Towards Verifying Robotic Systems using Statistical Model Checking in STORM

Marco Lampacrescia[1], Michaela Klauck[1], and Matteo Palmas[1,2]
firstname.lastname@de.bosch.com

[1] Robert Bosch GmbH, Bosch Research
[2] University of Genova

Abstract. Robust autonomy and interaction of robots with their environment, even in rare or new situations, is an ultimate goal of robotics research. We settle on Statistical Model Checking (SMC) for the benefit of robustness of robot deliberation and base our implementation on STORM, one of the most performant and comprehensive open-source model checkers, so far lacking an SMC extension. The SMC extension introduced in this paper offers various statistical methods, from which the user can choose to find the best trade-off between accuracy of the result and runtime. We demonstrate the efficiency of our SMC implementation by comparing it to other state-of-the-art SMC tools on well-established benchmarks and on a robotics-related example. The results indicate that our implementation, which will be continuously extended in the future to improve support for robotics use cases, is performant enough to bridge the gap between robotic systems and model checking in industry.

Keywords: Statistical Model Checking · STORM · Robotics

1 Introduction

Despite the improvements of both hardware and software capabilities, robots are still struggling to cope robustly with unknown situations in unstructured environments, e.g., when cleaning in households or public spaces. This is due to a lack of methods and tools to validate complex robotic systems efficiently *as a whole*, despite the longstanding problem of verification and validation (V&V) of autonomous systems. With the growing need for reliable robots, model checking (MC) is becoming an increasingly relevant tool for the robotics community. So far, it has only been taken into consideration on small parts of a full robotic system, because it was unclear how to get consistent full models that are verifiable in a reasonable time [24,50,44,6,22,21], though recent works are starting to look into this modeling problem [33]. In the present paper, our aim is to pave the way for efficient verification of complex models with different interacting parts like the ones representing full robotic systems.

MC is a fully automated formal method for finding flaws in system functionalities [3], enabling developers to produce ever more robust systems [28]. Based on a formal model of a system it is checked whether the system satisfies a property expressed in a temporal logic formula. For probabilistic models, the probability that the formula is satisfied is determined [17]. Model checking comes in two flavors. The classical approach [18] builds the complete state space of the model to validate the property under investigation. The need to construct the model and explore it entirely makes explicit-state model checking extremely sensitive to the system's complexity and size [16]. Its usefulness for industrial developers is often reduced because of an exponential explosion in the number of elements to explore, known as the state space explosion problem.

To solve this problem, statistical model checking (SMC) [32,41,37,42,51] reduces the model checking procedure to a matter of statistical inference, where the model is evaluated by simulating sample executions (or traces) to get statistical evidence whether a property holds. This is also called Monte Carlo Simulation [46]. The model of the system is not completely explored, but iteratively sampled, and the results are statistically evaluated. The resulting information is used to determine the model checking result within a certain confidence interval defined by the user. This makes it possible to perform MC on systems that are too large to be evaluated using numerical model checking, providing developers with an alternative to unit and real-life system tests that is more reliable and can provide statistical guarantees on the result. In general, this approach is only applicable to deterministic models because during simulation nondeterminism would have to be resolved to continue the traces. Therefore, we concentrate on discrete-time Markov chains (DTMCs) in this first work. There are approaches to find the best scheduler wrt. the property under investigation to resolve nondeterminism in Markov decision processes (MDPs) when performing SMC [7,9,30,23,2], but the consideration of those techniques is left for future extensions.

In the state of the art there are many tools [4,11] implementing efficient flavors of statistical model checking algorithms for different types of quantitative models, like PRISM [34], STORM [31], The MODEST Toolset [27], and UPPAAL [40].

The PRISM model checker [34] is a tool that has been applied in many application domains over years, including communication and multimedia protocols, for formal modelling and analysis. It is implemented in Java and its code is open source. It is accompanied by many benchmarks in the PRISM language [36] and contains an SMC engine which is taken into consideration for the performance comparison in Sec. 3.

The STORM model checker [31] is a tool for the analysis of systems involving random or probabilistic phenomena supporting several automata-based formalisms. It is implemented in C++, and is also available open source. STORM provides multiple engines implementing both classical and semi-statistical methods to evaluate models that, depending on the type, can be provided in several formats. This includes the JANI [10] and the PRISM [36] formats, that are the ones supporting DTMCs and MDPs. As demonstrated in competitions [11], it is highly performant. On the other hand, it does not natively provide a SMC implementation. StormDftRes [1], the only extension of STORM using SMC techniques, targets a very specific set of models (Dynamic Fault Trees) and no DTMCs. This motivates the contributions of this work, that aims at providing an open source tool performant enough to be used on running systems, for which the use of a performant model checker such as STORM seems to be beneficial.

The MODEST Toolset [27,7,9] also supports a wide range of automata types centered around stochastic hybrid automata in both classical and statistical methods for model checking. It is implemented in C# and the executable is publicly available for research use but it is closed source. The toolset accepts inputs in its own MODEST language [26] as well as JANI and offers various analysis backends. The various statistical methods implemented in modes [9], the SMC tool of The MODEST Toolset, were the main source of inspiration for the algorithms implemented for smc_storm in this work.

UPPAAL [5,38,39,40] is a comprehensive tool consisting of several engines dealing with networks of timed automata modeling real-time systems. This makes it especially interesting for industrial use cases as shown in many case studies. It is implemented in Java and the executable is publicly available for academic use but it is closed source. The tool comes with its own non-deterministic guarded command input language, a simulator and a symbolic as well as a statistical model checking engine [20]. It is the only tool listed here which provides graphical user interaction via a system editor, a graphical simulator, and a requirement specification editor.

Our goal in this work is to contribute a performant, open source SMC tool accessible for many users by a commonly used input format. Such a well-established input format is JANI [10], whose aim is to foster tool interaction and interoperability in the quantitative verification community. Converters to and from the PRISM language exist. The well-established Quantitative Verification Benchmark Set (QVBS) [29] contains plenty of JANI models for tool comparisons. They are used for example in the QComp competition [11,25] comparing quantitative verification tools. Based on that, we also contribute a performance comparison to other state-of-the-art SMC tools in the evaluation in Sec. 3, inspired by QComp. To allow for comparison with other established model checkers, we decided to start with the evaluation of discrete-time Markov chain (DTMC) models and later extend the functionalities of the implementation.

In the following, we present smc_storm, our SMC tool based on the open source STORM model checker, including its technical details and the available configuration parameters (Sec. 2), as well as a benchmarking comparison to other state-of-the-art SMC engines on the QVBS [29], but also on a model of a robotic system with realistic handling of geometric behavior and constraints in an environment (Sec. 3). The smc_storm code is publicly available online [3].

