





# Scalable Reliability Analysis by Lazy Verification

Shahid Khan<sup>1</sup>  , Joost-Pieter Katoen<sup>1</sup> , Matthias Volk<sup>1</sup> ,  
and Marc Bouissou<sup>2</sup> 

<sup>1</sup> RWTH Aachen University, Aachen, Germany  
{shahid.khan,katoen,matthias.volk}@cs.rwth-aachen.de  
<sup>2</sup> Électricité de France, Palaiseau, France  
marc.bouissou@edf.fr

**Abstract.** This paper presents an iterative method to analyse system reliability models. The key idea is to analyse a partial state space of a reliability model in a conservative and an optimistic manner. By considering unexplored states as being always operational or, dually, already failed, our analysis yields sound upper- and lower-bounds on the system’s reliability. This approach is applied in an iterative manner until the desired precision is obtained. We present details of our approach for Boolean-logic driven Markov processes (BDMPs), an expressive fault tree variant intensively used in analysing energy systems. Based on a prototypical implementation on top of the probabilistic model checker STORM, we experimentally compare our technique to two alternative BDMP analysis techniques: discrete-event simulation obtaining statistical bounds, and a recent closed-form technique for obtaining pessimistic system lifetimes. Our experiments show that mostly only a fragment of the state space needs to be investigated enabling the reliability analysis of models that could not be handled before.

## 1 Introduction

*Reliability Analysis of Safety-Critical Systems.* Reliability analysis is concerned with analysing system models to determine measures-of-interest such as the mean-time-to-failure (MTTF) and the system’s reliability, i.e., the probability that the system is continuously operational up to a given mission time? Model-based analysis such as the numerical evaluation of Markov chains suffer from the state-space explosion problem. A possible remedy is discrete-event simulation. Simulation is applicable to a large class of reliability models, e.g., it supports general failure and repair rates, and has a low memory footprint as only the current model state needs to be kept in memory. Simulation results though come with statistical bounds only<sup>1</sup> and excessively many simulation runs are needed for rare events, events that happen with very low probability. Failures in safety-critical

---

S. Khan—supported by a HEC-DAAD scholarship.

<sup>1</sup> Known as confidence intervals.

systems such as autonomous cars, nuclear power plants, satellites, launchers, etc., are (supposed to be) rare events, and standards such as ISO 26262 require hard guarantees—safe lower- and upper-bounds.

*Lazy Verification.* The challenge is to come up with a reliability analysis technique that provides hard guarantees, can deal with rare events, and preferably provides results with numerical accuracy.

This paper proposes to use *lazy verification* for reliability analysis. The idea is conceptually simple: generate a *partial* state space of a reliability model description and carry out a fast analysis that takes a conservative and an optimistic perspective. By considering unexplored states as being always operational or, dually, already failed, the analysis yields a sound upper- and lower-bound ( $ub$  and  $lb$ ) on the system’s reliability, see Fig. 1. Fast analysis is done using the state-of-the-art probabilistic model-checking techniques [2, 4, 25] for continuous-time Markov models. If the gap between the lower and upper-bound is below an a priori user-defined tolerance  $\epsilon$ , i.e.,  $ub-lb \leq \epsilon$ , the analysis halts: the system reliability is certainly between these bounds. In case the results are not tight enough,  $ub-lb > \epsilon$ , the partial state space is extended with some unexplored states. This iterative approach thus has a *lazy* character: only a state-space fragment required to obtain the system’s reliability (or measures such as MTTF) with a given accuracy is generated and explored.

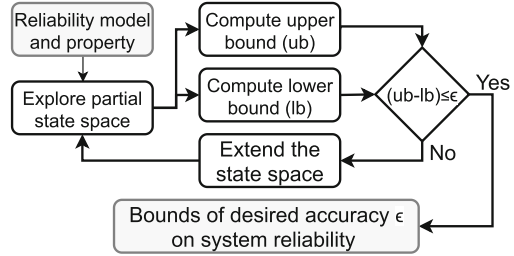


Fig. 1. Lazy verification for reliability

*Lazy Verification of BDMPs.* Our lazy approach is applicable to a wide range of *dynamic* reliability models, in particular those containing *state-dependent* failure mechanisms such as temporal orderings, spare management, and failure dependencies. This includes, e.g., dynamic fault trees [17], state-event fault trees [24], dynamic reliability block diagrams [16], and Pandora temporal fault trees [31]. Our lazy verification approach is also applicable to the *static* (non-state-dependent) reliability models such as static fault trees and reliability block diagrams. However, our approach will not be competitive to the binary decision diagram-based analysis of these models. We present details of our lazy verification approach for *Boolean logic-driven Markov processes* (BDMPs), an expressive dynamic fault tree variant extensively used by engineers at Électricité de France (EDF) [9]. EDF is one of the world’s largest producers of electricity and is active in technologies such as nuclear power, hydropower, wind power, solar and geothermal energy. BDMPs contain VOTing gates (generalisation of AND and OR gates), priority AND-gates, basic events that model system com-

ponents whose lifetime is exponentially distributed, instantaneous events, and two forms of triggers. While the general definition of BDMPs in [9] allows the use of arbitrary Markov processes for defining basic events, we restrict ourselves to the commonly used exponential distributions. The semantics of BDMPs has been translated into Markov automaton in [26], and generalised stochastic Petri nets in [27]. The underlying stochastic process of a BDMP is a continuous-time Markov chain (CTMC). Polynomial-time model-checking algorithms for computing lifetimes on CTMCs have been given [4] and are part of probabilistic model checkers such as PRISM [28] and STORM [22]. The main questions for lazy BDMP verification are “which fragment of the state space needs to be explored?” and “how much to extend a partial state space in an iteration of Fig. 1?” The first question is answered by a probabilistic criterion, i.e., the states with the highest reachability probabilities are selected for exploration. Regarding the second question, all one-step-successors of the selected state(s) are explored and the exploration is stopped based on an exploration threshold, e.g., if all remaining states in iteration  $i$  have reachability probability  $< 2^{-i}$ .

*Experimental Evaluation.* We implemented the BDMP lazy verification approach on top of the probabilistic model checker STORM [22] whose performance is among the best as witnessed in recent tool competitions (see [qcomp.org](http://qcomp.org)). Distinguishing features of STORM are its modular set-up enabling the rapid exchange of solvers, its facility to generate counterexamples, and its support for multiple modelling languages such as the reliability models dynamic fault trees and BDMPs. To validate our results, we compare to a free discrete-event simulation tool [7]. To indicate the “goodness” of the obtained bounds, we compare to initiator and all barriers (I&AB), a recent closed-form technique for obtaining pessimistic system reliability of BDMPs [11]<sup>2</sup>. We distinguish between *non-repairable* and *repairable* reliability models, as some analysis techniques perform better for a particular class. The main findings of the experimental evaluation:

- Exploring small state-space fragments mostly suffices, in particular for repairable models. This extends the findings of lazy verification of dynamic fault trees that do not include repairs, as in [29].
- Reliability model sizes that could not be handled before come within range.
- In contrast to I&AB, lazy verification is generally applicable (non-repairable and repairable models, arbitrary mission times, acyclic models, and models with loops), and provides sound *lower-* and *upper-bounds* within a given accuracy  $\varepsilon$ .

