RealySt: A C++ Tool for Optimizing Reachability Probabilities in Stochastic Hybrid Systems

Joanna Delicaris[1(✉)], Jonas Stübbe[1], Stefan Schupp[2], and Anne Remke[1]

[1] Westfälische Wilhelms-Universität, 48149 Münster, Germany
{joanna.delicaris,jonas.stuebbe,anne.remke}@uni-muenster.de
[2] TU Wien, 1040 Wien, Austria
stefan.schupp@tuwien.ac.at

Abstract. This paper presents the open-source C++ tool RealySt for effectively computing optimal time-bounded reachability probabilities for subclasses of hybrid automata extended with random clocks. The tool explicitly resolves the underlying nondeterminism and computes reachable state sets exactly. The error of the computed results solely stems from the multi-dimensional integration. The architecture of RealySt is extensible and allows to easily integrate other classes of hybrid automata extended by random clocks. RealySt relies on the HyPro library to perform flowpipe construction, and on GSL for multi-dimensional integration.

Keywords: Tool · (Optimal) reachability probabilities · Stochastic hybrid automata

1 Introduction

Stochastic hybrid systems (SHS) combine discrete, continuous and stochastic behavior. Especially in systems which affect humans or where humans are in the loop, safety is of major importance and formal approaches, such as time-bounded reachability analysis allow to make validated statements about the safety of a system. The reachability analysis of SHS poses a diverse set of challenges, rooted in the combination of discrete-continuous behavior with stochasticity. Nondeterminism, which arises naturally e.g., in concurrent systems, is often resolved probabilistically in (purely) stochastic models [1,6,20], and is usually not explicitly resolved in non-stochastic hybrid systems. Maintaining the discrete and continuous nondeterminism present, e.g., in initial sets, dynamic behavior or time delays in a stochastic hybrid system, enables us to optimize reachability probabilities, leading to *maximum* and *minimum* reachability probabilities.

Supported by DFG project 471367371.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2024
Published by Springer Nature Switzerland AG 2024. All Rights Reserved
E. Kalyvianaki and M. Paolieri (Eds.): VALUETOOLS 2023, LNICST 539, pp. 170–182, 2024.
https://doi.org/10.1007/978-3-031-48885-6_11

RealySt is an open-source tool that is specifically designed to optimize discrete and continuous nondeterminism in SHS. The tool is taylored for stochastic variants of subclasses of hybrid automata, namely *singular automata with random clocks (SAR)* [22,26] and *rectangular automata with random clocks (RAR)* [11]. Currently, RealySt resolves discrete and continuous nondeterminism via maximum prophetic schedulers and computes optimal time-bounded reachability probabilities. The computations performed in RealySt rely on the library HyPro for geometric operations on convex polytopes and the construction of the reachable state space as a flowpipe [3]. By using the rational number format GMP, these state sets are guaranteed to be exact. Multi-dimensional integration is currently done via Monte Carlo VEGAS as provided by GSL [12] which computes a probability and statistical error for a predefined number of samples.

We illustrate the capabilities of RealySt on an existing case study [11] with considerably improved computation times.

Related Work The tool HPNMG [18] is a model checker for hybrid Petri nets with general transitions (HPnGs) [14], which exhibit discrete and continuous behavior, as well as stochasticity in the form of *general transitions* modeling random delays. Earlier work resolved the inherent discrete nondeterminism of HPnGs via weights [14], and has later been optimized for both prophetic and non-prophetic schedulers in [21]. Faust^2 [27] only supports *discrete-time* stochastic systems. Its successor StocHy [8] then performs quantitative analysis of *discrete-time* SHS by constructing abstractions in the form of (interval) Markov decision processes (MDPs). The MODEST toolset [16] is a collection of tools for the analysis of stochastic timed and hybrid systems. Using ProHVer [15] as a backend, MODEST analyses reachability in stochastic hybrid systems with general probability distributions and (nonlinear) continuous dynamics. It performs a discretization of the support of continuous stochastic delays and an abstraction for discrete probabilistic behaviour. Together this results in an overapproximation of the actual reachability probabilities. ProbReach [25] provides algorithms to analyse *parametric* SHS that allow for nonlinear continuous dynamics, where parameters can be random as well as nondeterministic. Here, bounded reachability analysis is encoded as a first-order logic formula and solved using dReach [19], which provides a rigorous enclosure that includes the sought reachability probability. Recent computations within the ARCH competition [2] show that ProbReach is not competitive with RealySt on small examples. UPPAAL [5] offers a wide range of tools and verification techniques for networks of timed automata. Stochastic extensions for more expressive models can be evaluated with *statistical model checking* and optimal strategies can be identified and constructed using Stratego [10] and Tiga [4].