To demonstrate the potential of smc_storm, we compared it to the SMC tools provided by PRISM and The MODEST Toolset. The supported input format (no PRISM or JANI), the modeling formalism (timed automata), and the comparability to other tools (QComp/QVBS) lead to the decision to not further consider UPPAAL in the experiments of this paper. Nevertheless, it is

[3] https://github.com/convince-project/smc_storm

definitely of interest for future investigations. The same holds for additional tools that extend existing third-party simulators with SMC capabilities, like Plasma-Lab [8], that is implemented in Java and supports, among other formats, PRISM. There is also MultiVeStA [48], implemented in Java, that is designed to verify agent-based models. Though it would have been interesting to consider those implementations, we opted for a comparison of tools running the QVBS models off-the-shelf in this paper.

The long term goal of this work is to bridge the gap between model checking and complex robotic applications. This will be achieved by providing a performant verification toolbox useable for complex industrial settings based on STORM and JANI [33].

2 Functionalities of smc_storm

In this section we introduce the technical details of smc_storm, the SMC tool we developed on top of the STORM framework. The smc_storm implementation can be split into two parts: (1) the trace generation engine, starting from the model exploration engine already present in STORM and (2) the statistical analysis methods to determine the amount of traces required to compute the desired results (both with a fixed number of iterations [43] and with sequential methods [47,15,13], evaluated in each iteration). Additionally, performance improvements using parallelization and optimizations are done to make the SMC engine applicable and useful for developers of complex (autonomous) robotic systems.

The following sections introduce the structure of the smc_storm implementation and the input parameters the user can provide.

2.1 Algorithm Structure

The structure of smc_storm has been inherited from the existing *exploration* engine in STORM, that already provided tools to evaluate probabilities (P) on simple path properties. The existing code was extended to support SMC, and to be able to evaluate reward (E) properties and more complex path properties. The main steps of the implementation are summarized in Algorithm 1.

The algorithm starts by loading the model and the property, making sure all provided constants are assigned, the initial state is provided, and traces can be generated starting from it. Once the model and the task are loaded, an object to collect the results of the generated samples is prepared (see line 2), and the main loop of the algorithm is started (see line 5). This loop can be parallelized, with each iteration being responsible to generate a batch of a fixed number of samples (see line 7) that are added to the collector at the end of the loop (see line 10). As soon as the collector contains enough samples to provide the desired result with enough confidence, the algorithm stops to sample new traces and computes the result. To prevent possible biases in the result, the collector object ensures

Algorithm 1 Simplified Flow of smc_storm

```
 1: procedure SMC_STORM(model, property)
 2:     Initiate sampling_results
 3:     for all threads do
 4:         Initiate expl_model                        ▷ Exploration Information
 5:         while sampling_results.converged() == False do
 6:             Initiate results_batch
 7:             while results_batch.not_full() do
 8:                 path_result = generateNewPath(model, property, expl_model)
 9:                 results_batch.add(path_result)
10:             sampling_results.add(results_batch)
11:     result = sampling_results.calculateResult()
```

that each thread contributes the same amount of batches by using a buffer to synchronize the different threads, as suggested in previous SMC works [12,52].

In the following, we introduce the contributed components that we consider to be the most relevant for the implementation of smc_storm.

The *PropertyDescription* class introduced in this work converts an input path property to a bounded until (condition $U_{[a,b]}$ target) property, generating a *condition* expression, a *target* expression, the *lower bound a* and the *upper bound b*. In addition, a *terminal_verified* flag indicates whether a path reaching a terminal state without verifying the target expression should be considered as verifying the path property or not. This approach allows to unify the path evaluation part by focusing on a single, consistent interface that remains valid across all supported path properties.

The existing *ExplorationInformation* class is meant to store the explored model in an incremental fashion, in order to reuse previously computed information when generating new samples. We extended it to store explored rewards and explicitly assign the result of the path property evaluation to each expanded state. The evaluation result consists of any combination of the following flags: *no_info*, *is_terminal*, *break_condition*, and *satisfy_target*. With the additional information the algorithm is capable of evaluating bounded path properties and computing rewards on the generated samples.

Lastly, the *SamplingResults* class was introduced to collect the results from the generated samples, to define when the algorithm generated enough samples to compute a final result with the requested precision and confidence (detailed in Sec. 2.2), and to perform the final result calculation. To properly evaluate the existing samples, this module needs to distinguish between P and E properties. Additionally, to lower the need for thread synchronization and hence improve parallelization, each thread collects the computed results in an own *BatchResult* object of fixed *batch_size*, before adding it to the main *SamplingResults* collector, that will add that result in a *BatchBuffer* object to ensure each thread provides the same amount of samples. A higher *batch_size* reduces the need of synchronization between multiple threads, at the expense of potentially producing more samples than needed.

2.2 Provided Statistical Methods

The crucial point when performing formal verification using SMC, is to evaluate the statistical relevance of the computed result. In other terms, it needs to be determined if, given the existing samples, Eq. 1 is satisfied.

$$Prob(|X - Y| > \varepsilon) < (1 - C) \tag{1}$$

Eq. 1 is used to make sure that the probability that the computed result differs too much from the actual result is low. In the equation, X is the correct result, Y is the one estimated from the samples, ε is the maximum absolute error, and C is the minimum confidence level that the error $|X - Y|$ is lower than ε. The concept of this equation is closely related to the confidence interval (CI) that determines the interval containing the true result with a given confidence level on the basis of the amount and nature of the existing samples. ε represents the desired half width of the CI, and our goal is to converge to a CI whose width is smaller than 2ε.

In the literature multiple methods exist to determine whether, given a set of samples, Eq. 1 is satisfied. They can be divided into two main categories: 1) methods requiring a fixed number of iterations and 2) sequential methods, evaluated at runtime and generally requiring a lower amount of samples. Additionally, since those methods usually rely on the assumption that the samples stem from a specific statistical distribution, it is important to distinguish between samples generated to evaluate a P property and those to evaluate an E property. In the first case, the result is binary (whether the trace satisfies the property or not), so we can assume that the results are coming from a binomial distribution. In the second case, the result is a number (the reward/cost obtained on that trace) coming from a continuous interval with unknown bounds. Therefore we need to assume the results are coming from a different distribution, e.g. a normal distribution.

When using smc_storm, the user can define the desired confidence level C and maximum error ε of the computed result, together with the statistical method to use for determining when to stop generating new samples. The complete list of implemented statistical methods can be found in Table 1 together with their compatibility with P and E properties.

