FIG: the Finite Improbability Generator v1.3

Carlos E. Budde †
University of Trento, Italy
carlosesteban.budde@unitn.it

ABSTRACT

This work presents version 1.3 of the Finite Improbability Generator (FIG): a statistical model checker to estimate transient and steady-state reachability properties in stochastic automata. Specialised in rare event simulation via importance splitting, FIG implements RESTART and Fixed Effort algorithms. Its distinctive feature is push-button automation: users need not define an importance function, because FIG can derive it from the property query and model specifications. The input models are Input/Output Stochastic Automata with Urgency, written either in the native IOSA syntax or in the JANI exchange format. The theory backing FIG has demonstrated good efficiency, comparable to optimal importance splitting implemented ad hoc for specific systems. Written in C++, FIG is FOSS released under the GPLv3 in https://git.cs.famaf.unc.edu.ar/dsg/fig.

Keywords

Rare event simulation; automatic importance function.

1. INTRODUCTION

In formal analysis of stochastic systems, statistical model checking (SMC [1]) emerges as an alternative to numerical techniques such as exhaustive probabilistic model checking. Its partial, on-demand state exploration offers an alternative to exhaustive approaches, with a smaller and constant memory footprint at the cost of typically longer runtimes.

At its core, SMC integrates Monte Carlo simulation with formal models, where—for stateful models—traces of states are generated dynamically e.g. via discrete event simulation. Such traces are samples of the states that a stochastic model usually visits. Via the generation and analysis of these stochastic samples, SMC can estimate the degree to which a model satisfies a property.

For instance, consider a stateful system with stochastically-chosen sojourn times, and a temporal logic property φ that characterises a subset of the model states $S_\varphi \subset S$. From these two inputs an SMC analysis can yield an estimate $\hat{\gamma} \in [0,1]$ of the actual probability γ with which the model satisfies φ. A typical example are steady-state properties, $\varphi = \mathsf{S}(\mathsf{cond})$, where $\hat{\gamma}$ represents the proportion of time spent by the system in states that satisfy the Boolean condition cond in the long run.

Besides producing $\hat{\gamma}$, SMC can quantify the statistical error incurred via two numbers, $\delta \in (0,1)$ and $\varepsilon > 0$, such that $\hat{\gamma} \in [\gamma - \varepsilon, \gamma + \varepsilon]$ with probability δ. Thus, if $n \in \mathbb{N}$ traces are sampled, the full SMC outcome is the tuple $(n, \hat{\gamma}, \delta, \varepsilon)$.

With this statistical quantification—usually output as a confidence interval (CI) around $\hat{\gamma}$—an idea of the quality of an estimation is conferred [2]. Higher quality means *more precision* (smaller ε) or *more confidence* (bigger δ). The usual approach is to fix δ prior to experimentation: then the more sample traces are drawn, the narrower the CI becomes. So n is inversely proportional to ε and thus to the CI width, showing how SMC trades memory for either runtime or precision when compared to exhaustive methods [25].

A main advantage of SMC and its trade-off is that it can trivially analyse systems where the sojourn times in states are governed by non-Markovian stochastic distributions. The vast majority of this field is out of reach for most other model checking approaches, making SMC unique [12].

A main disadvantage are rare events: most traces sampled will not visit S_φ if there is a very low probability γ to satisfy φ. Then the estimate $\hat{\gamma}$ is either (incorrectly) 0, or, if a few traces do visit S_φ, statistical error quantification makes ε skyrocket. In sum, drawing an insufficient number of samples n easily degenerates estimations to yield the trivial confidence interval $= [0,1]$. To counter such phenomena n must increase as γ decreases. Unfortunately, for typical estimates such as the sample mean, it takes $n \geqslant {}^{384}/_{\gamma}$ to build a standard CI where $\delta = 0.95$ and $\varepsilon = \frac{\gamma}{10}$. If e.g. $\gamma \approx 10^{-8}$ then $n \geqslant 38400000000$ traces are needed, causing trace-sampling times to grow unacceptably long. To tackle this issue, rare event simulation (RES) methods have emerged in many different scientific disciplines [6].

Roughly speaking, RES can be divided in the two main areas importance sampling (IS) and importance splitting (ISPLIT). Methods from IS compromise the aforementioned advantage as they tamper the stochastic transitions of the model [4]. In view of this, and since the study of non-Markovian systems is a chief reason to use SMC, the statistical model checker **FIG** specialises in RES by implementing ISPLIT methods. To deploy an efficient implementation, however, both importance sampling and splitting require expert knowledge. The novelty of **FIG** lies on its automatic derivation of the importance function (and thresholds and splitting values) required by ISPLIT. This derivation exploits the model and property under study, resulting in a push-button application of RES for SMC.

