International Journal on Software Tools for Technology Transfer (2022) 24:821–841
https://doi.org/10.1007/s10009-022-00675-x

GENERAL

Special Issue: TACAS 2020

Analysis of non-Markovian repairable fault trees through rare event simulation

Carlos E. Budde[1] · Pedro R. D'Argenio[2,3,4] · Raúl E. Monti[5] · Mariëlle Stoelinga[5,6]

Accepted: 29 August 2022 / Published online: 4 November 2022

Abstract

Dynamic fault trees (DFTs) are widely adopted in industry to assess the dependability of safety-critical equipment. Since many systems are too large to be studied numerically, DFTs dependability is often analysed using Monte Carlo simulation. A bottleneck here is that many simulation samples are required in the case of rare events, e.g. in highly reliable systems where components seldom fail. Rare event simulation (RES) provides techniques to reduce the number of samples in the case of rare events. In this article, we present a RES technique based on importance splitting to study failures in highly reliable DFTs, more precisely, on a variant of repairable fault trees (RFT). Whereas RES usually requires meta-information from an expert, our method is fully automatic. For this, we propose two different methods to derive the so-called importance function. On the one hand, we propose to cleverly exploit the RFT structure to compositionally construct such function. On the other hand, we explore different importance functions derived in different ways from the minimal cut sets of the tree, i.e., the minimal units that determine its failure. We handle RFTs with Markovian and non-Markovian failure and repair distributions—for which no numerical methods exist—and implement the techniques on a toolchain that includes the RES engine FIG, for which we also present improvements. We finally show the efficiency of our approach in several case studies.

Keywords Rare event simulation · Fault tree analysis · Statistical model checking · System reliability

1 Introduction

Reliability engineering is an important field that provides methods and tools to assess and mitigate the risks related to complex systems. *Fault tree analysis* (FTA) is a prominent technique here. Its application encompasses a large number of industrial domains that range from automotive and aerospace system engineering to energy and telecommunication systems and protocols. *Fault trees.* A *fault tree* (FT) describes how component failures occur and propagate through the system, eventually generating system-wide failures. Technically, an FT is a directed acyclic graph whose leaves model component failures and whose other nodes

The authors are listed in alphabetical order. This work was partially supported by the EU Grant Agreement 101008233 (MISSION), ANPCyT PICT-2017-3894 (RAFTSys), and SeCyT project 33620180100354CB (ARES). Funded also by the EU Grant Agreement 101067199 (ProSVED). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or The European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

✉ Carlos E. Budde
carlosesteban.budde@unitn.it

Pedro R. D'Argenio
pedro.dargenio@unc.edu.ar

Raúl E. Monti
r.e.monti@utwente.nl

Mariëlle Stoelinga
m.i.a.stoelinga@utwente.nl

[1] Department of Information Engineering and Computer Science, University of Trento, Trento, Italy

[2] FAMAF, Universidad Nacional de Córdoba, Córdoba, Argentina

[3] CONICET, Córdoba, Argentina

[4] Department of Computer Science, Saarland University, Saarbrücken, Germany

[5] Formal Methods and Tools, University of Twente, Enschede, The Netherlands

[6] Department of Software Science, Radboud University, Nijmegen, The Netherlands

(called gates) model failure propagation. Using fault trees, one can compute dependability metrics to quantify how a system fares w.r.t. certain performance indicators. Two common metrics are system *reliability*—the probability that there are no system failures during a given mission time—and system *availability*—the average percentage of time that a system is operational. *Static fault trees* (also known as standard FTs) contain a few basic gates, like AND and OR gates. This makes them easy to design and analyse but also limits their expressivity. *Dynamic fault trees* (DFTs [26,57]) are a common and widely applied extension of standard FTs, catering for more complex dependability patterns, like spare management and causal dependencies. Such gates make DFTs more difficult to analyse. In static FTs, it only matters whether or not a component has failed, so they can be analysed with Boolean methods, such as binary decision diagrams [38]. Dynamic fault trees, on the other hand, crucially depend on the failure order, so Boolean methods are insufficient. Moreover, and on top of these two classes, *repairable fault trees* (RFT [7]) permit components to be repaired after they have failed. Repairs are not only crucial in fault-tolerant and resilient systems, they are also an important cost driver. Hence, repairable fault trees allow one to compare different repair strategies with respect to various dependability metrics. In this article we consider repairable fault trees. *Fault tree analysis.* The reliability and availability of a fault tree can be computed via numerical methods, such as probabilistic model checking. This involves exhaustive explorations of state-based models such as interactive Markov chains [54]. Since the number of states (i.e. system configurations) is exponential in the number of tree elements, analysing large trees remains a challenge today [1,38]. Moreover, numerical methods are usually restricted to exponential failure rates and combinations thereof, like Erlang and acyclic phase-type distributions [54].

Alternatively, fault trees can be analysed using standard Monte Carlo simulation, which embedded in formal system modelling is typically called *statistical model checking* (SMC [30,52,54]). Here, a large number of simulated system runs (*samples*) are produced. Reliability and availability are then statistically estimated from the resulting sample set. Such sampling does not involve storing the full state space so, although the result provided can only be correct with a certain probability, SMC is much more memory efficient than numerical techniques. Furthermore, SMC is not restricted to exponential probability distributions.

However, a known bottleneck of SMC are rare events: when the event of interest has a low probability, which is typically the case in highly reliable systems, millions of samples may be required to observe the event. It is a well-known limitation of SMC that producing these samples can take an unacceptably long simulation time.

Rare event simulation. To alleviate this problem, the field of *rare event simulation* (RES) provides techniques that reduce the number of samples required to produce a useful estimate [49]. These techniques can be classically categorised as importance sampling and importance splitting.

Importance sampling means that the method will tweak the probabilities in the model, then compute the metric of interest for the changed system, and finally adjust the analysis results to the original model [33,47]. Unfortunately, this approach has specific requirements on the stochastic model: in particular, it is generally applicable to models with exponential probability distributions only.

Importance splitting, deployed in this paper, does not have this limitation. Importance splitting relies on rare events that arise as a sequence of less rare intermediate events [3,39]. It exploits this fact by generating more (partial) samples on paths where such intermediate events are observed.

As a simple example, consider a biased coin whose probability of heads is $p = \frac{1}{80}$. Suppose we flip it eight times in a row, and say we are interested in observing at least three heads. If head comes up at the first flip (H), then we are on a promising path. We can then clone (*split*) the current path H, generating e.g. 7 copies of it, each clone evolving independently from the second flip onwards. Say one clone observes three heads—the copied H plus two more. Then, this observation of the rare event (three heads) is counted as $\frac{1}{7}$ rather than as one observation to account for the splitting where the clone was spawned. Now, if a clone observes a new head (HH), this is even more promising than H, so the splitting can be repeated. If we make five copies of the HH clone, then observing three heads in any of these copies counts as $\frac{1}{35} = \frac{1}{7} \cdot \frac{1}{5}$. Alternatively, observing tails as second flip (HT) is less promising than heads. One could then decide not to split such a path.

This example highlights a key ingredient of importance splitting: the *importance function*, which indicates for each state how promising it is w.r.t. the event of interest. This function, as well as other parameters such as thresholds [27], are used to choose e.g. the number of clones spawned when a simulation run visits certain state. An importance function for our example could be the number of heads seen thus far. Another one could be such a number, multiplied by the number of coin flips yet to come. The goal is to give *higher importance* to states from which observing the *rare event is more likely*. The efficiency of an importance splitting implementation increases as the importance function better reflects such property.

Rare event simulation has been successfully applied in several domains [5,6,48,59,60,65]. However, a key bottleneck is that it critically relies on expert knowledge. In particular, for importance splitting, finding a good importance function is a well-known, highly non-trivial task [36,49].

Our contribution: rare event simulation for fault trees. This article presents an importance splitting method to analyse RFTs. In particular, we automatically derive the importance function by exploiting the description of a system as a fault tree. This is crucial, since the importance function is normally given manually in an ad hoc fashion by a domain or RES expert. We use two general approaches to derive the importance function.

The first approach builds local importance functions for the (automata-semantics of the) nodes of the tree. Then these local functions are aggregated into an importance function for the full tree. Aggregation uses structural induction in the layered description of the tree, and the way they are aggregated depends strongly on the gate at the top of the subtree combining the propagation of the fault below it.

The second approach is based on (minimal) cut sets. Cut sets are sets of basic events such that, if all elements in any cut set fail, the tree fails—we note that cut sets are defined for static fault trees; we conservatively extend these to dynamic fault trees. Thus, the more elements in a cut set that have failed, the higher its importance. Since a fault tree usually has multiple cut sets, we take the maximum importance over all cut sets. We also explore some variants of this idea, where we also normalise the cut sets by their maximum weights, or prune them based on their cardinality or failure probability.

Using such importance functions, we implement importance splitting methods to run RES analyses. We use a variety of RES algorithms to estimate system unreliability and unavailability. Our approach converges to precise estimations in increasingly reliable systems at much faster pace than standard Monte Carlo. This method has four advantages over earlier analysis methods for RFTs—which we overview in the related work section 7—namely: (1) we are able to estimate both the system reliability and availability; (2) we can handle arbitrary failure and repair distributions; (3) we can handle rare events; and (4) we can do it in a fully automatic fashion.

We implemented our theory in a full-stack toolchain that is also presented here. Within this toolchain, and in addition to the new importance function generation techniques, we introduce the language Kepler as an extension of Galileo [55,56] to describe repairable fault trees in textual format. We also changed the RES engine on the core of the `FIG` statistical model checker [11] by resampling time values at the moment of splitting. This requires considering algorithms specifically tailored to generate conditional pseudorandom variates. We show how this modification provides significant improvements to the performance of the tool.

With this toolchain, we computed confidence intervals for the unreliability and unavailability of several case studies. Our case studies are RFTs whose failure and repair times are governed by arbitrary continuous probability density functions (PDFs). Each case study was analysed for a fixed runtime budget and in increasingly resilient configurations. In all cases, our approach could estimate the narrowest intervals for the most resilient configurations.

Summarising, the contributions reported in this work are:

1. two methods to automatically generate importance functions, one based recursively on the structure of the RFT, and the other on its collection of minimal cut sets;
2. a toolchain including the previous methods that, given the input RFT, estimates the required dependability metric through RES in a fully automated fashion;
3. an improvement of the `FIG` statistical model checker thorough resampling of the clocks at the moment of splitting;
4. the textual language Kepler to describe RFT having failure and repair time with arbitrary distributions; and
5. an extensive validation of the techniques and tools in a variety of case studies.

Paper outline. This article is structured as follows. In Sec. 2, we discuss the concepts related to fault trees which are fundamental to our contribution, while in Sec. 3 we recall the basics of RES and the importance splitting techniques. Sec. 4 focuses on our fundamental contributions, namely, the techniques for automatically deriving importance functions and the technique for improving the engine of `FIG`. Sec. 5 describes the toolchain with particular attention on the language Kepler and its translation to IOSA, the input language of `FIG`. Using the toolchain, we performed an extensive experimental evaluation that we present in Sec. 6. We overview related work in Sec. 7 and conclude our contributions in Sec. 8.

We remark that this article merges and extends the contributions reported in [12,19].

2 Fault tree analysis

2.1 Syntax

A fault tree '△' is a directed acyclic graph that models how component failures propagate and eventually cause the full system to fail. We consider repairable fault trees (RFTs), where the occurrence time of failures and repairs is governed by arbitrary probability distributions.

Basic elements. The leaves of the tree, called basic events or *basic elements* (BEs), model the failure behaviour of components. Thus, a BE b is equipped with a failure distribution F_b that governs the probability for b to fail before time t, and a repair distribution R_b that governs its repair time. Some BEs are used as spare components. The *spare basic elements* (SBEs) replace a primary component when it fails. SBEs are also equipped with a dormancy distribution D_b, since spares fail less often when *dormant*, i.e. not in use. Only if an SBE becomes active, its failure distribution is given by F_b.

Gates. Non-leave nodes are called *intermediate events* and are labelled with *gates*, that describe how combinations of

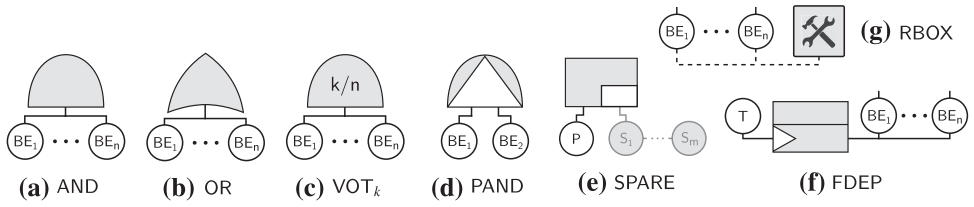


Fig. 1 Fault tree gates and the repair box

lower failures propagate to upper levels. Fig. 1 shows their syntax. Their meaning is as follows. The AND gate fails if all its children fail, the OR gate fails if at least one of its children fails, and the VOT_k gate fails if k of its m children fail (with $1 \leq k \leq m$). The latter is called the *voting* or *k out of m* gate. Note that VOT_1 is equivalent to an OR gate, and VOT_m is equivalent to an AND. If any of these gates is in a fail state, it becomes repaired if the condition that produces the failure is falsified. Thus, for instance, a failing AND is repaired if at least one of its failing children is repaired.

Note that the three gates we presented so far react only based on changes in the combination of the inputs provided by their children. These are called *static gates* and are already present in Static Fault Trees. In contrast, the following gates are called *dynamic gates* and react to the change of state of their children taking into account also other aspects like timing and dependence.

The *priority-and gate* (PAND) is an AND gate that only fails if its children fail orderly from left to right (though adjacent children may also fail simultaneously). PAND gates express failures that can only happen in a particular order, e.g. a general electric failure can only happen if first the circuit breaker fails and then a short circuit occurs. A failing PAND gate gets repaired whenever its right-most failed child becomes repaired [45,46].

SPARE gates have one *primary* child and one or more *spare* children: spares replace the primary when it fails. A SPARE gate fails if the primary (or current active) child fails and it does not succeed to find an operational spare child. It becomes repaired whenever the primary child is repaired or an operational spare child becomes available.

The FDEP gate has a *trigger* child and several *dependent* children: all dependent children become unavailable when the trigger fails. Note that the dependent children do not necessarily fail as a cause of the failure of the trigger child and that they become available again as soon as the trigger child is repaired. FDEPs can model, for instance, network elements that become unavailable if their connecting bus fails.

Repair boxes. An RBOX determines which basic element is repaired next according to a given policy. Thus all its inputs are BEs or SBEs. Unlike gates, an RBOX has no output since it does not propagate failures.

Top-level event. A full-system failure occurs if the *top event* (i.e. the root node) of the tree fails.

Example. The repairable fault tree in Fig. 2 models a railway-signal system, which fails if its high voltage and relay cabinets fail [31,53]. Thus, the top event is an AND gate with children HVcab (a BE) and Rcab. The latter is a SPARE gate with primary P and spare S. All BEs are managed by one RBOX with repair priority HVcab > P > S.

