A Storm is Coming: A Modern Probabilistic Model Checker

Christian Dehnert(✉), Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk

RWTH Aachen University, Aachen, Germany
dehnert@cs.rwth-aachen.de

Abstract. We launch the new probabilistic model checker Storm. It features the analysis of discrete- and continuous-time variants of both Markov chains and MDPs. It supports the Prism and JANI modeling languages, probabilistic programs, dynamic fault trees and generalized stochastic Petri nets. It has a modular set-up in which solvers and symbolic engines can easily be exchanged. It offers a Python API for rapid prototyping by encapsulating Storm's fast and scalable algorithms. Experiments on a variety of benchmarks show its competitive performance.

1 Introduction

In the last five years, we have developed our in-house probabilistic model checker with the aim to have an easy-to-use platform for experimenting with new verification algorithms, richer probabilistic models, algorithmic improvements, different modeling formalisms, various new features, and so forth. Although open-source probabilistic model checkers do exist, most are not flexible and modular enough to easily support this. Our efforts have led to a toolkit with mature building bricks with simple interfaces for possible extensions, and a modular set-up. It comprises about 100,000 lines of `C++` code. The time has come to make this toolkit available to a wider audience: this paper presents Storm.

Like its main competitors Prism [1], MRMC [2], and iscasMC [3], Storm relies on numerical and symbolic computations. It does not support discrete-event simulation, known as statistical model checking [4]. The main characteristic features of Storm are:

– it supports *various native input formats*: the Prism input format, generalized stochastic Petri nets, dynamic fault trees, and conditioned probabilistic programs. This is not just providing another parser; state-space reduction and generation techniques as well as analysis algorithms are partly tailored to these modeling formalisms;
– in addition to Markov chains and MDPs, it supports *Markov automata* [5], a model containing probabilistic branching, non-determinism, and exponentially distributed delays;

R. Majumdar and V. Kunčak (Eds.): CAV 2017, Part II, LNCS 10427, pp. 592–600, 2017.
DOI: 10.1007/978-3-319-63390-9_31

- it can do *explicit state* and *fully symbolic* (BDD-based) model checking as well as a *mixture* of these modes;
- it has a *modular* set-up, enabling the easy exchange of different solvers and distinct decision diagram packages; its current release supports about 15 solvers, and the BDD packages `CUDD` [6] and multi-threaded `Sylvan` [7];
- it provides a *Python API* facilitating easy and rapid prototyping of other tools using the engines and algorithms in Storm;
- it provides the following functionalities under one roof: the synthesis of counterexamples and permissive schedulers (both MILP- and SMT-based), game-based abstraction of infinite-state MDPs, efficient algorithms for conditional probabilities and rewards [8], and long-run averages on MDPs [9];
- its performance in terms of verification speed and memory footprint on the Prism benchmark suite is mostly better compared to Prism.

Although many functionalities of Prism are covered by Storm, there are significant differences. Storm does not support LTL model checking (as in iscasMC and Prism) and does not support the Prism features: probabilistic timed automata, and an equivalent of Prism's "hybrid" engine (a crossover between full MTBDD and Storm's "hybrid" engine), a fully symbolic engine for continuous-time models, statistical model checking, and the analysis of stochastic games as in Prism-GAMES [10].

2 Features

Model Types. Storm supports Markov chains and Markov decision processes (MDPs), both in two forms: discrete time and continuous time. This yields four different models: classical discrete-time (DTMCs) and continuous-time Markov chains (CTMCs), as well as MDPs and Markov automata (MA) [5], a compositional variant of continuous-time MDPs. The MA is the richest model. CTMCs are MAs without non-determinism, while MDPs are MAs without delays; DTMCs are CTMCs without delays, cf. [11]. All these models are extensible with rewards (or dually: costs) to states, and – for non-deterministic models – to actions. Most probabilistic model checkers support Markov chains and/or MDPs; MAs so far have only been supported by few tools [12,13].