† This work was funded by EU grants 830929 (*H2020-CyberSec4Europe*) and 952647 (*H2020-AssureMOSS*).

This paper presents the latest version of the tool, FIG 1.3, demonstrating its interface and most salient capabilities.

Outline and contributions. § 2 formalises the analysis setting from § 1. The input syntax (model & properties) of FIG 1.3 is introduced in § 3. Functional additions w.r.t. to previous versions are then highlighted in § 4, followed by installation instructions and a demonstration of the command line interface of FIG, presented here for the first time.

Related work. Other statistical model checkers offer RES methods to some degree of automation. Plasma Lab implements automatic IS and semiautomatic ISPLIT for Markov chains (DTMCs) [14]. A wizard guides users to choose an importance function based on a layered decomposition of the property query—not the system model. Via APIs, the ISPLIT engine of Plasma Lab may be extended beyond DTMCs. $\mathcal{S}$BIP 2.0 [19] implements an equivalent (semiautomatic, property-based) engine for DTMCs, while offering a richer set of temporal logics to define the property query in. Cosmos [9] and DFTRES [24] implement importance sampling on Markov chains, the latter specialising in systems described as repairable Dynamic Fault Trees (DFTs). All these tools can operate directly on Markovian models, and none offers fully automated ISPLIT. Instead, the SMC tool modes [25] supports non-Markovian probability distributions and is much closer to the capabilities of FIG, offering a similar degree of automation. As a matter of fact, all core RES algorithms in modes were inspired in or motivated by the theory behind FIG. On the one hand, FIG is restricted to fully-stochastic (IOSA) models, while modes uses the LSS algorithm [**succeeds**] to handle nondeterminism as well. On the other hand, FIG offers more full- and semi-automatic heuristics to build importance functions and thresholds, and it has an operation mode specialised for DFT analysis.

Previous versions of this tool have been used for scientific experimentation and research: FIG 1.0 was the first to implement and exercise the theory of [13]; FIG 1.1 was presented in [16] and used in [25]; FIG 1.2 was introduced in [23].

2. THE FIG APPROACH TO RES

Let S be the countable state space of a formal system model, where sojourn times in every $s \in S$ are described by (possibly non-Markovian) stochastic distributions [1]. RES methods can make more traces visit the *rare states* $S_\varphi \subset S$ that satisfy a temporal property φ, to reduce the variance of SMC estimators. For a fixed budget of traces n, this yields narrower CIs than crude Monte Carlo simulation (CMC).

From a plethora of RES methods FIG implements importance splitting, which can work on non-Markovian systems without special considerations. ISPLIT splits S into layers that wrap S_φ like an onion. Reaching the rare event, i.e. generating a trace that begins at the outer layer of S and reaches some state in S_φ, is then broken down into many steps, each involving partial traces called *retrials*. A retrial starts a discrete event simulation from the current layer i, and tries to reach an inner layer $(i+1)$ closer to S_φ. Thus, the i-th step estimates the conditional probability to reach layer $i+1$ from the outer layer i. This stepwise estimation of conditional probabilities can be much more efficient than trying to go at once from the surface of the onion to S_φ [5].

More formally, let S be the states of a model M with initial states S_0 and rare states S_φ. ISPLIT Works on a partition $\biguplus_{i=0}^{M} S_i = S$, where $S_\varphi = S_M$. To estimate the probability $\gamma = \mathit{Prob}(S_\varphi \,|\, S_0)$, each conditional probability $\gamma_i = \mathit{Prob}(S_i \,|\, S_{i-1})$ is estimated separately via CMC. Then simply $\hat{\gamma} = \prod_{i=1}^{M} \hat{\gamma}_i \approx \prod_{i=1}^{M} \gamma_i = \gamma$.

This approach is correct, i.e. it yields an unbiased estimator $\hat{\gamma} \xrightarrow{n\to\infty} \gamma$. However, it is efficient iff $\forall_{i=1}^{M} . \gamma_i \gg \gamma$, which depends on how the S_i layers where chosen. For this, an *importance function* $f\colon S \to \mathbb{R}_{\geqslant 0}$ and *thresholds* $\ell_i \in \mathbb{R}_{\geqslant 0}$ are defined: then $S_i = \{s \in S \mid \ell_i \leqslant f(s) < \ell_{i+1}\}$, where $\ell_0 = 0$, and S_φ are the states with highest *importance*, i.e. $f(s) \geqslant \ell_M$. The efficiency of ISPLIT is thus delegated to the choice of $\{\ell_i\}_{i=1}^{M}$ and the importance function f.