Note that solving reliability models is much more difficult when the model includes repairs, for several reasons:

- Merging failure states into a single state is not correct for repairs any more, because these states are usually associated with different repair rates.

<sup>2</sup> [https://www.lr.org/en/riskspectrum/support-and-downloads/#accordion-rsat3.4.5\\_released\(riskspectrumpsa1.4.0/rsat3.4.5\)\(15june2020\)](https://www.lr.org/en/riskspectrum/support-and-downloads/#accordion-rsat3.4.5_released(riskspectrumpsa1.4.0/rsat3.4.5)(15june2020)).

- The existence of repairs can create new states that cannot be reached from the initial state by a path containing only failures. This is exemplified by the case of the PAND-gate given in Sect. 2.
- The failure and repair rates usually differ by several orders of magnitude creating stiffness problems for solvers based on matrix calculations.

## 2 Boolean Logic-Driven Markov Processes

This section briefly explains the main principles of BDMPs; for more details we refer to [9]. BDMPs extend static fault trees with triggers and associate a pair of CTMCs with leaves to model various failure modes. Triggers model activation mechanisms that are useful to model dynamic failure dependencies, e.g., failure on-demand, mutual exclusion, and causal failure dependencies. Semantically, BDMPs augment the *failure* predicate of static fault trees with *activation* and *trimming* predicates. While activation predicates govern the activation status (active or dormant) of BDMP elements, the trimming predicates curtail the BDMP’s state space, e.g., by inhibiting the failure of non-failed inputs of OR-gates once the gate fails. (If such input is shared with other parts of the BDMP, then it is not pruned.) Fig. 2 depicts the main BDMP elements.

*Gates* (the first row of Fig. 2). A VOT-gate has two parameters: the number  $N$  of inputs and the number  $1 \leq K \leq N$  of inputs that need to fail for the VOT gate to fail. The AND- and OR-gates are special cases of VOT with  $K = N$  and  $K = 1$ , respectively. The priority-AND (PAND) gate fails once both its inputs fail in a left-to-right order. Simultaneous input failures of PAND do not lead to a gate failure. (Other fault tree variants use inclusive PAND [23].) Four repair strategies for PAND exist in BDMPs: (1) on the repair of the first input (*repair-first*), (2) on the repair of the second input (*repair-last*), (3) on the repair of both inputs (*repair-all*), or (4) on the repair of any input (*repair-any*).

*Basic events* (the second row of Fig. 2). The EXP-type basic events fail and are repaired following an exponential distribution. They can fail only in active mode; their repair is independent of their activation status. The STDBY-type basic events can switch between active and passive failure rates depending on their activation status. The INST-type basic events fail upon activation with probability  $\gamma$ . The repair behavior of both STDBY and INST is the same as for EXP. The failure of the top event TOP represents the system’s failure.

*Triggers and links* (the third row of Fig. 2). The trigger TRIG activates its target when its source fails and if at least one parent of TRIG’s target is active.

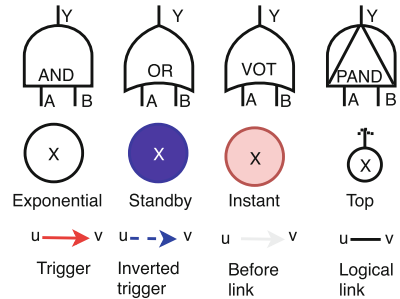
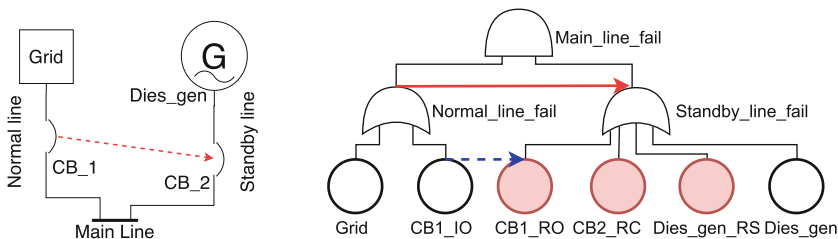


Fig. 2. BDMP elements

The inverted trigger *InvTRIG* achieves the inverse behavior, i.e., it activates the target node when at least one parent of the target node is active and the origin of *InvTRIG* is not failed. The *BeforeLINK* forces an order on simultaneously enabled *INST* leaves to reduce the number of combinations to be examined thanks to the trimming mechanism. The *LogicLINK* propagates the failure and activation status among *BDMP* elements.

*Example 1.* As a running example we consider a reconfiguration strategy adopted from [11]. The system, see Fig. 3 (left), has two power sources for the *Main\_line*: (1) a *Grid*, and (2) a diesel generator *Dies\_gen*. The red dotted line indicates the reconfiguration strategy. Initially, the *Normal\_line* is active and powers the *Main\_line*. On its failure, the load is switched to the *Standby\_line*. A repair will switch back the load to the *Normal\_line*. Reconfigurations are realised by circuit breakers; they can fail due to: (1) inadvertent failure during normal operation, (2) a refuse-to-open failure, or (3) a refuse-to-close failure. The latter two modes are failure on-demand as they happen while switching the load from the *Normal\_line* to *Standby\_line* and vice versa. *Dies\_gen* has two failure modes: failure in-operation, and failure on-demand. Whereas, *Grid* can only fail in-operation.

A *BDMP* model is shown in Fig. 3 (right). The red (blue) arrow represents *TRIG* (*InvTRIG*). The inverted trigger models mutually exclusive failure modes of the circuit breaker *CB1* (*CB1\_IO*) that is, the inadvertent opening of *CB1* (*CB1\_IO*) preempts its refuse-to-open failure mode *CB1\_RO*. In the *BDMP*, initially either *Grid* or *CB1\_IO* can fail. Any of these failures causes a failure of *Normal\_line\_fail*, which in turn activates *Standby\_line\_fail*. This also activates its children. A failure of an *INST* leaf causes the events *Standby\_line\_fail* and the *Main\_line\_fail* to fail. After a successful reconfiguration, *Dies\_gen* can fail. We point out that a failing sequence initiated by *CB1\_IO* does not lead to *CB1\_RO* being tested. For the sake of simplicity, the basic event *CB2\_IO* is omitted from the model. Moreover, *CB2\_RO* and *CB1\_RC* are omitted due to their negligible failure probability.