Notation. The nodes of a tree $\triangle$ are given by the set

$$nodes(\triangle) = \{0, 1, \ldots, n-1\}.$$

We let v, w range over $nodes(\triangle)$. A function

$$type^{\triangle} : nodes(\triangle) \to \left\{ \begin{array}{l} \mathsf{BE}, \mathsf{SBE}, \mathsf{AND}, \mathsf{OR}, \mathsf{VOT}_k, \\ \mathsf{PAND}, \mathsf{SPARE}, \mathsf{FDEP}, \mathsf{RBOX} \end{array} \right\}$$

yields the type of each node in the tree. A function

$$chil^{\triangle} : nodes(\triangle) \to nodes(\triangle)^*$$

returns the ordered list of children of a node. If clear from context, we omit the superscript $\triangle$ from function names.

2.2 Semantics

Following [45] we give semantics to RFT as *Input/Output Stochastic Automata* (IOSA), so that we can handle arbitrary probability distributions. Each state in the IOSA represents a system configuration, indicating which components are operational and which have failed. Transitions among states describe how the configuration changes when failures or repairs occur.

More precisely, a *state* of the IOSA derived from an RFT is a tuple $\boldsymbol{x} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}) \in \mathcal{S} \subseteq \mathbb{N}^n$, where $\mathcal{S}$ is the *state space* and $\boldsymbol{x}_v$ denotes the state of node v in $\triangle$. The possible values for $\boldsymbol{x}_v$ depend on the type of v. The *output* $z_v \in \{0, 1\}$ of node v indicates whether it is operational (z_v=0) or failed (z_v=1) and is calculated as follows:

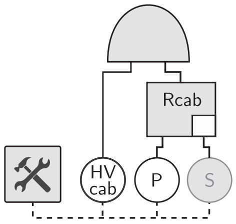
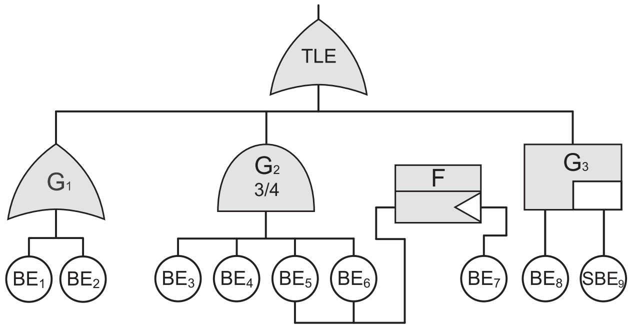


Fig. 2 Tiny RFT

- BEs (white circles in Fig. 1) have a binary state: $\boldsymbol{x}_v = 0$ if BE v is operational and $\boldsymbol{x}_v = 1$ if it is failed. The output of a BE is its state: $z_v = \boldsymbol{x}_v$.
- SBEs (gray circles in Fig. 1e) have two additional states: $\boldsymbol{x}_v = 2, 3$ if a *dormant* SBE v is respectively operational or failed. Here $z_v = \boldsymbol{x}_v \bmod 2$.
- ANDs have a binary state. The AND gate v fails iff all children fail: $\boldsymbol{x}_v = \min_{w \in chil(v)} z_w$. An AND gate outputs its internal state: $z_v = \boldsymbol{x}_v$.
- OR gates are analogous to AND gates, but fail iff any child fail, i.e. $z_v = \boldsymbol{x}_v = \max_{w \in chil(v)} z_w$ for OR gate v.
- VOT gates also have a binary state: a VOT$_k$ gate fails whenever $1 \leq k \leq m$ children fail, thus $z_v = \boldsymbol{x}_v = 1$ if $k \leq \sum_{w \in chil(v)} z_w$, and $z_v = \boldsymbol{x}_v = 0$ otherwise.
- PAND gates admit multiple states to represent the failure order of the children. For PAND v with two children we let $\boldsymbol{x}_v$ equal: 0 if both children are operational; 1 if the left child failed, but the right one has not; 2 if the right child failed, but the left one has not; 3 if both children have failed, the right one first; 4 if both children have failed, otherwise. The output of PAND gate v is $z_v = 1$ if $\boldsymbol{x}_v = 4$ and $z_v = 0$ otherwise. PAND gates with more children are handled by exploiting the fact that $\mathsf{PAND}(w_1, w_2, w_3) = \mathsf{PAND}(\mathsf{PAND}(w_1, w_2), w_3)$.
- SPARE gate v leftmost child is its primary BE. All other (spare) inputs are SBEs. SBEs can be shared among several SPARE gates. When the primary of v fails, it is replaced with an *available* SBE. An SBE is unavailable if it is failed or if it is replacing the primary BE of another SPARE. The output of v is $z_v = 1$ if its primary is failed and no spare is available. Else $z_v = 0$.
- An FDEP gate has no output. Its leftmost input is the trigger. We consider non-destructive FDEPs [8]: if the trigger fails, the output of all other inputs is set to 1, without affecting the internal state. Since this can be modelled by a suitable combination of OR gates [45], we omit the details.

For example, the RFT from Fig. 2 starts with all elements operational, so the initial state is $\boldsymbol{x}^0 = (0, 0, 2, 0, 0)$. If then P fails, $\boldsymbol{x}_{\mathsf{P}}$ and z_{P} are set to 1 (failed) and S becomes $\boldsymbol{x}_{\mathsf{S}} = 0$ (active and operational spare), so the state changes to $\boldsymbol{x}^1 = (0, 1, 0, 0, 0)$. The traces of the IOSA are given by $\boldsymbol{x}^0 \boldsymbol{x}^1 \cdots \boldsymbol{x}^n \in \mathcal{S}^*$, where a change from $\boldsymbol{x}^j$ to $\boldsymbol{x}^{j+1}$ corresponds to transitions triggered in the IOSA.

Dynamic fault trees may exhibit nondeterministic behaviour as a consequence of underspecified failure behaviour [23,37]. This can happen e.g. when two SPAREs have a single shared SBE: if all elements are failed and the SBE is repaired first, the failure behaviour depends on which SPARE gets the SBE. Monte Carlo simulation, however, requires fully stochastic models and cannot cope with nondeterminism. To overcome this problem, we deploy the theory from [24,45]. If a fault tree adheres to some mild syntactic conditions, then its IOSA semantics is *weakly deterministic*, meaning that all resolutions of the nondeterministic choices lead to the same probability value. In particular, we require that (1) each BE is connected to at most one SPARE gate, and that (2) BEs and SBEs connected to SPAREs are not connected to FDEPs. In addition to this, some semantic decisions have been fixed. Notably, the semantics of PAND, which normally has some ambiguity and has rarely been discussed in the context of repairs, is here fully specified. Besides, policies should be provided for RBOX and spare assignments.



(a) An RFT without PANDs

Familiy	Minimal cut sets included

$\mathscr{M}(\triangle)$	{BE$_1$} {BE$_2$} {BE$_3$,BE$_4$,BE$_5$}
	{BE$_3$,BE$_4$,BE$_6$} {BE$_3$,BE$_5$,BE$_6$} {BE$_4$,BE$_5$,BE$_6$}
	{BE$_3$,BE$_7$} {BE$_4$,BE$_7$} {BE$_8$,SBE$_9$}

$\mathscr{M}_{<3}(\triangle)$	{BE$_1$} {BE$_2$} {BE$_3$,BE$_7$} {BE$_4$,BE$_7$} {BE$_8$,BE$_9$}

$\mathscr{M}_{>\frac{1}{4}}(\triangle)$	{BE$_2$} {BE$_3$,BE$_4$,BE$_5$} {BE$_3$,BE$_4$,BE$_6$}
	{BE$_3$,BE$_5$,BE$_6$} {BE$_4$,BE$_5$,BE$_6$}

(b) Minimal cut sets of Fig. 3a

Fig. 3 Fault trees & minimal cut sets

2.3 Minimal cut sets

Cut sets are a well known qualitative technique in FTA for static FTs. A *cut set* is a set of basic elements (BEs and SBEs) whose joint failure will cause a top-level event. A *minimal cut set* (MCS) is a cut set of which no subset is a cut set. These concepts can be lifted to dynamic fault trees (and RFTs) in general, but this requires introducing an order to capture temporal dependencies, plus several other subtleties [37,54]. Nevertheless, RFTs as defined above rule out several issues raised by cut sets in DFTs, such as event simultaneity [37]. Furthermore, for FT analysis related to MCS, we exclude order dependence by considering RFTs without PAND gates.

For an illustration, Fig. 3b lists all minimal cut sets of the tree $\triangle$ from Fig. 3a. We use the notation: $\mathscr{M}(\triangle)$ for the

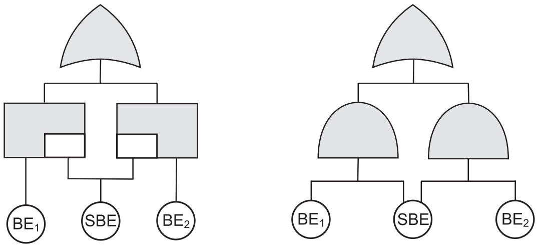


Fig. 4 Replacing SPAREs with ANDs may loose MCSs

family of all minimal cut sets of the FT $\triangle$; $\mathscr{M}_{<N}(\triangle)$ for the subset of $\mathscr{M}(\triangle)$ that excludes cut sets with N or more BEs (called *pruning* of order N); $\mathscr{M}_{>\lambda}(\triangle)$ for the subset of $\mathscr{M}(\triangle)$ that excludes cut sets where the product of the failure rate of the BEs is $\leq \lambda \in \mathbb{R}_{>0}$. The latter is only defined when all BEs and SBEs in the RFT have exponential failure and dormancy distributions. To obtain $\mathscr{M}_{>\frac{1}{4}}(\triangle)$ in Fig. 3b, we make this the case with failure rates: $\frac{1}{4}$ for BE$_1$ and BE$_7$, $\frac{6}{20}$ for BE$_2$, $\frac{2}{3}$ for $\{\text{BE}_i\}_{i=3}^{6}$, and $\frac{1}{2}$ for BE$_8$ and SBE$_9$.

Cut set pruning as in $\mathscr{M}_{<N}(\triangle)$ and $\mathscr{M}_{>\lambda}(\triangle)$ is a standard way to speed up FT analyses [57]. The goal is to ignore the most unlikely (and hard to compute) cut sets: pruning of order N assumes that the top-level event will most likely occur by cut sets with less than N BEs; pruning by rate $\leq \lambda$ assumes that the top-level event will occur first by cut sets where BEs have higher rates and thus fail faster. Choosing such N and λ to prune irrelevant MCS of a given tree depends on its structure and the BEs failure, dormancy, or repair distributions.

An RFT without PAND or SPARE gates can be translated into a static FT in such a way that they have exactly the same set of MCSs (notice that RBOXs have no effect on minimal cut sets). Obtaining the MCSs from static FT can be easily done, e.g. by translating it into an FT in disjunctive normal form and taking the minimal clauses as the minimal cut sets. For this work, we also opt to translate SPARE gates into AND gates. However, this translation does not preserve all MCSs, yielding, on the translated FT, a subset of the MCSs of the original RFT. For example, the RFT on the left of Fig. 4 has three MCSs. In particular, since SBE cannot be operational in both SPARE gates at the same time, $\{\text{BE}_1, \text{BE}_3\}$ is one of these MCSs. However, this set is not an MCS in the FT on the right side of Fig. 4, which replaces the SPARE gates with AND gates.

2.4 Dependability metrics

An important use of fault trees is to compute relevant dependability metrics. Let X_t denote the random variable that represents the state of the top event at time t [22]. Two popular metrics are:

- *system reliability*: the probability of observing no top event failure before some mission time $T > 0$, viz.

$$\text{REL}_T = Prob\left(\forall_{t\in[0,T]} \,.\, X_t = 0\right);$$

- *system availability*: the proportion of time that the system remains operational in the long run, viz.

$$\text{AVA} = \lim_{t\to\infty} Prob\,(X_t = 0)\,.$$

System *unreliability* and *unavailability* are the complements of these metrics, i.e. $\text{UNREL}_T = 1 - \text{REL}_T$ and $\text{UNAVA} = 1 - \text{AVA}$.

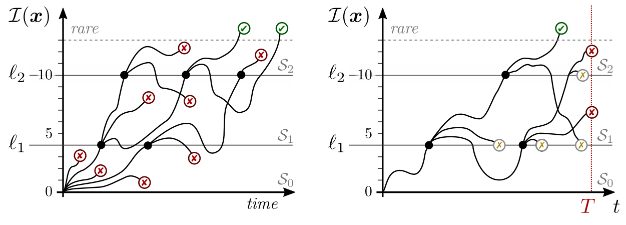
3 Stochastic simulation for fault trees

Standard Monte Carlo simulation (SMC). Monte Carlo simulation takes random samples from stochastic models to estimate a (dependability) metric of interest. For instance, to estimate the unreliability of a tree $\triangle$we sample N independent traces from its IOSA semantics. An unbiased statistical estimator for $p = \text{UNREL}_T$ is the proportion of traces observing a top-level event, that is, $\hat{p}_N = \frac{1}{N}\sum_{j=1}^{N} X^j$ where $X^j = 1$ if the j-th trace exhibits a top-level failure before time T and $X^j = 0$ otherwise. The statistical error of $\hat{p}$ is typically quantified with two numbers δ and ε s.t. $\hat{p} \in [p-\varepsilon, p+\varepsilon]$ with probability δ. The interval $\hat{p} \pm \varepsilon$ is called a *confidence interval* (CI) with coefficient δ and precision 2ε.

Such procedures scale linearly with the number of tree nodes and cater for arbitrary PDFs, i.e. not restricted to exponential ones. However, they encounter a bottleneck to estimate *rare events*: if $p \approx 0$, very few traces observe $X^j = 1$. Therefore, the variance of estimators like $\hat{p}$ becomes huge, and CIs become too wide, easily degenerating to the trivial interval [0, 1]. Increasing the number of traces alleviates this problem, but even standard confidence interval settings—where ε is relative to p—require sampling an unacceptably large number of traces [49]. Rare event simulation techniques solve this specific problem.

Rare Event Simulation (RES). RES techniques [49] increase the number of traces that observe the rare event, e.g. a top-level event in an RFT. Two prominent classes of RES techniques are *importance sampling*, which adjusts the PDF of failures and repairs, and *importance splitting* (ISPLIT) [43], which samples more (partial) traces from states that are closer to the rare event. We focus on ISPLIT due to its flexibility with respect to the probability distributions.

ISPLIT can be efficiently deployed as long as the rare event γ, here defined as the set of states characterising the rare event, can be described as a nested sequence of less-rare events $\gamma = \gamma_M \subsetneq \gamma_{M-1} \subsetneq \cdots \subsetneq \gamma_0 = \mathcal{S}$. This



(a) FE$_5$ for *Prob*(¬✘U✔) **(b)** RST$_{\text{ES}}$ for UNREL$_T$

Fig. 5 Importance splitting algorithms fixed effort & RESTART

decomposition allows ISPLIT to study the conditional probabilities $p_k = Prob(\gamma_{k+1}\,|\,\gamma_k)$ separately, to then compute $p = Prob(\gamma) = \prod_{k=0}^{M\text{-}1} Prob(\gamma_{k+1}\,|\,\gamma_k)$. Moreover, ISPLIT requires all conditional probabilities p_k to be much greater than p, so that estimating each p_k can be done efficiently with SMC.

