Effective Static and Dynamic Fault Tree Analysis

Ola Bäckström[1], Yuliya Butkova[2(✉)], Holger Hermanns[2], Jan Krčál[2], and Pavel Krčál[1(✉)]

[1] Lloyd's Register Consulting, Stockholm, Sweden
Pavel.Krcal@lr.org
[2] Computer Science, Saarland University, Saarbrücken, Germany
{butkova,hermanns}@cs.uni-saarland.de

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 support for time-dependent behaviour, e.g., repairs or dynamic dependencies. This enables more realistic and more precise modelling, and can thereby avoid overly pessimistic analysis results. But analysis of DFT is so far limited to substantially smaller models than those required for instance in the domain of nuclear power safety. This paper considers so called *SD fault trees*, where the user is free to express each equipment failure either statically, without modelling 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 employs automatic abstraction techniques effectively, and thereby scales similarly to static analysis algorithms, albeit allowing for a more realistic modelling and analysis. We demonstrate the applicability of the method by an experimental evaluation on fault trees of nuclear power plants.

1 Introduction

Fault trees are a very prominent formalism for inductive failure modelling. They underly safety assessments in a wide spectrum of technical systems, ranging from nuclear power production [9,17], over chemical and process industry [7] to automotive and aerospace [14] systems.

A fault tree decomposes the failure potential of a complete system into failures of its subcomponents, sub-sub-components, and sub-sub-subcomponents, up to the level of so-called *basic events*. The latter represent individual equipments, atomic external events, operator errors, etc. These are assumed to be quantifiable wrt. estimates of failure frequencies or probabilities, achieved by statistical methods from operation history or simulations or even by engineering computations. Originally, fault trees describe a static view on a system, we thus call them static fault trees (SFTs). Static fault trees pair simplicity in modelling with efficiency in analysis techniques.

A particularly effective analysis technique characterises all fault combinations leading to the complete failure of an SFT, and returns their minimal-sized

© Springer International Publishing Switzerland 2016
A. Skavhaug et al. (Eds.): SAFECOMP 2016, LNCS 9922, pp. 266–280, 2016.
DOI: 10.1007/978-3-319-45477-1_21

representation, in the form of so called *minimal cutsets*. Even though the number of minimal cutsets can be exponential in the number of basic events, it is possible to appropriately employ the cutoff on low probability cutsets to reduce the size of the problem. This minimal cutset analysis is in daily use for instance in the safety analyses of nuclear power plants [9,17], where SFTs with several thousands of basic events are routinely processed, supported by tools such as Saphire [18] or RiskSpectrum [13].

It has been argued [2–4,11,14] that the static system view supported by SFTs is often very rough (though conservative), in the sense that a more precise analysis is possible if the fault tree formalism provides support for representation and analysis of the changes in state of the system in operation. In the nuclear safety domain, this means that the dynamics of an accident and possible countermeasures can be detailed. The promised gain in precision is of industrial relevance, for instance for analyses with longer mission time, such as probabilistic Level 2 [10] (and consequently Level 3) studies in nuclear power plants. After the Fukushima accident, the interest in analyses studying 'safe state' rather than a fixed mission time has increased. This will increase the need to properly treat long mission times also within Level 1 [9] probabilistic safety assessment.

Over the years, several kinds of dynamic fault trees have been proposed, starting with the work of Dugan [2]. However, dynamic analysis techniques need to implicitly or explicitly explore the state space spanned by the system dynamics. This space tends to be prohibitively large; often it is of exponential size, relative to the number of basic events. With previous techniques, models with more than a few hundred basic events are impossible to process. This means that these approaches cannot be directly applied to large scale industrial fault tree models such as those of nuclear power plants.

SD fault trees [11] (SD-FTs) have lately been proposed to provide a potential way forward. They extend SFTs with features to model *some* parts of the system dynamically, without the need to construct the induced state space of the entire fault tree. This means that it remains possible to utilize efficient solver technology for SFTs, and combine this with less efficient, but focused analysis for the dynamic parts. The new features can capture (1) *sequential* application of elementary safety functions and (2) *repairs* of failed components. Basic events can be considered either static or dynamic. Dynamic dependencies are expressed via a triggering mechanism, whereby a safety function failure may *activate* other safety functions and failed components can be repaired (and thus continue to perform their function).

In this paper, we build on the SD-FT concept. We attack the problem that the focused analysis needed for the dynamic parts may still suffer from state space explosion, exponential in the amount of dynamic basic events. Indeed, the algorithm originally developed for the SD-FT formalism [11] is efficient only if restricting the triggering logic severely in expressiveness. This is rooted in the fact that the algorithm calculates the dynamic failure probability exactly, which in turn requires considering all possible accident progression scenarios, including

consecutive failures and repairs of components. This becomes quickly infeasible for increasingly intricate triggering patterns induced by a richer triggering logic.

