A Statistical Model Checker for Nondeterminism and Rare Events

Carlos E. Budde[1], Pedro R. D'Argenio[2,3,4], Arnd Hartmanns[1](✉), and Sean Sedwards[5]

[1] University of Twente,
Enschede, The Netherlands
{c.e.budde,a.hartmanns}@utwente.nl
[2] Universidad Nacional de Córdoba,
Córdoba, Argentina
dargenio@famaf.unc.edu.ar
[3] CONICET, Córdoba, Argentina
[4] Saarland University, Saarbrücken, Germany
[5] University of Waterloo, Waterloo, Canada
sean.sedwards@uwaterloo.ca

Abstract. Statistical model checking avoids the state space explosion problem in verification and naturally supports complex non-Markovian formalisms. Yet as a simulation-based approach, its runtime becomes excessive in the presence of rare events, and it cannot soundly analyse nondeterministic models. In this tool paper, we present modes: a statistical model checker that combines fully automated importance splitting to efficiently estimate the probabilities of rare events with smart lightweight scheduler sampling to approximate optimal schedulers in nondeterministic models. As part of the Modest Toolset, it supports a variety of input formalisms natively and via the Jani exchange format. A modular software architecture allows its various features to be flexibly combined. We highlight its capabilities with an experimental evaluation across multi-core and distributed setups on three exemplary case studies.

1 Introduction

Statistical model checking (SMC [30,49]) is a formal verification technique for stochastic systems. Using a formal stochastic model, specified as e.g. a continuous-time Markov chain (CTMC) or a stochastic Petri net (SPN), SMC can answer questions such as "what is the probability of system failure between two inspections" or "what is the expected time to complete a given workload". It is gaining popularity for complex applications where traditional exhaustive probabilistic model checking is limited by the state space explosion problem and by its inability to efficiently handle non-Markovian formalisms or complex continuous dynamics. At its core, SMC

This work is supported by the 3TU.BSR project, ERC grant 695614 (POWVER), the JST ERATO HASUO Metamathematics for Systems Design project (JPMJER1603), the NWO SEQUOIA project, and SeCyT-UNC projects 05/BP12 and 05/B497.

D. Beyer and M. Huisman (Eds.): TACAS 2018, LNCS 10806, pp. 340–358, 2018.
https://doi.org/10.1007/978-3-319-89963-3_20

is the integration of classical Monte Carlo simulation with formal models. By only sampling concrete traces of the model's behaviour, its memory usage is effectively constant in the size of the state space, and it is applicable to any behaviour that can effectively be simulated.

The result of an SMC analysis is an *estimate* $\hat{q}$ of the actual quantity q together with a statistical statement on the potential error. A typical guarantee is that, with probability δ, any $\hat{q}$ will be within $\pm\,\epsilon$ of q. To strengthen such a guarantee, i.e. increase δ or decrease ϵ, more samples (that is, simulation runs) are needed. Compared to exhaustive model checking, SMC thus trades memory usage for accuracy or runtime. A particular challenge lies in *rare events*, i.e. behaviours of very low probability. Meaningful estimates need a small *relative* error: for a probability on the order of 10^{-19}, for example, ϵ should reasonably be on the order of 10^{-20}. In a standard Monte Carlo approach, this would require infeasibly many simulation runs.

SMC naturally works for formalisms with non-Markovian behaviour and complex continuous dynamics, such as generalised semi-Markov processes (GSMP) and stochastic hybrid Petri nets with many generally distributed transitions [42], for which the exact model checking problem is intractable or undecidable. As a simulation-based approach, however, SMC is incompatible with nondeterminism. Yet (continuous and discrete) nondeterministic choices are desirable in formal modelling for concurrency, abstraction, and to represent absence of knowledge. They occur in many formalisms such as Markov decision processes (MDP) or probabilistic timed automata (PTA [38]). In the presence of nondeterminism, quantities of interest are defined w.r.t. optimal *schedulers* (also called policies, adversaries or strategies) that resolve all nondeterministic choices: the verification result is the *maximum* or *minimum* probability or expected value ranging over *all* schedulers. Many SMC tools that appear to support nondeterministic models as input, e.g. Prism [37] and Uppaal smc [14], implicitly use a single hidden scheduler by resolving all choices randomly. Results are thus only guaranteed to lie *somewhere* between minimum and maximum. Such implicit resolutions are a known problem affecting the trustworthiness of simulation studies [36].

In this paper, we present a statistical model checker, modes, that addresses both of the above challenges: It implements *importance splitting* [45] to efficiently estimate the probabilities of rare events and *lightweight scheduler sampling* [39] to statistically approximate optimal schedulers. Both methods can be combined to perform rare event simulation for nondeterministic models.

Rare Event Simulation. The key challenge in rare event simulation (RES) is to achieve a high degree of automation for a general class of models. Current approaches to automatically derive the importance function for importance splitting, which is critical for the method's performance, are mostly limited to restricted classes of models and properties, e.g. [7,18]. modes combines several importance splitting techniques with the compositional importance function construction of Budde et al. [5] and two different methods to derive levels and splitting factors [4]. These method combinations apply to arbitrary stochastic models with a partly discrete state space. We have shown them to work well across different Markovian and non-Markovian automata- and dataflow-based formalisms [4]. We present details on modes' support for RES in Sect. 3. Alongside Plasma lab [40], which implements automatic *importance sampling* [33] and semi-automatic importance splitting [32,34] for Markov chains (with APIs allowing for extensions to other models), modes is one of the most automated tools for RES on formal models today. In particular, we are not aware of any other tool that provides fully automated RES on general stochastic models.

Nondeterminism. Sound SMC for nondeterministic models is a hard problem. For MDP, Brázdil et al. [3] proposed a sound machine learning technique to incrementally improve a partial scheduler. Uppaal Stratego [13] explicitly synthesises a "good" scheduler before using it for a standard SMC analysis. Both approaches suffer from worst-case memory usage linear in the number of states as all scheduler decisions must be stored explicitly. Classic memory-efficient sampling approaches like the one of Kearns et al. [35] address discounted models only. modes implements the lightweight scheduler sampling (LSS) approach introduced by Legay et al. [39]. It is currently the only technique that applies to reachability probabilities and undiscounted expected rewards—as typically considered in formal verification—that also keeps memory usage effectively constant in the number of states. Its efficiency depends only on the likelihood of sampling near-optimal schedulers. modes implements the existing LSS approaches for MDP [39] and PTA [10,26] and supports unbounded properties on Markov automata (MA [16]). We describe modes' LSS implementation in Sect. 4.