**Fig. 3.** Running example of a system (left) and a *BDMP* modeling it (right) (Color figure online)

### 3 Lazy Probabilistic Verification

This section introduces the partial state-space exploration for continuous-time Markov chains (CTMCs), the underlying model of BDMPs. CTMCs are finite transition systems with a designated initial state, whose transitions are labelled with rates (positive reals) of exponential distributions. For state  $s$  with transition rates  $R(s, u) = \lambda$  and  $R(s, v) = \mu$ , say, the probability to move from  $s$  to  $u$  is  $\lambda/\lambda+\mu$ , and the state residence time is random: the probability to stay for  $t$  time units in  $s$  equals  $1-e^{-(\lambda+\mu)\cdot t}$ . To enable CTMC analysis by model checking [4], states are labelled with sets of atomic propositions.

#### 3.1 State-Space Generation

The (compositional) semantics of BDMPs in terms of CTMCs is fully explained in [26,27]. We present the general idea by an example.

*Example 2 (CTMC for BDMP).* Consider again the BDMP in Fig. 3 (right). Figure 5 depicts a fragment of its corresponding CTMC. Initially, the BDMP elements *Grid* and *CB1\_IO* can fail. The initial state thus has two outgoing transitions labelled with the failure rates of *Grid* and *CB1\_IO*. The failure of *Grid* causes the testing of three independent Bernoulli trials through the trigger. Thus, the failure of *Grid* is succeeded by  $2^3 = 8$  probabilistic transitions. The failure of *CB1\_IO* causes two independent Bernoulli trials, as testing of *CB1\_RO* is inhibited by the inverted trigger. We combine these probabilistic transitions to the preceding exponential transitions. This gives rise to 12 outgoing transitions in the initial state, e.g.,  $s_1$  represents the scenario where *CB1\_IO* fails, but both instantaneous events *CB1\_RO* and *CB2\_RC* have not failed. The failure rate of *CB1\_IO* is 0.0001 and the failure probabilities of the two instantaneous events are 0.0001. We have a transition rate of  $0.0001 \cdot (1-0.0001)^2 = 9.998e^{-5}$ . We indicate the failure of component *CB1\_IO* by adding the corresponding label  $\{CB1\_IO\}$  to state  $s_1$ . The other transition rates are obtained similarly. For state  $s_1$ , there are two possibilities: either *CB1\_IO* is repaired, or *Dies\_gen* fails. The transitions have rates 0.1 and 0.0001, respectively.

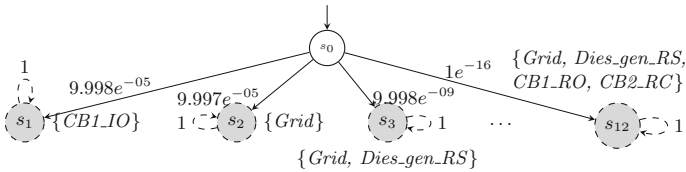


Fig. 4. State space after the first iteration

On completing the state-space generation process, the resulting CTMC is amenable to standard CTMC analysis techniques such as model checking [3].

This is supported by state-of-the-art model checkers such as STORM [22]. Time-bounded properties are important for reliability analysis. The *unreliability* is the probability of system failure within  $T$  time units. Expressed in a timed probabilistic temporal logic such as CSL [4]:

$$\mathbb{P}\left(\diamond^{[0, T]} \text{“failed”}\right),$$

where “failed” represents the failure of the top-level event in the BDMP. The *unavailability* is the probability that the system is failed at a given time point  $t$ :

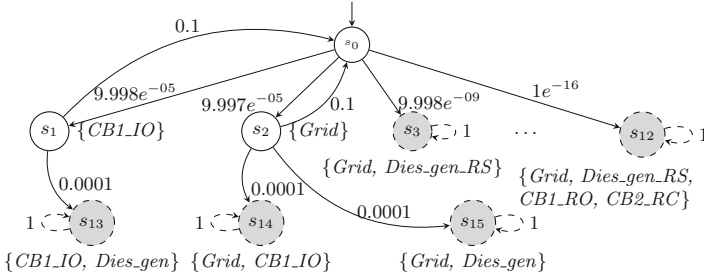
$$\mathbb{P}\left(\diamond^{[t, t]} \text{“failed”}\right).$$

### 3.2 Lazy Verification

While efficient probabilistic model-checking algorithms exist [21], the state-space explosion remains a limiting factor. This issue can partially be mitigated by using bisimulation minimization [5], but this requires the entire state space to be generated first. We resort to partial state-space generation. This is supported by STORM to analyse *non-repairable* dynamic fault trees [29] and yields safe upper- and lower-bounds of unreliability (and other measures). In this work, we generalise this to *repairable* models described by BDMPs.

The idea is to perform *iterative* state-space exploration. Let us explain this using Fig. 4. In each iteration, we explore a certain number of unexplored states—prioritized according to some heuristic. In our example of Fig. 4, we start by first exploring the initial state  $s_0$  which yields successor states  $s_1, \dots, s_{12}$ . State  $s_0$  is now explored (white), and all other states are discovered (gray). All discovered states are equipped with a self-loop in order to distinguish them from deadlock states. Deadlocks are attributed to modelling errors. The resulting partial state space is used to obtain safe lower and upper bounds. As both unreliability and unavailability use the label “failed” we pursue as follows. To obtain a *lower bound*  $lb$ , we mark all *explored* states that correspond to a system failure with the label “failed” and analyse the unreliability (or unavailability) of the resulting CTMC. For the *upper bound*  $ub$ , we additionally label all *discovered* states with label “failed” as well and analyse the resulting CTMC. For our example, we obtain the interval  $[0, 0.8646]$  for the unreliability. If  $ub-lb$  is less than a user-defined accuracy  $\epsilon$ , then the procedure terminates. Otherwise, it continues with the next iteration and explores more states. In our example, we continue exploration as indicated in Fig. 5. In the second iteration, we explore states  $s_1$  and  $s_2$  and obtain three new states. Note that now also “repair” transitions returning to state  $s_0$  are inserted. Computing the refined bounds on the extended state space yields  $[0, 0.0044]$ . (The nominal unreliability of this example is 0.0024).

*“Failed” Labels for Discovered States.* Note that we could have already added “failed” labels to many discovered states of the first iteration as they already represent system-level failure, e.g., state  $s_{12}$ . While this will help to significantly



**Fig. 5.** State space after the second iteration

tighten the bounds for the unreliability computation, it can give non-monotonic behavior for the unavailability. A discovered state which is marked as failed can never be left due to the self-loop. However, further exploration might introduce repair transitions such that non-failed states can be reached now. In such scenarios, the lower bound for the unavailability would decrease between iterations, which could yield unsound bounds. We, thus, keep our bound differences strictly non-increasing by applying a more conservative approach for discovered states.

*Exploration Heuristics.* The order in which states are explored and the threshold when to stop exploration for an iteration are determined by *exploration heuristics*. We use two types of heuristics in STORM: *depth-based* and *probability-based*. The *depth-based* heuristic explores the state space up to a predefined depth, i.e., distance to the initial state. This method is beneficial if one wants to analyse a system for a certain number of consecutive failures. The *probability-based* heuristic orders the states by their probability to eventually reach the state from the initial state. That way, we give priority to states which are more likely to occur and disregard unlikely events.