However, our analysis of real-life safety analysis models has made apparent that most of these scenarios turn out to be rather unrealistic. This is reflected by their relatively low probability compared to a few dominating simple scenarios. The present paper exploits this observation to leap to a generally applicable method. The crux of this leap lies in abstracting away from unrealistic event sequences in a controlled manner. This allows us to obtain an over- and under-approximation, safely bounding the exact value. As a result, the present work lifts the triggering restrictions in their entirety, enabling efficient analysis of SD-FTs with arbitrary triggering logic. We present this approach on a mildly restricted subclass of SD-FTs that limits the shape of dynamic basic events, in contrast to restrictions on the triggering logic.

As we will demonstrate by means of several examples, the resulting method scales very well to industrial-size systems, even from the nuclear power domain, and with high precision guarantees. The restrictions we need to impose on SD-FTs do not affect their adequacy for the application context as they cover all standardly used reliability models of basic events.

2 Static and Dynamic Fault Trees

We focus our work on a class of *static and dynamic (SD) fault trees*, introduced in [11]. It allows the modelling of components of the system either statically or dynamically. The behaviour of dynamic components are modelled via *continuous-time Markov chains*.

Definition 1. *A* failure continuous-time Markov chain (failure CTMC, or fCTMC) *is a tuple* $\mathcal{C} = (S, R, \nu, F)$ *where* S *is a finite state space,* ν *is the initial distribution over* S*,* $R : S \times S \to \mathbb{R}_{\geq 0}$ *is the rate matrix, and* $F \subseteq S$ *is the set of failed states.*

At initialisation time, the system chooses a state according to initial distribution ν. The amount of time the system spends in some state s is distributed exponentially (with the *rate* parameter of the distribution $\lambda = \sum_{s' \in S} R(s, s')$). After this delay the system moves from the current state to successor s' with probability $R(s, s') / \sum_{s'' \in S} R(s, s'')$.

The set F of states of a fCTMC $\mathcal{C}$ corresponds to failed states of a component. The complement set represents properly functioning ones. Failure of the component is modelled by transitions from functioning to failed states, and repair - from failed to functioning. An example fCTMC is depicted in Fig. 1.

The SD-FT formalism allows one to model redundant back-up components as well. Whenever a component is failed, its back-up substitute can be used by the system until the main component gets fixed. This feature is modelled with the help of *triggered CTMCs*:

Definition 2. *A* triggered CTMC *(tCTMC) is a fCTMC with states partitioned into* $S^{off} \uplus S^{on}$ *and with total functions* $on : S^{off} \to S^{on}$ *and* $off : S^{on} \to S^{off}$. *We require* $F \subseteq S^{on}$ *and* $\{s \in S \mid \nu(s) > 0\} \subseteq S^{off}$, *i.e. only an* on *state can be considered failed, and only at* off *states the system can be initialized.*

A component represented by a tCTMC can be either *switched on* or *off*. Figure 1 displays an example of a tCTMC. Dashed transitions, representing the effect of functions *on* and *off*, are called *triggering transitions*. Being currently in an *on* or *off* state, a tCTMC behaves in the same way as an fCTMC. Triggering transitions are ignored unless an external event arrives (e.g. failure of another component). In this case the tCTMC takes *instantaneously* the corresponding triggering *off* or *on* transition.

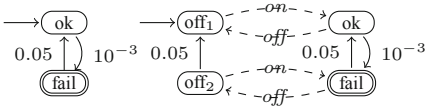

Fig. 1. An example fCTMC (left) and tCTMC (right). Double circles indicate F states. States *ok* (left) and off_1 (right) are initial.

Definition 3 (SD fault trees [11]). *A* static and dynamic fault tree *(SD-FT) is a finite directed acyclic graph where its leaves are partitioned into sets* B_s, *called* static basic events, *and* B_d, *called* dynamic basic events. *Its inner nodes* G *are called* gates *where a distinguished root node is denoted* g_{top}. *Additionally,*

- *each gate is either of type* AND *or of type* OR,
- *each gate* g *has a set of dynamic basic events* $\mathrm{trig}(g)$ *that are* triggered *by* g,
- *each static basic event* a *is specified by its probability of failing* $p(a)$,
- *each dynamic basic event* a *is specified by* $T(a)$ *which is a tCTMC iff* a *is triggered by some gate, and an ordinary fCTMC, otherwise.*

SD-FT can be considered as a specific subclass of BDMPs [4], albeit at the price of dropping the distinction between static and dynamic events. In fact it is this distinction that we exploit to conquer and intertwine static and dynamic analysis steps effectively.

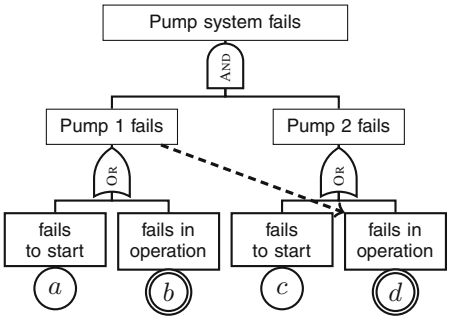

Fig. 2. An example of a SD fault tree.