The key idea behind ISPLIT is to define the events γ_k via a so called *importance function* $\mathcal{I}\colon \mathcal{S} \to \mathbb{N}$ that assigns an *importance* to each state $s \in \mathcal{S}$. The higher the importance of a state, the closer it is to the rare event γ_M. Event γ_k collects all states with importance at least ℓ_k, for certain sequence of *threshold levels* $0 = \ell_0 < \ell_1 < \cdots < \ell_M$. Formally: $\gamma_k = \{s \in \mathcal{S} \mid \mathcal{I}(s) \geq \ell_k\}$.

To exploit the importance function $\mathcal{I}$ in the simulation procedure, ISPLIT samples more (partial) traces from states with higher importance. Two well-known methods are deployed and compared in this paper: RESTART and Fixed Effort. *Fixed Effort* (FE) [27] samples a predefined amount of traces in each region $\mathcal{S}_k = \gamma_k \backslash \gamma_{k+1} = \{s \in \mathcal{S} \mid \ell_{k+1} > \mathcal{I}(s) \geq \ell_k\}$ for $0 \leq k < M$. Thus, starting at γ_0 it first estimates the proportion of traces that reach γ_1, i.e. the probability $p_0 = Prob(\gamma_1\,|\,\gamma_0) = Prob(\mathcal{S}_0)$. Next, from the states that reached γ_1 new traces are generated to estimate $p_1 = Prob(\mathcal{S}_1)$, and so on until p_M. Fixed Effort thus requires that (i) each trace has a clearly defined end, so that estimations of each p_k finish with probability 1, and (ii) all rare events reside in the uppermost region $\mathcal{S}_M = \{s \in \mathcal{S} \mid \mathcal{I}(s) \geq \gamma_M\}$.

Example. Fig. 5a shows Fixed Effort estimating the probability to visit states labelled ✔ before others labelled ✘. States ✔ have importance >13, and thresholds $\ell_1 = 4$ and $\ell_2 = 10$ partition the state space in regions $\{\mathcal{S}_i\}_{i=0}^{2}$ s.t. all ✔ $\in \mathcal{S}_2$. The effort is 5 simulations per region, for all regions: we call this algorithm FE$_5$. In region $\mathcal{S}_0$, 2 simulations made it from the initial state to threshold ℓ_1, i.e. they reached some state with importance 4 before visiting a state ✘. In $\mathcal{S}_1$, starting from these two states, 3 simulations reached ℓ_2. Finally, 2 out of 5 simulations visited states ✔ in $\mathcal{S}_2$. Thus, the estimated rare event probability of this run of FE5 is $\hat{p} = \prod_{i=1}^{2} \hat{p}_i = \frac{2}{5}\frac{3}{5}\frac{2}{5} = 9.6 \times 10^{-2}$.

RESTART (RST) [62,63] is another RES algorithm, which starts one trace in γ_0 and monitors the importance of the states visited. If the trace up-crosses threshold ℓ_1, the first state visited in $\mathcal{S}_1$ is saved and the trace is cloned, aka *split*—see Fig. 5b. This mechanism rewards traces that get closer to the rare event. Each clone then evolves independently, and if one up-crosses threshold ℓ_2 the splitting mechanism is repeated. Instead, if a state with importance below ℓ_1 is visited, the trace is *truncated* (⊗ in Fig. 5b). This penalises traces that move away from the rare event. To avoid truncating all traces, the one that spawned the clones in region $\mathcal{S}_k$ can go below importance ℓ_k. To deploy an unbiased estimator for p, RESTART measures how much split was required to visit a rare state [62]. In particular, RESTART does not need the rare event to be defined as γ_M [58], and it was devised for steady-state analysis [63] (e.g. to estimate UNAVA) although it can also be used for transient studies as depicted in Fig. 5b [59].

4 Importance splitting for FTA

The effectiveness of ISPLIT crucially relies on the importance function $\mathcal{I}$, as well as the threshold levels $\ell_1, \ldots, \ell_M$ [43]. Traditionally, these are given by domain and/or RES experts, thus requiring considerable expert knowledge. To alleviate this requirement, we focus on techniques that are able to obtain importance functions and splitting thresholds automatically. In particular, we introduce a first technique that derives $\mathcal{I}$ from the structure of the given RFT, and a second technique that defines $\mathcal{I}$ based on the MCSs of the RFT. We also discuss methods to select the threshold levels ℓ_k and an improvement on `FIG` based on resampling time values at the moment of splitting simulation runs.

4.1 Compositional importance functions for fault trees

The core idea behind importance splitting is that states that are more likely to lead to the rare event should have a higher importance. To achieve this, the key lies in defining an importance function $\mathcal{I}$ and thresholds ℓ_k that are sensitive to both the state space $\mathcal{S}$ and the transition probabilities of the system. For us, $\mathcal{S} \subseteq \mathbb{N}^n$ are all possible states of an RFT. Its top event fails when certain nodes fail in certain order, and remains failed until certain repairs occur. To exploit this for ISPLIT, the structure of the tree must be embedded into $\mathcal{I}$.

The strong dependence of the importance function $\mathcal{I}$ on the structure of the tree is easy to see in the following example. Take the RFT △from Fig. 2 and let its current state $\boldsymbol{x}$ be s.t. P is failed and HVcab and S are operational. If the next event is a repair of P, then the new state $\boldsymbol{x}'$ (where all basic elements are operational) is farther from a failure of the top event. Hence, a good importance function should satisfy $\mathcal{I}(\boldsymbol{x}) > \mathcal{I}\left(\boldsymbol{x}'\right)$. Oppositely, if the next event had been a failure of S leading to state $\boldsymbol{x}''$, then one would want that $\mathcal{I}(\boldsymbol{x}) < \mathcal{I}\left(\boldsymbol{x}''\right)$. The key observation is that these inequalities depend on the structure of $\triangle$as well as on the failures/repairs of basic elements.

In view of the above, any attempt to define an importance function for an arbitrary fault tree $\triangle$ must put its (gate) structure in the forefront. In Table 1 we introduce a compositional heuristic for this, which defines *local importance functions* distinguished per node type. The importance function associated to node v is $\mathcal{I}_v\colon \mathbb{N}^n \to \mathbb{N}$. We define the *global importance function* of the tree ($\mathcal{I}_{\mathsf{STR}}$) as the local importance function of the top event node of $\triangle$.

Thus, $\mathcal{I}_v$ is defined in Table 1 via structural induction in the fault tree. It is defined so that it assigns to a failed node v its highest importance value. Functions with this property deploy the most efficient ISPLIT implementations [43], and some RES algorithms (e.g. Fixed Effort) require this property [27].

In the following we explain our definition of $\mathcal{I}_v$. If v is a failed BE or SBE, then its importance is 1; else it is 0. This matches the output of the node, thus $\mathcal{I}_v(\boldsymbol{x}) = z_v$. Intuitively, this reflects how failures of basic elements are positively correlated to top event failures. The importance of AND, OR, and VOT_k gates depends exclusively on their input. The importance of an AND is the sum of the importance of their children, and scaled by a normalisation factor (explained below). This reflects that AND gates fail when all their children fail, and each failure of a child brings an AND closer to its own failure, hence increasing its importance. Instead, since OR gates fail as soon as a single child fails, their importance is the maximum importance among its children. The importance of a VOT_k gate is the sum of the k (out of m) children with highest importance value.

Omitting normalisation may yield an undesirable importance function. To understand why, suppose a binary AND gate v with children l and r, and define $\mathcal{I}_v^{\text{naive}}(\boldsymbol{x}) = \mathcal{I}_l(\boldsymbol{x}) + \mathcal{I}_r(\boldsymbol{x})$. Suppose that $\mathcal{I}_l$ takes it highest value in $\max_l^{\mathcal{I}} = 2$ while $\mathcal{I}_r$ in $\max_r^{\mathcal{I}} = 6$ and assume that states $\boldsymbol{x}$ and $\boldsymbol{x}'$ are s.t. $\mathcal{I}_l(\boldsymbol{x}) = 1$, $\mathcal{I}_r(\boldsymbol{x}) = 0$, $\mathcal{I}_l(\boldsymbol{x}') = 0$, $\mathcal{I}_r(\boldsymbol{x}') = 3$. This means that in both states one child of v is "good-as-new" and the other is "half-failed" and hence the system is equally close to fail in both cases. Hence we expect $\mathcal{I}_v^{\text{naive}}(\boldsymbol{x}) = \mathcal{I}_v^{\text{naive}}(\boldsymbol{x}')$ when actually $\mathcal{I}_v^{\text{naive}}(\boldsymbol{x}) = 1 \neq 3 = \mathcal{I}_v^{\text{naive}}(\boldsymbol{x}')$. Instead, $\mathcal{I}_v$ operates with $\frac{\mathcal{I}_l(\boldsymbol{x})}{\max_l^{\mathcal{I}}}$ and $\frac{\mathcal{I}_r(\boldsymbol{x})}{\max_r^{\mathcal{I}}}$, which can be interpreted as the "percentage of failure" of the children of v. To make these numbers integers we scale them by lcm_v, the *least common multiple* of their max importance values. In our case $\text{lcm}_v = 6$ and hence $\mathcal{I}_v(\boldsymbol{x}) = \mathcal{I}_v(\boldsymbol{x}') = 3$. Similar problems arise with all gates, hence normalisation is applied in all cases.

SPARE gates with m children—including its primary—behave similarly to AND gates: every failed child brings the gate closer to failure, as reflected in the left operand of the max in Table 1. However, SPAREs fail when their primaries fail and no SBEs are *available* (as opposed to *failed*, e.g. possibly being used by another SPARE). This means that the gate could fail despite some children being operational. To account for this we exploit the gate output: multiplying z_v by m we give the gate its maximum value when it fails, even when this happens due to unavailable—but possibly operational—SBEs.

For a PAND gate v we have to carefully look at the states. If the left child l fails first, then the right child r contributes positively to the failure of the PAND and hence the importance function of the node v. If instead the right child has failed *before the left child*, then the PAND gate will not fail and hence we let it contribute negatively to the importance function of v. Thus, we multiply $\frac{\mathcal{I}_r(\boldsymbol{x})}{\max_r^{\mathcal{I}}}$ (the normalized importance function of the right child) by -1 in the later case, i.e. when state $\boldsymbol{x}_v \notin \{1, 4\}$. Instead, the left child always contributes positively. Finally, the max operation is twofold: on the one hand, $z_v \cdot 2$ ensures that the importance value remains at its maximum while failing (PANDs remain failed even after the left child is repaired); on the other, it ensures that the smallest possible value while operational is 0 (since importance values cannot be negative.)

4.2 Importance functions based on minimal cut sets

In the previous section we introduced a compositional technique to derive the importance function. The key concept is that, in general, importance should reflect proximity to the rare event. This means that *the importance of a gate should increase as the gate approaches its own failure*. Note that the type of a gate defines how, as its children fail, the gate approaches its own failure. Following the structure of the RFT, its importance function is determined by the importance of its top-level gate.

In this section we follow a different approach based on the set of minimal cut sets. Recall that the joint failure of all basic elements in a MCS determines the failure of the top event. Thus, by counting how many BEs have failed in a MCS, we have an idea of the proximity to the occurrence of the top-level event. This suggests the following importance function: given a state of the RFT, its importance is defined by the maximum number of failed BEs in any MCS.

Formally, let $\triangle^*$ be the FT rewrite of the original RFT $\triangle$ as described in Sec. 2.3—i.e. replacing SPARE and FDEP gates resp. for AND and OR—and let $\mathscr{M}(\triangle^*)$ be the set of all its minimal cut sets. Then, the importance function described above is given by function $\mathcal{I}_{\mathsf{MCS}}$ in Table 2.

We further consider pruned variants of $\mathcal{I}_{\mathsf{MCS}}$ that discards cut sets based on their cardinality ($\mathcal{I}_{\mathsf{MCS\text{-}P}}$) or, if BE failures are exponentially distributed, based on the product of the

Table 1 Compositional ("structural") importance function for RFTs

type(v)	$\mathcal{I}_v(\boldsymbol{x})$
BE, SBE	z_v
AND	$\mathrm{lcm}_v \cdot \sum_{w \in chil(v)} \frac{\mathcal{I}_w(\boldsymbol{x})}{\max_w^{\mathcal{I}}}$
OR	$\mathrm{lcm}_v \cdot \max\limits_{w \in chil(v)} \left\{ \frac{\mathcal{I}_w(\boldsymbol{x})}{\max_w^{\mathcal{I}}} \right\}$
VOT$_k$	$\mathrm{lcm}_v \cdot \max\limits_{W \subseteq chil(v), \lvert W \rvert = k} \left\{ \sum_{w \in W} \frac{\mathcal{I}_w(\boldsymbol{x})}{\max_w^{\mathcal{I}}} \right\}$
SPARE	$\mathrm{lcm}_v \cdot \max \left(\sum_{w \in chil(v)} \frac{\mathcal{I}_w(\boldsymbol{x})}{\max_w^{\mathcal{I}}}, z_v \cdot m \right)$
PAND	$\mathrm{lcm}_v \cdot \max \left(\frac{\mathcal{I}_l(\boldsymbol{x})}{\max_l^{\mathcal{I}}} + ord \frac{\mathcal{I}_r(\boldsymbol{x})}{\max_r^{\mathcal{I}}}, z_v \cdot 2 \right)$
	where $ord = 1$ if $\boldsymbol{x}_v \in \{1, 4\}$ and $ord = -1$ otherwise

with $\max_v^{\mathcal{I}} = \max_{\boldsymbol{x} \in \mathcal{S}} \mathcal{I}_v(\boldsymbol{x})$ and $\mathrm{lcm}_v = \mathrm{lcm} \left\{ \max_w^{\mathcal{I}} \,\middle|\, w \in chil(v) \right\}$

Table 2 Importance functions for automatic RES in fault trees

Name	Expression	Description
$\mathcal{I}_{\mathsf{MCS}}(\boldsymbol{x}) =$	$\underset{\mathrm{MCS} \in \mathscr{M}(\Delta^*)}{max} \left\{ \sum\limits_{v \in \mathrm{MCS}} z_b \right\}$	For each MCS of the tree, $\mathcal{I}_{\mathsf{MCS}}$ counts the number of BEs that have failed in the current state $\boldsymbol{x}$. (Recall that a basic element b in $\boldsymbol{x}$ has failed if $z_b = \boldsymbol{x}_b = 1$.) The importance $\mathcal{I}_{\mathsf{MCS}}(\boldsymbol{x})$ of the current state of the tree is the maximum among these counts.
$\mathcal{I}_{\mathsf{MCS\text{-}P}}(\boldsymbol{x}) =$	$\underset{\mathrm{MCS} \in \mathscr{M}_{<N}(\Delta^*)}{max} \left\{ \sum\limits_{v \in \mathrm{MCS}} z_b \right\}$	$\mathcal{I}_{\mathsf{MCS\text{-}P}}$ operates similarly to function $\mathcal{I}_{\mathsf{MCS}}$ above, but here the maximum ranges over a *pruned* set of MCS, discarding cut sets with N or more BEs.
$\mathcal{I}_{\mathsf{MCS\text{-}PR}}(\boldsymbol{x}) =$	$\underset{\mathrm{MCS} \in \mathscr{M}_{>\lambda}(\Delta^*)}{max} \left\{ \sum\limits_{v \in \mathrm{MCS}} z_b \right\}$	Similar to $\mathcal{I}_{\mathsf{MCS\text{-}P}}$ but using the failure *rates* for pruning, $\mathcal{I}_{\mathsf{MCS\text{-}PR}}$ considers only MCS where the product of the failure rate of all BEs is greater than λ. Applicable only to FTs whose failure and dormancy distributions are Markovian.
$\mathcal{I}_{\mathsf{MCSN}}(\mathbf{x}) =$	$\underset{\mathrm{MCS} \in \mathscr{M}(\Delta^*)}{max} \left\{ \mathrm{lcm} \cdot \sum\limits_{v \in \mathrm{MCS}} \frac{z_b}{\lvert \mathrm{MCS} \rvert} \right\}$	$\mathcal{I}_{\mathsf{MCSN}}$ is a normalised version of $\mathcal{I}_{\mathsf{MCS}}$. The normalisation follows a similar procedure to Sec. 4.1, where lcm is the least common multiple of the cardinality of every MCS in $\mathscr{M}(\Delta^*)$.

failure rates of the BEs in the cut set ($\mathcal{I}_{\mathsf{MCS\text{-}PR}}$). The only difference between these functions and $\mathcal{I}_{\mathsf{MCS}}$ is the range of the max operator, which reflects the pruning of some minimal cut sets.