The modes Tool. modes is part of the Modest Toolset [24], which also includes the explicit-state model checker mcsta and the model-based tester motest [21]. It inherits the toolset's support for a variety of input formalisms, including the high-level process algebra-based Modest language [22] and xSADF [25], an extension of scenario-aware dataflow. Many other formalisms are supported via the Jani interchange format [6]. As simulation is easily and efficiently parallelisable, modes fully exploits multi-core systems, but can also be run in a distributed fashion across homogeneous or heterogeneous clusters of networked systems. We describe the various methods implemented to make modes a correct and scalable statistical model checker that supports classes of models ranging from CTMC to stochastic hybrid automata in Sect. 2. We focus on its software architecture in Sect. 5. Finally, Sect. 6 uses three very different case studies to highlight the varied kinds of models and analyses that modes can handle.

Previous Publications. modes was first described in a tool demonstration paper in 2012 [2]. Its focus was on the use of partial order and confluence reduction-based techniques [27] to decide on-the-fly if the nondeterminism in a model is spurious, i.e. whether maximum and minimum values are the same and an implicit randomised scheduler can safely be used. modes was again mentioned as a part of the Modest Toolset in 2014 [24]. Since then, modes has been completely redesigned. The partial order and confluence-based methods have been replaced by LSS, enabling the simulation of non-spurious nondeterminism; automated importance splitting has been implemented for rare event simulation; support for MA and a subset of stochastic hybrid automata (SHA [22]) has been added; and the statistical evaluation methods have been extended and improved. Concurrently, advances in the shared infrastructure of the Modest Toolset, now at version 3, provide access to new modelling features and formalisms as well as support for the Jani specification.

2 Ingredients of a Statistical Model Checker

A statistical model checker performs a number of tasks to analyse a given formal model w.r.t. to a property of interest. In this section, we describe these tasks, their challenges, and how modes implements them. All random selections in an SMC tool are typically resolved by a *pseudo*-random number generator (PRNG). For brevity, we write "random" to mean "pseudo-random" in this section.

Simulating Different Model Types. The most basic task is *simulation*: the generation of random samples—*simulation runs*—from the probability distribution over behaviours defined by the model. modes contains simulation algorithms specifically optimised for the following types of models:

- For deterministic **MDP** (Markov decision processes), i.e. DTMC (discrete-time Markov chains), simulation is simple and efficient: Obtain the current state's probability distribution over successors, randomly select one of them (using the distribution's probabilities), and continue from that state.
- Deterministic **MA** (Markov automata [16]) are CTMC. Here, the situation is similar: Obtain the set of enabled outgoing transitions, randomly select a delay from the exponential distribution parameterised by the sum of their rates, then make a random selection of one transition weighted by the transitions' rates.
- **PTA** (probabilistic timed automata [38]) extend MDP with clock variables, transition guards and location invariants as in timed automata. Like MA, they are a continuous-time model, but explicitly keep a memory of elapsed times in the clocks. They admit finite-state abstractions that preserve reachability probabilities and allow them to essentially be simulated as MDP. modes implements region graph- and zone-based simulation of PTA as MDP [10,26]. With fewer restrictions, they can also be treated as SHA:
- **SHA** extend PTA with general continuous probability distributions and continuous variables with dynamics governed by differential equations and inclusions. modes supports deterministic SHA where all differential equations are of the form $\dot{v} = e$ for a continuous variable v and an expression e over *discrete* variables. This subset can be simulated without the need for approximations; it corresponds to deterministic rectangular hybrid automata [29]. For each transition, the SHA simulator needs to compute the set of time points at which it is enabled. These sets can be unions of several disjoint intervals, which results in relatively higher computational effort for SHA simulation.

Properties and Termination. SMC computes a value for the property on every simulation run. A run is a finite trace; consequently, standard SMC only works for linear-time properties that can be decided on finite traces. modes supports

- **transient** (**reachability**) queries of the form $\mathbb{P}(\neg \mathit{avoid}\,\mathtt{U}\,\mathit{goal})$ for the probability of reaching a set of states characterised by the state formula *goal* before entering the set of states characterised by state formula *avoid*, and
- **expected reward** queries of the form $\mathbb{E}(\mathit{reward} \mid \mathit{goal})$ for the expected accumulated reward (or cost) over the reward structure *reward* when reaching a location in the set of states characterised by *goal* for the first time.

Transient queries may be time- and reward-bounded. A state formula is an expression over the (discrete and continuous) variables of the model without any temporal operators. A reward structure assigns a rate reward $r(s) \in \mathbb{R}$ to every state s and a branch reward $r(b) \in \mathbb{R}$ to every probabilistic branch b of every transition. An example transient query is "what is the probability to reach a destination (*goal*) within an energy budget (a reward bound) while avoiding collisions (*avoid*)". Expected reward queries allow asking for e.g. the expected number of retransmissions (the reward) until a message is successfully transmitted (*goal*) in a wireless network protocol. Every query q can be turned into a *requirement* $q \sim c$ by adding a comparison $\sim \in \{\leq, \geq\}$ to a constant value $c \in \mathbb{R}$.

A simulation run ends when the value of a property is decided. For transient properties, this is the case when reaching an *avoid* state or a deadlock (value 0), or a *goal* state (value 1). To ensure termination, the probability of eventually encountering one of these events must be 1. modes additionally implements cycle detection: it keeps track of a configurable number n of previous visited states. When a run returns to a previous state without intermediate steps of probability <1, it will loop forever on this cycle and the run has value 0. modes uses $n = 1$ by default for good performance while still allowing models built for model checking, which avoid deadlocks but often contain terminal states with self-loops, to be simulated. For expected rewards, when entering a *goal* state, the property is decided with the value being the sum of the rewards along the run.