In the following list, we briefly describe the most common statistical methods:

– **Chernoff Bounds** [14,43]: This method defines the amount of required samples depending on the provided C and ε, together with the width of the interval containing the possible outcomes of each sample. For P properties, the interval width is 1 (i.e., [0, 1]), so the number of required samples can be determined directly from the start. For E properties the width of the interval is not known at start, and it needs to be estimated during runtime. This is

Name	ID	P Support	E Support
Chernoff Bounds	chernoff	✓	✓
Wald Interval	wald	✓	✗
Agresti-Coull Interval	agresti	✓	✗
Wilson Score Interval	wilson	✓	✗
Corrected Wilson Score Interval	wilson_corrected	✓	✗
Clopper-Pearson Interval	clopper_pearson	✓	✗
Arcsine Transformation	arcsine	✓	✗
New Adaptive Sampling Method	adaptive	✓	✗
Chow-Robbins Confidence Interval	chow_robbins	✓	✓

Table 1: Statistical methods available in STORM SMC with their ID and support for P and E properties.

done by keeping track of the minimum and maximum sampled reward and increasing the amount of required samples each time the tracked minimum and maximum values change, assuming the new values are the boundaries of the interval of all possible outcomes. This method usually results in a higher number of samples compared to the sequential methods described in the following.

- **Wald Interval** [49]: The simplest sequential method designed for P properties, approximating the binomial distribution to a normal distribution. Though it keeps the number of required samples low, the result's error is often higher than the provided ε, especially in case the expected result is close to 0 or 1.
- **Clopper-Pearson Interval** [19]: Another sequential method designed for P properties, that computes the exact interval from the binomial distribution and is more accurate than the Wald Interval, though it requires a higher number of samples.
- **New Adaptive Sampling Method** [13]: An alternative sequential method introduced in the context of machine learning, that requires more samples than the two sequential methods introduced before, but provides more accurate results. Since it assumes sample results coming from a binomial distribution, it can be used only for P properties. It is the default method used in modes and smc_storm.
- **Chow-Robbins Method** [15]: This is a sequential method for samples drawn from a continuous distribution. It assumes samples are drawn from a normal distribution and computes the width of the confidence interval based on the variance and the amount of sampled results. It is the default method used by PRISM SMC, modes, and smc_storm for evaluating E properties. In case of P properties, it approximates the binomial distribution to a normal distribution, resulting into the Wald interval method described above.

From our experience, the statistical methods selected by default provide a good trade-off between result accuracy and performance for all the considered use cases in our evaluation. However, sequential methods might converge to the wrong results, and cannot provide with any guarantee on the quality of the results based on asymptotic behavior. For this reason, if the quality of the result is more important than the computation time, we recommend to use the Chernoff Bounds both for P and E properties, that assures the number of generated samples is high enough to satisfy Eq. 1 for the given C and ε.

2.3 Running smc_storm

Currently, smc_storm is provided in a standalone repository[4] that uses STORM as an external library. It depends on the official STORM repository[5], that needs to be installed before compiling smc_storm. smc_storm supports only Linux.

Once installed, smc_storm can be executed from the command line as follows:

```
smc_storm
    --model <path>
    --properties-names <string>
    --constants <string>
    --epsilon <real>
    --confidence <real>
    --stat-method <string>
    --max-trace-length <int>
    --n-threads <int>
    --batch-size <int>
    --show-statistics
```

The following list provides the description of the available parameters:

* `--model`: Path to the file containing the model and properties description (jani or prism format).
* `--properties-names`: Comma-separated list of the properties to evaluate.
* `--properties-file`: File containing the available properties (for prism models only).
* `--constants`: Values for constant parameters of the loaded model.
* `--epsilon`: Maximum absolute error of the computed result, given a certain confidence level.
* `--confidence`: The confidence level that the error of the computed result is below the requested epsilon.
* `--stat-method`: The statistical method used to determine how many samples we need to generate. IDs given in Tab. 1.
* `--max-trace-length`: Maximum length of a single generated trace. Set this parameter to 0 to make traces unbounded.
* `--n-threads`: Number of threads to use for the generation of the samples.
* `--batch-size`: Number of samples a thread generates before adding them to the main collector.

[4] https://github.com/convince-project/smc_storm
[5] https://github.com/moves-rwth/storm

* `--show-statistics`: Flag to print a report on the statistics (i.e., number of generated traces, number of satisfying traces, min/max reward) of the generated samples at the end of the evaluation.

3 Experimental Evaluation

In this section we evaluate the performance of smc_storm against the SMC implementations in PRISM [35] and modes, the SMC tool from The MODEST Toolset [7]. In Sec. 3.1 we evaluate the tools using all applicable DTMC models with their related properties from the QVBS (i.e., we only considered models with a single initial state providing atomic probability and reward properties). In addition, we defined a first robotics-related example as explained in Sec. 3.2 to show how we can make use of smc_storm to formally verify industrially relevant properties on it. All tests have been performed on a standard laptop running on Ubuntu 22.04 with an Intel Core i7-11850H CPU and 32GB of RAM.

3.1 Evaluation on Standard DTMC Benchmarks of the QVBS

To demonstrate the capabilities of smc_storm we selected all DTMC models with their related properties from the QVBS [29] that are compatible with it (i.e., single initial state, atomic property). The list of models used for this evaluation with the IDs shown in the plots, together with the related constants and properties, can be found in Table 2.

Because of the different statistical methods in use (see Sec. 2.2), we split our tests in two groups, depending on the property under verification: the first group consists of all available probability (P) properties and the second one consists of all available reward (E) properties. Additionally, for smc_storm and modes, we measure the performance both in single-threaded and multi-threaded mode, to evaluate the impact of parallelization on the performance of the tools. In the plots shown in the following, a comparison to PRISM is always provided, though it doesn't support multi-threading, and therefore we can compare only to the results from the single-threaded runs.

All experiments are performed using $\varepsilon = 0.01$ and $C = 0.95$ from Eq. 1. Each run evaluates exactly one property of a model. For the multi-threaded runs of storm_smc and modes, we use five parallel threads and a batch size of 100 samples. All remaining parameters have been left to their default values. This includes the selection of which statistical method is used to determine when to stop generating new samples. For P properties, smc_storm and modes use the New Adaptive Sampling Method from Chen and Xu [13], while PRISM computes the CI width assuming the generated samples are coming from the Student's t-distribution [45]. For E properties, all three tools make use of the Chow-Robbins Method to evaluate the CI width [15]. It is also worth mentioning that no tool used rare event simulation strategies during these tests, a feature currently provided only by modes.