A concept already discussed in Sec. 4.1 is *importance normalisation*. We also experiment with it here: in Table 2, $\mathcal{I}_{\mathsf{MCSN}}$ stands for the normalised version if $\mathcal{I}_{\mathsf{MCS}}$. By dividing each summation by its maximum possible value, namely $\lvert MCS \rvert$, we compute the percentage of failure introduced by MCS in the current state. The scaling factor lcm ensures that the resulting value is an integer as required by our tooling framework. Finally, although omitted in Table 2, we further define the functions $\mathcal{I}_{\mathsf{MCSN\text{-}P}}$ and $\mathcal{I}_{\mathsf{MCSN\text{-}PR}}$ as the normalised versions of the functions $\mathcal{I}_{\mathsf{MCS\text{-}P}}$ and $\mathcal{I}_{\mathsf{MCS\text{-}PR}}$, respectively.

4.3 Automatic importance splitting for FTA

The techniques to provide importance functions introduced in the previous subsections are based on the the distribution of operational/failed basic elements in the fault tree, being it through the structure of the tree or the contribution to its minimal cut sets. This follows the core idea of importance splitting: the more failed BEs/SBEs (in the right order), the closer a tree is to its top event failure.

However, the ISPLIT strategy is to run more simulations from those states that have a higher *probability* to lead to rare states. This is only partially reflected by whether basic element b is failed. Probabilities lie also in the failure, repair and dormancy distributions (F_b, R_b, D_b). These distributions govern the transitions among states $\boldsymbol{x} \in \mathcal{S}$, and can be exploited for importance splitting.

Thus, after determining the importance function, we run "pilot simulations" on the importance-labelled states of the tree. Running simulations exercises the fail and repair distributions of BEs and SBEs, imprinting this information in the thresholds ℓ_k. Several algorithms can do such *selection of thresholds*. They operate sequentially, starting from the initial state—a fully operational tree—which has importance $i_0 = 0$. For instance, Expected Success [13] runs N finite-life simulations. If $K < \frac{N}{2}$ simulations reach the next smallest importance $i_1 > i_0$, then the first threshold will be $\ell_1 = i_1$. Next, N simulations start from states with importance i_1, to determine whether the next importance i_2 should be chosen as threshold ℓ_2, and so on.

Expected Success also computes the *effort* per splitting region $\mathcal{S}_k = \{\boldsymbol{x} \in \mathcal{S} \mid \ell_{k+1} > \mathcal{I}(\boldsymbol{x}) \geq \ell_k\}$. For Fixed Effort, "effort" is the base number of simulations to run in region $\mathcal{S}_k$. For RESTART, it is the number of clones spawned when threshold ℓ_{k+1} is up-crossed. In general, if K out of N pilot simulations make it from ℓ_{k-1} to ℓ_k, then the k-th effort is $\lceil \frac{N}{K} \rceil$. This is chosen so that, during RES estimations, one simulation makes it from threshold ℓ_{k-1} to ℓ_k on average.

Thus, using the method from [14,15] based on any of our importance functions, we compute (automatically) the thresholds and their effort for the given RFT. This is all the meta-information required to apply importance splitting RES [14,27,28].

4.4 Sampling conditional variables

The engine of the simulation tool `FIG` is based on standard discrete-event simulation [42]: time advances in discrete steps until the first occurrence of an event and, at this moment, the occurrence time of the newly enabled events are sampled. Thus, once the occurrence time of an event is fixed, it remains unaltered until it actually occurs. This has negative implications on the ISPLIT method. Since each time a simulation run up-crosses a threshold, the current state is cloned and hence the occurrence time of all active (i.e. enabled but yet to occur) events are also copied on the splitting simulation runs. This means that there exists some dependence among the splitting runs with the consequent adverse impact on the variance of the parameter calculated during simulation.

To improve the convergence of RESTART and FE, it is therefore convenient that at each splitting point the occurrence time of every active event is resampled *conditioned to the time elapsed since it became active*. For this, we have implemented two different methods in `FIG`, one based on the inverse transform method and the other using a rejection technique [42].

If the random variable X that determines the occurrence of an event has a cumulative distribution function (CDF) F whose inverse F^{-1} can be written in a closed form, then we use the inverse-transform method as described in the following. Let t be the value that we want to sample from X conditioned to the fact that t_e units of time have passed. Then

$$P(X \leq t \mid X > t_e) = \frac{F(t) - F(t_e)}{1 - F(t_e)}.$$

More precisely, we are interested on sampling the remaining time t_r such that $t = t_e + t_r$. Hence, we are interested in the random variable $Y = X - t_e$ with CDF F_{t_e} such that:

$$F_{t_e}(t_r) = P(X \leq t_e + t_r \mid X > t_e) = \frac{F(t_e + t_r) - F(t_e)}{1 - F(t_e)}.$$

Following the inverse-transform method, a (pseudo) random value for Y can be obtained by generating a value $u \sim u[0,1]$ taking $t_r = F_{t_e}^{-1}(u) = F^{-1}(u + (1-u) \cdot F(t_e)) - t_e$.

Oppositely, for the case in which F^{-1} cannot be written in a closed formula (e.g. log-normal and gamma distributions), we use the rejection method. The method is simple:

1. generate $t \sim X$, and
2. if $t > t_e$ output t, otherwise repeat from 1.

However, as t_e increases, the likelihood of repeating the loop increases as well, which could turn the algorithm inefficient. In fact, if N is the number of repetitions until successfully choosing t, then $E(N) = \frac{1}{1-F(t_e)}$. In view of this, we only run the algorithm if $F(t_e) \leq 0.75$, in which case $E(N) \leq 4$. Otherwise we simply keep the previously sampled value.

5 Toolchain

Figure 6 outlines the complete toolchain implemented to deploy the theory described above. To model the input RFT, we introduce the new Kepler textual format, which extends the Galileo textual format [21,55,56] which is a widespread syntax to describe fault trees [9,25,44]. Kepler follows the syntax of Galileo adding support for repairs and non-Markovian distributions. We present Kepler in Sec. 5.1 and give its complete grammar in Appendix A.

The toolchain is structured as follows. The Kepler specification file is given as input to a Python converter script that produces three outputs: the IOSA specification that encodes the semantics of the tree, the property queries for unreliability or unavailability synthesised for the tree, and our compositional importance functions in terms of variables of the IOSA semantic model. This information is dumped into a single text file and fed to `FIG`, a statistical model checker specialised in importance splitting RES. `FIG` interprets this importance function, deploying it into its internal model representation, which results in a global function for the whole tree. `FIG` can then use ISPLIT algorithms such as RESTART and Fixed Effort via the automatic methods described above. The result are confidence intervals that estimate the reliability or availability of the RFT.

5.1 The Kepler language

Standard Galileo supports three PDF families, namely exponential, Weibull, and log-normal. Kepler extends

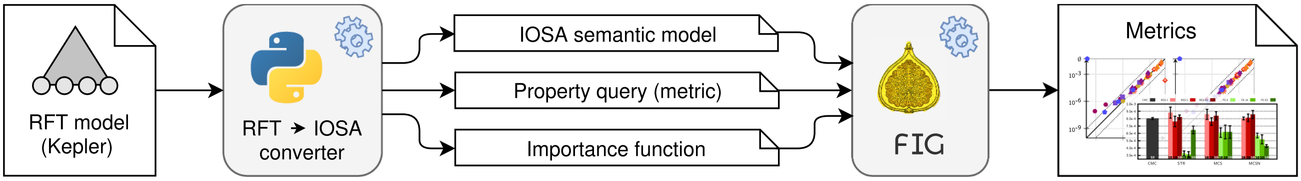


Fig. 6 Toolchain: from fault tree specification to the estimation of rare dependability metrics

```
toplevel "Gate2";
"Gate2" spare "BE_C" "BE_D";
"BE_C" fail~rayleigh(6.0E-2);
"BE_D" fail~exponential(1.11E-3) dorm~erlang(3,9);
```

Code 1 Spare with independent dormancy PDF

```
1  toplevel "Gate3";
2  "Gate3" and "BE_E" "BE_F" "BE_G";
3  "BE_E" fail~exponential(6.0E-5) repair~uniform(8,24);
4  "BE_F" fail~exponential(7.0E-5) repair~uniform(8,24);
5  "BE_G" fail~exponential(6.0E-5) repair~uniform(8,12);
6  "RB1" rbox prio "BE_E" "BE_F" "BE_G";
```

Code 2 Simple Kepler tree with a repair box

Galileo with arbitrary failure distributions—we introduce its full syntax in Code 5 on page 20. The current definition of Kepler supports a particular set of distributions but it can be straightforwardly extended to support others.

Kepler is a declarative language. Each line describes a node in the tree by its name, type, some extra characteristics and the names of its children. When declaring a (spare) basic element, we use the keyword `fail` to precede the definition of its failure distribution, `repair` for the repair distribution and `dorm` for its dormant failure distribution. The presence of a dormancy failure distribution is the only distinguishable factor between the definition of a BE and the definition of a SBE. SPAREs are defined by the keyword `spare`.

Code 1 provides an example of a SPARE gate (`Gate2`) with a primary basic element (`BE_C`) and a spare one (`BE_D`). Their respective failure PDFs are Rayleigh ($\sigma = 0.06$) and exponential ($\lambda = 0.00111$). Notice that, unlike Galileo, we allow the dormancy PDF of an SBE to be independent of its failure PDF. Thus we define the dormancy of `BE_D` as an Erlang($k = 3, \lambda = 9$).

Furthermore, Kepler supports the use of (multiple) repair boxes with the keywords `rbox`. The first parameter after `rbox` defines the policy for serving the queue of failed BEs and SBEs. Currently, only non-preemptive priority policies are provided through the keyword `prio`. Other policies are proposed in [45]. For example, in Code 2, all BEs are repairable, with repair time uniformly distributed. Line 6 of the code defines the RBOX of the system, which handles one repair at a time with the priority given by the order of the list. Thus, for instance, if `BE_E` and `BE_F` fail while `BE_G` is being repaired, `BE_E` will be chosen next.

5.2 Compiling Kepler to IOSA

We developed a Python textual converter that takes as input an RFT modelled in Kepler. The converter automatically produces 3 outputs:

1. The IOSA model of the input RFT,
2. A property specification for evaluating the unreliability or unavailability of the tree, and
3. The importance functions for the tree.

The translation of a Kepler model to a IOSA specification follows the semantics of RFTs defined in [45,46]. As an example, Code 3 shows the IOSA module corresponding to the basic element `BE_E` in Code 2.

The syntax of IOSA is close to that of PRISM [41] but includes primitives to handle stochastic timing. In IOSA, systems are modelled as a set of interacting processes which communicate by synchronising equally named transitions. Transitions are split into input and output. Output transitions are generative while input are reactive and only take place by synchronising with output transitions of other modules. IOSA presents a discrete-event continuous-time semantics. All clock variables in a IOSA model count down at the same rate and can be set to values sampled from their associated probability distribution. An example of setting clock `BE_E_rc` is seen at the end of line 10 in Code 3. When a clock variable expires (i.e. reaches zero), it may enable an output transition. We indicate this by preceding the clock name with the symbol `@` in the guard of a transition—see, for example, expression `@ BE_E_fc` at line 6 of Code 3.

The unreliability and unavailability queries are encoded as variants of PCTL [32] and CSL [2]. To do so, the script automatically identifies the state characterisation of the top-level event. This state condition depends solely on the semantic model of the top-level gate. For instance, Code 4 shows the unavailability property generated for the RFT of Code 2. Here, `Gate3_count == 3` characterises the failing of the top-level AND gate with 3 children.

Finally, all importance functions are calculated following Tables 1 and 2. We recall that $\mathcal{I}_{\mathrm{MCS}}$ and $\mathcal{I}_{\mathrm{MCSN}}$ are only available for models without PANDs.

```
module BE_BE_E
  BE_E_fc : clock;
  BE_E_rc : clock;
  BE_E_broken : [0..2] init 0;

  [BE_E_fl!] BE_E_broken==0 @ BE_E_fc
             -> (BE_E_broken'=1);
  [BE_E_r??] BE_E_broken==1
             -> (BE_E_broken'=2)
              & (BE_E_rc'=uniform(8,24));
  [BE_E_up!] BE_E_broken==2 @ BE_E_rc
             -> (BE_E_broken'=0)
              & (BE_E_fc'=exponential(6.0E-5));
endmodule
```

Code 3 IOSA model of BE_E from Code 2

```
properties
  S( Gate3_count == 3 )
endproperties
```

Code 4 Unavailability property query for Code 2

5.3 FIG: RES to estimate rare dependability metrics

The FIG tool was devised to study temporal logic queries of IOSA models [10], described either in their native syntax or in the JANI model exchange format [16]. Using RES embedded in statistical model checking, FIG computes CIs that estimate the value with which a model satisfies (1) transient and (2) steady-state properties. An example of (1) is the probability of observing a system failure before a given mission time T, i.e. the unreliability value UNREL_T. An example of (2) is the proportion of time that a repairable system remains inoperative, i.e. the unavailability value UNAVA.