Statistical Evaluation of Samples. n simulation runs provide a sequence of independent values $v_1, \ldots, v_n$ for the property. $\hat{v}_n = \frac{1}{n}\sum_{i=1}^{n} v_i$ is an unbiased estimator of the actual probability or expected reward v. An SMC tool must stop generating runs at some point, and quantify the statistical properties of the estimate $\hat{v} = \hat{v}_n$ returned to the user. modes implements the following methods:

- For a given half-width w and confidence δ, the **CI** method returns a confidence interval $[x, y]$ that contains $\hat{v}$, with $y - x = 2 \cdot w$. Its guarantee is that, if the SMC analysis is repeated many times, $100 \cdot \delta\,\%$ of the confidence intervals will contain v. For transient properties, where the v_i are sampled from a Bernoulli distribution, modes constructs a binomial proportion confidence interval. For expected rewards, the underlying distribution is unknown, and modes uses the standard normal (or Gaussian) confidence interval. This relies on the central limit theorem for means, assuming a "large enough" n. modes requires $n \geq 50$ as a heuristic. modes requires the user to specify δ plus either of w and n. If n is not specified, the CI method becomes a sequential procedure: generate runs until the with of the interval for confidence δ is below $2 \cdot w$. The CI method can be turned into a hypothesis test for requirements $q \sim c$ by checking whether $\hat{v} \geq y$ or $\hat{v} \leq x$, and returning undecided if $\hat{v}$ is inside the interval. When n is unspecified, this is the Chow-Robbins sequential test [44]. Finally, modes can be instructed to interpret the value of w as a relative half-width, i.e. the final interval will have width $\hat{v} \cdot 2 \cdot w$. This is useful for rare events.
- The **APMC** [30] method, based on the Okamoto bound [41], guarantees for error ϵ and confidence δ that $\mathbb{P}(|\hat{v} - v| > \epsilon) < \delta$. It only applies to the Bernoulli-distributed samples for transient properties here. modes requires the user to

specify any two of ϵ, δ and n, out of which the missing value can be computed. The APMC method can be used as a hypothesis test for $\mathbb{P}(\cdot) \sim c$ by checking whether $\hat{v} \geq c + \epsilon$ or $\hat{v} \leq c - \epsilon$, and returning undecided if neither is true.
- modes also implements Wald's **SPRT**, the sequential probability ratio test [47]. As a sequential hypothesis test, it has no predetermined n, but decides on-the-fly whether more samples are needed as they come in. It is a test for Bernoulli-distributed quantities, i.e. it only applies to transient requirements of the form $\mathbb{P}(\cdot) \sim c$. For indifference level ϵ and error α, it stops when the collected samples so far provide sufficient evidence to decide between $\mathbb{P}(\cdot) \geq c + \epsilon$ or $\mathbb{P}(\cdot) \leq c - \epsilon$ with probability $\leq \alpha$ of wrongly accepting either hypothesis.

For a more detailed description of these and other statistical methods and especially hypothesis tests for SMC, we refer the interested reader to [44].

Distributed Sample Generation. Simulation is easily and efficiently parallelisable. Yet a naïve implementation of the statistical evaluation—processing the values from the runs in the order they flow in—risks introducing a bias in a parallel setting. Consider estimating the probability of system failure when simulation runs that encounter failure states are shorter than other runs, and thus quicker. In parallel simulation, failure runs will tend to arrive earlier and more frequently, thus overestimating the probability of failure. To avoid such bias, modes uses the adaptive schedule first implemented in Ymer [48]. It adapts to differences in the speed of nodes by scheduling to process more future results from fast nodes when current results come in quickly. It always commits to a schedule *a priori* before the actual results arrive, ensuring the absence of bias. It is thus well-suited for heterogeneous clusters of machines with significant performance differences.

3 Automated Rare Event Simulation

With the standard confidence of $\delta = 0.95$, we have $n \approx 0.37/\epsilon^2$ in the APMC method: for every decimal digit of precision, the number of runs increases by a factor of 100. If we attempt to estimate probabilities on the order of 10^{-4}, i.e. $\epsilon \approx 10^{-5}$, we need billions of runs and days or weeks of simulation time. This is the problem tackled by rare event simulation (RES) techniques [45]. modes implements RES for transient properties via *importance splitting*, which iteratively increases the simulation effort for states "closer" to the goal set. Closeness is represented by an *importance function* $f_I \colon S \to \mathbb{N}$ that maps each state in S to its importance in $\{ 0, \ldots, \max f_I \}$. The performance, but not the correctness, of all splitting methods hinges on the quality of the importance function.

Deriving Importance Functions. Traditionally, the importance function is specified ad hoc by a RES expert. Striving for usability by *domain* experts, modes implements the compositional importance function generation method of [5] that is applicable to any compositional stochastic model $\mathtt{M} = \mathtt{M}_1 \parallel \ldots \parallel \mathtt{M}_n$ with a partly discrete state space. We write $s|_i$ for the projection of state s of $\mathtt{M}$ to the discrete local variables of component $\mathtt{M}_i$. The method works as follows [4]:

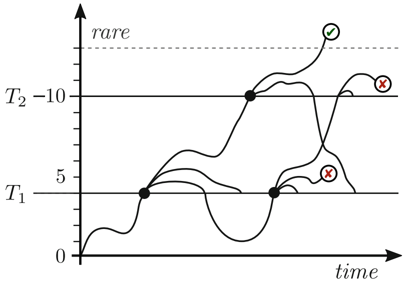

Fig. 1. Illustration of RESTART [4]

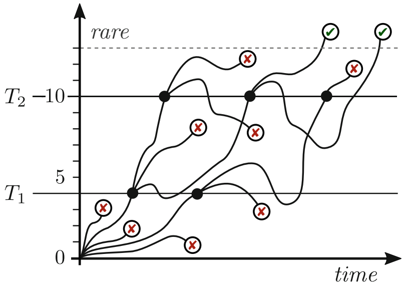

Fig. 2. Illustration of fixed effort [4]

1. Convert the goal set formula *goal* to negation normal form (NNF) and associate each literal $goal^j$ with the component $\mathtt{M}(goal^j)$ whose local state variables it refers to. Literals must not refer to multiple components.
2. Explore the *discrete part* of the state space of each component $\mathtt{M}_i$. For each $goal^j$ with $\mathtt{M}_i = \mathtt{M}(goal^j)$, use reverse breadth-first search to compute the local minimum distance $f_i^j(s|_i)$ of each state $s|_i$ to a state satisfying $goal^j$.
3. In the syntax of the NNF of *goal*, replace every occurrence of $goal^j$ by $f_i^j(s|_i)$ with i such that $\mathtt{M}_i = \mathtt{M}(goal^j)$, and every Boolean operator $\wedge$ or $\vee$ by $+$. Use the resulting formula as the importance function $f_I(s)$.

The method takes into account both the structure of the goal set formula and the structure of the state space. This is in contrast to the approach of Jégourel et al. [32], implemented in a semi-automated fashion in PLASMA LAB [34,40], that only considers the structure of the (more complex linear-time) logical property. The memory usage of the compositional method is determined by the number of discrete local states (required to be finite) over all components. Typically, component state spaces are small even when the composed state space explodes.