Outline After introducing the class of models in Sect. 2, the tool is described in Sect. 3. The evaluation of the case study is presented in Sect. 4 and the paper is concluded in Sect. 5.

2 Model Classes

Hybrid automata (HA) [3] are a modeling formalism for systems which exhibit mixed discrete-continuous behavior. A subclass of HA are rectangular automata [17], where rates and conditions are described via intervals.

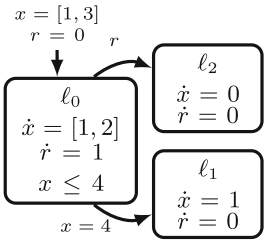

Fig. 1. RAR with random clock r.

Recently, rectangular automata with random clocks (RAR) have been proposed as a stochastic extension for this subclass of HA, where the duration of random delays is measured explicitly via stopwatches and constraints on the syntax ensure that the resulting probability space is sound. We refer the interested reader to [11] for a detailed presentation of their syntax and semantic.

RealySt implements the algorithm proposed in [11] to maximize reachability probabilities in rectangular automata with random clocks (RAR) for history-dependent prophetic schedulers. RAR allow for (i) *initial nondeterminism* in the choice of the initial state, (ii) *time nondeterminism* when time can elapse but also jumps are enabled during the whole flow, and (iii) *rate nondeterminism* when continuous variables can evolve with different rates. In addition to these continuous types of nondeterminism, RAR also contain (iv) *discrete nondeterminism* when different jumps are enabled simultaneously. An example RAR with random clock r is shown in Fig. 1, see Appendix A for explanation. We use prophetic schedulers [9] to resolve nondeterminism, which have full information not only on the history but also on the future expiration times of all random clocks, as introduced in [22]. While prophetic scheduling may seem unrealistic, it is well-suited to perform a *worst-case* analysis, especially when uncontrollable uncertainties are modeled nondeterministically.

For a bounded number of jumps, reachability is decidable for non-initialized rectangular automata (RA) [3]. Hence, forward flowpipe construction computes exact reachable state-sets, e.g. using convex polytopes as a state-set representation. Jump-bounded reachability can also be computed for RAR,, when stochasticity is disregarded, i.e., stochastic edges that represent the expiration of a random delay are interpreted as regular edges without guard. Since it has been shown that RA can be transformed into singular automata (SA), stopwatch automata (SWA) and timed automata (TA), respectively [3,17], all of those model classes can be extended with random clocks, similarly to RAR.

3 The RealySt Tool

We present RealySt, an open source `C++` tool available online via go.unims.de/realyst, for the efficient computation of optimal reachability probabilities in stochastic hybrid automata. Currently, RealySt relies on the state-space representation in terms of convex polytopes. While this representation is exact for the classes of hybrid automata that are currently supported by RealySt, the geometric operations are computationally involved. Hence, the tool architecture is extensible and can easily include other state-space representations like zonotopes or support functions, in the future [23], as well as more efficient multi-dimensional integration [28].

This section is further organized as follows: Sect. 3.1 presents the overall program flow, Sect. 3.2 the necessary details on backward refinement and integration and Sect. 3.3 introduces dependendicies and usage parameters.

3.1 Program Flow

Figure 2 illustrates the program flow for a given automaton.

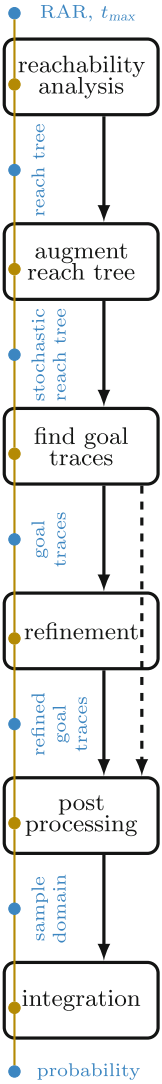

Fig. 2. Data flow in RealySt.