In the second iteration of our example in Fig. 5, we explored states  $s_1$  and  $s_2$ , because they have the highest incoming transition probabilities. We set the exploration threshold to  $10^{-5}$  for iteration 2. That means, the other states are not explored in this iteration, because their probabilities are below the threshold.

**Proposition 1.** *The lazy verification technique provides sound lower and upper bounds, i.e., the exact unreliability (or unavailability) lies within  $[lb, ub]$ .*

This can be seen as follows. The approximation accounts for both extremes. Either all of the discovered states lead to the failure (upper bound) or none of the discovered states lead to a failure (lower bound). Moreover, the difference between the upper and the lower bounds is strictly non-increasing for increasing iterations. The algorithm terminates at the latest when the complete state space is explored. The lower and upper bound then equal the exact result (up to the machine precision) as no discovered states are present anymore.



## 4 The Initiator and All Barriers Method

*Context.* The I&AB method [10, 11] is an approximation-based analysis approach. It aims to compute a conservative approximation of the BDMP's unreliability for a given mission time. It is included in the commercial RISK SPECTRUM PSA 1.4.0/RSAT 3.4.5 software tool to assess repairs in real-life nuclear probabilistic safety assessments (PSAs) (cf. footnote <sup>4</sup>). It is based on two assumptions:

- all standby redundancies become active upon the failure of an initiator, and
- the repair of initiator  $i$  inhibits the system failure (due to the sequence initiated by  $i$ ).

As the I&AB method relies on the repair of initiators, it is applicable to repairable systems only. It requires the repair rates to be at least ten times higher than the failure rates. We describe the four steps of the I&AB method and demonstrate each step on the sample BDMP in Fig. 3 (right):

*Step-1: Marking initiators and barriers.* The I&AB method starts by partitioning the set of basic events into *initiators* and *barriers*. Initiators are the basic events that can take the system out of its perfect state (aka: initial state). Barriers are basic events that get activated upon the failure of an initiator. Once an initiator fails, all non-failed initiators are—as barriers—activated.

*Example 3.* As *Grid* and *CB1\_IO* are the only basic events that can be active at the start, we mark them as initiators. All other basic events are barriers.

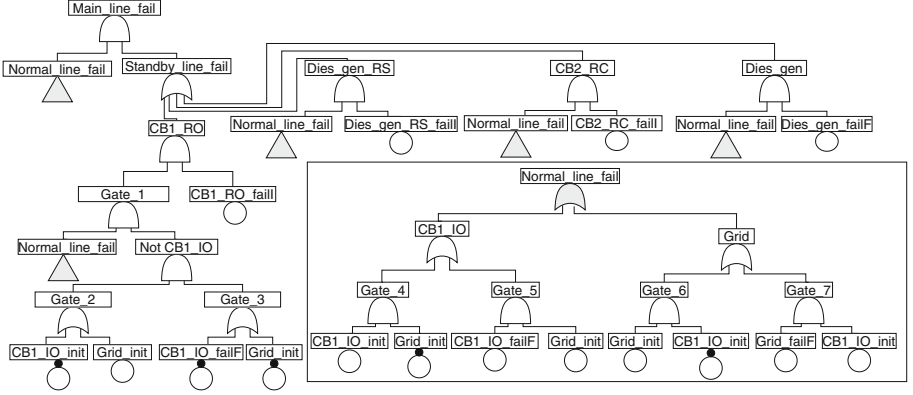
*Step-2: Generating minimal contents of sequences.* After declaring the initiators, the I&AB method computes the *set of minimal contents of sequences* (MCSS). This set contains sequences of failures—initiated by an initiator—that cause the top event to fail. Such sequence  $m$  is minimal in the sense that none of its proper prefixes causes a top event failure. To compute the MCSS, an SFT with NOT-gates is derived from the BDMP while respecting the triggers and precedence links. A PAND-gate is replaced by an AND-gate. The minimal cut sets<sup>3</sup> of the SFT computed using SCRAM<sup>4</sup> correspond to MCSS of the BDMP.

*Example 4.* Applying the proprietary EDF-tool KB3<sup>5</sup> to our running example yields the SFT of Fig. 6. The black circles above basic events represent negated literals. The two top layers of the SFT are as in the BDMP. The top event AND *Main\_line\_fail* has two inputs: *Normal\_line\_fail*, and *Standby\_line\_fail*. Consider the boxed fragment of the SFT. *Normal\_line\_fail* occurs when either *CB1\_IO* or *Grid* fails. Since they are initiators, their failures as initiators are mutually exclusive. This is captured by the negated literals, e.g., the left input of *CB1\_IO* fails if *CB1\_IO\_init* fails and *Grid\_init* does not. As *CB1\_IO* is an initiator and

<sup>3</sup> A cut set of an SFT is a set of basic events that cause the top event to fail.

<sup>4</sup> <https://github.com/rakhimov/scram>.

<sup>5</sup> <https://www.edf.fr/en/the-edf-group/who-we-are/activities/research-and-development/design-codes/design-code-kb3>.



**Fig. 6.** SFT generated for the running example

its mode of failure switches from *init* to *failF* (failure in-function) when *Grid\_init* fails, the right input of *CB1\_IO* has *Grid\_init* and *CB1\_IO\_failF* as inputs.

Now consider the right input of *Main\_line\_fail*, i.e., *Standby\_line\_fail*. The *Standby\_line\_fail* of Fig. 3 (right) has four components, and it is the target of a trigger. Correspondingly, four AND gates in the SFT capture the failure of each component. *Dies\_gen\_RS*, *CB2\_RC*, and *Dies\_gen* fail once the trigger source *Normal\_line\_fail* fails. The failure conditions of *CB1\_RO* are more involved, as it is a target of an inverted trigger. The right input of *CB1\_RO* depicts that *CB1\_RO\_failI* (*failI* is a failure on-demand) must fail for *CB1\_RO* to fail. Whereas the left input of *CB1\_RO* ensures that the trigger source, i.e., *Normal\_line\_fail* has failed and this failure is caused by *Grid*. The SFT's minimal cut sets are:

$\{CB1\_IO\_init, Dies\_gen\_failF\}$ ,  $\{CB1\_IO\_init, Dies\_gen\_RS\_failI\}$ ,  
 $\{CB1\_IO\_init,$   
 $CB2\_RC\_failI\}$ ,  $\{Grid\_init, CB1\_RO\_failI\}$ ,  $\{Grid\_init, CB2\_RC\_failI\}$ ,  
 $\{Grid\_init, Dies\_gen\_RS\_failI\}$ , and  $\{Grid\_init, Dies\_gen\_failF\}$ .