Levels and Splitting Factors. We also need to specify *when* and *how much* to "split", i.e. increase the simulation effort. For this purpose, the values of the importance function are partitioned into *levels* and a *splitting factor* is chosen for each level. Splitting too much too often will degrade performance (oversplitting), while splitting too little will cause starvation, i.e. few runs that reach the rare event. It is thus critical to choose good levels and splitting factors. Again, to avoid the user having to make these choices ad hoc, modes implements two methods to compute them automatically. One is based on the sequential Monte Carlo splitting technique [8], while the other method, named *expected success* [4], has been newly developed for modes. It strives to find levels and factors that lead to one run moving up from one level to the next in the expectation.

Importance Splitting Runs. The derivation of importance function, levels and splitting factors is a preprocessing step. Importance splitting then replaces the simulation algorithm by a variant that takes this information into account to more often encounter the rare event. modes implements three importance splitting techniques: Restart, fixed effort and fixed success.

Restart [46] is illustrated in Fig. 1: As soon as a Restart run crosses the threshold into a higher level, $n_\ell - 1$ new child runs are started from the first state in the new level, where n_ℓ is the splitting factor of level ℓ. When a run moves below its creation level, it ends. It also ends on reaching an *avoid* or *goal* state. The result of a Restart run—consisting of a main and several child runs—is the number of runs that reach *goal* times $1/\prod_\ell n_\ell$, i.e. a rational number ≥ 0.

Runs of the *fixed effort* method [17,19], illustrated in Fig. 2, are rather different. They consist of a fixed number of partial runs on each level, each of which ends when it crosses into the next higher level or encounters a *goal* or *avoid* state. When all partial runs for a level have ended, the next round starts from the previously encountered initial states of the next higher level. When a fixed effort run ends, the fraction of partial runs started in a level that moved up approximates the conditional probability of reaching the next level given that the current level was reached. If *goal* states exist only on the highest level, the overall result is the product of all of these fractions, i.e. a rational number in $[0, 1]$.

Fixed success [1] is a variant of fixed effort that generates partial runs until a fixed number of them have reached the next higher level. For all three methods, the average of the result of many runs is again an unbiased estimator for the probability of the transient property [19]. However, each run is no longer a Bernoulli trial. Of the statistical evaluation methods offered by modes, only CI with normal confidence intervals is thus applicable. For a deeper discussion of the challenges in the statistical evaluation of rare event simulation results, we refer the interested reader to [43]. To the best of our knowledge, modes is today the most automated rare event simulator for general stochastic models. In particular, it defaults to the combination of Restart with the expected success method for level calculation, which has shown consistently good performance [4].

4 Scheduler Sampling for Nondeterminism

Resolving nondeterminism in a randomised way leads to estimates that only lie *somewhere* between the desired extremal values. In addition to computing probabilities or expected rewards, we also need to find a (near-)optimal scheduler.

Lightweight Scheduler Sampling. modes implements the lightweight scheduler sampling (LSS) approach for MDP of [39] that identifies a scheduler by a single integer (typically of 32 bits). This allows to randomly select a large number m of schedulers (i.e. integers), perform standard or rare event simulation for each, and report the maximum and minimum estimates over all sampled schedulers as approximations of the actual extremal values. We show the core of the lightweight approach—performing a simulation run for a given scheduler identifier σ—for MDP and transient properties as Algorithm 1. An MDP consists of a countable set of states S, a transition function T that maps each state to a finite *set* of probability distributions over successor states, and an initial state

```
Input: MDP $\langle S, T, s_0 \rangle$, transient property $\phi$, scheduler id $\sigma \in \mathbb{Z}$
1 $s := s_0$, $\pi := s_0$
2 while $\phi(\pi) =$ undecided do
3     $\mathcal{U}_{\mathrm{nd}}$.initialise($\mathcal{H}(\sigma.s)$)            // use hash of $\sigma$ and $s$ as seed for $\mathcal{U}_{\mathrm{nd}}$
4     if $T(s) = \varnothing$ then return false                    // end of run due to deadlock
5     $\mu := \lceil \mathcal{U}_{\mathrm{nd}} \cdot |T(s)| \rceil$-th element of $T(s)$         // use $\mathcal{U}_{\mathrm{nd}}$ to select transition
6     $s' := \mu \circ \mathcal{U}_{\mathrm{pr}}$.next()       // use $\mathcal{U}_{\mathrm{pr}}$ to select successor state according to $\mu$
7     $\pi := \pi.s'$, $s := s'$                       // append $s'$ to $\pi$ and continue from $s'$
8 return $\phi(\pi)$
```

Algorithm 1. Simulation for an MDP and a fixed scheduler id [10]

s_0. The algorithm uses two PRNG: $\mathcal{U}_{\mathrm{pr}}$ to simulate the probabilistic choices (line 6), and $\mathcal{U}_{\mathrm{nd}}$ to resolve the nondeterministic ones (line 5). We want σ to represent a deterministic memoryless scheduler: within one simulation run as well as in different runs for the same value of σ, $\mathcal{U}_{\mathrm{nd}}$ must always make the same choice for the same state s. To achieve this, $\mathcal{U}_{\mathrm{nd}}$ is re-initialised with a seed based on σ and s in every step (line 3). The overall effectiveness of the lightweight approach only depends on the likelihood of selecting a σ that represents a (near-)optimal scheduler. We want to sample "uniformly" from the space of all schedulers to avoid actively biasing against "good" schedulers. Algorithm 1 achieves this naturally for MDP.

Beyond MDP. LSS can be adapted to any model and type of property where the class of optimal schedulers only uses *discrete* input to make its decision for every state [26]. This is obviously the case for discrete-space discrete-time models like MDP. It means that LSS can directly be applied to MA and *time-unbounded* properties, too. In addition to MDP and MA, modes also supports two LSS methods for PTA, based on a variant of forwards reachability with zones [10] and the region graph abstraction [26], respectively. While the former includes zone operations with worst-case runtime exponential in the number of clocks, the latter implements all operations in linear time. It exploits a novel data structure for regions based on representative valuations that performs very well in practice [26]. Extending LSS to models with general continuous probability distributions such as stochastic automata [11] is hindered by optimal schedulers requiring non-discrete information (the values and expiration times of all clocks [9]). modes currently provides prototypical LSS support for SA encoded in a particular form and various restricted classes of schedulers as described in [9].