Without loss of generality, we assume that each dynamic basic event is triggered by at most one gate. The case of multiple triggering gates $g_1, g_2, \ldots g_k$ can be reduced to only one by adding an OR gate over $g_1, g_2, \ldots, g_k$, and making only this OR gate triggering. We also require that there are no cyclic dependencies in the triggering structure. Scenarios excluded by this requirement are exactly "deadlocks" situations where none from a group of several dynamic events can fail before all others have failed.

Example 1. Figure 2 depicts an example SD-FT. Dynamic basic events b and d are denoted by double circles, and their CTMCs are given in Fig. 1 (non-triggered for b and triggered for d). Failure of pump 1 triggers the event d from the pump 2, depicted by the dashed edge.

Behaviour of a SD-FT. At time zero each static event a either fails with probability $p(a)$ or succeeds with probability $1-p(a)$. Dynamic events randomly choose their initial states according to their initial distributions and proceed as described above. Failures and repairs of basic events instantaneously propagate up through the SD fault tree according to the rules of Boolean logic. We call a gate *failed* or *functioning*, if the logic beneath the gate is failed or functioning. Whenever a triggering gate becomes failed, or gets repaired, it instantaneously triggers the corresponding triggered basic events, which each instantaneously take a transition labelled by *on* or *off*, respectively.

Semantics. For the formal definition of the SD-FT semantics we refer to [11]. Informally, it is given in terms of a product Markov chain $\mathcal{C}_{FT} = (S, R, \nu, \mathbf{F})$. To this end, first, each static basic event a is represented as an equivalent Markov chain. It consists of only two states *ok* and *fail*, has no transitions between them, and $\nu(fail) = p(a)$. Then, the product Markov chain is built over the product state space of all its basic events. Transitions between states occur according to parallel interleaving, i.e. only one basic event can transit at a time. The failure state set $\mathbf{F}$ of the $\mathcal{C}_{FT}$ is formed by those states in which failures of the respective components jointly induce a failure of the top gate.

Probability of Failure. We are interested in the probability of the top gate of the fault tree FT to fail within some fixed time horizon t. We will denote this value as $p(FT)$. This value corresponds to the reachability property [1] of the product Markov chain $\mathcal{C}_{FT}$, which is the probability of the Markov chain to reach the set of goal states $\mathbf{F}$ within time t. Thus

$$p(FT) = \Pr_{\mathcal{C}_{FT}}\left[\text{Reach}^{\leqslant t}\mathbf{F}\right]$$

3 SD-FT Analysis

Existing Techniques. An effective computational method for SD-FT analysis has been proposed in [11], albeit with restrictions: The price of the computation speed is paid by severe constraints on the triggering logic. These constraints exclude, for instance, multiple dynamic basic events in different subtrees below OR gates or any occurrence of dynamic basic events in subtrees of AND gates. Furthermore, for nested triggering, they enforce that all dynamic events under a (nested) triggering gate are triggered by the same trigger. Relinquishing any of these constraints makes the algorithm not scale well. For regular industrial

systems the application of this algorithm is therefore limited. However, the algorithm will be our natural reference for comparison in the experimental evaluation in Sect. 4.

A More General and More Efficient Approach. In order to successfully apply SD-FTs to real world applications we thus need a more general and more efficient approach. In this section we present a new simple and efficient algorithm for solving SD-FTs. The approach overcomes constraints on the triggering logic in their entirety. It uses abstractions so as to cope with the state space explosion problem. In doing so, it introduces a reasonable and controllable error margin, and comes at the price of mildly restricting tCTMCs appearing as triggered basic event behaviours. These restrictions are not prohibitive at all with respect to models currently used in practice. This is rooted in the lack of available statistical data. Models of basic events that are used in real world application need the data of failure and/or repair rates for a specific component. These values are gathered statistically and so far are mostly available for very simple basic events, like those depicted in Fig. 1. Due to this, designing a finer model of a basic event is in most cases not possible.

Our algorithm is built upon the ideas of static fault tree analysis and is centred around the notion of *minimal cutsets*. A set of basic events C is a *cutset* if whenever all of the basic events in C are simultaneously in a failed state then the top gate is failed as well. A cutset C is *minimal* if there is no smaller cutset contained in C. For instance, in Example 1 the set $C = \{a, b, c\}$ is a cutset, while $C = \{a, c\}$ is a minimal cutset (MCS). A *failure probability* of a cutset $p(C) := \Pr_{\mathcal{C}_{FT}}\left[\text{Reach}^{\leqslant t}\mathbf{F}(C)\right]$, where $\mathbf{F}(C)$ are those states of the product CTMC $\mathcal{C}_{FT}$ in which all events from C are failed. The set of minimal cutsets $L(FT)$ of a tree FT represents exactly the failure scenarios of a system, i.e. $\text{Reach}^{\leqslant t}\mathbf{F} = \bigcup_{C\in L(FT)} \text{Reach}^{\leqslant t}\mathbf{F}(C)$. Thus, $p(FT) = \Pr_{\mathcal{C}_{FT}}\left[\bigcup_{C\in L(FT)} \text{Reach}^{\leqslant t}\mathbf{F}(C)\right]$ and can be computed via minimal cutsets and the inclusion-exclusion principle.