Modeling Languages. Storm supports various symbolic and an explicit input format to specify the aforementioned model types: (i) Most prominently, the Prism input language [1] (ii) the recently defined JANI format [14], a universal probabilistic modeling language; (iii) as the first tool *every*[1] generalized stochastic Petri net (GSPN) [17] via both a dedicated model builder as well as an encoding in JANI; (iv) dynamic fault trees (DFTs) [18,19] – due to dedicated state-space generation and reduction techniques for DFTs, Storm significantly

[1] Existing CSL model checkers for GSPNs such as GreatSPN [15] and MARCIE [16] are restricted to confusion-free Petri nets; Storm does not have this restriction as it supports MA.

outperforms competing tools in this domain [20]; (v) pGCL probabilistic programs [21] extended with observe-statements [22], an essential feature to describe and analyze e.g., Bayesian networks; (vi) in the spirit of MRMC [2], models can be provided in a format that explicitly enumerates transitions.

Properties. Storm focusses on probabilistic branching-time logics, i.e. PCTL [23] and CSL [24] for discrete-time and continuous-time models, respectively. To enable the treatment of reward objectives such as expected and long-run rewards, Storm supports reward extensions of these logics in a similar way as Prism. In addition, Storm supports conditional probabilities and conditional rewards [8]; these are, e.g., important for the analysis of cpGCL programs.

Engines. Storm features two distinct in-memory representations of probabilistic models: *sparse matrices* allow for fast operations on small and moderately sized models, multi-terminal binary decision diagrams (MTBDDs) are able to represent gigantic models, however with slightly more expensive operations. A variety of engines built around the in-memory representations is available, which allows for the more efficient treatment of input models. Both Storm's *sparse* and the *exploration* engine purely use a sparse matrix-based representation. While the former amounts to an efficient implementation of the standard approaches, the latter one implements the ideas of [25] which scrutinizes the state space with machine learning methods. Three other engines, *dd*, *hybrid* and *abstraction-refinement*, use MTBDDs as their primary representation. While *dd* exclusively uses decision diagrams, *hybrid* also uses sparse matrices for operations deemed more suitable on this format. The *abstraction-refinement* engine abstracts (possibly infinite) discrete-time Markov models to (finite) stochastic games and automatically refines the abstraction as necessary.

Parametric Models. Storm was used as backend in [26,27]. By using the dedicated library CArL [28] for the representation of rational functions and applying novel algorithms for the analysis of parametric discrete-time models, it has proven to significantly outperform the dedicated tool PARAM [29] and parametric model checking in Prism.

Exact Arithmetic. Several works [30,31] observed that the numerical methods applied by probabilistic model checkers are prone to numerical problems. Storm therefore supports enabling exact arithmetic to obtain *precise results*.

Counterexample Generation. For probabilistic models, several counterexample representations have been proposed [32]. Storm implements the MILP-based counterexample technique [33], as well as the MaxSat-based generation of high-level counterexamples on Prism models [34]. These algorithms go beyond the capabilities of dedicated, stand-alone counterexample generation tools such as DiPro [35] and COMICS [36]. In particular, the synthesis of high-level counterexamples facilitates to obtain counterexamples as Prism code, starting from a Prism model and a refuted property.

APIs. Storm can be used via *three* interfaces: a command-line interface, a C++ API, and a Python API. The command-line interface consists of several binaries that provide end-users access to the available settings for different tasks. Advanced users can utilize the many settings to tune the performance. Developers may either use the C++ API that offers fine-grained and performance-oriented access to Storm's functionality, or the Python API which allows rapid prototyping and encapsulates the high-performance implementations within Storm.

Availability. Storm's source code is available as open source and can be obtained along with additional information at http://www.stormchecker.org.

3 Architecture

Figure 1 depicts the architecture of Storm. Solid arrows indicate the flow of control and data, dashed lines represent a "uses" relationship. After the initial parsing step, it depends on the selected engine whether a model building step is performed: for all but the *exploration* and *abstraction-refinement* engines, it is necessary to build a full in-memory representation of the model upfront. Note that the available engines depend on the input format and that both PRISM and GSPN input can be either treated natively or transformed to JANI.