Bounds and Error Accumulation. The results of an SMC analysis with LSS are lower bounds for maximum and upper bounds for minimum values up to the specified statistical error and confidence. They can thus be used to e.g. *disprove* safety (the maximum probability to reach an unsafe state is above a threshold) or *prove* schedulability (there is a scheduler that makes it likely to complete the workload in time), but not the opposite. The accumulation of statistical error introduced by the repeated simulation experiments over multiple schedulers must also be accounted for. [12] shows how to modify the APMC method accordingly and turn the SPRT into a correct sequential test *over schedulers*. In addition to these, modes allows the CI method to be used with LSS by applying the standard Šidák correction for multiple comparisons. This enables LSS for expected rewards and RES. All the adjustments essentially increase the required confidence depending on the (maximum) number of schedulers to be sampled.

Two-Phase and Smart Sampling. If an SMC analysis for fixed statistical parameters would need n runs on a deterministic model, it will need significantly more than $m \cdot n$ runs for a nondeterministic model when m schedulers are sampled due to the increase in the required confidence. modes implements a two-phase approach and smart sampling [12] to reduce this overhead. The former's first phase consists of performing n simulation runs for each of the m schedulers. The scheduler that resulted in the maximum (or minimum) value is selected, and independently evaluated once more with n runs to produce the final estimate. The first phase is a heuristic to find a near-optimal scheduler before the second phase estimates the value under this scheduler according to the required statistical parameters. Smart sampling generalises this principle to multiple phases, dropping only the "worst" *half* of the evaluated schedulers between phases. It starts with an informed guess of good initial values for n and m. For details, see [12]. Smart sampling tends to find more extremal schedulers faster while the two-phase approach has predictable performance as it always needs $(m + 1) \cdot n$ runs. We thus use the two-phase approach for our experiments in Sect. 6.

5 Architecture and Implementation

modes is implemented in C# and works on Linux, Mac OS X and Windows systems. It builds on a solid foundation of shared infrastructure with other tools of the Modest Toolset. This includes input language parsers that map Modest, xSADF and Jani input into a common internal metamodel for networks of stochastic hybrid automata with rewards and discrete variables. Before simulation, every model is compiled to bytecode, making the metamodel executable. The same compilation engine is also used by the mcsta and motest tools.

The architecture of the SMC-specific part of modes is shown as a class diagram in Fig. 3. Boxes represent classes, with rounded rectangles for abstract classes and invisible boxes for interfaces. Solid lines are inheritance relations. Dotted lines are associations, with double arrows for collection associations. The architecture mirrors the three distinct tasks of a statistical model checker: the generation of individual simulation runs and per-run evaluation of properties, implemented in modes by *RunGenerator* and *RunEvaluator*, respectively; the coordination of simulation over multiple threads across CPU cores and networked machines, implemented by classes derived from *Worker* and *IWorkerHost*; and the statistical evaluation of simulation runs, implemented by *PropertyEvaluator*.

The central component of modes' architecture is the *Master*. It compiles the model, derives the importance function, sends both to the workers (on the same

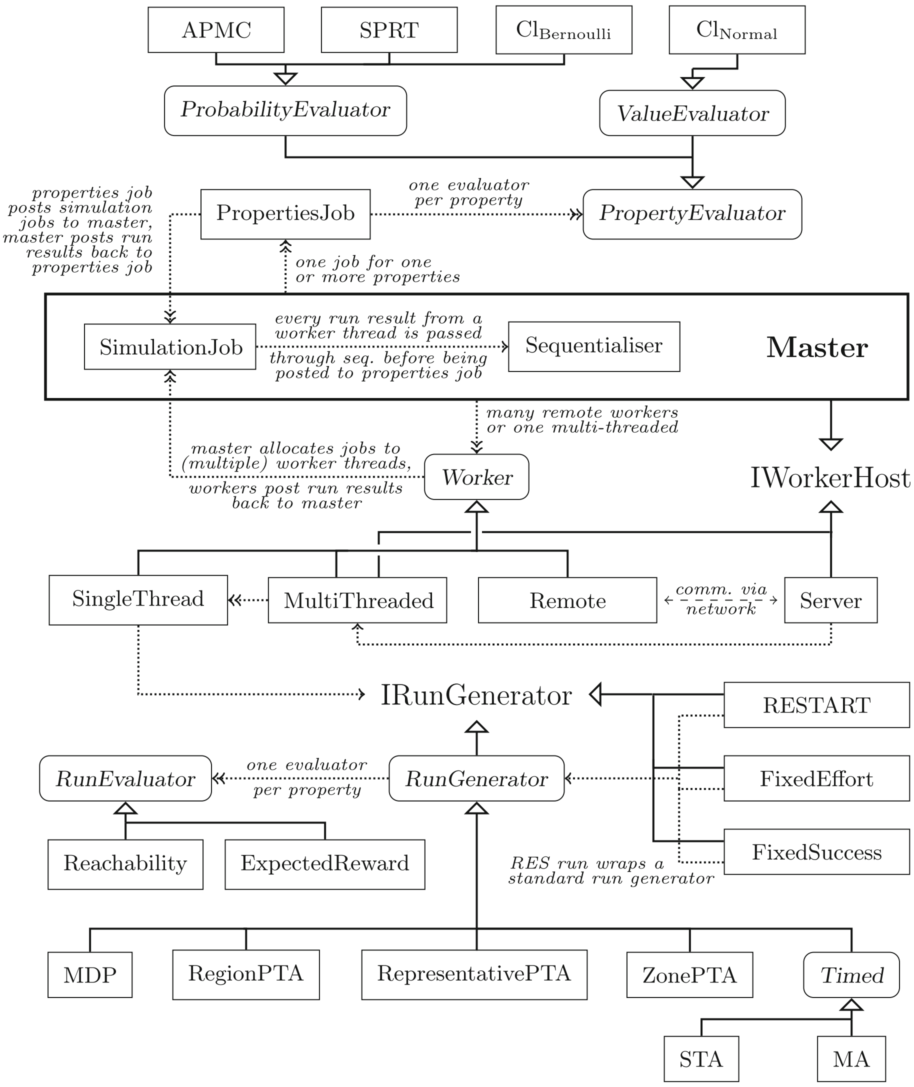
APMC
SPRT
Cl_Bernoulli
Cl_Normal
ProbabilityEvaluator
ValueEvaluator
properties job posts simulation jobs to master, master posts run results back to properties job
PropertiesJob
one evaluator per property
PropertyEvaluator
one job for one or more properties
SimulationJob
every run result from a worker thread is passed through seq. before being posted to properties job
Sequentialiser
Master
many remote workers or one multi-threaded
master allocates jobs to (multiple) worker threads, workers post run results back to master
Worker
IWorkerHost
SingleThread
MultiThreaded
Remote
comm. via network
Server
IRunGenerator
RESTART
FixedEffort
FixedSuccess
RunEvaluator
one evaluator per property
RunGenerator
Reachability
ExpectedReward
RES run wraps a standard run generator
MDP
RegionPTA
RepresentativePTA
ZonePTA
Timed
STA
MA