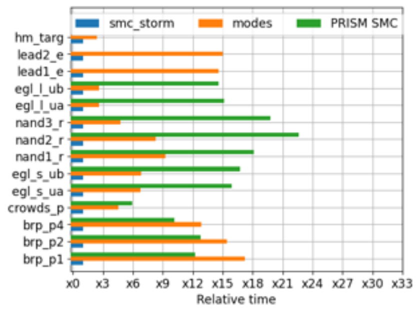
Model	Constants	Property	Instance id
brp	N=64, MAX=5	p1 (P)	brp_p1
		p2 (P)	brp_p2
		p4 (P)	brp_p4
crowds	TotalRuns=6, CrowdSize=20	positive (P)	crowds_p
egl	N=5, L=2	messagesA (E)	egl_s_ma
		messagesB (E)	egl_s_mb
		unfairA (P)	egl_s_ua
		unfairB (P)	egl_s_ub
	N=5, L=8	messagesA (E)	egl_l_ma
		messagesB (E)	egl_l_mb
		unfairA (P)	egl_l_ua
		unfairB (P)	egl_l_ub
leader_sync	N=3, K=2	eventually_elected (P)	lead1_e
		time (E)	lead1_t
	N=5, K=4	eventually_elected (P)	lead2_e
		time (E)	lead2_t
nand	N=20, K=1	reliable(P)	nand1_r
	N=20, K=2	reliable(P)	nand2_r
	N=40, K=2	reliable(P)	nand3_r
oscillators	N=6, T=6, R=1, mu=0.1, λ=1, ϵ=0.1,	time_to_synch (E)	osc_t
		power_consumption (E)	osc_p
haddad-monmege	N=20, p=0.7	target(P)	hm_targ

Table 2: QVBS models and properties used for the evaluation.

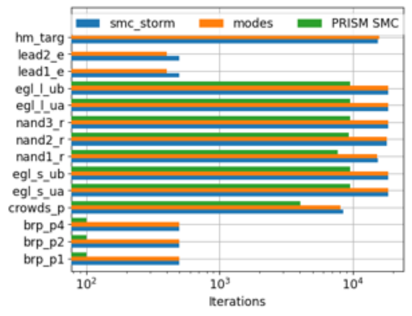
The four metrics we use to compare the different SMC implementations are the computation time, the number of generated samples needed to converge to the result, the maximum amount of used memory and the result error. These metrics are computed for each test instance, consisting of a model, the value of the constants and the property under evaluation. The complete list of test instances is provided in Table 2. For each test, we set a timeout of 20 minutes. When the timeout is reached, we consider the test as skipped.

The results for the multi-threaded tests on P and E properties are summarized in Figure 1 and Figure 2, respectively.

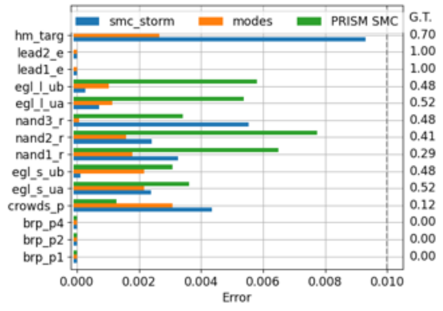
When evaluating P properties, smc_storm is on average 8 times faster than modes and 14 times faster than PRISM SMC (that does not provide a multi-threaded mode). The number of generated samples in smc_storm is comparable to modes and higher than PRISM SMC, which is expectable from the statistical methods in use. Regarding the error of the computed results, all three tools keep the error below the requested limit ε for all the tested benchmarks. Looking at



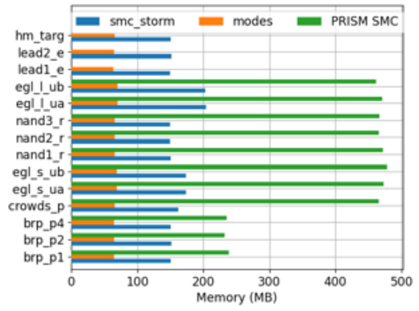
(a) Runtime relative to smc_storm.



(b) Number of generated samples.



(c) Absolute error w.r.t. ground truth.



(d) Max. used memory during runtime.

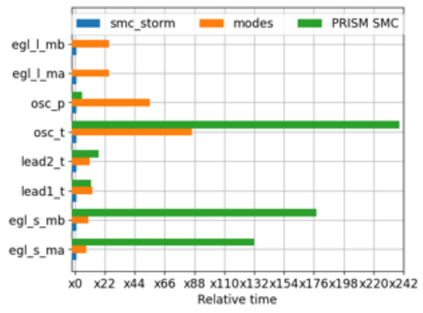
Fig. 1: Performance evaluation of smc_storm, modes, and PRISM SMC (y-axis) on P properties using multi-threading where available. If no bar is present, the test was skipped (20 min limit).

the memory usage, smc_storm uses on average double the memory of modes, but still less than PRISM SMC.

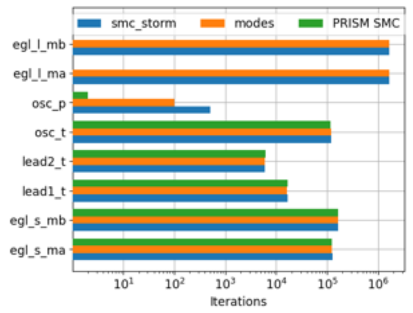
In the case of E properties, smc_storm runs on average 29 times faster than modes and 97 times faster than PRISM SMC, while generating a comparable number of samples to both modes and PRISM SMC. This fulfills the expectations because the same statistical methods are in use. Since for E properties the algorithm needs to keep the state and action rewards in memory, and there is a separated copy for each thread, smc_storm needs on average 6 times the memory used by modes in this case. For what concerns the computed results, the error of all three tools never exceeds ε.

For a fairer comparison, we also provide the results for the single-threaded runs of the tools in Figure 3 and Figure 4.

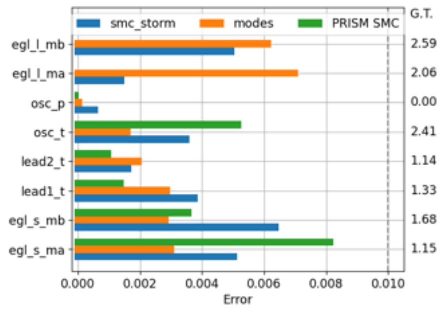
When evaluating P properties, smc_storm is on average 11 times faster than modes and 12 times faster than PRISM SMC, and uses on average as much memory as modes, while still using less memory than PRISM SMC. All other metrics are comparable to the multi-threaded results.



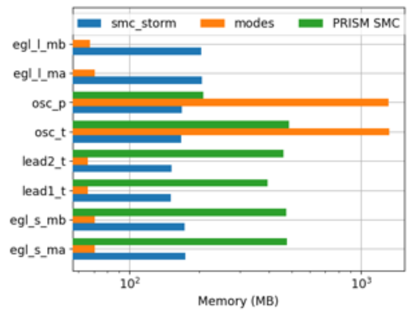
(a) Runtime relative to smc_storm.



(b) Number of generated samples.



(c) Absolute error w.r.t. ground truth.



(d) Max. used memory during runtime.

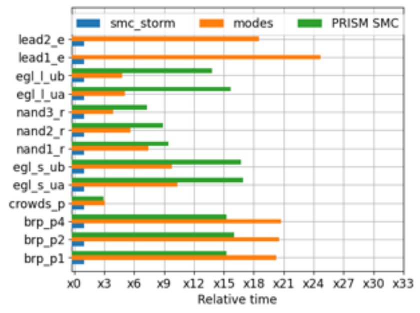
Fig. 2: Performance evaluation of smc_storm, modes, and PRISM SMC (y-axis) on E properties using multi-threading where available. If no bar is present, the test was skipped (20 min limit).