These choices are the key challenge in ISPLIT [5]. Many theoretical developments assume f is given [3, 10], and applications define it ad hoc via (RES and domain) expert knowledge [11, 15]. There is, nonetheless, one general rule of thumb: *importance must be proportional to the probability of reaching* S_φ. Thus for $s, s' \in S$, if a trace that visits s' is the most likely to observe a rare state, one wants $f(s) \leqslant f(s')$. This means that f depends both on the model M and the property φ that defines S_φ, which is what FIG exploits.

FIG, an SMC tool, uses the formal definitions of M and φ to derive f and $\{\ell_i\}_{i=1}^{M}$ in an heuristic approximation to the rule of thumb. For this, FIG runs a breadth-first search from S_φ on the (inverted) transitions of M. This computes the number-of-transitions distance from each state to S_φ. The heuristic importance function of FIG, $f^\star$, is the inverse of this distance, stored as an array the size of S.

To avoid the state explosion FIG works on modular formalisms, deriving a local $f_i^\star$ for each M_i whose parallel composition forms M. $f^\star$ is an aggregation of these functions, which in its most basic form adds the local $f_i^\star$ of every M_i whose variables appear explicitly in φ. Details are in [16] and [25]: the difference with the (later) implementation in modes is that FIG uses the disjunctive normal form of φ.

Note that $f^\star$ is solely based on the distance measured in number of transitions of M. All stochastic behaviour that is omitted by $f^\star$, such as probabilistic weights in the transitions, is captured in the thresholds ℓ_i. To choose these thresholds automatically—called *thresholds building*—FIG runs dynamic analyses using either Expected Success [21], or a variant of the Sequential Monte Carlo algorithm [10, 13]. In both cases finite-life simulations start from S_0, to estimate roughly the probability to reach states with higher importance via lightweight statistical analyses.

As an illustration say k_1 out of m simulations visit states with importance $i_1 > i_0 = f^\star(s)$ with $s \in S_0$. Then 1 out of $e_1 = \lceil \frac{m}{k_1} \rceil$ simulations are expected to reach threshold $\ell_1 = i_1$. If $e_1 > 1$, the algorithm builds *threshold* ℓ_1 with its corresponding *effort* e_1. All threshold-building algorithms in FIG are, essentially, a sequential repetition of this principle.

This is how, from the same input than CMC, i.e. the stochastic model M and temporal property φ, FIG derives $f^\star$ and $\{\ell_i, e_i\}_{i=1}^{M}$ automatically to deploy RES via ISPLIT.

3. MODELLING FORMALISM AND INPUT LANGUAGES

3.1 Input/Output Stochastic Automata

FIG studies Input/Output Stochastic Automata with urgency (IOSA [18]). These can be described as an extension of Generalised semi-Markov processes amenable to composition, where a model is formed of modules that run in parallel.

```
1  module M1
2      fc  : clock;
3      rc  : clock;
4      inf : [0..2];
5      brk : [0..2] init 0;
6      [fl!] brk==0 @ fc -> 0.9: (inf'=1) & (brk'=brk+1)
7                        + 0.1: (fc'=μ);
8      [r??] brk==1       -> (brk'=2) & (rc'=ν);
9      [up!] brk==2 @ rc -> (inf'=2) & (brk'=0) & (fc'=μ);
10     [f!!] inf==1       -> (inf'=0);
11     [u!!] inf==2       -> (inf'=0);
12 endmodule
```

Code 1: IOSA module in FIG 1.3

An IOSA module contains local continuous random variables called clocks. Every clock samples a positive value according to its probability density function (PDF). As time evolves, the clocks in all modules count down at the same rate, and the first to reach zero is said to expire. On expiration a clock can trigger (a) local events in its *active module*—e.g. new sampling of clock values, variables assignment—and (b) synchronisations with other *passive modules*. The single active module whose clock expired broadcasts an *output* action, that synchronises with homonymous *input* actions in the passive modules. IOSA is an input-enabled formalism.

Actions are orthogonally classified as (non-) urgent, where urgent outputs have maximal progress. IOSA can thus exhibit nondeterminism: to allow simulation, [20] gives conditions that ensure its absence modulo weak bisimulation.

IOSA variables in FIG have local scope and can be of type `clock`, `bool`, or ranged integer e.g. `[0..2]`. Constants can also be `float` and have global scope. FIG supports array variables and can compute e.g. a-random/the-smallest value.

Code 1 shows the guarded command language of FIG models, which was inspired in PRISM [8]. Semicolons terminate lines inside of modules. A *variable declaration line* has a unique variable name, its type, and an optional initialisation. A *behavioural line* has a (possibly empty) action in square brackets, a Boolean precondition, and a postcondition formed of a probabilistic choice of options (after `->` and separated by `+`) where single options have probability 1.