0. Initially, a rectangular automaton with random clocks, a time bound t_{max} and a goal specification is given.
1. The `reachabilityAnalysis` results in a *reach tree* up to the time bound t_{max}, where *nodes* link to state sets represented by convex polytopes. Here, random clocks are treated as continuous variables. The computed reachable state space then contains all possible valuations of the random clocks, regardless of their distributions' domain.

2. The method `augmentReachTree` augments the *reach tree* with stochastic information. Each random delay is modeled as a *stochastic edge* with a corresponding random clock, which is *active* in the source location of that edge. This information from the model is embedded in the nodes of the reach tree, resulting in a *stochastic reach tree*.
3. Next, `selectGoalStates` identifies traces of the *(stochastic) reach tree* that reach the goal. Each node containing goal states then induces a *goal trace* starting from the root of the reach tree, and links to the *refined segment*, which results from the intersection of the nodes state set with the goal.
4. The `backwardRefinement` starts from the refined segments and effectively computes a scheduler partitioning on the state space regarding their ability to reach the goal. This results in *refined* goal traces, which hold a sequence of convex polytopes representing all states that lead to the goal.
5. Method `findIntegrationBounds` prepares the obtained state sets for integration by (i) projection onto the stochastic domain, (ii) adaptation of upper bounds to incorporate later expiration of random delays, (iii) intersection of all resulting constraints per trace. For each goal trace, we obtain a (possibly unbounded) convex polytope that contains all sample values that allow to follow that trace to the goal. The *sample domain* is given by the union of these polytopes.
6. The `numericalIntegration` over the union of polytopes is done via Monte Carlo VEGAS and results in a optimal reachability probability, as well as a *statistical* error and an *integration* error.

3.2 Backward Refinement and Integration

Backward refinement restricts the reachable state space to the fragment from which the goal can be reached. This iterative process alternates the computation of *refined segments* and *intermediate goal segments* on all traces in the reach tree that lead to goal states. A scheduler that stays in the refined segment hence resolves all continuous nondeterminism, effectively creating a purely stochastic version of the model. A partioning of the schedulers is then given by each trace in the refined segments: all schedulers that make similar choices and enable the same trace in the reach tree belong to the same equivalence class. Note that backward refinement is only needed if the model contains continuous nondeterminism, e.g., in the initial set, dynamics, or time delays. Otherwise, e.g. as in the *urgent* version of SAR (c.f. [26]), the backward refinement step can be skipped.

Integration and error. Multidimensional integration over unbounded polytopes is not possible in general. Hence, we restrict the integration domain by a predefined *integration bound* $t_{int} \geq t_{max}$. This results in a new integration region $\mathcal{P}_{max}$, computed by intersection of the union of possibly unbounded polytopes with $[0, t_{int}]^d$. Further, we obtain an overapproximative error:

$$e_\infty = 1 - \int_{[0,t_{int}]^d} G(s)ds.$$

Here, $G = \prod_{r_n} Distr(r_n)$ is the joint probability density function for all random delays r_n with probability density functions $Distr(r_n)$. This error is exact if: (i) no random clock has ever expired upon reaching the goal set on all traces, or (ii) the support of all random clocks that have not expired upon reaching the goal is finite. Otherwise, e_∞ overapproximates the actual error. Clearly, increasing t_{int} decreases e_∞. RealySt implements Monte Carlo VEGAS for multi-dimensional integration, which introduces an additional statistical error e_{stat}, depending on the number of integration samples. This error is estimated based on the weighted average of independent samples and provided directly by GSL [12].

VEGAS uses *importance sampling* and *stratified sampling*, where a multidimensional weight function is used to concentrate the samples in regions with the highest peaks. VEGAS suffers from the curse of dimensionality, because it uses *rejection sampling* on the *bounding box* to obtain samples that are uniformly distributed over the polytopes.

3.3 Dependencies and Usage

RealySt is designed as a command line tool for Linux and our experiments have been run on the *Windows Subsystem for Linux*. The tool mainly depends on the libraries HyPro [24], GSL [12], and GMP [13]. Further dependencies include SPDLOG (logging), CEREAL (serialization), CLI11 (command line interface), and GTEST (unit testing) that are automatically fetched during configuration.

