>9.8s</td>
</tr>
<tr>
<td>+CCW trigger</td>
<td>$2.88 \cdot 10^{-5}$</td>
<td>10.4s</td>
</tr>
</tbody>
</table>

The big decrease of failure frequency is partly due to disregarding common cause failures in the analysis. Common cause failures are less influenced by timing dependencies and usually dominate the result. Still, this shows that sequencing and repairs in the model have a significant impact on the result.

Let us briefly turn our attention to the time complexity of the analysis. For the fully dynamic model, 5449, i.e. roughly a half of the minimal cutsets contain dynamic basic events and need to be analyzed dynamically. Among these cutsets, the average number of dynamic events is 3.02 out of which 1.78 are added because the triggering gates do not have static branching.

B. Industrial-size fault tree with static branching

To assess scalability of this method, we run a series of tests on two real life nuclear safety studies. We analyze different core damage scenarios with varying level of dynamic features. The aim of these experiments is to demonstrate that the dynamic quantification is feasible even for fault trees with thousands of basic events and tens of thousands of gates.
Fig. 2. Histograms depicting how many dynamic basic events are in different minimal cutsets. Six charts for varying amount of dynamic features.

Fig. 3. Time to analyze each Markov model of a MCS as it varies with the number of dynamic basic events in the MCS and with the number of phases per basic event. Note the logarithmic scale. The size of the Markov chain to be analyzed for each cutset is exponential where number of dynamic basic events in the cutset is the exponent and the number of phases influences the base.

Furthermore, we obtain a deeper understanding for the effect of various parameters on the analysis time.

**Model:** We consider two models, their basic parameters are summarized in the table below (again, we consider the cutoff threshold $10^{-15}$):

<table>
<thead>
<tr>
<th>Model</th>
<th># BE</th>
<th># gates</th>
<th># MCS</th>
<th>MCS generation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,995</td>
<td>52,213</td>
<td>74,130</td>
<td>4327s</td>
</tr>
<tr>
<td>2</td>
<td>2,040</td>
<td>56,863</td>
<td>76,921</td>
<td>16680s</td>
</tr>
</tbody>
</table>

In both models, we replace varying percentage of basic events by dynamic basic events. Always, the given percentage of events with the highest Fussel-Vesely importance factor \cite{28} is replaced. This way, we set dynamic behavior first to basic events where we expect the highest impact of dynamic modeling. We use the same quantity for setting triggers in the model. We assume that basic events playing role of symmetric redundant parts will have the same importance. Therefore, we create triggering chains from dynamic basic events with the same Fussel-Vesely importance factor (again, chains with highest importance first). We used the same models for dynamic basic events as for the smaller model.

**Experiments:** First, we observe how the failure frequency and the analysis time change when increasing the amount of dynamic features in model 1. Again, we fix the number of phases per basic event to 1 and the analysis horizon to 24 hours. Observe that adding the first 40 % of dynamic basic events has the highest impact on the frequency.

<table>
<thead>
<tr>
<th>% dyn. BE</th>
<th>% trigg. BE</th>
<th>failure freq.</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$1.50 \cdot 10^{-9}$</td>
<td>–</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>$1.45 \cdot 10^{-9}$</td>
<td>15s</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>$1.10 \cdot 10^{-5}$</td>
<td>40s</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>$6.45 \cdot 10^{-6}$</td>
<td>1m 53s</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>$5.89 \cdot 10^{-6}$</td>
<td>1m 26s</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>$5.78 \cdot 10^{-6}$</td>
<td>1m 36s</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>$5.71 \cdot 10^{-6}$</td>
<td>2m 12s</td>
</tr>
</tbody>
</table>

The analysis time does not substantially change after we reach 30 % of dynamic basic events. This phenomenon can be explained as follows. During each analysis run, tens of thousand of Markov chain models are evaluated. Analysis time for each model does not depend much on particular failure rates of the components; what matters is the size of the model (the number of dynamic basic events). The crucial observation is that the distribution of these sizes among the model set does not change much after the 30 % threshold is passed. For details, see Figure 2.

On both fault tree models, we also compared analysis times when varying the number of phases $k$ in the models of all dynamic basic event.
These numbers demonstrate several facts. First, the structure of the fault tree plays a crucial role. Models 1 and 2 have similar size and number of MCSs. Yet, analysis time differs substantially. Second, the analysis time grows exponentially when increasing the size of Markov models of MCSs. This is shown in a greater detail in Figure 3. It is clear that for larger models, it is infeasible to model each failure using Markov chains with many states. One has to carefully choose a small number of components where a non-exponential model of failure distribution is necessary.