Solvers. Storm's infrastructure is built around the notion of a *solver*. For instance, solvers are available for sets of linear or Bellman equations (both using sparse matrices as well as MTBDDs), (mixed-integer) linear programming (MILP) and satisfiability modulo theories (SMT). Note that Storm does not support stochastic games as input models, yet, but solvers for them are available because they are used in the *abstraction-refinement* engine. Offering these interfaces has several key advantages. First, it provides easy and coherent access to the tasks commonly involved in probabilistic model checking. Secondly, it

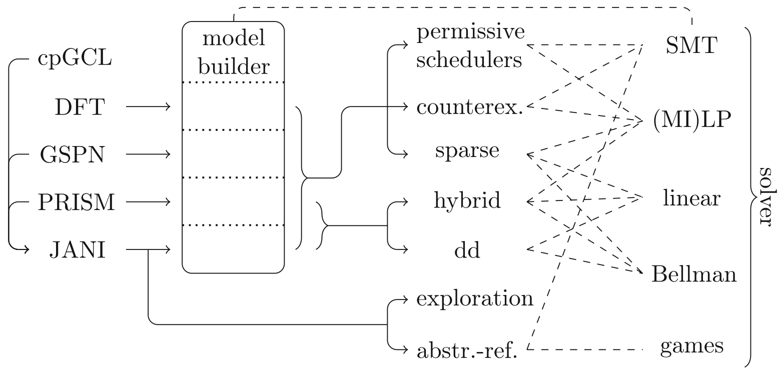

Fig. 1. Storm's architecture.

Table 1. Solvers offered by Storm.

Solver type	Available solvers
Linear equations (sparse)	Eigen [37], gmm++ [38], elim. [39], built-in
Linear equations (MTBDD)	CUDD [6], Sylvan [7]
Bellman equations (sparse)	Eigen, gmm++, built-in
Bellman equations (MTBDD)	CUDD, Sylvan
Stochastic games (sparse)	built-in
Stochastic games (MTBDD)	CUDD, Sylvan
(MI)LP	Gurobi [40], glpk [41]
SMT	Z3 [42], MathSAT [43], SMT-LIB

enables the use of dedicated state-of-the-art high-performance libraries for the task at hand. More specifically, as the performance characteristics of different backend solvers can vary drastically for the same input, this permits choosing the best solver for a given task. Licensing problems are avoided, because implementations can be easily enabled and disabled, depending on whether or not the particular license fits the requirements. Implementing new solver functionality is easy and can be done without knowledge about the global code base. Finally, it allows to embed new state-of-the-art solvers in the future. For each of those interfaces, several actual implementations exist. Table 1 gives an overview over the currently available implementations. Almost all engines and all other key modules make use of *solvers*. The most prominent example is the use of the equation solvers for answering standard verification queries. However, other modules use them too, e.g. model building (SMT), counterexample generation [33,34] (SMT, MILP) and permissive scheduler generation [44,45] (SMT, MILP).

4 Evaluation

Set-up. For the performance evaluation, we conducted experiments on a HP BL685C G7. Up to eight cores with 2.0 GHz and 8 GB of memory were available to the tools, but only Prism's garbage collection used more than one core at a time. We set a time-out of 1800 s.

Comparison with Prism. To assess Storm's performance on standard model-checking queries, we compare it with Prism on the Prism benchmark suite [46]. More specifically, we consider all DTMCs, CTMCs and MDPs (24 in total, and several instances per model) and all corresponding properties (82 in total). Note that we do not compare Storm with iscasMC as the latter one has a strong focus on more complex LTL properties.

Methodology. As both Prism and Storm offer several engines with different strengths and weaknesses, we choose the following comparison methodology. We compare engines that "match" in terms of the general approach. For example,