For state-set representations (e.g., convex polytopes), as well as reachability analysis, we rely on HyPro. By using a rational number format provided by GMP, the computation of the state sets is guaranteed to be exact. Integration via Monte Carlo VEGAS with GSL [12] leads to optimal reachability probabilities.

A command-line interface provides the option to analyze programmatically constructed models with different parameters such as the reachability time bound, the integration precision, as well as model-specific configurations. The models are specified in `C++` files.[1]

RealySt can be invoked via `./realyst [OPTIONS]` where the major parameters in `[OPTIONS]` are:

`-h` displays a complete help message with all parameters
`-t` sets the global timebound t_{max} for the reachability analysis
`-d` sets the bound for the jump depth for all traces
`-i` sets the considered intergration bound t_{int}
`-s` sets the number of samples for Monte Carlo integration
`-b` chooses a benchmark, requires further parameters based on choice
`-l` allows to choose a logging level
`-p` enables plotting, requires specifications of which dimensions to plot.

Depending on the chosen benchmark, further sets of parameters may be required. Apart from compiling the source, a user additionally can access a Docker container which holds a compiled version.[2] Once started, the tool will printout the

[1] See https://go.uni-muenster.de/iozrb for an exemplary model file.
[2] https://hub.docker.com/r/realyst/realyst.

chosen parameter settings as well as the final result, and, depending on the logging level, further output during the analysis.

4 Evaluation

We recompute results for the case study presented in [11] to illustrate the improved capabilities of RealySt. The case study models the state of charge of an e-car (c.f. Appendix B). The model contains random *charging* and *driving times*, as well as a scalable number of *detours*. In contrast to [11] we consider uniformly distributed charging and driving times, both following $\mathcal{U}(0, 15)$.

Table 1 shows the results for different model variants evaluated with RealySt and ProHVer [15]. The dimensionality of the variants is given by $K = (|Var_R|, |R|, |I_S|)$, where $|Var_R|$ is the number of random delays in the model, $|R|$ is the number of nodes in the reach tree and $|I_S|$ is the number of traces leading to the goal set. Using $1 \cdot 10^5$, $2 \cdot 10^6$ and $1 \cdot 10^7$ integration samples for $0, 1, 2$ detours respectively, RealySt computes maximum reachability probabilities for all model variants with 0 and 1 detour in less than 2 minutes.

For 2 detours the complexity of the model increases significantly. The computations in RealySt takes less than 45 minutes for the variant singular `A` and about 2 hours for rectangular `A`. In variants `AB` and `ABC` with 2 detours, the size of the state space becomes very large. RealySt is only able to complete the singular variant of `AB`, which takes about 13 hours instead of 83 hours as in [11]. For rectangular `AB` and both versions of `ABC`, flowpipe construction does not terminate anymore for 2 detours.

Table 1. Maximum reachability probabilities for benchmark from [11] with uniformly distributed *charging* and *driving time*, i.e., $c, d \sim \mathcal{U}(0, 15)$. Computation times t_{comp} provided for RealySt (R) with error e_{stat} and ProHVer (P). $K = (|Var_R|, |R|, |I_S|)$.

			0 detours			1 detour			2 detours
		Var.	`A`	`AB`	`ABC`	`A`	`AB`	`ABC`	`A`
		K	$(2, 8, 2)$	$(2, 12, 3)$	$(2, 16, 4)$	$(5, 38, 10)$	$(5, 88, 19)$	$(5, 167, 39)$	$(8, 128, 34)$
Rectangular	R	p_{max}	0.318 970	0.335 863	0.341 218	0.407 734	0.425 624	0.428 865	0.407 632
		e_{stat}	$1.193 \cdot 10^{-3}$	$4.363 \cdot 10^{-4}$	$5.984 \cdot 10^{-4}$	$4.031 \cdot 10^{-4}$	$5.066 \cdot 10^{-4}$	$1.636 \cdot 10^{-3}$	$4.122 \cdot 10^{-3}$
		t_{comp}	0.27 s	0.41 s	0.55 s	8.41 s	38.53 s	115.09 s	7260.44 s
	P	p_{max}	0.351 074	0.367 676	0.371 582	0.625 977	0.660 687	0.660 969	0.862 513
		t_{comp}	568.03 s	1162.00 s	1605.56 s	9907.90 s	21 373.98 s	38 688.62 s	23 273.99 s
Singular	R	p_{max}	0.319 275	0.337 199	0.341 095	0.407 481	0.424 928	0.431 253	0.406 746
		e_{stat}	$1.028 \cdot 10^{-3}$	$4.216 \cdot 10^{-4}$	$4.480 \cdot 10^{-4}$	$3.948 \cdot 10^{-4}$	$1.825 \cdot 10^{-3}$	$1.772 \cdot 10^{-3}$	$9.040 \cdot 10^{-4}$
		t_{comp}	0.33 s	0.41 s	0.51 s	6.78 s	24.67 s	61.90 s	2643.05 s
	P	p_{max}	0.351 074	0.367 676	0.371 582	0.623 630	0.643 771	0.652 361	0.862 513
		t_{comp}	368.76 s	535.77 s	674.30 s	7869.98 s	14 334.37 s	25 710.70 s	13 688.28 s