FIG was designed for automatic RES, implementing the algorithms from [14,15] to derive an importance function from the system model. For this, FIG uses the property query to identify the states of relevant IOSA modules that represent the rare event. However, in FTA the relevant information is in the structure of the tree, not in the query, so this strategy fails to produce useful importance functions.

FIG can also take a composition function as input, to aggregate the local importance functions of the system modules that are relevant for the rare event. We exploit this feature with our Kepler to IOSA compiler, thus instructing FIG how to implement the importance functions from Sects. 4.1 and 4.2. Note nonetheless that those functions depend at least on the state of BEs and SBEs or, in the general case, on the output z of all nodes v in Table 1 for which z appears in $\mathcal{I}_v$. So FIG offers an option to build local importance functions for those nodes, namely {BE, SBE, PAND, SPARE}, which are the base cases of the global importance function for the whole tree.

Thus from an FT model and via our toolchain, FIG can build the thresholds required to perform ISPLIT using several heuristics, and then run diverse RES algorithms to estimate rare event properties—see e.g. [11,13,17].

6 Experimental evaluation

To demonstrate the effectiveness of our theory we used this toolchain to compute the unreliability and unavailability of 27 highly-resilient repairable non-Markovian DFTs. These models come from seven literature case studies, that we enriched with RBOX elements and non-Markovian failure, dormancy, and repair stochastic distributions.

6.1 General setup

We estimated the UNAVA or UNREL_{10^3} of each tree in increasingly resilient configurations. Thus we show how *Crude Monte Carlo* (CMC) loses efficiency in comparison to our automatic implementations of RES, with an efficiency gap that increases as the dependability metrics become rarer.

Moreover, we compare the quality of the RES algorithms that result from three different importance functions, namely: the compositional function from Sec. 4.1 defined recursively in the structure of the tree $\triangle$, denoted *structural* ($\mathcal{I}_{\mathsf{STR}}$); function $\mathcal{I}_{\mathsf{MCS}}$ from Sec. 4.2 that works on the set of minimal cut sets of $\triangle$; and its normalised variant $\mathcal{I}_{\mathsf{MCSN}}$. We did not consider the pruned variants of $\mathcal{I}_{\mathsf{MCS}}$, which [12] reported as less promising.

To compare the efficiency of these functions empirically, we passed them as composition functions to FIG for different types of ISPLIT algorithms (*engines* in FIG terminology) and heuristics for thresholds selection. We tested the engines: Fixed Effort [27] with different effort values, i.e. different amounts of partial runs performed in each $\mathcal{S}_k$ region; standard RESTART [62], for different values of splitting per threshold; and RESTART with prolonged retrials of level 2 [61], RESTART-P_2, also for different values of splitting per threshold. For each engine we experimented three methods to build thresholds: a modified Sequential Monte Carlo algorithm [15] using global splitting 2 (for the RESTART variants) or global effort 8 (Fixed Effort); the same with values 5 and 16 resp.; the Expected Success algorithm [13], that computes splitting/effort values independently for each $\mathcal{S}_k$ region.

All these configurations result in nine ISPLIT variants: FE_m for $m = 8, 16, \textsc{es}$ and RST_n, $\mathrm{RST2}_n$ for $n = 2, 5, \textsc{es}$. If one of the importance functions $\mathcal{I}_{\mathsf{STR}}$, $\mathcal{I}_{\mathsf{MCS}}$, $\mathcal{I}_{\mathsf{MCSN}}$ shows better efficiency than the rest to analyse all case studies, in one or more of these ISPLIT variants without showing worse performance in the rest, then we conclude that it is (for our practices) a higher-quality function for RES.

More precisely, we define an *instance* $\mathfrak{y}$ as a combination of an algorithm *algo* (i.e. an ISPLIT variant using one of the three functions), an RFT, and a dependability metric. An RFT is identified by a case study (*CS*) and a parameter ($\mathfrak{p}$), where larger values of the parameter of the RFT CS_p indicate smaller dependability values p_{CS_p}. Running *algo* for a fixed simulation time, instance $\mathfrak{y}$ estimates the value $p_\mathfrak{y} \overset{\text{def}}{=} p_{CS_p}$. We fix the confidence coefficient $\delta = 0.95$, so this experiment produces a confidence interval (CI) $\hat{p}_\mathfrak{y}$ that has a certain width $\|\hat{p}_\mathfrak{y}\| \in [0, 1]$. The performance of *algo* can be measured by that width: the smaller $\|\hat{p}_\mathfrak{y}\|$, the more efficient the algorithm that achieved it.

This is a direct and standard approach to quantify efficiency: measure the confidence interval width for a fixed simulation budget. There are, however, two more dimensions to consider. First, the simulation budget may not suffice to observe rare events in certain cases, e.g. when using CMC on an instance with very low dependability value $p_\mathfrak{y}$. In such cases the `FIG` tool reports a null estimate $\hat{p}_\mathfrak{y} = [0, 0]$, which is an indication of poor performance. Second, the simulation of random events depends on the RNG—and its seed—used by `FIG`, so different runs yield CIs of different width. This is another indicator: the less variability observed for these widths, the better the algorithm—inasmuch the CI is narrow, i.e. relative to the first dimension mentioned.

We use these three dimensions to assess the performance of an *algo*: its capability to converge, the expected CI width achieved, and the variability of these widths. For this we repeated 10 times the estimation of $\hat{p}_\mathfrak{y}$ for each instance $\mathfrak{y}$, measuring: (i) how many times it yielded not-null estimates; (ii) what was the average width $\|\hat{p}_\mathfrak{y}\|$; and (iii) what was the standard deviation of those widths.

We performed 10 repetitions to ensure statistical significance: in the bar plots that we present in Sec. 6.3, a 95% CI for a bar is narrower than the whiskers and, in the hardest configuration of every *CS*, the whiskers of algorithms under comparison never overlap.

Case studies. Our seven parametric case studies are:

1. the synthetic model DSPARE$\mathfrak{p}$ [12], with $\mathfrak{p} \in \{3, 4, 5\}$ shared SBEs and 1 RBOX;
2. the synthetic model VOT$\mathfrak{p}$ [12], with $\mathfrak{p} \in \{1, \dots, 4\}$ shared BEs and 1 RBOX;
3. FTPP$\mathfrak{p}$ [26], where we study one triad with $\mathfrak{p} \in \{4, 5, 6\}$ shared SBEs, using one RBOX for the processors and another for the network elements;
4. HECS$\mathfrak{p}$ [57], with 2 memory interfaces, 4 RBOX (one per subsystem), $\mathfrak{p} \in \{1, \dots, 5\}$ shared spare processors, and 2$\mathfrak{p}$ parallel buses;
5. RC$\mathfrak{p}$ [30], with one RBOX and $\mathfrak{p} \in \{3, \dots, 6\}$ SPAREs;
6. HVC$\mathfrak{p}$ [31], with one RBOX and $\mathfrak{p} \in \{2, \dots, 4\}$ shared SBEs;
7. *RWC*$\mathfrak{p} \in \{4, \dots, 7\}$ [53], which is a non-trivial combination of RC$\mathfrak{p}$ and HVC$\mathfrak{p}$ via VOT and OR gates that maintains both independent RBOX elements.

In total we have 27 RFTs with PDFs that include exponential, Erlang, uniform, Rayleigh, Weibull, normal, and log-normal distributions. All details of these case studies can be found in the artifact accompanying this work, where we provide the fault trees written in the Kepler syntax, as well as their IOSA translations [18].

Transient vs. steady-state. From the nine ISPLIT variants, only the three based on standard RESTART (RST*n*) could be used to estimate both types of properties. Fixed Effort (FE_m) was not used for steady-state properties as this requires regeneration theory [27], which is not always feasible with non-Markovian models. Conversely, RESTART-P$_2$ (RST_{2n}) was not used for transient properties because there is no current support for this combination of engine and property in `FIG`.

Hardware. Our experiments ran in a PBS-administered cluster running Linux CenOS 7 (kernel 3.10.0-957), whose nodes have CPUs Intel® Xeon® E5-2680 v4@2.40 GHz (14 cores, 35M cache), each with 384 GB of DDR4 RAM @1600 MHz.

6.2 Experimental results: scatter plots

We start by presenting scatter plots, that offer a high-level overview of our general results, later refined into bar plots.

Fig. 7 compares all runs of CMC against all RES runs that used RESTART. We arrange the plots in a 3 × 3 matrix, where a row indicates a way to select thresholds, and a column indicates an importance function. For instance, the upper-left scatter plot of Fig. 7 compares CMC against RESTART using the $\mathcal{I}_{\text{STR}}$ function and thresholds built with Sequential Monte Carlo for global splitting 2. Instead, the lower-right scatter plot compares CMC to RESTART using the $\mathcal{I}_{\text{MCSN}}$ function and the Expected Success algorithm.

In each scatter plot, a mark at (x, y) coordinates corresponds to an instance whose CI width was x for RESTART and y for CMC. Thus, a mark above the solid diagonal line means that RESTART built a narrower CI than CMC in the same simulation time. Dotted diagonal lines indicate 10× narrower CIs (note that the axes are in logarithmic scale). A mark on the upper- or right-most bars labelled Ø respectively indicates that none of the 10 CMC or RESTART experiments managed to build a CI (e.g. CMC for VOT4).

These x, y values are the robust average of the corresponding 10+10 runs, computed via Z-score$_{m=2}$ to remove outliers [34]. Mark styles differentiate the case studies and properties, e.g. ⬟ compare unavailability for the four VOT$\mathfrak{p}$ case studies, while ♦ represent unreliability for RWC$\mathfrak{p}$.

Fig. 7 shows that in general and as expected, RESTART performs at least as well as CMC to estimate the dependability metrics, with an efficiency gap that increases as the RFT gains on resilience. However, this relative gain concerns UNAVA studies, which are the first-row models in the legend of Fig. 7, that have an A superscript and blue-tainted colour marks. For UNREL$_{1000}$ RESTART fails to surpass CMC significantly: we will show next that Fixed Effort is a much better ISPLIT algorithm in these cases.

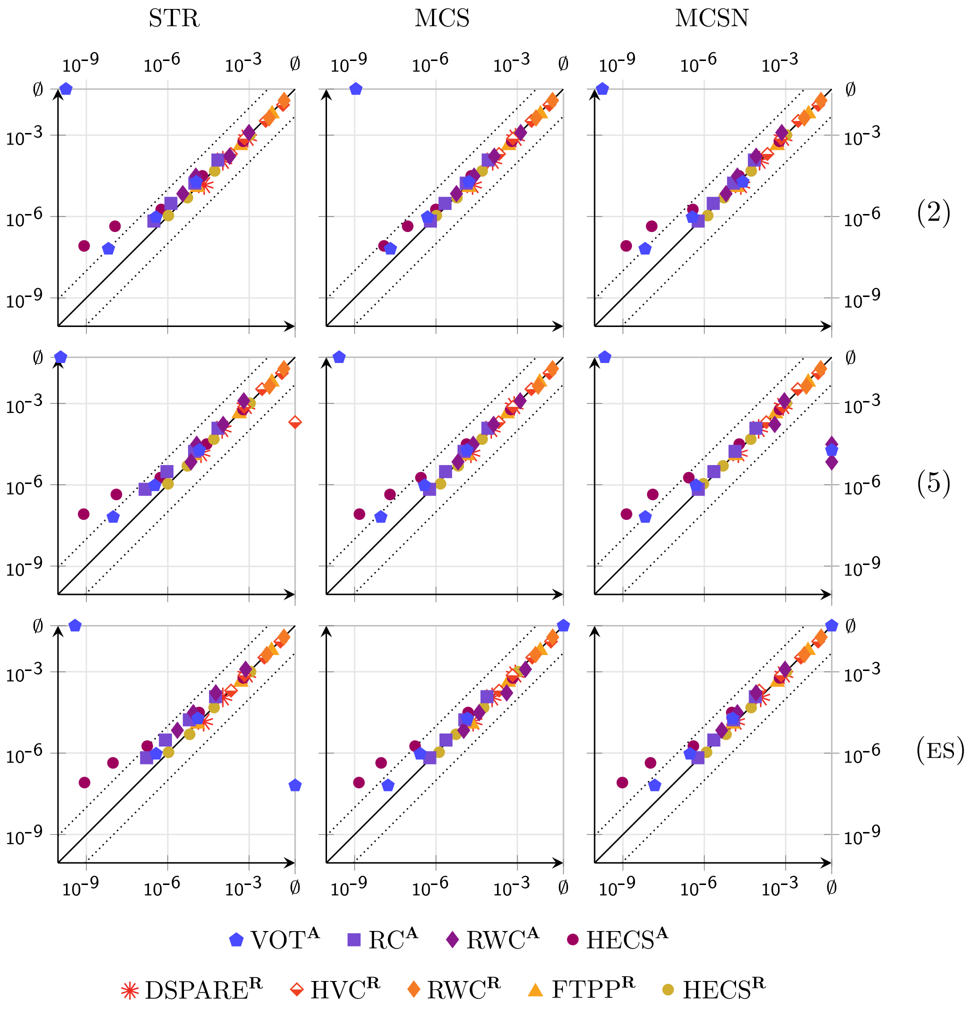


Fig. 7 CI precision of CMC vs. RESTART. A column of plots indicates the importance function used for experimentation with RESTART: structural ($\mathcal{I}_{\mathsf{STR}}$), minimal cut sets ($\mathcal{I}_{\mathsf{MCS}}$), or normalised MCS ($\mathcal{I}_{\mathsf{MCSN}}$). A row indicates the thresholds-building method used with that function: sequential Monte Carlo with global splitting 2 or 5, or Expected Success (ES). Darker marks in a scatter plot correspond to UNAVA experiments, for the case studies VOT, RC, RWC, HECS, represented with an A superscript in the legend of the plots (VOT$^{\mathrm{A}}$ . . .). Lighter marks correspond to UNREL$_{10^3}$ experiments, for the case studies DSPARE, HVC, RWC, FTPP, HECS, represented with an R superscript (DSPARE$^{\mathrm{R}}$ . . .). A mark above the solid diagonal line means that RESTART built a narrower CI than CMC in the same simulation time. Dotted diagonal lines indicate 10× narrower CIs. Both axes are in logarithmic scale.

When comparing importance functions, Fig. 7 shows a tendency that favours $\mathcal{I}_{\mathsf{STR}}$, evidenced in (each row of) plots by the higher mass of marks above the diagonal that occur on the column corresponding to $\mathcal{I}_{\mathsf{STR}}$. Nevertheless, there are two outliers in this trend that we discuss here.

First, RESTART with $\mathcal{I}_{\mathsf{STR}}$ and Expected Success failed to build any CI for UNAVA of the RFT VOT3 (mark ⬟ on the right-most bar of the bottom-left scatter plot). Studying the output logs of `FIG` it was found that Expected Success failed in the selection of a threshold at importance value 44 (out of 70). Although `FIG` has recovery heuristics for such situations, the result for this case was the selection of a splitting value = 2234. This is in contrast to the values observed for all other experiments, that seldom go over 200 and never over 400. Therefore, an oversampling occurred in the $\mathcal{S}_k$ region built above such threshold, which produced a bottleneck that could not be overcome in the 30 min of runtime allowed for this experiment, producing the lack of results.