For E properties, smc_storm is on average 45 times faster than modes and 61 times faster than PRISM SMC. All other metrics are comparable to the results obtained for P properties.

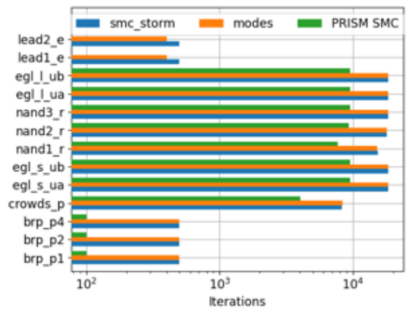
According to our evaluation, smc_storm is the fastest SMC available tool when it comes to evaluating P and E properties without sacrificing accuracy of the result. The statistical methods in use are comparable to the ones used by other tools, especially the ones from modes. In the next *Discussion* paragraph, we provide some thoughts on the possible reasons for the performance differences observed in the evaluation.

Discussion Overall, the results of the evaluation show that smc_storm is currently the fastest SMC tool for evaluating both P and E properties on DTMC models similar to the ones from our evaluation set, without sacrificing accuracy.

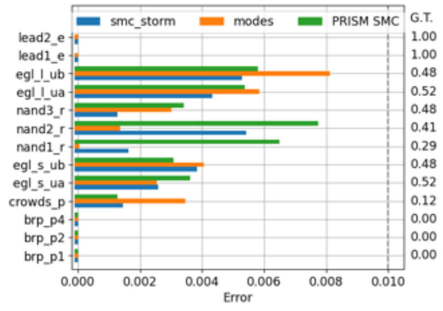
The absolute error plots might suggest that smc_storm is more accurate on P properties and modes is more accurate on E properties. However, this is not consistent over all test instances, so it is not possible to claim that one tool is more accurate than the other.



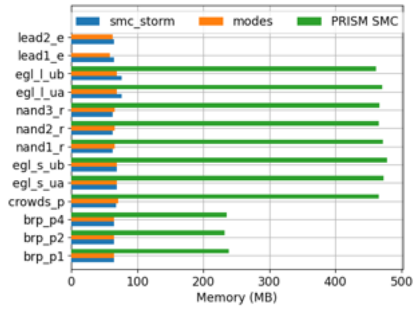
(a) Runtime relative to smc_storm.



(b) Number of generated samples.



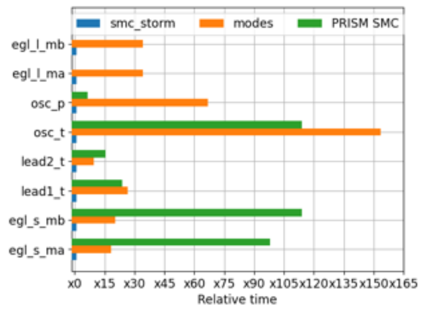
(c) Absolute error w.r.t. ground truth.



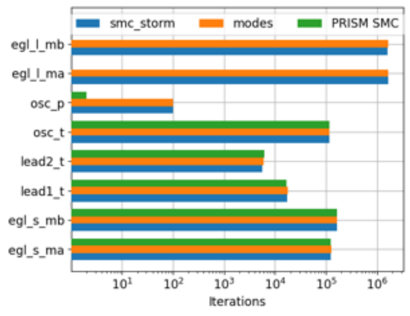
(d) Max. used memory during runtime.

Fig. 3: Performance evaluation of smc_storm, modes, and PRISM SMC (y-axis) on P properties running in single-threaded mode. If no bar is present, the test was skipped (20 min limit).

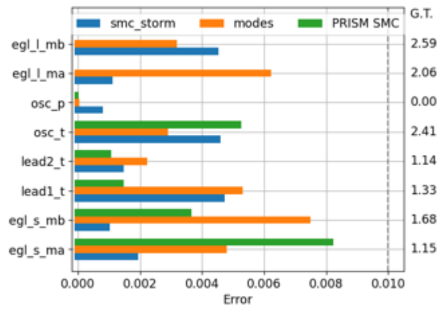
When looking at performance, the first thing to observe is the difference between single-threaded and multi-threaded runs of the tools: smc_storm proved to be faster in either case, but the performance improvement between single- and multi-threading in smc_storm is not as significant as for modes, that is able to get a higher speed-up from multi-threading while keeping its memory usage relatively constant. The measurements of the memory usage of modes suggest that the tool keeps a very limited set of information in memory, forcing it to expand the same states and edges of the model each time a new sample is generated, as opposed to smc_storm, that stores all the information computed from the model to avoid recomputing it when the same state is reached. This particular design choice has multiple implications: the amount of memory smc_storm uses when verifying models with a very large state space (e.g. in the robotics example introduced in Sec. 3.2) could be large, and the computation required to keep the information in memory could also increase with the amount of data. After a certain point, the gain of keeping information in memory might be outweighed by the expense of keeping them up to date. Comparisons between single- and multi-thread run-times of modes and smc_storm are depicted in Figure 5.



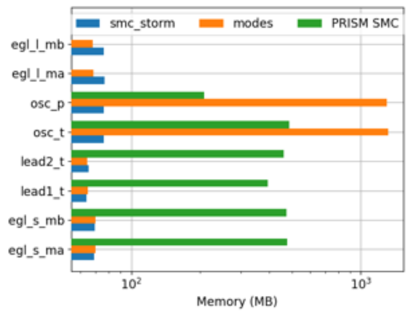
(a) Runtime relative to smc_storm.



(b) Number of generated samples.



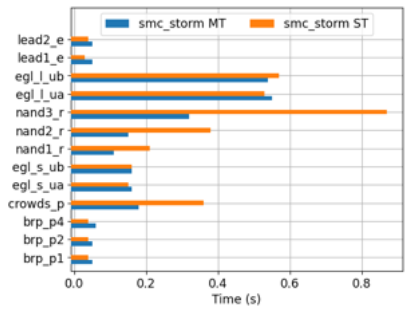
(c) Absolute error w.r.t. ground truth.



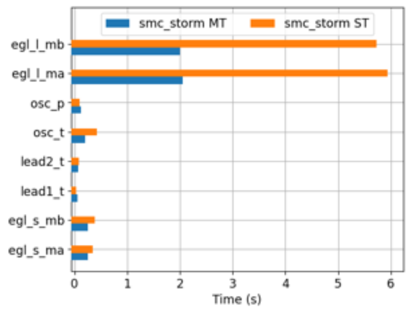
(d) Max. used memory during runtime.

Fig. 4: Performance evaluation of smc_storm, modes, and PRISM SMC (y-axis) on E properties running in single-threaded mode. If no bar is present, the test was skipped (20 min limit).