Due to more efficient memory allocation in RealySt, the computed results are on average 10 times faster than in [11]. Note that the impact of the chosen

distributions on the computation times is negligible. See Appendix C for further details on computation times. The charging rates in the singular variant equal the *lower bounds* of the rectangular interval. Due to the setup of the case study, the resulting maximum reachability probabilities are almost identical. This serves as further validation and illustrates the smaller computation times of the singular variant.

Similar to [11], results are validated by ProHVer and have been recomputed for the uniform distribution. While RealySt results are well supported by the overapproximation computed by ProHVer, computation times of ProHVer are considerably slower. Note that ProHVer is not able to refine its discretization for 2 detours, resulting in a significant overapproximation.

All experiments have been conducted on a machine equipped with an Intel® Core™ i7 with 1.70 GHz and 64 GiB RAM.

5 Conclusion and Future Work

With RealySt we present a tool that is able to optimize reachability probabilities in different types of SHS, potentially containing discrete and continuous nondeterminism, as well as stochastic behavior via random delays. The results of the feasibility study show that RealySt performs very well for up to five random variables. Reachability probabilities are highly accurate and obtained fast in comparison to ProHVer.

Future work includes the computation of *minimum* reachability probabilities, as well the integration of a dedicated analysis for a restricted version of SAR (c.f. [22]), where prophetic and non-prophetic schedulers are used to compute maximum and minimum reachability probabilites. Furthermore, we aim to improve scalability via other state set representations and include established input formats for SHS,, such as the JANI format [7].

Appendix A

Figure 1 illustrates an exemplary RAR with three locations that has two variables: random clock r and continuous variable x. The model contains initial nondeterminism in the valuation of x, which can be chosen from the interval $[1, 3]$. According to the syntax as presented in [11], the random clock r behaves as a stopwatch and has to be 0 initially.

The evolution of r is always either 1 or 0, since it can be *active* or *inactive*. In the given model, r is only active in location ℓ_0. Hence, it evolves with rate 1 in ℓ_0 and there is a stochastic transition from ℓ_0 to ℓ_2 modelling the expiration of the random delay. The continuous variable x, however, can evolve with a rate $\in [1, 2]$ in ℓ_0, which again models nondeterministic behavior. If it reaches a valuation of 4, location ℓ_0 has to be left, either with the stochastic transition or with the transition from ℓ_0 to ℓ_1. In ℓ_1, x is evolving with rate 1 and r is inactive and hence cannot expire anymore.

Appendix B

For completeness, we have included Fig. 3 illustrating the RAR model of the case study evaluated in Sect. 4. Note that the figure is taken from [11].

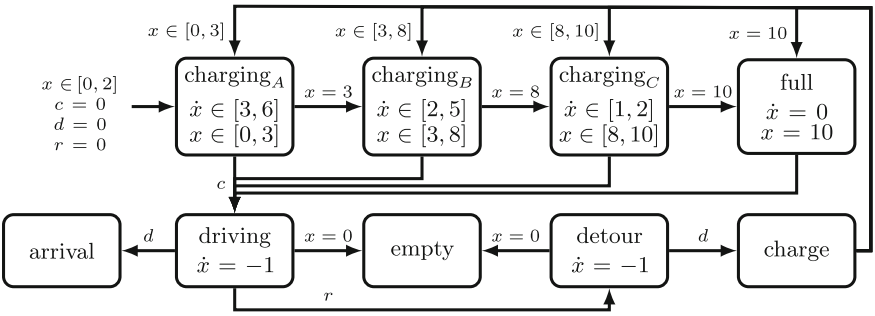

Fig. 3. Car model with detours. Random clock c is active ($\dot{c} = 1$) in the charging locations, d is active in locations *driving* and *detour* and r is active in location *driving*. The state of charge x is restricted to $[0, 10]$ in all locations unless stated otherwise. No time is spent in location *charge* due to invariants not shown.