Second, RESTART with $\mathcal{I}_{\mathsf{STR}}$ and Sequential Monte Carlo for global splitting 5 failed to build any CI for HVC7 when studying UNREL$_{1000}$. This is less surprising given the lower efficiency of RESTART to study unreliability in our experiments, and the nature of this case study that we discuss next when analysing Fixed Effort. Still we studied the output logs of `FIG`, and in this case found an *under*sampling caused by thresholds set too far apart by Sequential Monte Carlo. This is a known issue for this algorithm, that uses a single splitting value for all thresholds.

Thus, both outliers have roots on a combination of the algorithms to build thresholds, and their current implementation in `FIG`. The better performance of RESTART with $\mathcal{I}_{\mathsf{STR}}$, for the rest of the instances when compared to the columns of $\mathcal{I}_{\mathsf{MCS}}$ and $\mathcal{I}_{\mathsf{MCSN}}$, tilt the scale in favour of the structural function $\mathcal{I}_{\mathsf{STR}}$ defined in Sec. 4.1.

Figure 8 shows our experimental results for RESTART-P$_2$, which essentially delays the truncation of simulation retrials for 2 threshold levels. In comparison RESTART can be defined as RESTART-P$_0$, which truncates a retrial as soon as it visits a state whose importance is below the level of creation of the retrial. Empirical and theoretical studies have shown RESTART-P$_1$ and RESTART-P$_2$ to be more efficient than RESTART in the analysis of queueing systems [17,61].

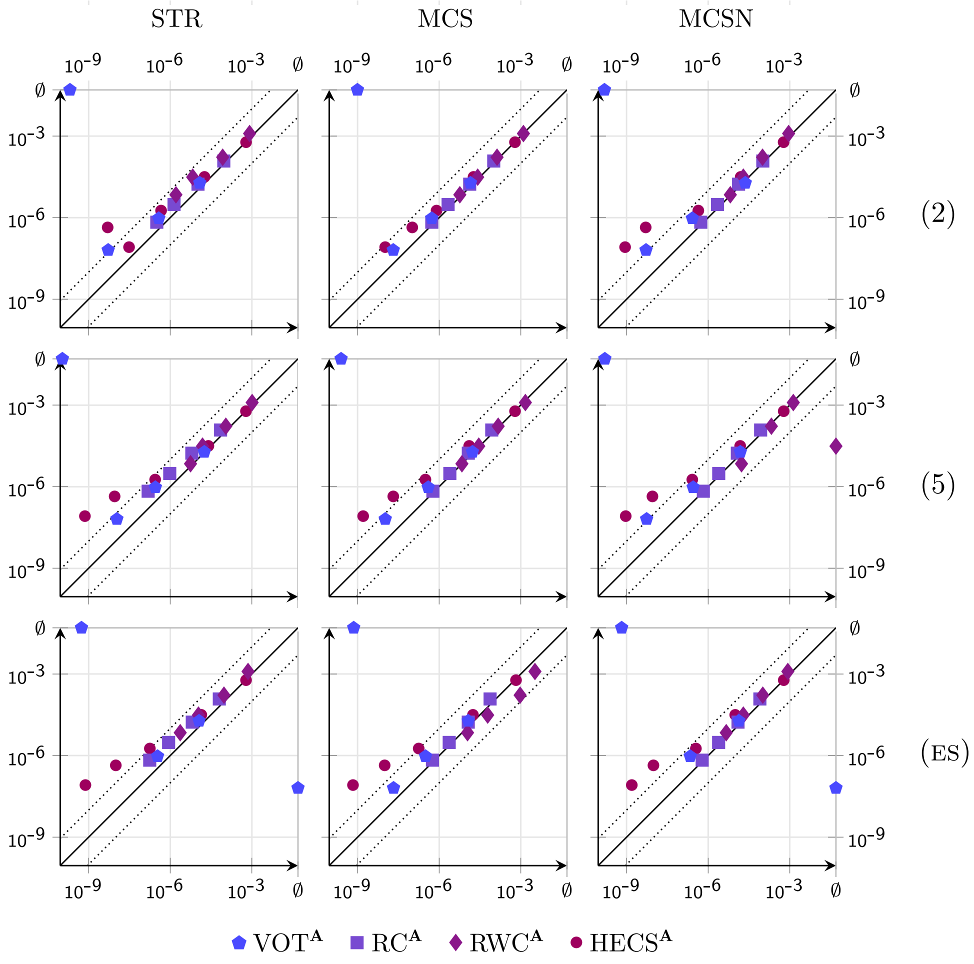


Fig. 8 *CI precision of CMC vs.* RESTART-P$_2$. A column of plots indicates the importance function used for experimentation with RESTART-P$_2$: structural ($\mathcal{I}_{\mathsf{STR}}$), minimal cut sets ($\mathcal{I}_{\mathsf{MCS}}$), or normalised MCS ($\mathcal{I}_{\mathsf{MCSN}}$). A row indicates the thresholds-building method used with that function: sequential Monte Carlo with global splitting 2 or 5, or Expected Success (ES). These are UNAVA experiments on the parameterised case studies VOT, RC, RWC, HECS. A mark above the solid diagonal line means that RESTART-P$_2$ built a narrower CI than CMC in the same simulation time. Dotted diagonal lines indicate 10× narrower CIs. Both axes are in logarithmic scale.

Here we experiment with the latter, within the current capabilities of `FIG` (UNAVA properties only), to further validate our previous discussion for $\mathcal{I}_{\mathsf{STR}}$.

For that function, Fig. 8 shows a similar trend than Fig. 7, with the following difference. When thresholds are selected with Sequential Monte Carlo for global splitting 2 (upper-left scatter plots in the figures), RESTART-P$_2$ managed to build narrower CIs than RESTART, most notably for the RWC$\mathfrak{p}$ case studies. The opposite effect is observed (but to a lower degree) when thresholds are selected with the Expected Success algorithm.

However interesting, this does not change the fact that for any row of scatter plots, those produced with the $\mathcal{I}_{\mathsf{STR}}$ function in Fig. 8 generally produced the narrower CIs. Also, being more sensitive than RESTART to the choice of thresholds via Expected Success, RESTART-P$_2$ puts in evidence that $\mathcal{I}_{\mathsf{MCS}}$ can result in ISPLIT algorithms that perform worse than CMC, e.g. the lower-central scatter plot of Fig. 8. Thus, all in all these results increase the evidence in favour of $\mathcal{I}_{\mathsf{STR}}$ as the most efficient of the three importance functions tested, at least for steady-state analysis.

Finally, Fig. 9 shows our experimental results when using Fixed Effort to study UNREL$_{1000}$. Something that jumps to sight is the high number of failures of the algorithm for all importance functions, but exclusively for the HVC$\mathfrak{p}$ case studies (plus some RWC$\mathfrak{p}$), and paradoxically for the less rare instances of those models.

To understand this we study the structure of these RFTs, whose smallest instance (HVC4) require 6 basic elements to fail in order to trigger a top event. This is not too favourable for ISPLIT, specially considering the fast repair times (uniformly distributed in $[0.15, 0.45]$) with respect to failure times (Erlang$(3, 0.25)$ and Rayleigh(1.999)). Still this did not stop RESTART to at least match the performance of CMC.

In general, the issue with Fixed Effort is that it is more structured, and thus more brittle than RESTART (and CMC), as shown here. By conditioning the success of the whole run on the chained success on every $\mathcal{S}_k$ region, a single failed step produces a 0 estimate and starts all estimations anew. This can be very efficient—see e.g. the CIs ≈ 100× narrower than CMC for HECS5—as it avoids to waste effort in unpromissing simulations. In HVC however, where repair

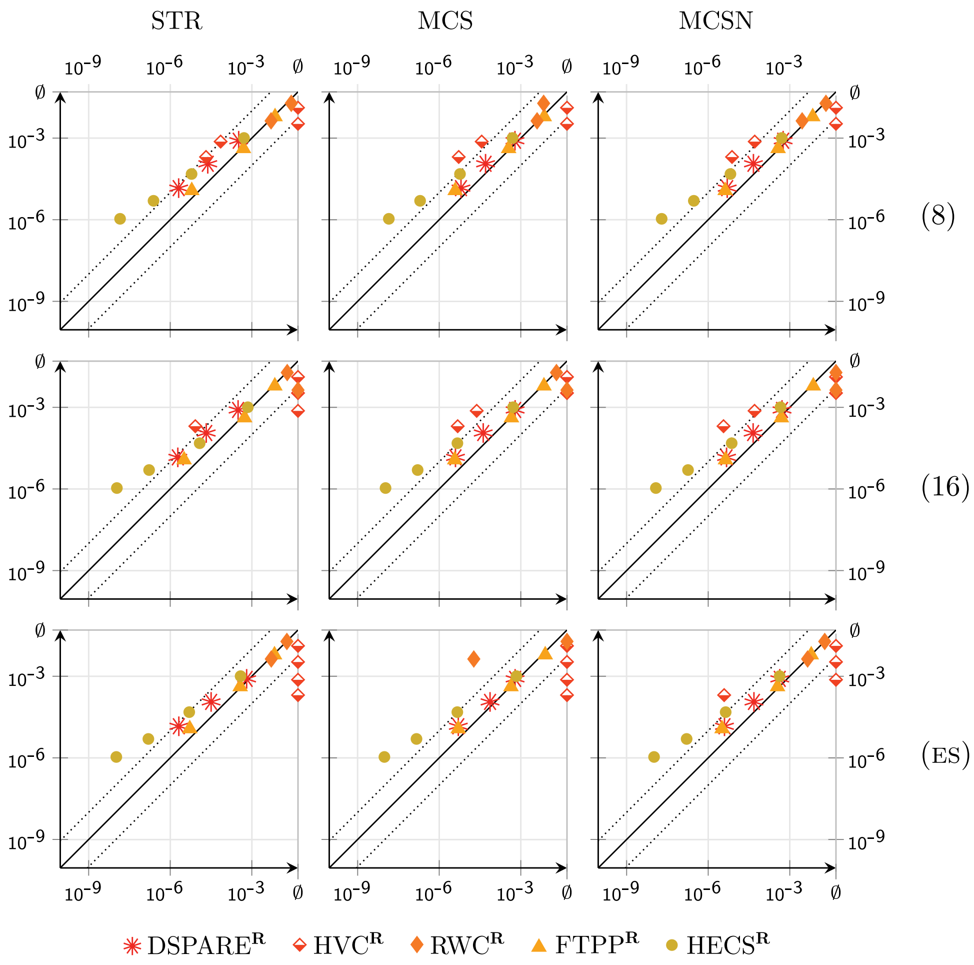


Fig. 9 *CI precision of CMC vs. Fixed Effort.* A column of plots indicates the importance function used for experimentation with Fixed Effort: structural ($\mathcal{I}_{\mathsf{STR}}$), minimal cut sets ($\mathcal{I}_{\mathsf{MCS}}$), or normalised MCS ($\mathcal{I}_{\mathsf{MCSN}}$). A row indicates the thresholds-building method used with that function: sequential Monte Carlo with global effort 8 or 16, or Expected Success (ES). These are UNREL$_{1000}$ experiments on the parameterised case studies DSPARE, HVC, RWC, FTPP, HECS. A mark above the solid diagonal line means that Fixed Effort built a narrower CI than CMC in the same simulation time. Dotted diagonal lines indicate 10× narrower CIs. Both axes are in logarithmic scale.

times are extremely faster than failures, such reset condition happens almost always before the top event of the tree. This does not affect RESTART and CMC so badly, which continue simulations as before. But for Fixed Effort and given the short runtimes allowed for the smallest instances—HVC4 and HVC5 are truncated after 90 and 300 s respectively—it results in null estimates as observed in Fig. 9. This is related to the fact that none of our importance functions considers time—the value of IOSA clocks—as a factor for splitting. We touch upon this subject in the conclusions.

For the rest of the cases, where redundancy (rather than fail vs. repair time) is the root factor for resilience, Fig. 9 shows an excellent performance of Fixed Effort to estimate unreliability. The dominance of $\mathcal{I}_{\mathsf{STR}}$ is not as clear here as it is for UNAVA studies via RESTART and RESTART-P$_2$; however, neither of the other two functions is clearly superior. Consider e.g. DSPARE$\mathfrak{p}$, where $\mathcal{I}_{\mathsf{STR}}$ shows consistent better results than $\mathcal{I}_{\mathsf{MCS}}$ and $\mathcal{I}_{\mathsf{MCSN}}$; and also the best-performance cases, namely HECS$\mathfrak{p}$, where all functions perform similarly for the different ways of choosing thresholds.

Therefore, our general observations remain favourable for $\mathcal{I}_{\mathsf{STR}}$, as the importance function that produces the most efficient ISPLIT implementations in general scenarios.

6.3 Experimental results: bar plots

Unlike the scatter plots in Figs. 7 to 9, the bar plots in this section show the variance of the CI widths produced by each algorithm. These are plotted as whiskers on top of the bars, where the height of a bar indicates the width of the CI achieved by the corresponding instance. Numbers in the range [0, 10] at the base of the bars tell how many of the 10 experimental repetitions managed to build a not-null CI. The label "$p_{\mathfrak{p}} \approx \mu \pm \sigma^2$" at the right of each plot shows the robust mean and variance estimated for the corresponding dependability metric, computed from the complete series of runs (190 independent experiments per case study).