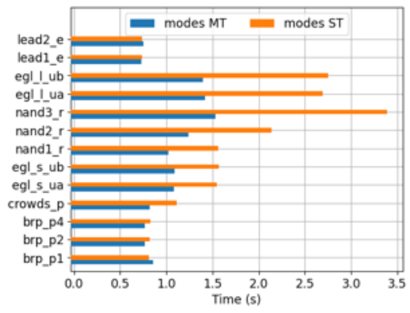
PRISM SMC does not have multi-thread capabilities, but its performance in the single-threaded runs is comparable to modes, while requiring the largest amount of memory. The comparison highlights that the smaller tests in smc_storm (i.e. the ones running in less than a second) often are faster in single-threaded than in multi-threaded mode. We observed similar results in modes as well. This is the case when the traces can be computed very quickly and the largest overhead in the algorithm comes from creating the threads and the synchronization of the results across them. However, most use cases are large enough to make multi-threading worth it, and given the usual performance improvement from multi-threading in smc_storm, we encourage to use it whenever possible. This is particularly true when evaluating E properties, that require on average a larger amount of samples to converge to the desired result, making the performance improvement more significant. Lastly, we observed that the performance increase resulting from using smc_storm is much higher when evaluating E properties compared to P properties: this could be explained by the fact that smc_storm stores the explored rewards in memory, and therefore it has to compute them



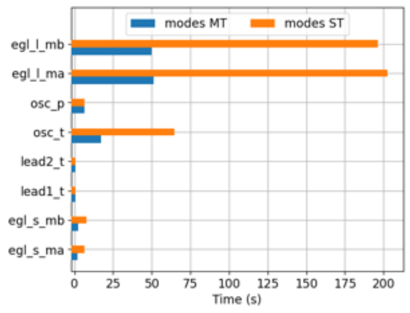
(a) Runtime smc_storm on P prop.



(b) Runtime smc_storm on E prop.



(c) Runtime modes on P prop.



(d) Runtime modes on E prop.

Fig. 5: Multi- (MT) vs. single-threaded (ST) smc_storm and modes.

only once. However, it is just an hypothesis that requires further investigations to be confirmed.

The reported results suggest that smc_storm can still be improved when it comes to multi-threading, especially in terms of memory and synchronization efficiency, to reduce memory usage and, at the same time, get even higher performance.

3.2 Evaluation on Robotics Example

In addition, we run smc_storm on a first robotic example, modeling the full robotic deliberation behavior in a realistic environment.

Our current example consists of a mobile robot operating in a 2D environment. The robot description consists of its shape, that stays constant, and its 2D pose (x and y-coordinates plus its orientation), that changes over time. The robot's position also represents the current state of the DTMC model under investigation. At each transition, the robot can either drive forward or turn on the spot, with equal probabilities.

In addition, the model describes the environment the robot is acting in. The environment consists of its boundaries, that are constant and represent, e.g., the walls of the room, and of obstacles, that currently can only be stationary, but we plan to make them dynamically moveable in the future.

The coordinates of the robot and the environment are given using real numbers, and the most complex operations in the model are the ones defining the intersections of the robot's trajectory with the boundaries and obstacles of the environment, to determine the robot's position after driving. Additionally, given the limitation of STORM not supporting real non-transient variables, we discretize the state space (i.e., the robot pose) to represent the x- and y-coordinates in centimeters and the orientation in degrees, such that the state of the model can be represented using only integers. With that solution, each transition starts by converting the state back to real numbers (i.e., using meters and radians) and finishes by converting back to integers. This approach is also useful to prevent the number of generated states from growing too large.

The described model is completely modeled in JANI, but given the amount of operations required to express the geometric operations introduced above, we implemented a new robotics flavored version of JANI, including syntactic sugar to express geometries, to describe the robot's properties and the environment, and to define geometric operators not available in the original JANI definition. Models written in *robotics-JANI* can be automatically expanded into original JANI v.1 with a Python script, that will also be provided open-source, to enable verification of such models with the model checking tool of choice.

To verify the robotics model including the geometric aspects, we extended STORM to support some trigonometry operators that, until now, were not supported. The new operators are already available in the latest version of STORM.

Using smc_storm and modes, we evaluated the robotic model on a simple property, checking whether the robot eventually reaches a target position in a fixed number of steps (i.e., 10000 steps). The result calculated by smc_storm and modes is the same (up to ε), but given the number of states generated during the evaluation (almost eight million states), the tools showed very specific behaviors in terms of memory usage and runtime. In particular, smc_storm used roughly 1GB in single-threaded mode and 3GB when using five threads, while modes used roughly 80MB in both cases. The runtime of the two tools was rather comparable, with smc_storm being slightly faster in single-threaded mode and modes being slightly faster in multi-threaded mode. These observations are in line with what we saw in the evaluation of the QVBS benchmarks. In the discussion paragraph above, we already provided some thoughts on the possible reasons for the behavior, that is largely affected by the size of the state space.

In the future we plan to extend the robotics model to verify more complex functionalities of the robot, with particular focus on the deliberation algorithms defining its high level behavior in challenging environments. Properties of interest are for example that the robot eventually finds a way back to its docking station or that the robot is able to drive away from a very cluttered area in less then a predefined number of time steps.

4 Conclusions

We introduced smc_storm, a SMC extension of STORM offering diverse statistical methods and evaluated its performance on QVBS benchmarks and a first robotics example with realistic geometric calculations against other state-of-the-art statistical model checkers. Already at the current early development stage smc_storm outperforms other implementations on most of the benchmarks. In the next development steps further improvements in terms of memory efficiency and improved multi-threading mechanisms will be evaluated. In addition, studies on further real-life industrial use cases from the robotics domain are ongoing. This will lead to extensions and adaptions of the implementation with the integration of further algorithmic improvements, e.g., an extension of the SMC engine towards nondeterministic models like MDPs, or timed models, and the connection to other modeling languages are considered.

The overarching goal of our work is to bring together existing MC and robotic tools including modeling languages by an open-source framework which also commercial providers can link their solutions to. For this, STORM with its new SMC extension provides a stable basis. In this context, we are also working on a feature extension of the JANI-format to make the expression of robotic functionalities and environment models more natural and user-friendly. Combining those two lines of work, the overall goal is a framework to which robotics engineers can link their solutions to with lightweight adapters and small tools on top of existing software.

Acknowledgments. The work was supported by the European Union's Horizon Europe Research & Innovation Program under Grant 101070227 (CONVINCE). We thank Arnd Hartmanns for providing us with a license allowing to integrate The MODEST Toolset into the benchmarking comparisons.