Appendix C

Figure 4 shows the computation times of the different model variants, where rectangular model variants are indicated with R and singular model variants with S. The computation times are separated for flowpipe construction (step 1), refinement (step 2 and 3), extraction of the sample domain (step 4) and the multi-dimensional integration (step 5). As the figure illustrates, RealySt is considerably quicker for all model variants in comparison to the computations from [11], and especially the flowpipe construction and integration perform much better.

Note that due for to the absence of peaks in the uniform distribution, VEGAS is unable to perform importance sampling. This particularly occurred during the variant singular AB for 2 detours, hence a larger number of integration samples had to be used in the computation. This resulted in a probability of 0.425 528 with $e_{stat} = 1.203 \cdot 10^{-05}$.

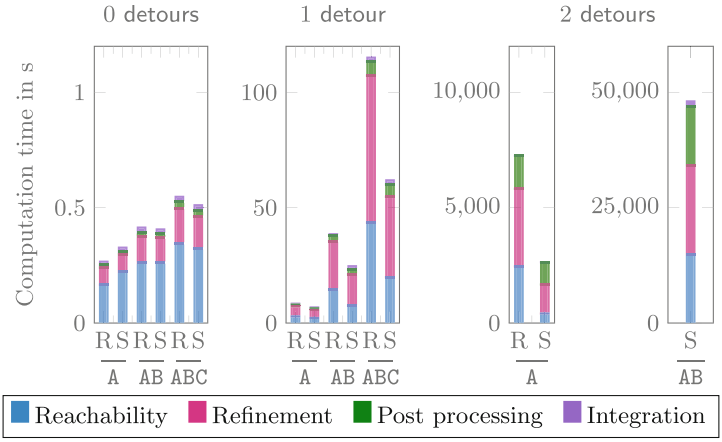

Fig. 4. Computation times for different models and number of detours with RealySt.