*Step-3: Computing minimal products.* The MCSS are arranged as *minimal products*, partial minimal sequences associated to initiators. It amounts to stripping initiator *ie* from sequence  $m \in MCSS$  and designating  $m \setminus ie$  as minimal product associated to *ie*. The minimal products are arranged as dictionary structure. The minimal products for MCSS are:

*CB1\_IO\_init*:  $\{\{Dies\_gen\_failF\}, \{Dies\_gen\_RS\_failI\}, \{CB2\_RC\_failI\}\}$  and  
*Grid\_init*:  $\{\{CB1\_RO\_failI\}, \{CB2\_RC\_failI\}, \{Dies\_gen\_RS\_failI\},$   
 $\{Dies\_gen\_failF\}\}$ .

*Step-4: Computing unreliability.* Based on minimal products, a closed-form formula approximates the BDMP's unreliability. To that end, the system failure distribution is assumed to be exponential, and the key is to approximate its failure rate  $\lambda_{eq}$ . For mission time *t*, the unreliability is approximated by  $1 - e^{-\lambda_{eq} \cdot t}$ .

The rate  $\lambda_{eq}$  is approximated as  $\sum_{ie \in \text{init}} \lambda_{ie} \cdot P_{ie}$ , where  $\text{init}$  is the set of initiators and  $P_{ie}$  is the probability of system failure due to minimal products associated to  $ie \in \text{init}$ . An upper bound is obtained by  $P_{ie} \leq \sum_{c=1}^k \bar{R}_c(\infty)$ , where  $k$  is the number of minimal products with initiator  $ie$  and  $\bar{R}_c(\infty)$  is the steady-state unreliability of a hypothetical parallel system composed of components  $\text{comp} \in c$ . Since BDMPs can have both exponential (failure in-function) and instantaneous (failure on-demand) failures, a minimal product  $c$  can be: (1) mixed type, i.e., both exponential and instantaneous type components, (2) exponential type, or (3) instantaneous type. Closed-forms for  $\bar{R}_c(\infty)$  for these types are given in [10]. As type (2) and (3) are special cases of type (1), we consider (1) in the following.

Using Murchland approximation [15] we have:  $P_{ie} \leq \sum_{c=1}^k E(N_c(\infty))$ , where  $E(N_c(\infty))$  is the expected number of system failures due to minimal product  $c$  within time interval  $(0, \infty)$ . Consider a minimal product  $c$  involving  $\ell$  instantaneous and  $m$  exponential components. Let  $\lambda_{c,i}$  denote the failure rate of the  $i^{\text{th}}$  exponential component,  $\gamma_{c,i}$  denote the failure probability of the  $i^{\text{th}}$  instantaneous component and  $\mu_{c,i}$  denote the repair rate of the  $i^{\text{th}}$  component. Using  $r_{c,i} = \lambda_{c,i} + \mu_{c,i}$  and  $\mu_c = \mu_{c,ie} + \sum_{j=1}^{\ell} \mu_{c,j}$ , where  $\mu_{c,ie}$  is the repair rate of initiator  $ie$ ,  $E(N_c(\infty))$  equals:

$$\overbrace{\prod_{i=1}^{\ell} \gamma_{c,i} \sum_{i=1}^m \lambda_{c,i} \left( \prod_{\substack{j=1 \\ j \neq i}}^m \frac{\lambda_{c,j}}{r_{c,j}} \int_0^{\infty} \underbrace{e^{-\sum_{k=1}^{\ell} \mu_{c,k} \cdot x} \sum_{\substack{j=1 \\ j \neq i}}^m (1 - e^{-r_{c,j}x}) dx}_{\text{expr}_1}} \right)}^{(INST)} - \prod_{j=1}^m \frac{\lambda_{c,j}}{r_{c,j}} \int_0^{\infty} \underbrace{e^{-\sum_{k=1}^{\ell} \mu_{c,k} \cdot x} \sum_{j=1}^m (1 - e^{-r_{c,j}x}) dx}_{\text{expr}_2}} \quad (1)$$

Intuitively,  $\text{expr}_1$  represents the probability that all components of  $c$  except component  $i$  failed. The term  $\text{expr}_2$  represents the probability that all components of  $c$  failed. Their difference is the contribution of component  $i$  to  $c$ 's failure probability. The repair of instantaneous components also contributes to these probabilities (these parts are identified by overbraces). Solving  $\text{expr}_2$  yields:

$$\frac{1}{\mu_c} - \sum_{i=1}^m \frac{1}{\mu_c + r_{c,i}} + \sum_{i=1}^m \sum_{j>i}^m \frac{1}{\mu_c + r_{c,i} + r_{c,j}} - \sum_{i=1}^m \sum_{j>i}^m \sum_{k>j}^m \frac{1}{\mu_c + r_{c,i} + r_{c,j} + r_{c,k}} + \dots + (-1)^m \frac{1}{\mu_c + \sum_{i=1}^m r_{c,i}}$$

The formula for  $\text{expr}_1$  is similar except that index  $i$  is ignored.

*Example 5.* In our running example, let the failure rate (probability) of all exponential (instantaneous) events be 0.0001, and the repair rate be 0.1. We have that  $c_1 = \{\text{Grid\_init} : \{\text{CB2\_RC\_failI}\}\}$  is instantaneous, and  $c_2 = \{\text{CB1\_IO\_init} : \{\text{Dies\_gen\_failF}\}\}$  is exponential. For  $c_1$  and  $c_2$  we obtain:

$$P_{\text{Grid\_init}}(c_1) = \lambda_{\text{Grid}} \cdot \gamma_{\text{CB2\_RC\_failI}} = 1 \cdot 10^{-8}, \text{ and} \\ P_{\text{CB1\_IO\_init}}(c_2) = \lambda_{\text{CB1\_IO}} \cdot E(N_{c_2}(\infty)) = 9.995 \cdot 10^{-8},$$

where for  $\lambda_1 = \lambda_{\text{Dies\_gen\_failF}}$ ,  $\mu_1 = \mu_{\text{Dies\_gen\_failF}}$ ,  $\mu = \mu_{\text{CB1\_IO}}$ ,  $r_1 = \lambda_1 + \mu_1$ :

$$E(N_{c_2}(\infty)) = \lambda_1 \cdot ((1/\mu) - (\lambda_1/\mu_1 \cdot (1/\mu - 1/(\mu_1 + r_1))).$$

As there are two exponential and five instantaneous components, all of the same cardinality and equal reliability parameters:

$$\lambda_{eq} = 5 \cdot (1 \cdot 10^{-8}) + 2 \cdot (9.995 \cdot 10^{-8}) = 2.499 \cdot 10^{-7}.$$

The system unreliability for mission time  $t = 10,000$  thus is:  $1 - e^{-t \cdot \lambda_{eq}} = 0.0025$ . (The nominal unreliability of this example is 0.0024, see Sect. 3.2.)

## 5 Case Studies

We first tested our implementation on 24 BDMP test cases available online<sup>6</sup>. Though useful as a sanity check for our implementation, the test cases are too small to meet our objective: investigating scalable analysis on real-world case studies. There is only one larger test case in the literature: a power supply of a nuclear power plant [8], so other test cases were obtained by a semi-automatic translation from DFTs to BDMPs using the approach outlined in [6]. We selected DFTs from the online DFT repository<sup>7</sup>, and added repairs. This yields:

**Dual Processor Reactor Regulation System.** This DFT models a power regulator in a nuclear reactor [18]. The model available on the benchmark website is non-repairable and we adopted the repair rates from the original paper [18].