Decorators `?`/`!` after an action name indicate that it is input/output, e.g. `fl!` in line 6. Double decorators are for urgency, e.g. `r??` in line 8 is an urgent input. Character `@` indicates the expiration of the clock whose name succeeds it. So for instance `[fl!] brk==0 @ fc-> ···` (line 6 in Code 1) tells that, if `brk==0`, this module will output action `fl` on expiration of clock `fc`. In particular, non-urgent output actions such as `fl` can only be broadcast on clock expiration; instead urgent actions involve no time and thus no clock.

Each postcondition option, chosen probabilistically, is a sequence of effects concatenated by `&`. The left-hand side of `=` in an effect is a variable name with a `'` suffix. If the variable is a clock, the right-hand side is a PDF, e.g. `fc'=`μ in line 7 samples a new clock value.[†] Else, the right-hand side is an arithmetic expression, e.g. `1` and `brk+1` in line 6.

JANI. Besides its native input syntax, FIG 1.3 reads models written in the JANI exchange format [17]. Model types supported are continuous-time Markov chains, and a subset of Stochastic Timed Automata that matches the semantics of IOSA, i.e. with a single PDF per clock and that implement broadcast synchronisation. FIG also translate IOSA to JANI as Stochastic Timed Automata, to share models with tools such as modes, e.g. for complementation and comparisons [17, 21, 23, 27].

[†] In spite of the permissive syntax, IOSA clocks are random variables so they can sample values from a single PDF, cf. lines 7 and 9 in Code 1. Adding e.g. an effect `fc'=`ν (with $\nu \neq \mu$) would make FIG produce an error when compiling the model.

3.2 Properties

FIG estimates the probability with which a stochastic model satisfies temporal logic formulæ. A formula is specified as a transient or steady-state property query, within a properties environment in the model file.

```
1  properties
2    P( q2>0 U q2==8 )
3    S( q2>=8 )
4    S[9:999]( q2>=8 )
5  endproperties
```

Code 2: Property queries in FIG 1.3

Transient properties in FIG correspond to the PCTL-like query `P=?` in PRISM. For instance, the first property in Code 2 asks the probability of assigning value 8 to the variable `q2`, before it takes a value $\leqslant 0$.

Steady-state properties in FIG correspond to unbounded CSL-like queries `S=?` in PRISM, e.g. lines 3 and 4 in Code 2, which query the proportion of time that `q2` has a value $\geqslant 8$. FIG implements batch means for steady-state estimation, as usual in SMC. The initial transient simulation time to discard, as well as the batch time, can be heuristically computed by the tool. These values can also be input by the user, e.g. the last property specifies 9 and $999 - 9$ resp.

4. FIG v1.3

4.1 The software tool

The Finite Improbability Generator is written in C++14 and is now publicly available in its official website at https://git.cs.famaf.unc.edu.ar/dsg/fig. FIG is free open-source software (FOSS) released under the GNU GPLv3.

Requirements & setup. The native environment of FIG is Linux, where it can be compiled with the GNU compiler collection (gcc) $\geqslant$ 5.4, or the LLVM C-compiler (clang) $\geqslant$ 4.0. CMake $\geqslant$ 2.8.13 is also required, as well as the development versions of the bison $\geqslant$ 3.0.4 and flex $\geqslant$ 2.6.4 libraries, and the Z3 theorem prover $\geqslant$ 4.4.1 from Microsoft research.[*] Local copies of the corresponding `.so` files can be placed in the `lib` subdirectory of the code source, checked at compilation if the system does not provide a required library. To install FIG 1.3 simply download the codebase from the official repository and execute the `build_fig.sh` script at the base. A compile option allows choosing between the main fig project (`-p main`, which compiles the `fig` executable) and the tests (`-p tests`, which compiles `test`). Other options offered are the compiler (`-c gcc` and `-c clang`) and the compile mode (`-m release` and `-m debug`). Invoking the script wihtout arguments defaults to `-p main -m release -c clang`. In any case that clang is not found, the compiler falls back to the default version of gcc in the system.[‡]

[*] Arch Linux offers homonymous packages in its `core` and `extra` repositories; in Debian-based distros such as Ubuntu the corresponding packages are `cmake`, `libbison-dev`, `libfl-dev`, and `libz3-dev`.

[‡] These instructions are for Linux. Earlier FIG versions have ran in Mac: this has no current official support but the main developer can be contacted for it. There is no support planned for Windows.