References

1. Abate, A., Prandini, M., Lygeros, J., Sastry, S.: Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. Automatica **44**(11), 2724–2734 (2008). https://doi.org/10.1016/j.automatica.2008.03.027
2. Abate, A., Blom, H., Cauchi, N., Delicaris, J., Haesaert, S., van Huijgevoort, B., Lavaei, A., Remke, A., Schön, O., Schupp, S., Shmarov, F., Soudjani, S., Willemsen, L., Zuliani, P.: ARCH-COMP23 Category report: stochastic models. In: 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23). EPiC Series in Computing, EasyChair (2023), accepted for publication
3. Alur, R., Courcoubetis, C.A., Halbwachs, N., Henzinger, T.A., Ho, P., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoret. Comput. Sci. **138**(1), 3–34 (1995). https://doi.org/10.1016/0304-3975(94)00202-T
4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: Uppaal-tiga: Time for playing games! In: Computer Aided Verification, pp. 121–125. Springer, Berlin Heidelberg, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_14
5. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: Uppaal—a tool suite for automatic verification of real-time systems. In: Hybrid Systems III, pp. 232–243. Springer (1996)
6. Bertrand, N., Bouyer, P., Brihaye, T., Menet, Q., Baier, C., Größer, M., Jurdzinski, M.: Stochastic timed automata. Logical Methods Comput. Sci. **10**(4), 1–73 (2014). https://doi.org/10.2168/LMCS-10(4:6)2014
7. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: 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
8. Cauchi, N., Abate, A.: Stochy—automated verification and synthesis of stochastic processes: Poster abstract. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 258–259. Association for Computing Machinery (2019). https://doi.org/10.1145/3302504.3313349
9. D'Argenio, P.R., Gerhold, M., Hartmanns, A., Sedwards, S.: A hierarchy of scheduler classes for stochastic automata. In: Proceedings of FOSSACS'18. LNCS, vol. 10803, pp. 384–402. Springer (2018)
10. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal stratego. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 206–211. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_16
11. Delicaris, J., Schupp, S., Ábrahám, E., Remke, A.: Maximizing reachability probabilities in rectangular automata with random clocks. In: 17th International Symposium on Theoretical Aspects of Software Engineering. LNCS, vol. 13931, pp. 1–19. Springer (2023)
12. Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Alken, P., Booth, M., Rossi, F., Ulerich, R.: GNU Scientific Library Reference Manual (3rd Ed.), http://www.gnu.org/software/gsl/
13. Granlund, T.: The GNU Multiple Precision Arithmetic Library Reference Manual. https://gmplib.org/gmp-man-6.2.1.pdf
14. Gribaudo, M., Remke, A.: Hybrid petri nets with general one-shot transitions. Perform. Eval. **105**, 22–50 (2016). https://doi.org/10.1016/j.peva.2016.09.002
15. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst. Des. **43**(2), 191–232 (2013)
16. Hartmanns, A., Hermanns, H.: The modest toolset: an integrated environment for quantitative modelling and verification. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 593–598. Springer (2014)
17. Henzinger, T.A.: The theory of hybrid automata. In: Verification of Digital and Hybrid systems, NATO ASI Series, vol. 170, pp. 265–292. Springer (2000). https://doi.org/10.1007/978-3-642-59615-5_13
18. Hüls, J., Niehaus, H., Remke, A.: hpnmg: A C++ tool for model checking hybrid petri nets with general transitions. In: Lee, R., Jha, S., Mavridou, A. (eds.) NASA Formal Methods—12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11–15, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12229, pp. 369–378. Springer (2020). https://doi.org/10.1007/978-3-030-55754-6_22
19. Kong, S., Gao, S., Chen, W., Clarke, E.: dreach: δ-reachability analysis for hybrid systems. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 200–205. Springer (2015)
20. Lygeros, J., Prandini, M.: Stochastic hybrid systems: a powerful framework for complex, large scale applications. Eur. J. Control. **16**(6), 583–594 (2010). https://doi.org/10.3166/ejc.16.583-594
21. Pilch, C., Hartmanns, A., Remke, A.: Classic and non-prophetic model checking for hybrid petri nets with stochastic firings. In: HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21–24, 2020, pp. 10:1–10:11. ACM (2020). https://doi.org/10.1145/3365365.3382198

22. Pilch, C., Schupp, S., Remke, A.: Optimizing reachability probabilities for a restricted class of stochastic hybrid automata via flowpipe-construction. In: Quantitative Evaluation of Systems—18th International Conference, QEST 2021, Paris, France, August 23–27, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12846, pp. 435–456. Springer (2021). https://doi.org/10.1007/978-3-030-85172-9_23
23. Schupp, S.: State Set Representations and Their Usage in the Reachability Analysis of Hybrid Systems. Ph.D. thesis, RWTH Aachen University, Aachen (2019). https://doi.org/10.18154/RWTH-2019-08875
24. Schupp, S., Ábrahám, E., Makhlouf, I.B., Kowalewski, S.: HyPro: A C++ library of state set representations for hybrid systems reachability analysis. In: Proceedings of the 9th NASA Formal Methods Symposium (NFM'17). LNCS, vol. LNCS, pp. 288–294. Springer (2017). https://doi.org/10.1007/978-3-319-57288-8_20
25. Shmarov, F., Zuliani, P.: Probreach: verified probabilistic delta-reachability for stochastic hybrid systems. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, pp. 134–139 (2015)
26. da Silva, C., Schupp, S., Remke, A.: Optimizing reachability probabilities for a restricted class of stochastic hybrid automata via flowpipe-construction. Trans. Modeling Comput. Simul. (2023), accepted for publication
27. Soudjani, S.E.Z., Gevaerts, C., Abate, A.: FAUST ^{2} : Formal abstractions of uncountable-state stochastic processes. In: Tools and Algorithms for the Construction and Analysis of Systems—21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11–18, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9035, pp. 272–286. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_23
28. Stübbe, J., Remke, A.: Monte-Carlo integration on a union of polytopes. In: 19th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, Garmisch-Patenkirchen, Germany, June 20–23, 2023 (2023), accepted for publication