**Emergency Power Supply of Nuclear Power Plant.** This BDMP is given by EDF as a challenge for BDMP analysis tools [8]. The repairable model captures most of the scenarios encountered while analysing complex dynamic systems.

**Railway Crossing.** This DFT models a railway level crossing [20]. We consider variant *RC\_5\_5\_sc* in this paper. It consists of 5 barriers with independent motors, 5 groups of sensors, and the controller is modeled as a single basic event.

**Vehicle Guidance Case Study.** These DFTs stem from an industrial case study on safety concepts from the automotive domain [19]. We extended the original (non-repairable) DFTs by adding repair rates of 0.1 to all basic events. We consider all eight variants of this test case.

**Railway Station.** These DFTs model the influence of infrastructure failures in German railway station areas on the routability of trains [30]. We consider all six variants and made them repairable by adding repair rates of 0.1 to all basic events. Note that this model is not yet available on the benchmark website.

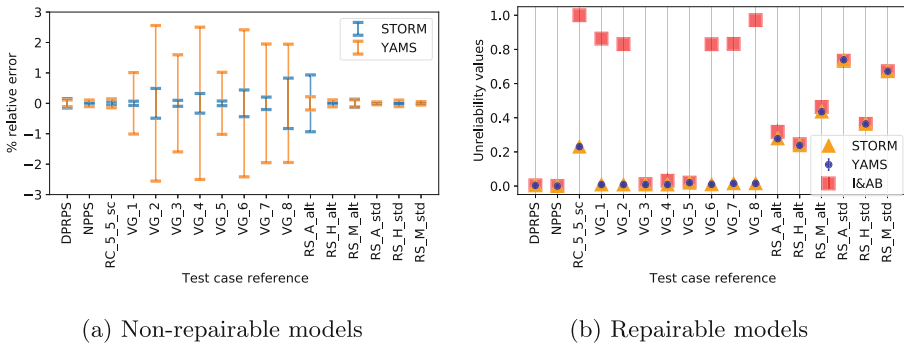
**Sensor Filter.** This DFT is automatically synthesized from an AADL description using the Compass tool-chain [13]. We use variant *sf\_3\_2* modelling a network consisting of three sensors and two filters.

<sup>6</sup> <https://sourceforge.net/projects/visualfigaro/files/Doc.and.examples/>.

<sup>7</sup> <https://dftbenchmarks.utwente.nl/>.

## 6 Results and Discussion

We implemented the incremental verification (Sect. 3.2) in the probabilistic model checker STORM [22], called STORM-APPROX. We also implemented the I&AB method (Sect. 4). We validated the outcomes of our open-source prototypical implementation with the commercial RISK SPECTRUM tool<sup>8</sup>. While the results coincide for both implementations, our implementation is not competitive in terms of computation time and scalability. (We calibrated our implementation using [8] as the RISK SPECTRUM results were online for this test case [12].) We compare both approaches with the Monte Carlo simulator YAMS [7] and evaluate both repairable and non-repairable BDMPs. We use the trimming feature [9]—which reduces the state space—to be able to treat large BDMPs. All the results discussed in this Section are available online<sup>9</sup>.



**Fig. 7.** Accuracy of results computed by STORM, YAMS and I&AB

### 6.1 Accuracy

*Non-repairable Models.* As the I&AB method is not applicable to non-repairable systems, we only compare YAMS and STORM-APPROX and give the results in Fig. 7a. Since the absolute deviations between both tools are negligible, we plot the % deviations relative to the respective mean value. YAMS provides a *mean value* and a confidence interval  $[\ell, u]$ . In Fig. 7a, the extremities of segments correspond to  $\frac{\ell - u}{2 \cdot \text{mean\_value}} \cdot 100$  and  $\frac{u - \ell}{2 \cdot \text{mean\_value}} \cdot 100$ . STORM-APPROX gives an *upper\\_bound* and a *lower\\_bound*. We compute % error bounds by using  $\frac{\text{upper\_bound} - \text{mean\_value}}{\text{mean\_value}} \cdot 100$  and  $\frac{\text{lower\_bound} - \text{mean\_value}}{\text{mean\_value}} \cdot 100$  where *mean\\_value* is the middle of the obtained bounds. Since we used  $10^7$  simulations for each test case with YAMS, the width of the confidence interval depends on the probability

<sup>8</sup> <https://www.lr.org/en/riskspectrum/>.

<sup>9</sup> <https://github.com/moves-rwth/dft-bdmp>.

**Table 1.** Computation time and state spaces statistics

Test case	BDMP			Non-repairable				Repairable				
	#BE	#Gates	#Trig.	State space		Comp. time		State space		Comp. Time		I& AB*
				#States	#Trans.	STORM	YAMS	#States	#Trans.	STORM	YAMS	
DPRPS	40	14	1	2.7 K	5.9 K	0.54 s	30 m	2.7 K	11.5 K	1.25 s	3.28 h	1.79 s
	(Non-trimmed version)			23.6 M	50 M	4.33 h	42.2 m	67.6 K	0.22 M	59 s	3.16 h	–
NPPS	81	54	12	10.3 M	21 M	9.5 h	3.8 h	45.5 M	96.6 M	11 m	3.76 h	11.3 m
RC.5.5_sc	41	28	25	12.3 M	35.9 M	3.6 h	2.4 h	5.3 M	18.2 M	1.7 h	4.7 h	0.52 s
VG_1	73	65	35	0.68 M	1.7 M	18 m	17 m	99.8 K	0.20 M	1.8 m	1.2 h	2.75 s
VG_2	67	63	31	0.30 M	75 M	5.6 m	7 m	39.5 K	82.6 K	33.4 s	41.8 m	11.14 s
VG_3	54	50	24	0.14 M	0.38 M	2.5 m	8.2 m	26.9 K	55.9 K	19.2 s	37.3 m	1.2 s
VG_4	54	52	25	37.2 K	86 K	30.8 s	5 m	4.9 K	10.1 K	3.2 s	10.7 m	1.2 s
VG_5	55	53	27	12.3 K	32.3 K	12.9 s	5.5 m	3.1 K	6.3 K	2.4 s	14 m	0 s
VG_6	61	58	21	0.11 M	0.29 M	2.1 m	6.5 m	37.3 K	78.2 K	30.6 s	38.5 m	0.52 s
VG_7	87	80	38	4.5 M	11.3 M	1.6 h	10.3 m	0.16 M	0.32 M	2.7 m	50.3 m	0 s
VG_8	99	95	44	18.9 M	45.8 M	8.8 h	13 m	0.87 M	1.8 M	18.5 m	3.45 h	0.31 s
RS.A.alt	556	531	111	15.6 M	20.6 M	18.2 h	8.5 h	7.0 M	19 M	11.2 h	8.45 h	2 h
RS.H.alt	194	161	38	18.6 K	39.2 K	25.6 s	2 h	3.8 K	8.9 K	6.8 s	7.5 h	4.18 s
RS.M.alt	520	492	104	8.5 M	11 M	8.6 h	8.8 h	0.87 M	2.18 M	1.4 h	8.8 h	1.2 h
RS.A.std	544	466	108	113	224	0.93 s	5.3 h	1134	224	1.03 s	5.3 h	3.8 m
RS.H.std	184	142	41	41	80	0.1 s	1.1 h	41	80	0.116 s	6.8 h	1.82 s
RS.M.std	450	338	90	91	180	0.56 s	3.93 h	91	180	0.705 s	8.8 h	468 s