New in this version. Three main features have been incorporated to the tool since its presentation in [23]: **(a)** From the modelling perspective and as illustrated in § 3.1, the postcondition of a transition can now branch among several options, with probabilistic weights quantifying the likelihood to choose each option. In principle this brings discrete-branching models such as DTMCs and (determinisic) Markov automata within the analysis scope of **FIG**. Although experimental results demonstrate this, e.g. the 7-nodes networks in [26], a formal theory showing algorithmic correctness is still lacking. **(b)** From the simulation perspective, the family of RESTART algorithms with prolonged retrials has been implemented to analyse steady-state properties [22, 26]. These algorithms delay the truncation of retrials, reducing the stochastic dependency among simulations but increasing the simulation overhead. The best trade-off was found at prolongation levels 1 and 2, where the original RESTART algorithm is defined as prolongation 0. **(c)** Also affecting simulations, a new feature for clock resampling has been added in **FIG** 1.3, that samples new values for the clocks of a retrial (conditioned on the time already expired) every time that the splitting mechanism is triggered. This is done via the inverse method for distributions whose conditional PDF has a known analytic form, i.e. exponential, uniform, Weibull, and Rayleigh. For other distributions **FIG** uses rejection sampling, bounding the effort based on the likelihood to sample a new valid value: if this would take more than four attempts on average then resampling is suppressed.

4.2 Command line interface

The basic invocation of **FIG** takes three mandatory options: the `<model>` file path, the simulation `<strategy>`, and the `<termination>` criteria. To see their full syntax and semantics call **FIG** with `--help` or see [16]. For instance:

```
>_ fig model.sa --cmc --stop-conf .95 .2
```

invokes the tool to run crude Monte Carlo simulations, to estimate the properties specified in the IOSA file `model.sa`. For each property, simulations will run until a 95%-confidence-level CI is built, whose width is 20% that of the point estimate value $\hat{\gamma}$ that is being computed.

This asks for a CI whose width is given as a proportion of an estimate unknown a priori. It is usual for RES to work with such *relative error*: when $0 < \gamma \ll 1$ for γ unknown, asking for fixed-width CIs easily becomes too lax (yielding useless estimates) or too tight (estimations take too long).

The main issue with this approach is that $\hat{\gamma}$—whose value defines the termination condition—is updated at the same time than the CI. Guaranteeing true CI coverage in such situations is an open problem in RES. Instead it is possible to fix the number of samples or runtime for the estimation, and report a CI in terms of relative error. For instance:

```
>_ fig model.sa --amono --stop-time 2m
```

makes **FIG** run estimations for two minutes per property, using RES with default parameters via an automatically built monolithic importance function (option `--amono`). The output of the tool reports the precision—i.e. CI width—achieved for confidence levels 80%, 90%, 95%, and 99%:

```
>_ Property: P( (q2>0) U (q2==8) )
    + RNG & seed:          mt64 & 32363521 (randomized)
    + importance function: monolithic
    + post-processing:     (null)
    + threshold builder:   hyb
    + simulation engine:   restart
    + resample on split:   yes
    [ 13 thresholds | global effort = 3 ]
    - Requested runtime:   00:02:00
    - Ends at:             21:59:10
    - Batch (ini):         64
   Time-out Interruption
      · Computed estimate: 5.38e-06 (124300416 samples)
      · Computed variance: 5.38e-06
      · 80% confidence
          - precision: 5.71e-07
          - interval: [ 5.10e-06, 5.67e-06]
      · 90% confidence
          ⋮
      · Estimation time: 120.00 s
```

The `<strategy>` used to run simulations can be CMC or RES. For the latter the user can pass an importance function with the `--adhoc <ifun>` option, using variables of the modules in the function expression, e.g. `--adhoc 'q1+2*q2'`. But the pith of **FIG** is its ability to build importance functions automatically: passing the `--amono` or `--acomp` options makes **FIG** build the importance function $f^\star$ described in § 2.

The compositional importance function takes a binary associative arithmetic operator as argument, to aggregate the local functions $f_i^\star$ built for the IOSA modules M_i. If none is passed **FIG** defaults to summation; `--acomp '*'` changes this to product. Alternatively, users can pass an arbitrary function whose variables are the modules for which **FIG** built the $f_i^\star$, e.g. `--acomp 'Queue1+2*Queue2'`. The difference with the previous ad-hoc example is that here the operands are *the importance functions* automatically built for these modules, rather than specific variables within the modules.

In particular, this is used to build importance functions for IOSA translated from (Dynamic) Fault Trees, where a composition strategy can be derived from the DFT structure. However, **FIG** only builds local importance functions for the modules whose variables appear in the property queries. Since the state of a DFT is fully defined by its Basic Elements (plus some special gates such as SPARE and PAND), **FIG** offers the `--ft` option to force the construction of all the $f_i^\star$ that are typically required for DFT analysis.