Due to the scale of systems, computation of the failure probability of a fault tree becomes rarely feasible. Instead, a value called *rare event approximation* [14] *with a cutoff* is usually targeted. This quantity is defined by $p_{rea}(FT) := \sum_{p(C)>c^*} p(C)$. Here c^* is called a *cutoff* constant. In static fault tree analysis it is usually set to values in the order of 10^{-10}. We call a MCS C *relevant* if $p(C) > c^*$. Following the best practices, we as well approximate the value $p_{rea}(FT)$ rather than $p(FT)$ in our analysis.

We will introduce now a subclass of triggered CTMCs that allows efficient analysis. It mildly restricts the structure of tCTMCs without sacrificing expressiveness.

Definition 4 (Simple SD-FT). *A SD fault tree is* simple *if the tCTMC of each triggered dynamic basic events satisfies the following:*

- $R(s, s') > 0 \Leftrightarrow s, s' \in S^{on}$ *or* $s, s' \in S^{off}$*;*
- *both* $on \circ off$ *and* $off \circ on$ *are identities;*

- *the projection of the tCTMC on* S^{on} *(or equivalently* S^{off}*) has one of the shapes depicted in Fig. 3 with* $k \geqslant 0$ *and* $l \geqslant 1$*;*
- *for any two states* s_{off} *and* s_{on}*, such that* $s_{off} = off(s_{on})$ *(or equivalently* $s_{on} = on(s_{off})$*):*

$$s_{off}, s_{on} \in S \backslash F \rightarrow R(s_{off}, succ(s_{off})) \leqslant R(s_{on}, succ(s_{on}))$$
$$s_{off}, s_{on} \in F \quad \rightarrow R(s_{off}, succ(s_{off})) \geqslant R(s_{on}, succ(s_{on})),$$

i.e. the rate of failing is higher when the component is turned on, than when it is off, and, analogously, the rate of repair is lower.

This definition in particular naturally allows for models that return to a stable configuration (on repair or similar). An example of a simple SD-FT is the tCTMC depicted in Fig. 1.

Remark. The correctness of our algorithm is rooted in properties of *open Interactive Markov Chains* (oIMCs) [5]. Nowadays, oIMC analysis has scalability issues, but it might benefit from recent advances in the field of *Continuous Time Markov Decision Processes* [6]. In this way, our approach can be lifted to the general class of tCTMCs, possibly retaining its effectiveness.

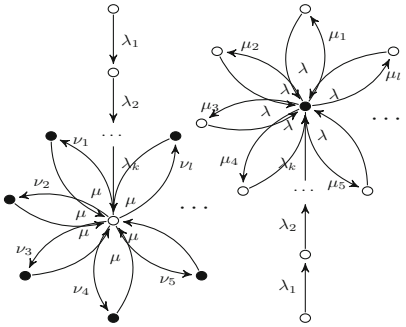

Fig. 3. Two possible shapes of CTMCs of triggered BE of a simple SD-FT. States filled with black denote failed states and the non-filled ones are functioning.

3.1 Quantification of a SD-FT

Let FT be a simple SD-FT and c^* be our cutoff constant. As mentioned before, we target the approximation of the value $p_{rea}(FT) := \sum_{p(C)>c^*} p(C)$. To quantify this value we need a list of relevant cutsets and a procedure to quantify the value $p(C)$ for each relevant cutset C. To efficiently obtain the list of relevant cutsets we can proceed in the same way as presented in [11]. To this end we use the MOCUS algorithm [8], which returns the set of relevant cutsets L_{c^*} as well as the bound ε on the error introduced by the cutoff c^*. We will thus skip this step and in the following concentrate on the algorithm to quantify each relevant cutset.

Quantification of Failure Probability of a MCS. As observed in [11], the failure probability $p(C)$ of a MCS C can be *exactly* expressed by the failure probability of a smaller SD-FT FT_C, which we will call *representative tree for* C. It is constructed as follows:

`BuildRepTree`(C)

1. Add to FT_C a new top AND gate with all basic events from C as inputs.

2. To track which gates we model in FT_C, label all gates of FT as *missing*.
3. While FT_C has a basic event that is in FT triggered by a *missing* gate g:
   (a) Calculate minimal cutsets $C_1, \ldots, C_k$ of the subtree of g.
   (b) Model in FT_C the gate g by a new OR gate that has as inputs new gates $g_1, \ldots, g_k$ where each g_i is an AND gate over basic events from C_i. (In this process, copy to FT_C all the newly referred basic events.)
   (c) Label g as *not missing*.
4. Finally, having modelled all triggering gates, add to FT_C all the trigger edges, i.e. between a basic event b and gate g if g triggers b in FT.