Fig. 3. The software architecture of the modes statistical model checker

or different machines), and instantiates a *PropertiesJob* for every partition of the properties to be analysed that can share simulation runs.[1] Each *PropertiesJob* then posts simulation jobs back to the master in parallel or in sequence. A simulation job is a description of how to generate and evaluate runs: which run type (i.e. *RunGenerator* derived class) to use, whether to wrap it in an importance

[1] Using the same set of runs for multiple properties is an optimisation at the cost of statistical independence. modes can also generate independent runs for each property.

splitting method, whether to simulate for a specific scheduler id, which compiled expressions to evaluate to determine termination and the values of the runs, etc. The master allocates posted jobs to available simulation threads offered by the workers, and notifies workers when a job is scheduled for one of their threads. As the result for an individual run is handed from the *RunEvaluator* by the *RunGenerator* via the workers to the master, it is fed into a *Sequentialiser* that implements the adaptive schedule for bias avoidance. Only after that, possibly at a later point, is it handed on to the *PropertiesJob* for statistical evaluation.

For illustration, consider a *PropertiesJob* for LSS with 10 schedulers, RES with Restart, and the expected success method for level calculation. It is given the importance function by the master, and its first task is to compute the levels. It posts a simulation job for fixed effort runs with level information collection to the master. Depending on the current workload from other *PropertiesJob*s, the master will allocate many threads to this job. Once enough results have come in, the *PropertiesJob* terminates the simulation job, computes the levels and splitting factors, and starts with the actual simulations: It selects 10 random scheduler identifiers and concurrently posts for each of them a simulation job for Restart runs. The master will try to allocate available threads evenly over these jobs. As results come in, the evaluation may finish early for some schedulers, at which point the master will be instructed to stop the corresponding simulation job. It can then allocate the newly free threads to other jobs. This scheme results in a maximal exploitation of the available parallelism across workers and threads.

Due to the modularity of this architecture, it is easy to extend modes in different ways. For example, to support a new type of model (say, non-linear hybrid automata) or a new RES method, only a new *(I)RunGenerator* needs to be implemented. Adding another statistical evaluation method from [44] means adding a new *PropertyEvaluator*, and so on.

In distributed simulation, an instance of modes is started on each node with the `--server` parameter. This results in the creation of an instance of the *Server* class instead of a *Master*, which listens for incoming connections. Once all servers are running, a master can be started with a list of hosts to connect to. modes comes with a template script to automate this task on slurm-based clusters.

6 Experiments

We present three case studies in this section. They have been chosen to highlight modes' capabilities in terms of the diverse types of models it supports, its ability to distribute work across compute clusters, and the new analyses possible with RES and LSS. None of them has been studied before with modes or the combinations of methods that we apply here. Our experiments ran on an Intel Core i7-4790 workstation (3.6–4.0 GHz, 4 cores), a homogeneous cluster of 40 AMD Opteron 4386 nodes (3.1–3.8 GHz, 8 cores), and an inhomogeneous cluster of 15 nodes with different Intel Xeon processors. All systems run 64-bit Linux. We use 1, 2 or 4 simulation threads on the workstation (denoted "1", "2" and "4" in our tables), and n nodes with t simulation threads each on the clusters (denoted

Table 1. Performance and scalability on the electric vehicle charging case study

	$n_{fail} = 2$		$n_{fail} = 3$		$n_{fail} = 4$		$n_{fail} = 5$	
	MC	RES	MC	RES	MC	RES	MC	RES
conf. interval	[6.4E–2, 7.8E–2]		[5.2E–3, 6.4E–3]		[2.7E–4, 3.2E–4]		[8.3E–6, 1.0E–5]	
1	2 s	4 s	30 s	19 s	585 s	206 s	—	
2	1 s	2 s	15 s	11 s	315 s	101 s		
4	1 s	1 s	8 s	5 s	163 s	69 s		
5 × 4	1 s	1 s	4 s	4 s	69 s	23 s	2241 s	496 s
5 × 8	1 s	2 s	2 s	3 s	40 s	16 s	1238 s	328 s
40 × 2	0 s	1 s	1 s	2 s	16 s	8 s	483 s	135 s
20 × 8	0 s	2 s	1 s	2 s	10 s	6 s	314 s	105 s
40 × 8	0 s	2 s	1 s	3 s	5 s	6 s	159 s	64 s

"$n \times t$"). We used a one-hour timeout, marked "—" in the tables. Note that runtimes cannot directly be compared between the workstation and the clusters.

Electric Vehicle Charging. We first consider a model of an electric vehicle charging station. It is a Modest model adapted from the "extended" case study of [42]: a stochastic hybrid Petri net with general transitions, which in turn is based on the work in [31]. The scenario we model is of an electric vehicle being connected to the charger every evening in order to be recharged the next morning. The charging process may be delayed due to high load on the power grid, and the exact time at which the vehicle is needed in the morning follows a normal distribution. We consider one week of operation and compute the probability that the desired level of charge is not reached on any $n_{fail} \in \{2, \ldots, 5\}$ of the seven mornings.

This model is not amenable to exhaustive model checking due to the non-Markovian continuous probability distributions and the hybrid dynamics modelling the charging process. However, it is deterministic. We thus applied modes with standard Monte Carlo simulation (MC) as well as with RES using Restart. We performed the same analysis on different configurations of the workstation and the homogeneous cluster. To compare MC and RES, we use CI with $\delta = 0.95$ and a relative half-with of 10 % for both. All other parameters of modes are set to default values, which implies an automatic compositional importance function and the expected success method to determine levels and splitting factors. The results are shown in Table 1. Row "conf. interval" gives the average confidence intervals that we obtained over all experiments.

RES starts to noticeably pay off as soon as probabilities are on the order of 10^{-4}. The runtime of Restart is known to heavily depend on the levels and splitting factors, and we indeed noticed large variations in runtime for RES over several repetitions of the experiments. The runtimes for RES should thus not be used to judge the speedup w.r.t. parallelisation. However, when looking at the MC runtimes, we see good speedups as we increase the number of threads per node, and near-ideal speedups as we increase the total number of nodes, as long as there is a sufficient amount of work.