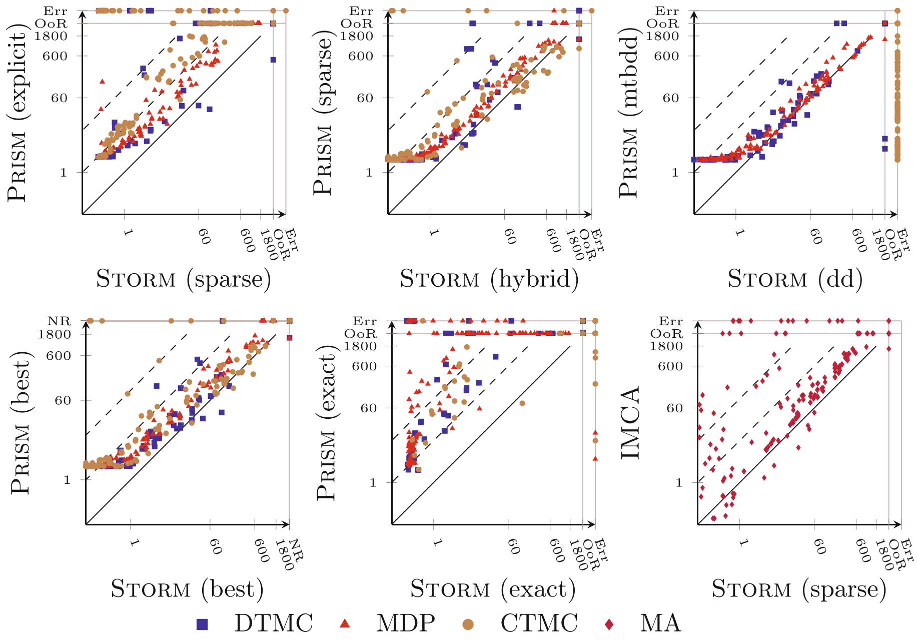

Fig. 2. Run-time comparison (seconds) of different engines/features.

Prism's *explicit* engine first builds the model in terms of a sparse matrix directly and then performs the model checking on this representation, which matches the approach of Storm's *sparse* engine. In the same manner, Prism's *sparse* engine is comparable to Storm's *hybrid* one and Prism's *mtbdd* engine corresponds to Storm's *dd* engine. Finally, we compare the run-times of Prism and Storm when selecting the best engine for each individual benchmark instance.

Results. Figure 2 (top-row) summarizes the results of the experiments in log-log scale. We plot the total time taken by the Storm engines versus the "matching" Prism engines. Data points above the main diagonal indicate that Storm solved the task faster. The two dashed lines indicate a speed-up of 10 and 100, respectively; "OoR" denotes memory- or time-outs, "Err" denotes that a tool was not able to complete the task for any other reason and "NR" stands for "no result".

Discussion. We observe that Storm is competitive on all compared engines. Even though the MTBDD-based engines are very similar and even use the same MTBDD library (CUDD), most of the time Storm is able to outperform Prism. Note that Storm currently does not support CTMCs in this engine. We observe a slightly clearer advantage of Storm' hybrid engine in comparison to Prism's sparse engine. Here, model building times tend to be similar, but most often the numerical solution is done more efficiently by Storm. However, for large

CTMC benchmarks, Prism tends to be faster than Storm. Storm's *sparse* engine consistently outperforms Prism due to both the time needed for model construction as well as solving times. For the overwhelming majority of verification tasks, Storm's best engine is faster than Prism's best engine. Storm solves 361 (out of 380) tasks, compared to 346 tasks Prism solves.

Exact Arithmetic. Figure 2 (bottom center) compares the *exact* modes of both tools. Storm outperforms Prism by up to three orders of magnitude.

Markov Automata. As Prism does not support the verification of MAs, we compare Storm with the only other tool capable of verifying MAs: IMCA [12]. We used the models provided by IMCA, results are depicted in Fig. 2 (bottom right). For most instances, Storm is significantly faster.

Acknowledgments. The authors would like to thank people that support(ed) the development of Storm over the years (in alphabetical order): Philipp Berger, Harold Bruintjes, Gereon Kremer, David Korzeniewski, and Tim Quatmann.

References

1. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47
2. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perform. Eval. **68**(2), 90–104 (2011)
3. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasMc: a web-based probabilistic model checker. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014. LNCS, vol. 8442, pp. 312–317. Springer, Cham (2014). doi:10.1007/978-3-319-06410-9_22
4. Larsen, K.G., Legay, A.: Statistical model checking: past, present, and future. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 3–15. Springer, Cham (2016). doi:10.1007/978-3-319-47166-2_1
5. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proceedings of LICS, pp. 342–351. IEEE CS (2010)
6. CUDD (2016). http://vlsi.colorado.edu/~fabio/CUDD/cudd.pdf
7. Dijk, T., Pol, J.: Sylvan: multi-core decision diagrams. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 677–691. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_60
8. Baier, C., Klein, J., Klüppelholz, S., Märcker, S.: Computing conditional probabilities in markovian models efficiently. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 515–530. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54862-8_43
9. de Alfaro, L.: How to specify and verify the long-run average behavior of probabilistic systems. In: Proceedings of LICS, pp. 454–465. IEEE CS (1998)
10. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: a model checker for stochastic multi-player games. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 185–191. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36742-7_13