Lemma 1. $p(C) = p(FT_C)$

In order to quantify $p(C)$ one can construct the semantical CTMC of the fault tree FT_C and apply a numerical algorithm for the reachability analysis on it [1]. However, the size of the fault tree FT_C depends on the triggering structure of FT and in the worst-case can be as large as FT, rendering the direct analysis of the semantical CTMC infeasible. For comparison, 100 dynamic basic events translates into 2^{100} states of the product CTMC, when modern tools for CTMC analysis (e.g. PRISM) can handle up to 2^{40} states at most. We will later show in the experimental evaluation section that this growth problem is not an exotic corner case, but is a real problem even for simple real world models. Our approach instead avoids the explosion by building conservative over- and under-approximations of the value $p(C)$. In this way we sacrifice precision but retain expressiveness and efficiency.

Over- and Under-Approximations of the MCS Failure Probability. We aim at decreasing the size of the state space by reducing the amount of basic events of FT_C and simplifying its triggering structure. Intuitively, we shall replace some of the dynamic basic events with trivial *static* ones, which are failed either always or never (for over- and under-approximations respectively). This will allow us to cancel out not only a number of dynamic basic events, but also some of the triggering gates completely, thereby significantly simplifying the analysis. We do so in a way that controls the error introduced by this replacement.

We need to differentiate between *immediate* and *nested* triggering gates, with respect to a cutset C. Immediate gates are those that trigger some BE from C directly, while nested gates trigger basic events indirectly through a sequence of failures and triggering of other gates. We will also introduce two new static basic events: e_{slow} with probability 0 and e_{fast} with probability 1. Intuitively, e_{slow} never fails, while e_{fast} is failed from the beginning.

We will now define the procedure that allows us to obtain an abstraction of the representative tree of a cutset. Let C be a cutset, the variable $dir \in \{over, under\}$ denotes the direction of abstraction (over- or under-approximation). The list of basic events to be cancelled out is called an *abstraction sequence*. The procedure we present is applicable for an arbitrary abstraction sequence. Later in this section we will present heuristics for obtaining abstraction sequences for over

and under-approximations, that we used in our experiments. In the following, whenever we perform an operation on a cutset (or a list of cutsets) we assume an equivalent operation to be performed on its representative tree and vice versa.

`AbstractTree`(C, c^*)

1. Using the `BuildRepTree` procedure, build the representative tree of C. In step 3(a) of `BuildRepTree` instead of using the set of cutsets of a gate g, use the set of relevant cutsets $L_{c^*}(g)$. The value $L_{c^*}(g)$ and the cutoff error bound ε_g can be obtained in the same way as described above using the MOCUS algorithm;
2. If $dir = over$ add to $L_{c^*}(g)$ the set $\{b_{\varepsilon_g}, e_{fast}\}$ where b_{ε_g} is a new static basic event with probability ε_g[1];
3. Choose an *abstraction sequence* $A = (b_1, G_1)(b_2, G_2)\dots(b_n, G_n)$, where b_i is a non-triggered basic event of FT_C and G_i is a set of gates;
4. Repeatedly for $i = 1..n$:
   (a) for each gate $g \in G_i$, for each cutset $C_g \in L_{c^*}(g)$ replace all occurrences of b_i by e_{slow} if $dir = under$, and by e_{fast} if $dir = over$[2];
   (b) remove from $L_{c^*}(g)$ cutsets that have become non-minimal (propagate these changes into the tree by removing respective gates);

Remark. Notably, after step 4(b) one can still perform a number of further reductions of the state-space of FT_C. For instance whenever an event b is replaced with e_{slow}, all the cutsets containing b can be immediately removed, since they will never fail. As a result of this procedure we obtain new trees $\overline{FT_C}$ and $\underline{FT_C}$ for over- and under-approximations.

Lemma 2. $p(\underline{FT_C}) \leqslant p(C) \leqslant p(\overline{FT_C})$

Depending on the chosen abstraction sequence, $\overline{FT_C}$ and $\underline{FT_C}$ can be of a much smaller size than the original FT_C, making it possible to apply the efficient CTMC analysis we discussed above directly to product CTMCs constructed separately for $\underline{FT_C}$ and $\overline{FT_C}$. Let $\underline{\mathbf{F}}$ and $\overline{\mathbf{F}}$ be failed states of $\underline{FT_C}$ and $\overline{FT_C}$, and let ε' be the error bound used by the CTMC algorithm. We thus define the over- and under-approximations as follows:

$$\underline{p}_{c^*}(C) := p(\underline{FT_C}) \qquad = \Pr_{\mathcal{C}_{\underline{FT_C}}}\left[Reach^{\leqslant t}(\underline{\mathbf{F}})\right]$$
$$\overline{p}_{c^*}(C) := p(\overline{FT_C}) + \varepsilon' = \Pr_{\mathcal{C}_{\overline{FT_C}}}\left[Reach^{\leqslant t}(\overline{\mathbf{F}})\right] + \varepsilon'$$

[1] This is to compensate for the cutoff error bound ε_g.

[2] Whenever the event b_i belongs to a cutset of a gate $g \notin G_i$, we create a copy of b_i and direct all the transitions from gates g to b_i to the new basic event. Thus whenever b_i is abstracted in gates $g \in G_i$, it is not abstracted away in gates $g \notin G_i$.

The Abstraction Sequence Heuristic. The abstraction sequence that one decides to use in the above procedure affects directly the error introduced by the approximation. We will now describe the heuristics for selecting an abstraction sequence that we find reasonable in practice and that we used for the experiments.