Table 2. Performance and results for the low-latency wireless network case study

		time	$\mathbb{P}(i < 4\,\mathtt{U}\,\mathit{failed})$	$\mathbb{P}(i < 4\,\mathtt{U}\,\mathit{offline}_{\{1\}})$	$\mathbb{P}(i < 4\,\mathtt{U}\,\mathit{offline}_{\{2\}})$
optimal			$[0.028, 0.472]$	$[0.026, 0.269]$	$[0 \quad , 0.424]$
1	100	3523 s	$[0.041, 0.363]$	$[0.030, 0.189]$	$[0.000, 0.309]$
2	100	2045 s			
4	100	1205 s			
20×8	1000	607 s	$[0.033, 0.383]$	$[0.028, 0.242]$	$[0.000, 0.327]$
40×8	1000	308 s			

Although this model was not designed with RES in mind and has only moderately rare events, the fully automated methods of modes could be applied directly, and they significantly improved performance. For a detailed experimental comparison of the RES methods implemented in modes on a larger set of examples, including events with probabilities as low as $4.8 \cdot 10^{-23}$, we refer the reader to [4].

Low-latency Wireless Networks. We now turn to the PTA model of a low-latency wireless networking protocol being used among three stations, originally presented in [15]. We take the original model, increase the probability of message loss, and make one of the communication links nondeterministically drop messages. This allows us to study the influence of the message loss probabilities and the protocol's robustness to adversarial interference. The model *is* amenable to model checking, as demonstrated in [15]. It allows us to show that modes can be applied to such models originally built for traditional verification, and since we can calculate the precise maximum and minimum values of all properties via model checking, we have a reference to evaluate the results of LSS.

We show the results of using modes with LSS on this model in Table 2. Row "optimal" lists the maximum and minimum probabilities computed via model checking for three properties: the probability that the protocol fails within four iterations, and that either the first or the second station goes offline. We used the two-phase LSS method with $m = 100$ schedulers on the workstation, and with $m = 1000$ schedulers on the homogeneous cluster. The intervals are the averages of the min. and max. values returned by all analyses. The statistical evaluation is APMC with $\delta = 0.95$ and $\epsilon = 0.0025$, which means that 59556 simulation runs are needed per scheduler.

Near-optimal schedulers for the minimum probabilities do not appear to be rare: we find good bounds for the minima even with 100 schedulers. However, for maximum probabilities, sampling more schedulers pays off in terms of better approximations. In all cases, the results are conservative approximations of the actual optima (as expected), and they are clearly more useful than the single value that would be obtained by other tools via a (hidden) randomised scheduler. Performance scales ideally with parallelism on the cluster, and still linearly on the workstation. For a deeper evaluation of the characteristics of LSS, including experiments on models too large for model checking, we refer the reader to the description of the original approach [12,39] and its extensions to PTA [10,26].

Table 3. Performance and results for the reliable database system case study

	uniform scheduler			lightweight scheduler sampling (20)			
R	MC	RES	conf. interval	MC	RES	min. conf. int.	max. conf. int.
2	1 s	4 s	[1.5E–2, 1.8E–2]	4 s	31 s	[1.4E–2, 1.7E–2]	[1.5E–2, 1.9E–2]
3	8 s	3 s	[1.0E–4, 1.3E–4]	181 s	26 s	[7.9E–5, 9.6E–5]	[1.3E–4, 1.6E–4]
4	816 s	13 s	[9.3E–7, 1.1E–6]	—	221 s	[6.3E–7, 7.6E–7]	[1.3E–6, 1.6E–6]
5	—	229 s	[1.1E–8, 1.3E–8]		3072 s	[6.2E–9, 7.6E–9]	[1.6E–8, 2.0E–8]

Redundant Database System. The redundant database system [20] is a classic RES case study. It models a system consisting of six disk clusters of $R + 2$ disks each plus two types of processors and disk controllers with R copies of each type. Component lifetimes are exponentially distributed. Components fail in one of two modes with equal probability, each mode having a different repair rate. The system is operational as long as fewer than R processors of each type, R controllers of each type, and R disks in each cluster are currently failed. The model is a CTMC with a state space too large and a transition matrix too dense for it to be amenable to model checking with symbolic tools like Prism [37].

In the original model, any number of failed components can be repaired in parallel. We consider this unrealistic, and extend the model by a *repairman* that can repair a single component at a time. If more than one component fails during a repair, then as soon as the current repair is finished, the repairman has to decide which to repair next. Instead of enforcing a particular repair policy, we leave this decision as nondeterministic. The model thus becomes an MA. We use LSS in combination with RES to investigate the impact of the repair policy. We study the scenario where one *component* of each kind (one disk, one processor, one controller) is in failed state, and estimate the probability for *system* failure before these components are repaired. The minimum probability is achieved by a perfect repair strategy, while the maximum results from the worst possible one.

Table 3 shows the results of our LSS-plus-RES analysis with modes using default RES parameters and sampling $m = 20$ schedulers. Due to the complexity of the model, we ran this experiment on the inhomogeneous cluster only, using 16 cores on each node for 240 concurrent simulation threads in total. We see that RES needs a somewhat rare event to improve performance. We also compare LSS to the uniform randomised scheduler (as implemented in many other SMC tools). It results in a single confidence interval for the probability of failure. With LSS, we get two intervals. They do not overlap when $R \geq 3$, i.e. the repair strategy matters: a bad strategy makes failure approximately twice as likely as a good strategy! Since the results of LSS are conservative, the difference between the worst and the best strategy may be even larger.

Experiment Replication. To enable independent replication of our experimental results, we have created a publicly available evaluation artifact [23]. It contains the version of modes and the model files used for our experiments, the raw experimental results, summarising tabular views of those results (from which we derived Tables 1, 2 and 3), and a Linux shell script to automatically replicate a subset of the experiments. Since the complete experiments take several hours to complete and require powerful hardware and computer clusters, we have selected a subset for the replication script. Using the virtual machine of the TACAS 2018 Artifact Evaluation [28] on typical workstation hardware of 2017, it runs to completion in less than one hour while still substantiating our main results.

7 Conclusion

We presented modes, the Modest Toolset's distributed statistical model checker. It provides methods to tackle both of the prominent challenges in simulation: nondeterminism and rare events. Its modular software architecture allows its various features to be easily combined and extended. For the first time, we used lightweight scheduler sampling with Markov automata, and combined it with rare event simulation to gain insights into a challenging case study that, currently, cannot be analysed for the same aspects with any other tool that we are aware of. modes is available for download at www.modestchecker.net.

Acknowledgments. The authors thank Carina Pilch and Sebastian Junges for their support with the *vehicle charging* and *wireless networks* case studies.

Data Availability. The data generated in our experimental evaluation is archived and available at DOI 10.4121/uuid:64cd25f4-4192-46d1-a951-9f99b452b48f [23].