References

1. Andriushchenko, R., Bork, A., Budde, C.E., Češka, M., Grover, K., Hahn, E.M., Hartmanns, A., Israelsen, B., Jansen, N., Jeppson, J., Junges, S., Köhl, M.A., Könighofer, B., Křetínský, J., Meggendorfer, T., Parker, D., Pranger, S., Quatmann, T., Ruijters, E., Taylor, L., Volk, M., Weininger, M., Zhang, Z.: Tools at the frontiers of quantitative verification (2024), https://arxiv.org/abs/2405.13583
2. Ashok, P., Křetínský, J., Weininger, M.: Pac statistical model checking for markov decision processes and stochastic games. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 497–519. Springer International Publishing, Cham (2019)
3. Baier, C., Katoen, J.P.: Principles of Model Checking (Representation and Mind Series). The MIT Press (2008), https://dl.acm.org/doi/book/10.5555/1373322
4. Bakir, M.E., Gheorghe, M., Konur, S., Stannett, M.: Comparative analysis of statistical model checking tools. In: Int. Conf. on Membrane Computing (2016). https://doi.org/10.1007/978-3-319-54072-6_8, https://api.semanticscholar.org/CorpusID:1161173

5. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA. pp. 125–126. IEEE Computer Society (2006). https://doi.org/10.1109/QEST.2006.59, https://doi.org/10.1109/QEST.2006.59
6. Biggar, O., Zamani, M.: A framework for formal verification of behavior trees with linear temporal logic. IEEE Robotics Autom. Lett. **5**(2), 2341–2348 (2020). https://doi.org/10.1109/LRA.2020.2970634, https://doi.org/10.1109/LRA.2020.2970634
7. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for modestly nondeterministic models. In: Schmitt, J.B. (ed.) Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance - 16th International GI/ITG Conference, MMB & DFT 2012, Kaiserslautern, Germany, March 19-21, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7201, pp. 249–252. Springer (2012). https://doi.org/10.1007/978-3-642-28540-0_20, https://doi.org/10.1007/978-3-642-28540-0_20
8. Boyer, B., Corre, K., Legay, A., Sedwards, S.: Plasma-lab: A flexible, distributable statistical model checking library. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) Quantitative Evaluation of Systems. pp. 160–164. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
9. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: An efficient statistical model checker for nondeterminism and rare events. Int. J. Softw. Tools Technol. Transf. **22**(6), 759–780 (2020). https://doi.org/10.1007/S10009-020-00563-2, https://doi.org/10.1007/s10009-020-00563-2
10. 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.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10206, pp. 151–168 (2017). https://doi.org/10.1007/978-3-662-54580-5_9, https://doi.org/10.1007/978-3-662-54580-5_9
11. Budde, C.E., Hartmanns, A., Klauck, M., Kretínský, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z.: On correctness, precision, and performance in quantitative verification - qcomp 2020 competition report. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 12479, pp. 216–241. Springer (2020). https://doi.org/10.1007/978-3-030-83723-5_15, https://doi.org/10.1007/978-3-030-83723-5_15
12. Bulychev, P., David, A., Guldstrand Larsen, K., Legay, A., Mikučionis, M., Bøgsted Poulsen, D.: Checking and distributing statistical model checking. In: Goodloe, A.E., Person, S. (eds.) NASA Formal Methods. pp. 449–463. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
13. Chen, J., Xu, J.: A new adaptive sampling method for scalable learning. In: Proceedings of the International Conference on Information and Knowledge Engineering (IKE). p. 1. The Steering Committee of The World Congress in Computer Science, Computer ... (2013)

14. Chernoff, H.: A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations. The Annals of Mathematical Statistics **23**(4), 493 – 507 (1952). https://doi.org/10.1214/aoms/1177729330, https://doi.org/10.1214/aoms/1177729330
15. Chow, Y.S., Robbins, H.: On the Asymptotic Theory of Fixed-Width Sequential Confidence Intervals for the Mean. The Annals of Mathematical Statistics **36**(2), 457 – 462 (1965). https://doi.org/10.1214/aoms/1177700156, https://doi.org/10.1214/aoms/1177700156
16. Clarke, E.M., Klieber, W., Nováček, M., Zuliani, P.: Model Checking and the State Explosion Problem, pp. 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35746-6_1, https://doi.org/10.1007/978-3-642-35746-6_1
17. Clarke, E.M., Zuliani, P.: Statistical model checking for cyber-physical systems. In: Bultan, T., Hsiung, P.A. (eds.) Automated Technology for Verification and Analysis. pp. 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
18. Clarke, E., Grumberg, O., Peled, D., Peled, D.: Model Checking. The Cyber-Physical Systems Series, MIT Press (1999), https://books.google.de/books?id=Nmc4wEaLXFEC
19. Clopper, C.J., Pearson, E.S.: The use of Confidence or Fiducial Limits illustrated in the case of the Binomial. Biometrika **26**(4), 404–413 (12 1934). https://doi.org/10.1093/biomet/26.4.404, https://doi.org/10.1093/biomet/26.4.404
20. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Uppaal SMC tutorial. Int. J. Softw. Tools Technol. Transf. **17**(4), 397–415 (2015). https://doi.org/10.1007/S10009-014-0361-Y, https://doi.org/10.1007/s10009-014-0361-y
21. Dust, L.J., Gu, R., Seceleanu, C., Ekström, M., Mubeen, S.: Pattern-based verification of ROS 2 nodes using UPPAAL. In: Cimatti, A., Titolo, L. (eds.) Formal Methods for Industrial Critical Systems - 28th International Conference, FMICS 2023, Antwerp, Belgium, September 20-22, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14290, pp. 57–75. Springer (2023). https://doi.org/10.1007/978-3-031-43681-9_4, https://doi.org/10.1007/978-3-031-43681-9_4
22. Ebert, S., Mey, J., Schöne, R., Götz, S., Aßmann, U.: DiNeROS: A model-driven framework for verifiable ROS applications with Petri nets. In: Proceedings of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) (2023)
23. Gros, T.P., Hermanns, H., Hoffmann, J., Klauck, M., Steinmetz, M.: Deep statistical model checking. In: Gotsman, A., Sokolova, A. (eds.) 40th International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE). Formal Techniques for Distributed Objects, Components, and Systems, vol. LNCS-12136, pp. 96–114. Springer International Publishing, Valletta, Malta (Jun 2020). https://doi.org/10.1007/978-3-030-50086-3_6, https://inria.hal.science/hal-03283238, part 1: Full Papers
24. Grunske, L., Lindsay, P.A., Yatapanage, N., Winter, K.: An automated failure mode and effect analysis based on high-level design specification with behavior trees. In: Romijn, J., Smith, G., van de Pol, J. (eds.) Integrated Formal Methods, IFM 2005. LNCS, vol. 3771, pp. 129–149. Springer (2005). https://doi.org/10.1007/11589976_9, https://doi.org/10.1007/11589976_9
25. Hahn, E.M., Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Kretínský, J., Parker, D., Quatmann, T., Ruijters, E., Steinmetz, M.: The 2019 comparison of tools for the analysis of quantitative formal models - (qcomp 2019 competition report). In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) Tools

and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11429, pp. 69–92. Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_5, https://doi.org/10.1007/978-3-030-17502-3_5