All these options operate with the discrete variables in a model. A clock, in contrast, has its stochastic behaviour encapsulated in its PDF, and its (continuous) values are not explicit in an IOSA module. Therefore, the $f^\star$ importance function is oblivious of it. Such stochastic information—crucial for an efficient implementation of ISPLIT—is instead conveyed by the thresholds and their effort.

FIG offers several mechanisms to build thresholds: besides `--thresholds-adhoc <list>`, the most prominent are a modified SMC algorithm [10] (`-t hyb`, the default), and Expected Success [21] (`-t es`). The former has proved efficient in many experiments, but it requires an (arbitrary) unique effort applied to all thresholds, making it a semi-automatic procedure. This global effort can be set with `-g <value>`, and optimal values exist for continuous state spaces [11]. In contrast, Expected Success is fully automatic as it computes the optimal effort for each threshold chosen (see § 2).

The final customisation of a RES strategy is the *simulation engine*. To estimate transient properties **FIG** 1.3 offers RESTART (`-e restart`) and Fixed Effort (`-e sfe`).

For steady-state properties users can choose RESTART or its prolonged variants (`-e restart1` through `restart6`). Although the optimal choice depends strongly on the model and property, [22] shows that RESTART with prolongation of 1 or 2 levels is expected to achieve narrower CIs for a fixed simulation budget. This was experimentally (and semi-independently) replicated in the experiments of [26]. Still, the default simulation engine of **FIG** is RESTART, which can analyse both types of properties.

JANI compatibility is transparent for estimations: the user can specify an (IOSA-compatible) JANI model and **FIG** will run estimations on the resulting IOSA translation. The `--to-jani` and `--from-jani` options make the tool translate models without running simulations. Moreover, the new option `--to-jani-modest` modifies the scope of variables and module synchronisation, to output an (IOSA incompatible) JANI Stochastic Automata that can be fed to modes.

Other relevant CLI options of **FIG** 1.3 are: `--no-resampling` to turn off the clock-resampling mechanism (on by default); `-r <mt64/pcg32/pcg64>` to select the internal RNG (defaults to `mt64` = 64-bit Mersenne-Twister); `--rng-seed <value>` to define the RNG seed[¶]; `--timeout <duration>` to truncate estimations (and output preliminary results) after the duration, which must be specified in the same format than the GNU coreutils but with integers only, i.e. `<int></s/m/h/d>`; and `--post-process` to modify the importance values prior to thresholds selection, e.g. `--post-process shift 2` increases by 2 every value of $f^\star$.

4.3 Demonstration

We conclude this work by showing the capabilities of the software tool, studying rare event properties in two small examples from its test suite. The first is a triple tandem queue with Erlang service times [22]: the IOSA model file is publicly available in the official website of **FIG** in the following path: `tests/models/3tandem_queue.sa`.

We compare CMC and two RES strategies with the monolithic importance function, i.e. $f^\star$ built on the composition of all IOSA modules. The first strategy uses all of **FIG** default parameters, and the second one requests Expected Success to build thresholds, and the RESTART engine with level-2 prolongations. The corresponding commands are:

```
>_ fig --stop-time 5m 3tandem_queue.sa --cmc
fig --stop-time 5m 3tandem_queue.sa --amono
fig --stop-time 5m 3tandem_queue.sa --amono -t es \
    -e restart2
```

We estimate the steady-state property `S(q3>=7)`, which asks the proportion of time that the third queue contains more than 7 elements. Comparisons were done for a fixed simulation budged, namely a wall-clock time horizon of 5 minutes. When the time is due, simulations stop and CIs are reported: the estimation that achieves the narrowest CI for a fixed confidence level is the most efficient one.

Running these experiments in an Intel(R) Xeon(R) E-2124G CPU @ 3.40GHz (Linux kernel 5.14.8-arch1-1) resulted in the following 95% CIs: $[3.81\text{E-}6, 4.52\text{E-}6]$ for CMC, $[4.15\text{E-}6, 4.36\text{E-}6]$ for **FIG** defaults, and $[4.25\text{E-}6, 4.40\text{E-}6]$ for the custom command. The corresponding widths of these intervals are 7.13E-7, 2.12E-7, and 1.53E-7.

[¶] The C++ standards do not specify reproducibility of the sequences produced by the distributions from its `random` library, so using the same RNG seed twice does not guarantee obtaining the same results.

All CIs overlap and contain the expected value 4.25E-6. However and as expected, RES can achieve tighter estimates for the same simulation budget. We highlight that the default **FIG** command is as bare as crude Monte Carlo, yet it produced an estimate more than three times more precise. The custom command shows that tweaking some parameters such as the simulation engine can further increase the gain, without special considerations about the model.

To end this section we experiment with a second model: a small repairable Fault Tree with non-Markovian failure and repair times (`FT.sa`), also available in the website of **FIG** as `tests/models/resampling_tiny_FT.sa`. The distribution families include exponential, Erlang, normal, and lognormal.