For nested gates, we abstract all basic events in an arbitrary order yielding $\overline{L_{c^*}}(g) = \{\{e_{fast}\}\}$ and $\underline{L_{c^*}}(g) = \{\{e_{slow}\}\}$[3]. As regards immediate gates, we use different heuristics for over- and under-approximations. We first introduce two new measures $\varepsilon^U(b_i) \geqslant 1$ and $\varepsilon^O(b_i) \geqslant 1$ of the impact of abstracting event b_i away. These measures are based on the notions of *risk increase(decrease) factor* [16]. The closer these values are to 1 the smaller is the loss of precision due to reduction of the respective event. We therefore aim at abstracting such events.

Let $err \geqslant 0$ be an allowed error parameter, $x \in \{O, U\}$. We assign each G_i to be the set of all immediate triggering gates. The following procedure applies to both over- and under-approximation (by using respective x):

1. Enumerate all the basic events b from FT_C except for those in C by their ascending $\varepsilon^x(b)$. The $\varepsilon^x(b)$ needs to be re-evaluated for every element in the sequence as abstracting all previous events changes the FT_C;
2. Stop once reducing the next basic event according to the given order would make the error $\prod_{reduced\ b} \varepsilon^x(b)$ exceed $err + 1$;

Remark. As a result of applying these abstraction sequences one may obtain a lot of cutsets of a specific shape. Those are either singleton cutsets, or pairs of the form $\{b, b_i\}, \{b, b_j\}$. In order to further reduce the state space one can add another abstraction step that lumps such cutsets together, while preserving the property of being an over- or under-approximation. We indeed defined such a lumping procedure for the class of dynamic basic events whose CTMC has one of the shapes depicted in Fig. 3, and used it in our experiments

4 Experimental Evaluation

This section presents the empirical evaluation of our approach. Since our focus is on an efficient approach that integrates well with the *industrial practice*, we do not consider small or medium-size synthetic examples whose homogeneous structure would enable to study model size vs. solution time tradeoffs. Instead we prefer to present results for realistic models from industrial practice, therefore serving as a proof of concept.

As an implementation of the MOCUS algorithm we use RiskSpectrum [13], and resort to the Prism tool [12] for the reachability analysis of the CTMCs. All the intermediate processing, mainly reductions and conversions, were implemented as Python scripts. All experiments are carried out on a single Intel Core

[3] Reduction of a triggered basic event is possible due to reduction of its triggering gate.

i7-4790 with 32 GB of RAM. The following abbreviations will appear throughout the section: *BE*, *DynE* and *TrigE* denote the overall number of basic events, dynamic basic events, respectively triggered events in a given SD-FT. The number of relevant minimal cutsets is denoted as *RelMCS*.

Models. We evaluate our approach on four simple and two larger reactor models. These are derived from models representing analyses built by safety engineering experts with all the modelling power that *static fault trees* offer. For each of these original models, a static top value p_{stat} can be computed (by RiskSpectrum) characterizing the state-of-the-art failure frequency estimate of the analysed scenario. We obtained SD-FT models from these static ones by adding dynamic features offered by SD-FT formalism in a realistic manner. For all the dynamic basic events we use repair rate 0.1, which is approximately in the order of magnitude of real component repair rates. We use the static values p_{stat} as reference values for comparison in our experiments.

Simple Reactor Models. These models are variations of a toy example of a probabilistic safety assessment model of a boiling water reactor. We always calculate a core damage consequence, which is a typical Level 1 analysis with a 24 h time horizon. The size of these models is tiny relative to real-life models. Their common characteristics are presented in Table 1 (first row), the variants differ in the triggering logic:

Table 1. Model characteristics.

	BE	*DynE*	*TrigE*	*RelMCS*
Simple reactor	40	13	7	Various
Ind-1	3000	220	168	3164
Ind-2	2215	599	12	96042

TwoTrains models a system with two redundant trains of separate equipment, such as pumps, diesel engines, SWS (Service Water System), and CCW (Component Cooling Water System). The second train is triggered whenever the first pump fails;

Diesel is a system where the two diesel engines are redundant per train. One diesel engine is enough to make the respective train function properly;

SWS+Diesel adds redundancy for SWS systems in addition and similar to the diesel engine redundancy;

CCW+SWS+Diesel supports redundancy for CCW, SWS, and diesel engines.

Industrial-size Reactor Models. These are two slightly adapted core damage consequence analysis cases from two different real-life probabilistic safety assessment models. We will further refer to them as Ind-1 and Ind-2. Table 1 shows some of the core characteristics of the models. The most significant adaptations concern (1) switching off the common cause failure treatment and (2) updating failure data for some static basic events. We have added dynamic dependencies between components which in reality represent redundant systems (such as pumps) where only a subset of components has to run in order to guarantee the safety function. Triggering gates were chosen in a way that can be considered induced by

Table 2. Runtime experiments for simple reactor models performed with $err = 1$.