26. 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). https://doi.org/10.1007/S10703-012-0167-Z, https://doi.org/10.1007/s10703-012-0167-z
27. Hartmanns, A., Hermanns, H.: The modest toolset: An integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51, https://doi.org/10.1007/978-3-642-54862-8_51
28. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: A practitioner's guide to mdp model checking algorithms. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 469–488. Springer Nature Switzerland, Cham (2023)
29. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 344–350. Springer International Publishing, Cham (2019)
30. Henriques, D., Martins, J.G., Zuliani, P., Platzer, A., Clarke, E.M.: Statistical model checking for markov decision processes. In: Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012. pp. 84–93. IEEE Computer Society (2012). https://doi.org/10.1109/QEST.2012.19, https://doi.org/10.1109/QEST.2012.19
31. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker storm. Int. J. Softw. Tools Technol. Transf. **24**(4), 589–610 (2022). https://doi.org/10.1007/s10009-021-00633-z, https://doi.org/10.1007/s10009-021-00633-z
32. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 73–84. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
33. Klauck, M., Lange, R., Henkel, C., Kchir, S., Palmas, M.: Towards robust autonomous robots using statistical model checking. In: Springer Proceedings in Advanced Robotics (SPAR), European Robotics Forum (ERF) (2024), accepted, to be published
34. Kwiatkowska, M., Norman, G., Parker, D.: Advances and challenges of probabilistic model checking. In: Proc. 48th Annual Allerton Conference on Communication, Control and Computing. pp. 1691–1698. IEEE Press (2010)
35. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011)
36. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: Ninth International Conference on Quantitative Evaluation of Systems, QEST

2012, London, United Kingdom, September 17-20, 2012. pp. 203–204. IEEE Computer Society (2012). https://doi.org/10.1109/QEST.2012.14, https://doi.org/10.1109/QEST.2012.14
37. Larsen, K.G., Legay, A.: Statistical model checking: Past, present, and future. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques. pp. 3–15. Springer International Publishing, Cham (2016)
38. Larsen, K.G., Lorber, F., Nielsen, B.: 20 years of UPPAAL enabled industrial model-based validation and beyond. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 11247, pp. 212–229. Springer (2018). https://doi.org/10.1007/978-3-030-03427-6_18, https://doi.org/10.1007/978-3-030-03427-6_18
39. Larsen, K.G., Lorber, F., Nielsen, B.: 20 years of real real time model validation. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E.P. (eds.) Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10951, pp. 22–36. Springer (2018). https://doi.org/10.1007/978-3-319-95582-7_2, https://doi.org/10.1007/978-3-319-95582-7_2
40. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**(1-2), 134–152 (1997). https://doi.org/10.1007/S100090050010, https://doi.org/10.1007/s100090050010
41. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G.J., Rosu, G., Sokolsky, O., Tillmann, N. (eds.) Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6418, pp. 122–135. Springer (2010). https://doi.org/10.1007/978-3-642-16612-9_11, https://doi.org/10.1007/978-3-642-16612-9_11
42. Legay, A., Lukina, A., Traonouez, L., Yang, J., Smolka, S.A., Grosu, R.: Statistical model checking. In: Steffen, B., Woeginger, G.J. (eds.) Computing and Software Science - State of the Art and Perspectives, Lecture Notes in Computer Science, vol. 10000, pp. 478–504. Springer (2019). https://doi.org/10.1007/978-3-319-91908-9_23, https://doi.org/10.1007/978-3-319-91908-9_23
43. Legay, A., Sedwards, S., Traonouez, L.: Estimating rewards & rare events in nondeterministic systems. Electron. Commun. Eur. Assoc. Softw. Sci. Technol. **72** (2015). https://doi.org/10.14279/TUJ.ECEASST.72.1023, https://doi.org/10.14279/tuj.eceasst.72.1023
44. Lindsay, P.A., Winter, K., Yatapanage, N.: Safety assessment using behavior trees and model checking. In: Fiadeiro, J.L., Gnesi, S., Maggiolo-Schettini, A. (eds.) 8th IEEE Int. Conf. on Soft. Engin. & Formal Methods, SEFM. pp. 181–190. IEEE Computer Society (2010). https://doi.org/10.1109/SEFM.2010.23, https://doi.org/10.1109/SEFM.2010.23
45. Nimal, V.: Statistical Approaches for Probabilistic Model Checking. MSc Mini-project Dissertation, Oxford University Computing Laboratory (2010)
46. Raychaudhuri, S.: Introduction to monte carlo simulation. In: Mason, S.J., Hill, R.R., Mönch, L., Rose, O., Jefferson, T., Fowler, J.W. (eds.) Proceedings of the 2008 Winter Simulation Conference, Global Gateway to Discovery, WSC 2008, InterContinental Hotel, Miami, Florida, USA, December 7-10, 2008. pp. 91–

100. WSC (2008). https://doi.org/10.1109/WSC.2008.4736059, https://doi.org/10.1109/WSC.2008.4736059

47. Reijsbergen, D., de Boer, P.T., Scheinhardt, W., Haverkort, B.: On hypothesis testing for statistical model checking. International Journal on Software Tools for Technology Transfer **17**(4), 377–395 (Aug 2015). https://doi.org/10.1007/s10009-014-0350-1, https://doi.org/10.1007/s10009-014-0350-1
48. Vandin, A., Giachini, D., Lamperti, F., Chiaromonte, F.: Automated and distributed statistical analysis of economic agent-based models. Journal of Economic Dynamics and Control **143**, 104458 (Oct 2022). https://doi.org/10.1016/j.jedc.2022.104458, http://dx.doi.org/10.1016/j.jedc.2022.104458
49. Wald, A.: Sequential Tests of Statistical Hypotheses. The Annals of Mathematical Statistics **16**(2), 117 – 186 (1945). https://doi.org/10.1214/aoms/1177731118, https://doi.org/10.1214/aoms/1177731118
50. Yatapanage, N., Winter, K., Zafar, S.: Slicing behavior tree models for verification. In: Calude, C.S., Sassone, V. (eds.) Theoretical Computer Science - 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings. IFIP Advances in Information and Communication Technology, vol. 323, pp. 125–139. Springer (2010). https://doi.org/10.1007/978-3-642-15240-5_10, https://doi.org/10.1007/978-3-642-15240-5_10
51. Younes, H.L.S., Kwiatkowska, M.Z., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. Int. J. Softw. Tools Technol. Transf. **8**(3), 216–228 (2006). https://doi.org/10.1007/S10009-005-0187-8, https://doi.org/10.1007/s10009-005-0187-8
52. Younes, H.L.S.: Verification and planning for stochastic processes with asynchronous events. Carnegie Mellon University (2004)