The case study is quite interesting because importance splitting has limited applications in FT analysis. Importance functions such as $f^\star$, that only observe failures and repairs of components, result in efficient RES applications iff the dominant failure can be layered, e.g. as the result of the conjunctive failure of many subcomponents. To exploit this we have developed heuristics that (automatically) derive a composition strategy from the FT structure. This is fed to **FIG** as the argument of `--acomp`.

In this case we estimate a transient property: the time-bounded probability of observing a system failure before 150 time units. Again we compare CMC and two RES strategies: **FIG** with the `--ft` switch, Expected Success thresholds, Fixed Effort simulation engine, and (a) the default compositional importance function, and (b) the heuristic FT-structure importance function. The commands are:

```
>_ fig --stop-time 5m FT.sa --cmc
fig --stop-time 5m FT.sa --ft -t es -e sfe --acomp +
fig --stop-time 5m FT.sa --ft -t es -e sfe --acomp \
    'BE_0+max(BE_1,BE_2)+BE_4'
```

Experiments ran as before, resulting in the following 95% CIs: $[1.93\text{E-}4, 3.02\text{E-}4]$, $[2.28\text{E-}4, 3.12\text{E-}4]$, and $[2.39\text{E-}4, 2.70\text{E-}4]$, whose widths are 1.09E-4, 8.41E-5, and 3.12E-5.

As before all CIs contain the expected value (2.65E-4), and RES achieved the tightest intervals for the same simulation budget. In this case, however, the difference between CMC and the default compositional strategy of **FIG** is much less pronounced than in the previous example. This is expected given the low redundancy required to cause a system failure (three components must be simultaneously failed).

Yet in spite of this, the heuristic composition strategy performed significantly better, producing a CI almost an order of magnitude narrower than CMC. Perhaps the most appealing feature of this strategy is that it is automatic: we compute it from the FT structure, from which we also create the IOSA modules on which **FIG** runs the simulations.

Final notes. Thus we demonstrate the feasibility to deploy efficient RES via importance splitting, to a degree of automation that approximates and can equal crude Monte Carlo. All is implemented in the **FIG** tool, freely available in https://git.cs.famaf.unc.edu.ar/dsg/fig. We end by highlighting that this is but a small showcase of the interface and capabilities of **FIG** 1.3. Deeper studies to back the statements in §§ 2 and 4, and the theory behind **FIG** in general, require experimental repetition to show statistical significance, and ideally also software artifacts that permit reproducing the results presented. That is out of the scope of this tool demo paper: interested readers are referred to

[21, 23, 26] for deeper performance and scientific analyses.

References

[1] Håkan L. S. Younes and Reid G. Simmons. "Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling". In: *CAV*. Vol. 2404. LNCS. Springer, 2002, pp. 223–235. DOI: 10.1007/3-540-45657-0_17.

[2] John A. Rice. *Mathematical Statistics and Data Analysis*. 3rd ed. 2007. ISBN: 0-534-39942-8.

[3] Thomas Dean and Paul Dupuis. "Splitting for rare event simulation: A large deviation approach to design and analysis". In: *Stochastic Processes and their Applications* 119.2 (2009), pp. 562–587. DOI: 10.1016/j.spa.2008.02.017.

[4] Pierre L'Ecuyer, Michel Mandjes, and Bruno Tuffin. "Importance Sampling in Rare Event Simulation". In: ed. by Gerardo Rubino and Bruno Tuffin. Wiley, 2009. Chap. 2, pp. 17–38. DOI: 10.1002/9780470745403.ch2.

[5] Pierre L'Ecuyer et al. "Splitting Techniques". In: ed. by Gerardo Rubino and Bruno Tuffin. Wiley, 2009. Chap. 3, pp. 39–61. DOI: 10.1002/9780470745403.ch3.

[6] Gerardo Rubino and Bruno Tuffin. "Introduction to Rare Event Simulation". In: ed. by Gerardo Rubino and Bruno Tuffin. Wiley, 2009. Chap. 1, pp. 1–13. DOI: 10.1002/9780470745403.ch1.

[7] Gerardo Rubino and Bruno Tuffin, eds. *Rare Event Simulation Using Monte Carlo Methods*. Wiley, 2009. DOI: 10.1002/9780470745403.

[8] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems". In: *CAV*. Vol. 6806. LNCS. Springer, 2011, pp. 585–591. DOI: 10.1007/978-3-642-22110-1_47.

[9] Benoît Barbot, Serge Haddad, and Claudine Picaronny. "Coupling and Importance Sampling for Statistical Model Checking". In: *TACAS*. Vol. 7214. LNCS. Springer Berlin Heidelberg, 2012, pp. 331–346. DOI: 10.1007/978-3-642-28756-5_23.