Steady-state studies for RC. Fig. 10 shows the widths of the CIs produced for unavailability estimation on the RC$\mathfrak{p}$ case studies. The plots illustrate how $\mathcal{I}_{\mathsf{STR}}$ performs better than both $\mathcal{I}_{\mathsf{MCS}}$ and $\mathcal{I}_{\mathsf{MCSN}}$ in every one of the ISPLIT variants tested. This difference between the RES implementations

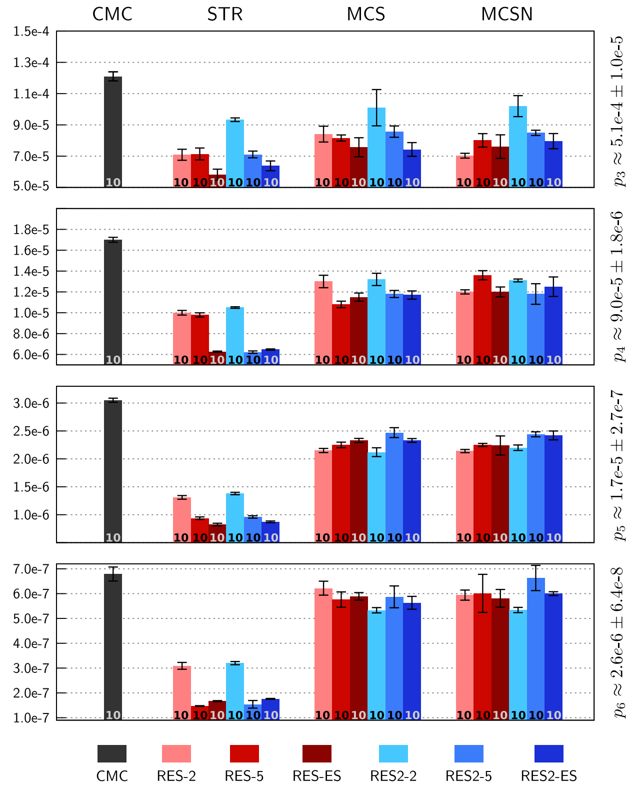


Fig. 10 *CI widths for $UNAVA$ studies of RC*$\mathfrak{p}$. The height of the bars indicates the width of (the robust mean of) the CI width, achieved with 10 runs of Crude Monte Carlo (CMC), RESTART with global splitting 2 (RES-2), ..., or RESTART-P$_2$ with Expected Success (RES2-ES). Lower is better. The whiskers on top of the bars indicate the standard deviation of these widths. The bars are clustered per importance function used with RESTART and RESTART-P$_2$: $\mathcal{I}_{\mathsf{STR}}$, $\mathcal{I}_{\mathsf{MCS}}$, and $\mathcal{I}_{\mathsf{MCSN}}$. The plots show experiments on RC3 (top plot) through RC6 (bottom plot)

resulting from $\mathcal{I}_{\mathsf{STR}}$ and the other functions increases for lower values of the steady-state property, and the variance of these results remains among the lowest of all cases. Moreover, here $\mathcal{I}_{\mathsf{STR}}$ is the only function with which all RESTART algorithms maintain or increase the efficiency gap that sets them apart from CMC.

Transient studies for DSPARE. Fig. 11 shows the widths of the CIs produced for unreliability estimation on the case studies DSPARE$\mathfrak{p}$. Once again we see $\mathcal{I}_{\mathsf{STR}}$ outperforming the other two functions in general, with an efficiency gap and accuracy that increases as the event becomes more rare. However and as discussed in Sec. 6.2, in this case this only happens with Fixed Effort variants, since RESTART fails to perform better than CMC. Yet the difference between the Fixed Effort implementations and the rest (specially with the $\mathcal{I}_{\mathsf{STR}}$ function) is remarkable, and in particular Fixed Effort is the only algorithm capable of producing consistently useful CIs (i.e. that exclude 0).

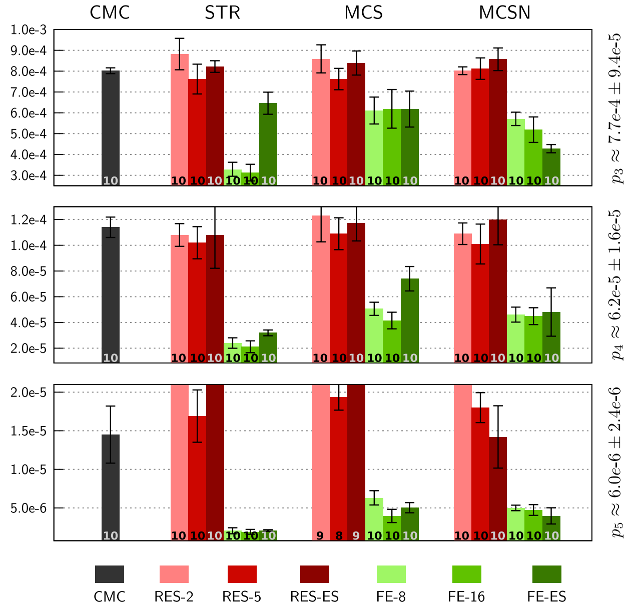


Fig. 11 *CI widths for $UNREL_{1000}$ studies of DSPARE*$\mathfrak{p}$. The height of the bars indicates the width of (the robust mean of) the CI width, achieved with 10 runs of Crude Monte Carlo (CMC), RESTART with global splitting 2 (RES-2), ..., or Fixed Effort with Expected Success (FE-ES). Lower is better. The whiskers on top of the bars indicate the standard deviation of these widths. The bars are clustered per importance function used with RESTART and Fixed Effort: $\mathcal{I}_{\mathsf{STR}}$, $\mathcal{I}_{\mathsf{MCS}}$, and $\mathcal{I}_{\mathsf{MCSN}}$. The plots show experiments on RC3 (top plot) through RC6 (bottom plot)

7 Related work

Most work on DFT analysis assumes discrete [4,57] or exponentially distributed [23,40] components failure. Furthermore, components repair is seldom studied in conjunction with dynamic gates [4,7,40,44,54]. In this work we addressed repairable DFTs, whose failure and repair times can follow arbitrary PDFs. More in detail, RFTs were first formally introduced as stochastic Petri nets in [7,20]. Our work stands on [45,46], which reviews [20] in the context of stochastic automata with arbitrary PDFs. In particular we also address non-Markovian continuous distributions: in Sec. 6 we experimented with exponential, Erlang, uniform, Rayleigh, Weibull, normal, and log-normal PDFs. Furthermore and for the first time (with the exclusion of [12,19] on which this work stands), we consider the application of [20,45] to study rare events.

Much effort in RES has been dedicated to study highly reliable systems, deploying either importance splitting or sampling. Typically, importance sampling can be used when the system takes a particular shape. For instance, a common assumption is that all failure (and repair) times are exponentially distributed with parameters λ^i, for some $\lambda \in \mathbb{R}$ and $i \in \mathbb{N}_{>0}$. In these cases, a favourable change of measure can be computed analytically [29,33,47,48,53,65].

In contrast, when events occur at times following less-structured or even arbitrary distributions, importance splitting is more easily applicable. As long as a full system failure can be broken down into several smaller failures, an importance splitting method can be devised. Of course, its efficiency relies heavily on the choice of importance function. This choice is typically done ad hoc for the model under study [43,58,60]. In that sense [14,15,35,36] are among the first to attempt a heuristic derivation of all parameters required to implement splitting, for which they exploit formal specifications of the model and property query.

Here we extended [10,14,15] in two different ways. One is the natural way in which we use the structure of the fault tree to define composition operands. With these operands we aggregate the automatically-computed local importance functions of the tree nodes. This aggregation results in an importance function for the whole model, that we present in Table 1 as the "structural" function $\mathcal{I}_{\mathsf{STR}}$.

The other extension relates to [58,59], where [58] initially defined a "state variable" (an importance function for the RESTART algorithm) $S(\boldsymbol{x}) = \max_i\{c_i(\boldsymbol{x})\}$, where $c_i(\boldsymbol{x})$ is the number of components of type i that are failed in state $\boldsymbol{x}$. Even though this is defined for a specific system, in essence and viewing the system as a fault tree, $S(\boldsymbol{x})$ counts the maximum number of failed components in any MCS. This was generalised in [59] using cut set analysis to define an importance function which, in our setting, is given by $\Phi(\boldsymbol{x}) = \min_{|MCS|} - \left(\min_{|MCS|} - \sum_{b\in \mathrm{MCS}} z_b\right)$, where in turn $\min_{|MCS|} = \left(\min_{MCS\in\mathscr{M}(\triangle)} |MCS|\right)$. Unlike $S(\boldsymbol{x})$, $\Phi(\boldsymbol{x})$ does not require all cut sets to have the same cardinality. However, both functions are hindered when the branches of the fault tree have different failure probabilities.

In [19] it was proposed to alleviate that issue via cut set pruning and importance normalisation, but the former strategy proved inefficient in the general case. Thus here we chose to experiment with the original and normalised variants of this function, i.e. $\mathcal{I}_{\mathsf{MCS}}$ and $\mathcal{I}_{\mathsf{MCSN}}$ in Table 2.

8 Conclusions

We have presented a theory to deploy automatic importance splitting (ISPLIT) for fault tree analysis of repairable dynamic fault trees (RFTs). This Rare Event Simulation approach supports arbitrary probability distributions of components failure and repair. The core of our theory lies on the general definition of importance functions. Thus, we provide the importance function $\mathcal{I}_{\mathsf{STR}}$, which is defined structurally on the given tree $\triangle$, and a family of importance functions, including $\mathcal{I}_{\mathsf{MCS}}$ and $\mathcal{I}_{\mathsf{MCSN}}$, derived from the collection of minimal cut sets of $\triangle$.

From such functions we have implemented ISPLIT algorithms and used them to estimate the unreliability and unavailability of highly-resilient RFTs. Setting itself apart from classical approaches, that define importance functions ad hoc using expert knowledge, our theory computes all metadata required for RES from the model and metric specifications. From this basis, we have shown how diverse ISPLIT algorithms can be automatically implemented from $\mathcal{I}_{\mathsf{STR}}$, $\mathcal{I}_{\mathsf{MCS}}$, or $\mathcal{I}_{\mathsf{MCSN}}$. Our experimentation shows that these algorithms can converge to narrower confidence intervals than crude Monte Carlo simulation (CMC).

The efficiency gap observed between CMC and the automatic ISPLIT implementations—i.e. how narrow are the CIs achieved for a fixed simulation budget—depends on a number of factors. These include the type of property, the specific RES algorithm, the relation between the fail and repair times of the BEs and SBEs of the tree, and (to a lesser degree) also the heuristic chosen to select thresholds. Nevertheless and in all cases, implementations with the structural importance function proved to be at least as performant as with those based on minimal cut sets, and in many cases consistently better, most prominently with RESTART variants for RES.

Another main advantage of $\mathcal{I}_{\mathsf{STR}}$ over $\mathcal{I}_{\mathsf{MCS}}$ and its variants is that the former is *linear in the size of the tree*, and its bottom-up computation from the tree structure (regardless of whether it is a proper tree or not) is likewise linear in the tree size. In contrast, $\mathcal{I}_{\mathsf{MCS}}$ and its variants are worse-case exponential in the size of the tree. This has not been a problem for the relatively small RFTs considered here, but in industrial applications with thousands of BEs this could quickly become a limiting factor. Moreover, importance functions are constantly evaluated during rare event simulation, so the computation overhead (see e.g. [64]) could easily tilt the scale against $\mathcal{I}_{\mathsf{MCS}}$ even in the cases where the exponential explosion is not observed.

Besides theoretical contributions and to provide an empirical basis to our studies, we have presented a toolchain that demonstrates our automatic RES applications. With the aim to improve a former toolchain we have also introduced Kepler, a textual format to represent RFTs, and changed the ISPLIT engine of FIG to include conditional sampling. This modification increases trace independence during splitting and had a positive impact on the performance of the importance splitting methods in FIG with respect to previous versions. All the experimental results here presented can be inspected and reproduced with the software artifact that we have made publicly available to that end [18].

There are several paths open for future development. First and foremost, we are looking into new ways to define the importance function, e.g. to cover more general categories of FTs such as fault maintenance trees [51]. In addition, we have defined $\mathcal{I}_{\mathsf{STR}}$, $\mathcal{I}_{\mathsf{MCS}}$, and $\mathcal{I}_{\mathsf{MCSN}}$ based on the tree structure alone. It would be interesting to further include stochastic information in this phase, and not only afterwards during the thresholds-selection phase. Finally, we are investigating enhancements in IOSA and our toolchain, to exploit the ratio between fail and dormancy PDFs of SBEs in warm SPARE gates.

Acknowledgements The authors thank Marco Biagi, for the many discussions that lead to the first implementation of the structural importance function in FIG. Thanks extend also to José and Manuel Villén-Altamirano, for fruitful discussions that helped to better understand the application scope of this approach.

A Kepler grammar for repairable DFTs

Kepler is a textual, human-readable syntax to describe Fault Trees with dynamic gates (PAND, SPARE, FDEP), repairs (RBOX), and general continuous distributions. Although a subset of such distributions is currently described, the extension to others is direct. We refer to these extended fault trees as RFTs.

The first line in a Kepler description of an RFT declares which is the top-level gate. Each subsequent line is either empty or it defines a node in the RFT. Each node can be described either with our newly introduced syntax, or by using the legacy Galileo syntax.

In Code 5 we describe the full Kepler syntax in standard BNF notation. Grammar symbols are capitalised words between carets, e.g. <THIS>. In particular, the entry point of the syntax is the production rule <TOPLEVEL>. The horizontal line '|' separates options. Other characters in the production rules are either literals or terminal symbols. Notice that our definition includes a formalisation of the legacy Galileo standard syntax.

Code 5 Kepler grammar for RFTs

```
<TOPLEVEL> ::= toplevel <NAME> ; <KEPLER>

// Kepler syntax

 <KEPLER> ::= <GATE>
            | <BE>
            | <SBE>
            | <GALILEO>
            | <KEPLER> <KEPLER>
   <GATE> ::= <NAME> or    <NAMES> ;
            | <NAME> and   <NAMES> ;
            | <NAME> <VOT> <NAMES> ;
            | <NAME> pand  <NAMES> ;
            | <NAME> fdep  <NAMES> ;
            | <NAME> spare <NAMES> ;
            | <NAME> rbox <RMODE> <NAMES> ;
     <BE> ::= <NAME> <FAIL> ;
            | <NAME> <FAIL> <REP> ;
    <SBE> ::= <NAME> <DORM> <FAIL> ;
            | <NAME> <DORM> <FAIL> <REP> ;
  <RMODE> ::= prio | fcfs | rand
   <FAIL> ::= fail   ~ <DIST>
    <REP> ::= repair ~ <DIST>
   <DORM> ::= dorm   ~ <DIST>

// Galileo

 <GALILEO> ::= <GGATE>
             | <GBE>
             | <GALILEO> <GALILEO>
   <GGATE> ::= <NAME> seq <NAMES> ;
             | <NAME> csp <NAMES> ;
             | <NAME> wsp <NAMES> ;
             | <NAME> hsp <NAMES> ;
     <GBE> ::= <NAME> <GFAIL> ;
             | <NAME> <GFAIL> <GDORM> ;
             | <NAME> <GDORM> <GFAIL> ;
   <GFAIL> ::= lambda = <PREAL>
             | prob   = <PROB>
             | res    = <PROB>
             | rate = <PREAL> shape = <PREAL>
             | mean = <REAL> stddev = <PREAL>
   <GDORM> ::= dorm = <PROB>

// Basic

 <NAMES> ::= <NAME> | <NAME> <NAMES>
  <NAME> ::= " <ALNUM> "
   <VOT> ::= <NAT>of<NAT>
  <DIST> ::= exponential ( <PREAL> )
           | erlang ( <NAT>, <PREAL> )
           | uniform ( <REAL> , <REAL> )
           | normal ( <REAL> , <PREAL> )
           | lognormal ( <REAL> , <PREAL> )
           | gamma ( <PREAL> , <PREAL> )
           | weibull ( <PREAL> , <PREAL> )
           | rayleigh ( <PREAL> )
  <PROB> ::= 0 | 1 | 0.<NAT>
  <REAL> ::= <SGN><PREAL>
   <INT> ::= <SGN><NAT>
 <PREAL> ::= <NAT>
           | <NAT>.<NAT>
           | <NAT>e<INT>
   <NAT> ::= <NUM>
           | <NUM><NAT>
 <ALNUM> ::= <ALPHA>
           | <NUM>
           | <NUM><ALNUM>
           | <ALPHA><ALNUM>
 <ALPHA> ::= A | B | ... | Z
           | a | b | ... | z | _
   <NUM> ::= 0 | 1 | ... | 9
   <SGN> ::=   | + | -
```