	T	T_{PRISM}	$RelMCS$	$AvDynE$	$AvTrigE$	$AvAdd$	$MaxSet$	$\#Set_{>8}$	[11]
TwoTrains	07:01	06:49	15061	4.8	0.1	0.2	15	818	>4 h
Diesel	30:04	29:53	10389	4.8	0.09	0.21	27	586	>4 h
SWS+Diesel	23:16	23:07	8007	4.8	0.09	0.20	27	501	>4 h
CCW+SWS+Diesel	15:42	15:34	5145	4.9	0.1	0.23	27	456	>4 h

a convenient modelling methodology. We chose gates corresponding to failures of complete systems and we did not simplify the logic under triggering gates by remodelling. All basic events with the mission time reliability model under the gates corresponding to the triggered systems were considered dynamic and triggered. Such a modelling requires only a high level understanding of dynamic relations between systems and components and knowledge about which gates model failures of these systems.

Experiments. In all the experiments we analyse a mission time of 24 h. The precision of time bounded reachability (computed by the PRISM tool) is set to 10^{-7}. In the tables presented, *AvDynE* (respectively *AvTrigE*) denotes the average amount of dynamic (respectively immediately triggered) events per cutset. When we report runtime, we use, unless otherwise stated, *min:sec* as format, and use T for overall runtime, and T_{PRISM} for the fragment thereof needed by PRISM. Value *AvAdd* denotes the average amount (over all cutsets C) of basic events, both static and dynamic, that have not been abstracted from FT_C (excluding the events from C itself). *MaxSet* refers to the maximum (over all cutsets) amount of basic events in a cutset tree that have been left after all abstractions, and $\#Set_{>8}$ shows the amount of cutsets, whose representative trees contain more than 8 basic events.

In order to evaluate our approach we use three measures: runtime, achieved accuracy and accuracy gain compared to a static analysis. To estimate the latter, we use the ratio of over-approximation $\overline{p}_{rea}$ to the value p_{stat} described above. This ratio can be expected to be lower than 1, since modelling the dynamics brings more accuracy and thus less pessimism. The runtime of the static analysis step is not reported. It was in the order of seconds for all experiments performed, given that the cutsets were precomputed by RISKSPECTRUM.

Influence of Model Parameters. We first want to estimate the effect of different parameters of the model itself on the running time of our algorithm. To do this we performed experiments on all the simple reactor models. These models share the same value of parameters *BE*, *DynE* and *TrigE* and differ mainly in their triggering logic. Each of the relevant cutsets contains at least one dynamic event. Table 2 summarizes the results of this experiment. As we can see, the existing algorithm from [11] is not competitive. The runtime of our algorithm is influenced by the maximum size of cutsets as well as the amount of large cutsets. More concretely, even though the amount of relevant cutsets for the

Table 3. Experiments with varying parameter *err* on TwoTrains, where $p_{stat} = 5.836344 \cdot 10^{-5}$.

err	*T*	$\underline{p}_{rea} \times 10^5$	$\overline{p}_{rea} \times 10^5$	*AvAdd*	*MaxSet*	$\#Set_{>8}$	$\overline{p}_{rea}/p_{stat}$
3	06:46	4.5747	4.6017	0.19	15	755	0.78
2	06:52	4.5769	4.6017	0.20	15	764	0.78
1	07:01	4.5848	4.6017	0.20	15	818	0.78
0.1	12:05	4.5927	4.6016	0.21	15	818	0.78
0.01	24:30	4.5961	4.6012	0.27	15	818	0.78
10^{-3}	38:10	4.5966	4.6012	0.34	15	818	0.78
10^{-4}	38:47	4.5966	4.6012	0.34	15	818	0.78
10^{-5}	38:46	4.5966	4.6012	0.34	15	818	0.78

Table 4. Experiments with varying parameter *err* for Ind-1, where $p_{stat} = 3.037881 \cdot 10^{-8}$.

err	*T*	$\underline{p}_{rea} \times 10^8$	$\overline{p}_{rea} \times 10^8$	*AvAdd*	*MaxSet*	$\#Set_{>8}$	$\overline{p}_{rea}/p_{stat}$
20	05:50	2.4790	2.5760	0.26	16	306	0.84
10	06:58	2.4790	2.4915	0.34	16	531	0.82
5	07:01	2.4790	2.4915	0.35	16	589	0.82
2	27:54	2.4798	2.4847	0.43	23	846	0.81
1	>6 h	2.4802	N/A	0.58	63	870	N/A

Table 5. Experiments with varying parameter *err* for Ind-2, where $p_{stat} = 7.342436 \cdot 10^{-7}$.

err	*T*(hrs:min:sec)	$\underline{p}_{rea} \times 10^7$	$\overline{p}_{rea} \times 10^7$	*AvAdd*	*MaxSet*	$\#Set_{>8}$	$\overline{p}_{rea}/p_{stat}$
20	02:16:10	4.8934	6.0541	0.05	14	103561	0.82
10	02:16:06	4.8934	6.0541	0.05	14	103561	0.82
5	03:01:27	4.8934	4.9301	0.1	14	107249	0.67
2	03:01:25	4.8934	4.9301	0.1	14	107249	0.67
1	03:01:27	4.8934	4.9301	0.1	14	107249	0.67

model TwoTrains is higher than for Diesel, the runtime on the latter model is notably higher due to the values *MaxSet* and $\#Set_{>8}$. As apparent from Table 2, the dominant portion of runtime is taken by the Prism processing. In further experiments we therefore do not report this value separately, and instead show only the overall running time of the algorithm.