[10] Frédéric Cérou et al. "Sequential Monte Carlo for rare event estimation". In: *Statistics and Computing* 22.3 (2012), pp. 795–808. DOI: 10.1007/s11222-011-9231-6.

[11] José Villén-Altamirano and Manuel Villén-Altamirano. "Rare event simulation of non-Markovian queueing networks using RESTART method". In: *Simulation Modelling Practice and Theory* 37 (2013), pp. 70–78. DOI: 10.1016/j.simpat.2013.05.012.

[12] Averill M. Law. *Simulation Modeling and Analysis*. 5th ed. 2015. ISBN: 978-0-07-340132-4.

[13] Carlos E. Budde, Pedro R. D'Argenio, and Raúl E. Monti. "Compositional Construction of Importance Functions in Fully Automated Importance Splitting". In: *VALUETOOLS*. ICST, 2016. DOI: 10.4108/eai.25-10-2016.2266501.

[14] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. "Plasma Lab: A Modular Statistical Model Checking Platform". In: *ISoLA*. Vol. 9952. LNCS. 2016, pp. 77–93. DOI: 10.1007/978-3-319-47166-2_6.

[15] Pietro Turati, Nicola Pedroni, and Enrico Zio. "Advanced RESTART method for the estimation of the probability of failure of highly reliable hybrid dynamic systems". In: *Reliability Engineering & System Safety* 154.C (2016), pp. 117–126. DOI: 10.1016/j.ress.2016.04.020.

[16] Carlos E. Budde. "Automation of Importance Splitting Techniques for Rare Event Simulation". PhD thesis. Córdoba, Argentina: Universidad Nacional de Córdoba, 2017.

[17] Carlos E. Budde et al. "JANI: Quantitative Model and Tool Interaction". In: *TACAS*. Vol. 10206. LNCS. Springer, 2017, pp. 151–168. DOI: 10.1007/978-3-662-54580-5_9.

[18] Pedro R. D'Argenio and Raúl E. Monti. "Input/Output Stochastic Automata with Urgency: Confluence and Weak Determinism". In: *ICTAC*. Vol. 11187. LNCS. Springer, 2018, pp. 132–152. DOI: 10.1007/978-3-030-02508-3_8.

[19] Braham Lotfi Mediouni et al. "$\mathcal{S}$BIP 2.0: Statistical Model Checking Stochastic Real-Time Systems". In: *ATVA*. Vol. 11138. LNCS. Springer, 2018, pp. 536–542. DOI: 10.1007/978-3-030-01090-4_33.

[20] Raúl E. Monti. "Stochastic Automata for Fault Tolerant Concurrent Systems". PhD thesis. Córdoba, Argentina: Universidad Nacional de Córdoba, 2018.

[21] Carlos E. Budde, Pedro R. D'Argenio, and Arnd Hartmanns. "Automated compositional importance splitting". In: *Science of Computer Programming* 174 (2019), pp. 90–108. DOI: 10.1016/j.scico.2019.01.006.

[22] José Villén-Altamirano. "An improved variant of the rare event simulation method RESTART using prolonged retrials". In: *Operations Research Perspectives* 6 (2019), pp. 100–108. DOI: 10.1016/j.orp.2019.100108.

[23] Carlos E. Budde. "FIG: The Finite Improbability Generator". In: *TACAS*. Vol. 12078. LNCS. Springer International Publishing, 2020, pp. 483–491. DOI: 10.1007/978-3-030-45190-5_27.

[24] Carlos E. Budde, Enno Ruijters, and Mariëlle Stoelinga. "The Dynamic Fault Tree Rare Event Simulator". In: *QEST*. Vol. 12289. LNCS. Springer International Publishing, 2020, pp. 233–238. DOI: 10.1007/978-3-030-59854-9_17.

[25] Carlos E. Budde et al. "An efficient statistical model checker for nondeterminism and rare events". In: *International Journal on Software Tools for Technology Transfer* 23 (6 2020), pp. 759–780. DOI: 10.1007/s10009-020-00563-2.

[26] Carlos E. Budde and Arnd Hartmanns. "Replicating RESTART with Prolonged Retrials: An Experimental Report". In: *TACAS*. Vol. 12652. LNCS. Springer International Publishing, 2021, pp. 373–380. DOI: 10.1007%2F978-3-030-72013-1_21.

[27] Carlos E. Budde et al. "On Correctness, Precision, and Performance in Quantitative Verification". In: *ISoLA*. Vol. 12479. LNCS. Springer International Publishing, 2021, pp. 216–241. DOI: 10.1007/978-3-030-83723-5_15.