As the last experiment, we compared analysis times when increasing the horizon of the evaluation. Here, we present results for the more computationally intensive model 2:

<table>
<thead>
<tr>
<th>horizon</th>
<th>failure frequency</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>24h</td>
<td>1.86 \times 10^{-6}</td>
<td>9m 31s</td>
</tr>
<tr>
<td>48h</td>
<td>4.67 \times 10^{-6}</td>
<td>12m 47s</td>
</tr>
<tr>
<td>72h</td>
<td>7.56 \times 10^{-6}</td>
<td>16m 59s</td>
</tr>
<tr>
<td>96h</td>
<td>1.05 \times 10^{-5}</td>
<td>19m 14s</td>
</tr>
</tbody>
</table>

One can see that the analysis time grows roughly linearly and that the method easily scales to the horizon of several days. In the light of post-Fukushima trends to increase the analysis horizon, this is an encouraging result.

As a concluding remark, note that for importance and uncertainty analyses, one needs to evaluate the list of minimal cutsets many times (e.g. once for each basic event). As this is easy to parallelize, our experiments provide us encouraging evidence that even this type of analyses is feasible with SD fault trees.

VII. RELATED WORK

The natural answer to timing concerns are dynamic fault trees (DFT) [4], [5]. This continuous-time stochastic formalism offers constructs with a great expressive power to model temporal patterns of failures. Despite advanced state space reduction techniques [7], a successful application of DFT has been reported only for small to middle-sized systems with around 100 basic events for general fault trees [1] and around 350 basic events for fault trees with repairs only [15]. Other solution methods based on translating DFT to Stochastic Petri Nets [13] or to Bayesian Networks [6] do not report scalability required for large industrial fault trees either.

The inspiration for our definitions of the dynamic aspects of the fault tree model comes from Boolean Driven Markov Processes (BDMP) [8]. Formal semantics of BDMP is also explained in [25]. The BDMP formalism allows repairs as well as unrestricted triggers. The evaluation is based first on a special algorithm for the transient Markov chain analysis and second on a massive state-space reduction; both exploiting the specific structure of Markov chains originating from BDMPs. Still, the methods only scale up to BDMPs of 100-300 basic events [9]. A later work [12] describes how to extract minimal cut sequences (minimal cutsets with temporal order information) from BDMP models in a systematic way. However, it does not give any evidence for an efficient procedure being able to handle fault trees for full-scale nuclear safety studies.

Mixed static and dynamic trees were proposed in [16]. The analysis is based on modularization and isolation of dynamic parts of the tree from the static ones. The dynamic parts are translated to Markov chains, while the rest is solved by a Binary Decision Diagram. Scalability of this method depends on dynamic subtrees being small and completely independent of the rest of the fault tree.

A cutset based method for dynamic evaluation of fault trees has been considered as well [27]. This approach has the same structure as our method, because it first generates cutsets and then quantifies them dynamically. However, this method only allows repairs and has thus a much lower expressiveness than SD fault trees. Going beyond simple repairs for a cutset based method is challenging and highly non-trivial. The concept of repairs in otherwise static fault trees has been also used to synthesize and evaluate repair policies [3], [24].

Authors of [30] also aim at utilization of static algorithms for dynamic fault trees. They restrict themselves to DFT with Priority-AND gates satisfying a special condition. Moreover, the paper presents only qualitative analysis of such trees, i.e., generation of minimal cut sequences. Neither PAND gates nor qualitative analysis address well the needs in the domain of our application. Another recent work [21] deals with a construct similar to our triggers – Spare gates. They develop an algebraic approach for quantification of these types of gates. The paper offers no arguments or experimental indication that the approach scales to large models. Various other extensions of static fault trees are reviewed in a recent survey paper [26].

VIII. CONCLUSIONS

We have presented a new fault tree model combining static and dynamic features. Under certain assumptions, we have developed a novel way of analysis of these fault trees which scales even for large industrial fault tree models of nuclear power plants. We demonstrate the feasibility of our approach by experiments with a prototype implementation which is able to handle such real life models. If most dynamic basic events have small Markov models then quantification times of minimal cutset lists are several orders of magnitude shorter than times for their generation. By this, we have extended modeling possibilities for any application area where large fault trees are needed.

As a future work, we want to lift the assumptions on the triggering gates and develop approximate algorithms for more general models. Failure probabilities may be under-approximated by disregarding interplays of several dynamic basic events. Dually, an over-approximation may be achieved by allowing dynamic basic events interfere irrespective of static basic events. In both cases, the incomplete Markov chain models can be analyzed by the methods of [10].

Acknowledgments: The authors would like to thank Ola Bäckström (Lloyd’s Register Consulting) for numerous discussions and for sharing his expert knowledge on probabilistic
safety studies of nuclear power plants. Furthermore, the authors are grateful for inspiring discussions with Wang Yi (Uppsala University), Holger Hermanns (Saarland University), Marc Bouissou (Électricité de France S.A., École Centrale Paris - LGI), and Mariëlle Stoelinga (University of Twente). Finally, we thank Peter Buchholz for carefully reading the manuscript. This work is partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center AVACS (SFB/TR 14), by the EU 7th Framework Programme under grant agreement no. 295261 (MEALS), 318490 (SENSATION) and 288175 (CERTAINTY), by the CAS/SAFEA International Partnership Program for Creative Research Teams, and by the Czech Science Foundation, grant No. P202/12/G061.

REFERENCES