\* I&AB computation time includes cut set computation and quantification times.

to be estimated. Figure 7a indicates that the magnitude of the unreliability values of both tools coincides. However, *the STORM bounds are significantly tighter than the simulation bounds of YAMS* with the notable exception of test case *RS.A.alt*. For this test case, STORM-APPROX took 18 h and explored 15.6 M states with 20.6 M transitions, cf. Table 1. This means that the number of states required to compute tighter bounds is too big to explore within the time-limit of 24 h.

*Repairable Models.* Figure 7b presents the (absolute) unreliability values obtained by STORM, YAMS and I&AB. The results of YAMS and STORM coincide. However, the results of the I&AB method are very pessimistic for six benchmarks. This is mostly due to the fact that I&AB does not work if there are looped interactions of structure functions through triggers, i.e., cyclic dependencies. Such loops inhibit the generation of SFTs for MCSS computation. One must manually break the loops by modifying the activation conditions of some instantaneous type basic events. The BDMPs of *VG*-variants contain such loops making them inappropriate for processing with I&AB. This yields trivial bounds.

## 6.2 Computation Time

Table 1 gives a detailed account of the model statistics and tool performances. For each test case, we list the number of BDMP elements. For both repairable and non-repairable variants, we give the explored state space sizes of STORM and the computation times required by the three tools. The error bound of STORM-APPROX is set to  $10^{-3}$  and the number of simulations for YAMS is  $10^7$ .

The state-space sizes might seem small at first glance, but they are the result of enabling trimming for the BDMP and partial exploration of these trimmed

state spaces. The effect of trimming can be seen for the DPRPS test case, where the state space without trimming is several orders of magnitudes larger.

The computation times of STORM and I&AB are within a few minutes for most of the repairable test cases whereas YAMS requires significantly more computation time. Note that some of these times could be considerably reduced by adapting the number of simulations to the probability to be calculated. The performance of STORM and I&AB is comparable for the test cases where I&AB returns accurate results. For *RS\_M\_alt*, I&AB is 5 times faster than STORM, whereas STORM is faster on *RS\_A\_std*. Note that the timings for STORM are not directly correlated to the state space sizes, compare, e.g., *NPPS* and *RC\_5\_5\_sc*. As failure and repair rates typically differ by at least one order of magnitude, computing unreliability becomes expensive due to the *stiffness* of the CTMC.

The I&AB method requires more than one hour for *RS\_A\_alt* and *RS\_M\_alt*. For both BDMPs, a high number of cut sets needs to be computed. We address this issue using SCRAM by limiting the cardinality of a cut set. But by doing so we introduce an error that cannot be mastered. The implementation of I&AB in RISK SPECTRUM would be much faster and yet more precise, as it is able to eliminate cut sets with probabilities lower than a given threshold during their generation.

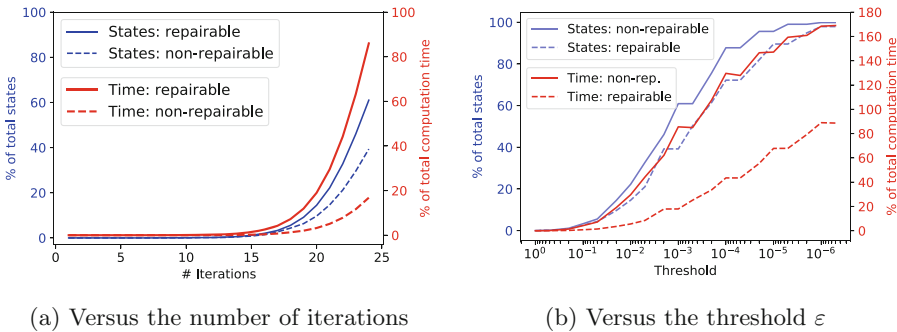


Fig. 8. Percentage of explored state space and computation time

### 6.3 Gained Insights on Lazy Verification

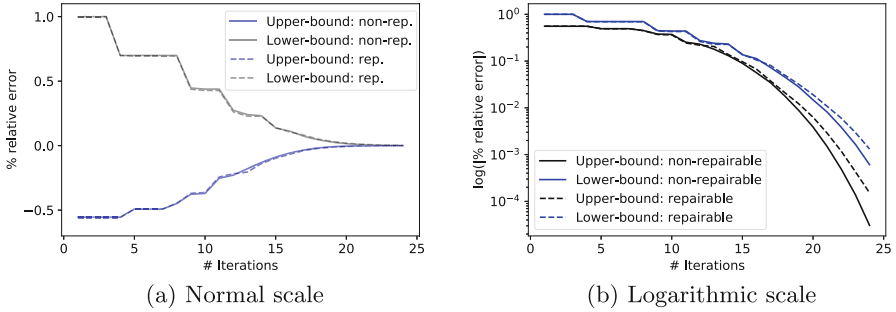
We provide insights into the lazy verification technique using the sensor filter *sf\_3\_2* test case. The observed trends for this benchmark are representative for the other benchmarks as well. The exhaustive non-trimmed state-space of the non-repairable (repairable) version consists of 5.19 M (5.19 M) states and 14 M (66 M) transitions and was verified in 10 m (56 m). The (approximate) trimmed state-space for non-repairable (repairable) case consists of the same number of states and 14 M (50 M) transitions and verified in 18 m (50 m).

*State Space Coverage Versus Number of Iterations.* The first trend we studied

is the percentage of the total state space explored in each iteration and the total computation time up to this iteration. The trends for both repairable and non-repairable models are shown in Fig. 8a. Both graphs are plotted for an accuracy of  $10^{-3}$ . Interestingly, the repairable model converges faster as compared to the non-repairable version and explores fewer states. One reason is the fact that while exploring the state space, we enter the region which does not lead to system failure but both our lazy verification and BDMP trimming are agnostic of this. Such region can exist due to, e.g., fail-safe behavior of a PAND-gate.

*State Space Coverage Versus Accuracy.* The next trend we studied is the dependency of the state space coverage on the accuracy. Figure 8b shows that a larger percentage of the state-space is explored for increasing accuracy. After the threshold of  $10^{-3}$ , the time for approximating the non-repairable model exceeds the verification time of the exact approach, i.e., 10 m. However, for the repairable case, the computation time is less than that required to fully explore the state space. This implies that our approximation technique, combined with the trimming feature, is more efficient for analyzing repairable systems.