References

1. Abate, A., Budde, C.E., Cauchi, N., Hoque, K.A., Stoelinga, M.: Assessment of maintenance policies for smart buildings: application of formal methods to fault maintenance trees. PHM Society European Conference **4**(1) (2018). https://www.phmpapers.org/index.php/phme/article/view/385
2. Baier, C., Katoen, J., Hermanns, H.: Approximate symbolic model checking of continuous-time Markov chains. In: CONCUR 1999, pp. 146–161 (1999). https://doi.org/10.1007/3-540-48320-9_12
3. Bayes, A.J.: Statistical techniques for simulation models. Aust. Comput. J. **2**(4), 180–184 (1970)
4. Beccuti, M., Codetta-Raiteri, D., Franceschinis, G., Haddad, S.: Non deterministic repairable fault trees for computing optimal

repair strategy. In: VALUETOOLS 2008 (2010). https://doi.org/10.4108/ICST.VALUETOOLS2008.4411

5. Blanchet, J., Mandjes, M.: Rare event simulation for queues. In: Rubino and Tuffin [50], pp. 87–124. https://doi.org/10.1002/9780470745403.ch5
6. Blom, H.A.P., Bakker, G.J.B., Krystul, J.: Rare event estimation for a large-scale stochastic hybrid system with air traffic application. In: Rubino and Tuffin [50], pp. 193–214. https://doi.org/10.1002/9780470745403.ch9
7. Bobbio, A., Codetta-Raiteri, D.: Parametric fault trees with dynamic gates and repair boxes. In: RAMS, pp. 459–465. IEEE (2004). https://doi.org/10.1109/RAMS.2004.1285491
8. Boudali, H., Crouzen, P., Haverkort, B.R., Kuntz, M., Stoelinga, M.: Architectural dependability evaluation with Arcade. In: DSN'08, pp. 512–521. IEEE Computer Society (2008). https://doi.org/10.1109/DSN.2008.4630122
9. Boudali, H., Dugan, J.B.: A new Bayesian network approach to solve dynamic fault trees. In: RAMS 2005, pp. 451–456. IEEE (2005). https://doi.org/10.1109/RAMS.2005.1408404
10. Budde, C.E.: Automation of importance splitting techniques for rare event simulation. Ph.D. thesis, FAMAF, Universidad Nacional de Córdoba, Córdoba, Argentina (2017). https://famaf.biblio.unc.edu.ar/cgi-bin/koha/opac-detail.pl?biblionumber=18143
11. Budde, C.E.: FIG: the finite improbability generator. In: TACAS, LNCS, vol. 12078, pp. 483–491. Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_27
12. Budde, C.E., Biagi, M., Monti, R.E., D'Argenio, P.R., Stoelinga, M.: Rare event simulation for non-markovian repairable fault trees. In: TACAS, LNCS, vol. 12078, pp. 463–482. Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_26
13. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Automated compositional importance splitting. Sci. Comput. Program. **174**, 90–108 (2019). https://doi.org/10.1016/j.scico.2019.01.006
14. Budde, C.E., D'Argenio, P.R., Hermanns, H.: Rare event simulation with fully automated importance splitting. In: EPEW 2015, LNCS, vol. 9272, pp. 275–290. Springer (2015). https://doi.org/10.1007/978-3-319-23267-6_18
15. Budde, C.E., D'Argenio, P.R., Monti, R.E.: Compositional construction of importance functions in fully automated importance splitting. In: VALUETOOLS 2016, pp. 30–37 (2017). https://doi.org/10.4108/eai.25-10-2016.2266501
16. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: TACAS, LNCS, vol. 10206, pp. 151–168. Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_9
17. Budde, C.E., Hartmanns, A.: Replicating RESTART with prolonged retrials: an experimental report. In: TACAS, LNCS, vol. 12652, pp. 373–380. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_21
18. Budde, C.E., Monti, R.E., D'Argenio, P.R.: Analysis of non-markovian repairable fault trees through rare event simulation. https://figshare.com/articles/software/Analysis_of_non-Markovian_repairable_fault_trees_through_rare_event_simulation_experimental_reproduction_package_/16907143 (2021). https://doi.org/10.6084/m9.figshare.16907143
19. Budde, C.E., Stoelinga, M.: Automated rare event simulation for fault tree analysis via minimal cut sets. In: MMB, LNCS, vol. 12040, pp. 259–277. Springer (2020). https://doi.org/10.1007/978-3-030-43024-5_16
20. Codetta-Raiteri, D., Iacono, M., Franceschinis, G., Vittorini, V.: Repairable fault tree for the automatic evaluation of repair policies. In: DSN, pp. 659–668. IEEE Computer Society (2004). https://doi.org/10.1109/DSN.2004.1311936
21. Coppit, D., Sullivan, K.J.: Galileo: A tool built from mass-market applications. In: Proceedings of the 2000 International Conference on Software Engineering 2000, pp. 750–753. IEEE (2000)
22. Coppit, D., Sullivan, K.J., Dugan, J.B.: Formal semantics of models for computational engineering: a case study on dynamic fault trees. In: ISSRE 2000, pp. 270–282 (2000). https://doi.org/10.1109/ISSRE.2000.885878
23. Crouzen, P., Boudali, H., Stoelinga, M.: Dynamic fault tree analysis using input/output interactive Markov chains. In: DSN 2007, pp. 708–717. IEEE Computer Society (2007). https://doi.org/10.1109/DSN.2007.37
24. D'Argenio, P.R., Monti, R.E.: Input/Output Stochastic Automata with Urgency: Confluence and weak determinism. In: ICTAC, LNCS, vol. 11187, pp. 132–152. Springer (2018). https://doi.org/10.1007/978-3-030-02508-3_8
25. Distefano, S., Puliafito, A.: Dependability modeling and analysis in dynamic systems. In: 2007 IEEE International Parallel and Distributed Processing Symposium, pp. 1–8 (2007). https://doi.org/10.1109/IPDPS.2007.370601
26. Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Fault trees and sequence dependencies. In: ARMS 1990, pp. 286–293. IEEE (1990). https://doi.org/10.1109/ARMS.1990.67971
27. Garvels, M.J.J.: The splitting method in rare event simulation. Ph.D. thesis, Department of Computer Science, University of Twente, Enschede, The Netherlands (2000). http://eprints.eemcs.utwente.nl/14291/
28. Garvels, M.J.J., van Ommeren, J.K.C.W., Kroese, D.P.: On the importance function in splitting simulation. Eur. Trans. Telecommun. **13**(4), 363–371 (2002). https://doi.org/10.1002/ett.4460130408
29. Goyal, A., Shahabuddin, P., Heidelberger, P., Nicola, V.F., Glynn, P.W.: A unified framework for simulating Markovian models of highly dependable systems. IEEE Trans. Comput. **41**(1), 36–51 (1992). https://doi.org/10.1109/12.123381
30. Guck, D., Katoen, J.P., Stoelinga, M., Luiten, T., Romijn, J.: Smart railroad maintenance engineering with stochastic model checking. In: Railways 2014, Civil-Comp Proceedings. Civil-Comp Press (2014). https://doi.org/10.4203/ccp.104.299
31. Guck, D., Spel, J., Stoelinga, M.: DFTCalc: Reliability centered maintenance via fault tree analysis (tool paper). In: ICFEM 2015, LNCS, vol. 9407, pp. 304–311. Springer (2015). https://doi.org/10.1007/978-3-319-25423-4_19
32. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Form. Asp. Comput. **6**(5), 512–535 (1994). https://doi.org/10.1007/BF01211866
33. Heidelberger, P.: Fast simulation of rare events in queueing and reliability models. ACM Trans. Model. Comput. Simul. **5**(1), 43–85 (1995). https://doi.org/10.1145/203091.203094
34. Iglewicz, B., Hoaglin, D.: How to detect and handle outliers. ASQC basic references in quality control. ASQC Quality Press (1993)
35. Jegourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: CAV 2013, LNCS, vol. 8044, pp. 576–591. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_38
36. Jégourel, C., Legay, A., Sedwards, S., Traonouez, L.M.: Distributed verification of rare properties using importance splitting observers. In: AVoCS 2015, ECEASST, vol. 72 (2015). https://doi.org/10.14279/tuj.eceasst.72.1024
37. Junges, S., Guck, D., Katoen, J., Stoelinga, M.: Uncovering dynamic fault trees. In: DSN 2016, pp. 299–310. IEEE Computer Society (2016). https://doi.org/10.1109/DSN.2016.35
38. Junges, S., Guck, D., Katoen, J.P., Rensink, A., Stoelinga, M.: Fault trees on a diet. In: SETTA 2015, LNCS, vol. 9409, pp. 3–18. Springer (2015). https://doi.org/10.1007/978-3-319-25942-0_1
39. Kahn, H., Harris, T.E.: Estimation of particle transmission by random sampling. Natl. Bur. Stand. Appl. Math. Ser. **12**, 27–30 (1951)
40. Katoen, J.P., Stoelinga, M.: Boosting fault tree analysis by formal methods, LNCS, vol. 10500, pp. 368–389. Springer (2017). https://doi.org/10.1007/978-3-319-68270-9_19

41. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic symbolic model checker. In: International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, pp. 200–204. Springer (2002)
42. Law, A.M.: Simulation modeling and analysis. McGraw-Hill (2014)
43. L'Ecuyer, P., Le Gland, F., Lezaud, P., Tuffin, B.: Splitting techniques. In: Rubino and Tuffin [50], pp. 39–61. https://doi.org/10.1002/9780470745403.ch3
44. Liu, Y., Wu, Y., Kalbarczyk, Z.: Smart maintenance via dynamic fault tree analysis: a case study on Singapore MRT system. In: DSN 2017, pp. 511–518. IEEE Computer Society (2017). https://doi.org/10.1109/DSN.2017.50
45. Monti, R.E.: Stochastic automata for fault tolerant concurrent systems. Ph.D. thesis, FAMAF, Universidad Nacional de Córdoba, Córdoba, Argentina (2018)
46. Monti, R.E., Budde, C.E., D'Argenio, P.R.: A compositional semantics for repairable fault trees with general distributions. In: LPAR, EPiC Series in Computing, vol. 73, pp. 354–372. EasyChair (2020). https://doi.org/10.29007/p16v
47. Nicola, V.F., Shahabuddin, P., Nakayama, M.K.: Techniques for fast simulation of models of highly dependable systems. IEEE Trans. Reliab. **50**(3), 246–264 (2001). https://doi.org/10.1109/24.974122
48. Ridder, A.: Importance sampling simulations of Markovian reliability systems using cross-entropy. Ann. Oper. Res. **134**(1), 119–136 (2005). https://doi.org/10.1007/s10479-005-5727-9
49. Rubino, G., Tuffin, B.: Introduction to rare event simulation. In: Rare event simulation using Monte Carlo methods [50], pp. 1–13. https://doi.org/10.1002/9780470745403.ch1
50. Rubino, G., Tuffin, B. (eds.): Rare event simulation using Monte Carlo methods. Wiley (2009)
51. Ruijters, E., Guck, D., Drolenga, P., Peters, M., Stoelinga, M.: Maintenance analysis and optimization via statistical model checking. In: QEST 2016, LNCS, vol. 9826, pp. 331–347. Springer (2016). https://doi.org/10.1007/978-3-319-43425-4_22
52. Ruijters, E., Guck, D., van Noort, M., Stoelinga, M.: Reliability-centered maintenance of the electrically insulated railway joint via fault tree analysis: a practical experience report. In: DSN 2016, pp. 662–669. IEEE Computer Society (2016). https://doi.org/10.1109/DSN.2016.67
53. Ruijters, E., Reijsbergen, D., de Boer, P.T., Stoelinga, M.: Rare event simulation for dynamic fault trees. Reliab. Eng. Syst. Saf. **186**, 220–231 (2019). https://doi.org/10.1016/j.ress.2019.02.004
54. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15–16**, 29–62 (2015). https://doi.org/10.1016/j.cosrev.2015.03.001
55. Sullivan, K., Dugan, J., Coppit, D.: The Galileo fault tree analysis tool. In: 29th Annual International Symposium on Fault-Tolerant Computing (Cat. No.99CB36352), pp. 232–235. IEEE (1999). https://doi.org/10.1109/FTCS.1999.781056
56. Sullivan, K.J., Dugan, J.B.: Galileo user's manual & design overview. https://www.cse.msu.edu/~cse870/Materials/FaultTolerant/manual-galileo.htm (1998). V2.1-alpha
57. Vesely, W., Stamatelatos, M., Dugan, J., Fragola, J., Minarick, J., Railsback, J.: Fault tree handbook with aerospace applications. NASA Office of Safety and Mission Assurance (2002). Version 1.1
58. Villén-Altamirano, J.: RESTART method for the case where rare events can occur in retrials from any threshold. Int. J. Electron. Commun. **52**(3), 183–189 (1998)
59. Villén-Altamirano, J.: Importance functions for RESTART simulation of highly-dependable systems. Simulation **83**(12), 821–828 (2007). https://doi.org/10.1177/0037549707081257
60. Villén-Altamirano, J.: RESTART vs splitting: a comparative study. Perform. Eval. **121–122**, 38–47 (2018). https://doi.org/10.1016/j.peva.2018.02.002
61. Villén-Altamirano, J.: An improved variant of the rare event simulation method RESTART using prolonged retrials. Oper. Res. Perspect. **6**, 100–108 (2019). https://doi.org/10.1016/j.orp.2019.100108
62. Villén-Altamirano, M., Martínez-Marrón, A., Gamo, J., Fernández-Cuesta, F.: Enhancement of the accelerated simulation method RESTART by considering multiple thresholds. In: Proc. 14_{th} Int. Teletraffic Congress, Teletraffic Science and Engineering, vol. 1, pp. 797–810. Elsevier (1994). https://doi.org/10.1016/B978-0-444-82031-0.50084-6
63. Villén-Altamirano, M., Villén-Altamirano, J.: RESTART: a method for accelerating rare event simulations. In: Queueing, Performance and Control in ATM (ITC-13), pp. 71–76. Elsevier (1991)
64. Villén-Altamirano, M., Villén-Altamirano, J.: Analysis of restart simulation: Theoretical basis and sensitivity study. Eur. Trans. Telecommun. **13**(4), 373–385 (2002). https://doi.org/10.1002/ett.4460130409
65. Xiao, G., Li, Z., Li, T.: Dependability estimation for non-Markov consecutive-k-out-of-n: F repairable systems by fast simulation. Reliab. Eng. Syst. Saf. **92**(3), 293–299 (2007). https://doi.org/10.1016/j.ress.2006.04.004