Influence of Parameter err. Parameter *err* is the only parameter of the heuristic that we use for reductions. We performed various experiments to evaluate the effect of it on the running time and accuracy of our algorithm. Tables 3, 4 and 5

show results of the experiments on one of the simple models and on both the industrial-size models. One can see that, as expected, with the increase of accuracy (decrease of *err*) the amount of added basic events increases as well. This in turn enlarges the state space of the product CTMC, what explains the increase of the running time. On the other hand, the abstractions become more and more precise. We achieved a gain of 22 % on the simple model, 19 % on IND-1 and 33 % on IND-2 compared to the static value p_{stat}. In some cases higher precision seems to come with slightly lower running time, e.g. in Table 5. This however is an artefact of runtime measurement inaccuracy, the actual computations performed are identical.

5 Concluding Comparison with Related Work

We have presented a generic analysis and approximation scheme for fault trees combining static and dynamic features. The key innovation is the use of bounding approximations for the underlying dynamic behaviour. The method enables to trade precision against runtime in an effective manner, so as to make it an industrial-scale dynamic safety analysis method.

Other available methods for solving fault trees with dynamic features suffer from either scalability or expressiveness issues [11,15]. Approaches with comparable expressiveness include Dynamic Fault Trees [2,3], Boolean Driven Markov Processes [4] and others. Analysis support for these models is limited to fault trees with at most 300 dynamic basic events, which is far from the sizes that one usually encounters in the nuclear safety domain. We have reported here on successful experiments for models with up to 600 dynamic basic events contained inside SD-FTs with several thousands of basic events in total.

Acknowledgments. This work is partly supported by the ERC Advanced Investigators Grant 695614 (POWVER), by the EU 7th Framework Programme under grant agreement no. 318490 (SENSATION) and 288175 (CERTAINTY), by the DFG Transregional Collaborative Research Centre SFB/TR 14 AVACS, by the CDZ project 1023 (CAP), and by the Czech Science Foundation, grant No. P202/12/G061.

References

1. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. **29**(6), 524–541 (2003)
2. Dugan, B.J., Bavuso, S.J., Boyd, M.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans. Reliab. **41**(3), 363–377 (1992)
3. Boudali, H., Crouzen, P., Stoelinga, M.: A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE Trans. Dependable Sec. Compt. **7**(2), 128–143 (2010)
4. Bouissou, M., Bon, J.L.: A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. Reliab. Eng. Syst. Saf. **82**(2), 149–163 (2003)

5. Brázdil, T., Hermanns, H., Krčál, J., Křetínský, J., Řehák, V.: Verification of open interactive Markov chains. In: FSTTCS. LIPIcs, vol. 18, pp. 474–485 (2012)
6. Butkova, Y., Hatefi, H., Hermanns, H., Krcál, J.: Optimal continuous time Markov decisions. In: Finkbeiner, B., et al. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 166–182. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24953-7_12
7. Center for Chemical Process Safety: Guidelines for Hazard Evaluation Procedures, 3rd edn. Wiley, Hoboken (2008)
8. Fussell, J.B., Vesely, W.E.: A new methodology for obtaining cut sets for fault trees. Trans. Am. Nucl. Soc. **15**, 262–263 (1972)
9. IAEA: Development and Application of Level 1 Probabilistic Safety Assessment for Nuclear Power Plants, IAEA Safety Standards Series No. SSG-3 (2010)
10. IAEA: Development and Application of Level 2 Probabilistic Safety Assessment for Nuclear Power Plants, IAEA Safety Standards Series No. SSG-4 (2010)
11. Krčál, J., Krčál, P.: Scalable analysis of fault trees with dynamic features. In: DSN 2015, pp. 89–100 (2015)
12. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
13. Lloyd's Register Consulting: RiskSpectrum, Theory Manual (2013)
14. NASA: Fault Tree Handbook with Aerospace Applications (2002)
15. Ruijters, E.J.J., Stoelinga, M.I.A.: Fault tree analysis: a survey of the state of the art in modeling, analysis and tools. Comput. Sci. Rev. **15**, 29–62 (2015)
16. Vesely, W., Davis, T., Denning, R., Saltos, N.: Measures of risk importance and their application (NUREG/CR-3385). US Nuclear Regulatory Commission (1983)
17. Vesely, W., Goldberg, F., Roberts, N., Haasl, D.: Fault Tree Handbook(NUREG/CR-0492). US Nuclear Regulatory Commission (1981)
18. Wood, S., Smith, C.L., Kvarfordt, K.J., Beck, S.: Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE): Summary Manual (NUREG/CR-6952, vol. 1). US Nuclear Regulatory Commission (2008)