*Convergence of Approximate Results.* Figure 9 plots the % relative errors of upper and lower bound to the exact value. We see that the bounds converge faster for increasing number of iterations and the error is negligible after 20 iterations.



**Fig. 9.** Iteration versus bound convergence

## 7 Conclusion

We presented an iterative verification procedure for BDMPs based on partial state-space exploration. Our evaluation shows that this approach allows scalable analysis of BDMPs while providing sound upper and lower bounds of the exact result. Our technique is applicable to other dynamic reliability models too. Lazy verification is a promising approach and we plan to further improve it by developing better exploration heuristics, e.g., using learning techniques [1, 14].



## References

1. Ashok, P., Butkova, Y., Hermanns, H., Křetínský, J.: Continuous-time Markov decisions based on partial exploration. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 317–334. Springer, Cham (2018). [https://doi.org/10.1007/978-3-030-01090-4\\_19](https://doi.org/10.1007/978-3-030-01090-4_19)
2. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 963–999. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-10575-8\\_28](https://doi.org/10.1007/978-3-319-10575-8_28)
3. Baier, C., Hahn, E.M., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Model checking for performability. *Math. Struct. Comput. Sci.* **23**(4), 751–795 (2013)
4. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov Chains. *IEEE Trans. Softw. Eng.* **29**(6), 524–541 (2003)
5. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
6. Bouissou, M.: A generalization of dynamic fault trees through Boolean logic driven markov processes (BDMP). In: Proceedings of the 16th European Safety and Reliability Conference (ESREL) (2007)
7. Bouissou, M.: A simple yet efficient acceleration technique for Monte Carlo simulation. In: Proceedings of the 22nd European Safety and Reliability Conference (ESREL), pp. 27–36 (2013)
8. Bouissou, M.: A benchmark on reliability of complex discrete systems: emergency power supply of a nuclear power plant. [arXiv:1703.06575](https://arxiv.org/abs/1703.06575) (2017)
9. Bouissou, M., Bon, J.L.: A new formalism that combines advantages of fault-trees and Markov models: Boolean logic Driven Markov Processes. *Rel. Eng. Sys. Safety* **82**(2), 149–163 (2003)
10. Bouissou, M., Hernu, O.: Boolean approximation for calculating the reliability of a very large repairable system with dependencies among components. In: Proceedings of the 25th European Safety and Reliability Conference (ESREL) (2016)
11. Bouissou, M., Hernu, O.: Estimation de la fiabilité d'un système industriel. French Patent FR3044787A1, June 2017. <https://worldwide.espacenet.com/patent/search/family/056321980/publication/FR3044787A1?q=FR3044787>
12. Bouissou, M., Khan, S., Katoen, J., Krcál, P.: Various ways to quantify BDMPs. In: MARS@ETAPS. EPTCS, vol. 316, pp. 1–14 (2020)
13. Bozzano, M., Cimatti, A., Katoen, J.P., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, dependability and performance analysis of extended AADL models. *Comput. J.* **54**(5), 754–775 (2011)
14. Brázdil, T., et al.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-11936-6\\_8](https://doi.org/10.1007/978-3-319-11936-6_8)
15. Collet, J., Bruyère, F.: An efficient tool for taking repairs into account in Boolean Models. In: Probabilistic Safety Assessment and Management, vol. 4 (1998)
16. Distefano, S., Puliafito, A.: Dynamic reliability block diagrams vs dynamic fault trees. In: Annual Reliability and Maintainability Symposium (RAMS), pp. 71–76. IEEE (2007)
17. Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Trans. Rel.* **41**(3), 363–377 (1992)

18. Durga Rao, K., Gopika, V., Sanyasi Rao, V., Kushwaha, H., Verma, A., Srividya, A.: Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment. *Reliab. Eng. Syst. Saf.* **94**(4), 872–883 (2009)
19. Ghadhab, M., Junges, S., Katoen, J.P., Kuntz, M., Volk, M.: Safety analysis for vehicle guidance systems with dynamic fault trees. *Reliab. Eng. Syst. Saf.* **186**, 37–50 (2019)
20. Guck, D., Katoen, J.P., Stoelinga, M.I., Luiten, T., Romijn, J.: Smart railroad maintenance engineering with stochastic model checking. In: *Proceedings of Railways*, pp. 950–953. Saxe-Coburg Publications (2014)
21. Hahn, E.M., et al.: The 2019 comparison of tools for the analysis of quantitative formal models. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) *TACAS 2019*. LNCS, vol. 11429, pp. 69–92. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-17502-3\\_5](https://doi.org/10.1007/978-3-030-17502-3_5)
22. Hensel, C., Junges, S., Katoen, J.P., Quatmann, T., Volk, M.: The probabilistic model checker storm. *CoRR* abs/2002.07080 (2020). <https://arxiv.org/abs/2002.07080>
23. Junges, S., Guck, D., Katoen, J.P., Stoelinga, M.: Uncovering dynamic fault trees. In: *DSN*, pp. 299–310. IEEE Computer Society (2016)
24. Kaiser, B., Gramlich, C., Förster, M.: State/event fault trees - a safety analysis model for software-controlled systems. *Reliab. Eng. Syst. Saf.* **92**(11), 1521–1537 (2007)
25. Katoen, J.P.: The probabilistic model checking landscape. In: *LICS*, pp. 31–45. ACM (2016). <https://doi.org/10.1145/2933575.2934574>
26. Khan, S., Katoen, J.-P., Bouissou, M.: A compositional semantics for repairable BDMPs. In: Casimiro, A., Ortmeier, F., Bitsch, F., Ferreira, P. (eds.) *SAFECOMP 2020*. LNCS, vol. 12234, pp. 82–98. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-54549-9\\_6](https://doi.org/10.1007/978-3-030-54549-9_6)
27. Khan, S., Katoen, J.P., Bouissou, M.: Explaining Boolean-logic driven Markov processes using GSPNs. In: *EDCC*, pp. 119–126. IEEE (2020)
28. 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). [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)
29. Volk, M., Junges, S., Katoen, J.P.: Fast dynamic fault tree analysis by model checking techniques. *IEEE Trans. Ind. Inf.* **14**(1), 370–379 (2018)
30. Volk, M., Weik, N., Katoen, J.-P., Nießen, N.: A DFT modeling approach for infrastructure reliability analysis of railway station areas. In: Larsen, K.G., Willemse, T. (eds.) *FMICS 2019*. LNCS, vol. 11687, pp. 40–58. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-27008-7\\_3](https://doi.org/10.1007/978-3-030-27008-7_3)
31. Walker, M.D.: Pandora: a logic for the qualitative analysis of temporal fault trees. Ph.D. dissertation, University of Hull, Kingston upon Hull, UK (2009)