References

1. Amrein, M., Künsch, H.R.: A variant of importance splitting for rare event estimation: fixed number of successes. ACM Trans. Model. Comput. Simul. **21**(2), 13:1–13:20 (2011)
2. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for Modestly nondeterministic models. In: Schmitt, J.B. (ed.) MMB&DFT 2012. LNCS, vol. 7201, pp. 249–252. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28540-0_20
3. Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_8
4. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Better automated importance splitting for transient rare events. In: Larsen, K.G., Sokolsky, O., Wang, J. (eds.) SETTA 2017. LNCS, vol. 10606, pp. 42–58. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69483-2_3
5. Budde, C.E., D'Argenio, P.R., Monti, R.E.: Compositional construction of importance functions in fully automated importance splitting. In: VALUETOOLS. ICST (2016)
6. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 151–168. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_9

7. Cérou, F., Guyader, A.: Adaptive multilevel splitting for rare event analysis. Stochast. Anal. Appl. **25**(2), 417–443 (2007)
8. Cérou, F., Moral, P.D., Furon, T., Guyader, A.: Sequential Monte Carlo for rare event estimation. Stat. Comput. **22**(3), 795–808 (2012)
9. D'Argenio, P.R., Gerhold, M., Hartmanns, A., Sedwards, S.: A hierarchy of scheduler classes for stochastic automata. In: FoSSaCS. LNCS, vol. 10803. Springer (2018, to appear)
10. D'Argenio, P.R., Hartmanns, A., Legay, A., Sedwards, S.: Statistical approximation of optimal schedulers for probabilistic timed automata. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 99–114. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33693-0_7
11. D'Argenio, P.R., Katoen, J.P.: A theory of stochastic systems part I: stochastic automata. Inf. Comput. **203**(1), 1–38 (2005)
12. D'Argenio, P.R., Legay, A., Sedwards, S., Traonouez, L.M.: Smart sampling for lightweight verification of Markov decision processes. STTT **17**(4), 469–484 (2015)
13. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal Stratego. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 206–211. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_16
14. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_27
15. Dombrowski, C., Junges, S., Katoen, J.P., Gross, J.: Model-checking assisted protocol design for ultra-reliable low-latency wireless networks. In: SRDS, pp. 307–316. IEEE (2016)
16. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS, pp. 342–351. IEEE Computer Society (2010)
17. Garvels, M.J.J., Kroese, D.P.: A comparison of RESTART implementations. In: Winter Simulation Conference, pp. 601–608 (1998)
18. Garvels, M.J.J., van Ommeren, J.C.W., Kroese, D.P.: On the importance function in splitting simulation. Eur. Trans. Telecommun. **13**(4), 363–371 (2002)
19. Garvels, M.J.J.: The splitting method in rare event simulation. Ph.D. thesis, University of Twente, Enschede, The Netherlands (2000)
20. 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)
21. Graf-Brill, A., Hartmanns, A., Hermanns, H., Rose, S.: Modelling and certification for electric mobility. In: Industrial Informatics (INDIN). IEEE (2017)
22. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst. Des. **43**(2), 191–232 (2013)
23. Hartmanns, A.: A Statistical Model Checker for Nondeterminism and Rare Events (artifact). 4TU.Centre for Research Data (2018). http://doi.org/10.4121/uuid:64cd25f4-4192-46d1-a951-9f99b452b48f
24. Hartmanns, A., Hermanns, H.: The Modest Toolset: an integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 593–598. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_51
25. Hartmanns, A., Hermanns, H., Bungert, M.: Flexible support for time and costs in scenario-aware dataflow. In: EMSOFT. ACM (2016)

26. Hartmanns, A., Sedwards, S., D'Argenio, P.R.: Efficient simulation-based verification of probabilistic timed automata. In: Winter Simulation Conference (2017)
27. Hartmanns, A., Timmer, M.: Sound statistical model checking for MDP using partial order and confluence reduction. STTT **17**(4), 429–456 (2015)
28. Hartmanns, A., Wendler, P.: TACAS 2018 Artifact Evaluation VM. Figshare (2018). https://doi.org/10.6084/m9.figshare.5896615
29. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998)
30. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24622-0_8
31. Hüls, J., Remke, A.: Coordinated charging strategies for plug-in electric vehicles to ensure a robust charging process. In: VALUETOOLS. ICST (2016)
32. Jégourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 576–591. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_38
33. Jégourel, C., Legay, A., Sedwards, S.: Command-based importance sampling for statistical model checking. Theor. Comput. Sci. **649**, 1–24 (2016)
34. Jégourel, C., Legay, A., Sedwards, S., Traonouez, L.M.: Distributed verification of rare properties using importance splitting observers. In: ECEASST, vol. 72 (2015)
35. Kearns, M.J., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large Markov decision processes. Machine Learn. **49**(2–3), 193–208 (2002)
36. Kurkowski, S., Camp, T., Colagrosso, M.: MANET simulation studies: the incredibles. Mob. Comput. Commun. Rev. **9**(4), 50–61 (2005)
37. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
38. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. **282**(1), 101–150 (2002)
39. Legay, A., Sedwards, S., Traonouez, L.-M.: Scalable verification of Markov decision processes. In: Canal, C., Idani, A. (eds.) SEFM 2014. LNCS, vol. 8938, pp. 350–362. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15201-1_23
40. Legay, A., Sedwards, S., Traonouez, L.-M.: Plasma Lab: a modular statistical model checking platform. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 77–93. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_6
41. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. Ann. Inst. Stat. Math. **10**(1), 29–35 (1959)
42. Pilch, C., Remke, A.: Statistical model checking for hybrid Petri nets with multiple general transitions. In: DSN, pp. 475–486. IEEE Computer Society (2017)
43. Reijsbergen, D., de Boer, P.-T., Scheinhardt, W.: Hypothesis testing for rare-event simulation: limitations and possibilities. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 16–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_2
44. Reijsbergen, D., de Boer, P., Scheinhardt, W.R.W., Haverkort, B.R.: On hypothesis testing for statistical model checking. STTT **17**(4), 377–395 (2015)
45. Rubino, G., Tuffin, B. (eds.): Rare Event Simulation Using Monte Carlo Methods. Wiley, New York (2009)

46. 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)
47. Wald, A.: Sequential tests of statistical hypotheses. Ann. Math. Stat. **16**(2), 117–186 (1945)
48. Younes, H.L.S.: Ymer: a statistical model checker. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 429–433. Springer, Heidelberg (2005). https://doi.org/10.1007/11513988_43
49. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_17