11. Katoen, J.P.: The probabilistic model checking landscape. In: Proceedings of LICS, pp. 31–46. ACM (2016)
12. Guck, D., Timmer, M., Hatefi, H., Ruijters, E., Stoelinga, M.: Modelling and analysis of markov reward automata. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 168–184. Springer, Cham (2014). doi:10.1007/978-3-319-11936-6_13
13. Guck, D., Hatefi, H., Hermanns, H., Katoen, J., Timmer, M.: Analysis of timed and long-run objectives for Markov automata. LMCS **10**(3), 1–29 (2014)
14. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 151–168. Springer, Heidelberg (2017). doi:10.1007/978-3-662-54580-5_9
15. Amparore, E.G., Beccuti, M., Donatelli, S.: (Stochastic) model checking in great-SPN. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 354–363. Springer, Cham (2014). doi:10.1007/978-3-319-07734-5_19
16. Schwarick, M., Heiner, M., Rohr, C.: MARCIE - model checking and reachability analysis done efficiently. In: Proceedings of QEST, pp. 91–100. IEEE CS (2011)
17. Eisentraut, C., Hermanns, H., Katoen, J.-P., Zhang, L.: A semantics for every GSPN. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 90–109. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38697-8_6
18. Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans. Reliab. **41**(3), 363–377 (1992)
19. Boudali, H., Crouzen, P., Stoelinga, M.I.A.: A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE Trans. Secure Distr. Comput. **7**(2), 128–143 (2010)
20. Volk, M., Junges, S., Katoen, J.-P.: Advancing dynamic fault tree analysis - get succinct state spaces fast and synthesise failure rates. In: Skavhaug, A., Guiochet, J., Bitsch, F. (eds.) SAFECOMP 2016. LNCS, vol. 9922, pp. 253–265. Springer, Cham (2016). doi:10.1007/978-3-319-45477-1_20
21. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Heidelberg (2005). doi:10.1007/b138392
22. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani.K.: Probabilistic programming. In: FOSE, pp. 167–181. ACM (2014)
23. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput. **6**(5), 512–535 (1994)
24. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. **29**(6), 524–541 (2003)
25. Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., Parker, D., Ujma, M.: Verification of markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). doi:10.1007/978-3-319-11936-6_8
26. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.-P., Ábrahám, E.: PROPhESY: A PRObabilistic ParamEter SYnthesis Tool. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 214–231. Springer, Cham (2015). doi:10.1007/978-3-319-21690-4_13
27. Quatmann, T., Dehnert, C., Jansen, N., Junges, S., Katoen, J.-P.: Parameter synthesis for markov models: faster than ever. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 50–67. Springer, Cham (2016). doi:10.1007/978-3-319-46520-3_4

28. CArL Website: http://smtrat.github.io/carl/ (2015)
29. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. STTT **13**(1), 3–19 (2010)
30. Haddad, S., Monmege, B.: Reachability in MDPs: refining convergence of value iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) RP 2014. LNCS, vol. 8762, pp. 125–137. Springer, Cham (2014). doi:10.1007/978-3-319-11439-2_10
31. Wimmer, R., Becker, B.: Correctness issues of symbolic bisimulation computation-for Markov chains. In: Müller-Clostermann, B., Echtle, K., Rathgeb, E.P. (eds.) MMB & DFT 2010. LNCS, vol. 5987, pp. 287–301. Springer, Berlin (2010)
32. Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.-P., Wimmer, R.: Counterexample generation for discrete-time Markov models: an introductory survey. In: Bernardo, M., Damiani, F., Hähnle, R., Johnsen, E.B., Schaefer, I. (eds.) SFM 2014. LNCS, vol. 8483, pp. 65–121. Springer, Cham (2014). doi:10.1007/978-3-319-07317-0_3
33. Wimmer, R., Jansen, N., Vorpahl, A., Ábrahám, E., Katoen, J.P., Becker, B.: High-level counterexamples for probabilistic automata. LMCS **11**(1), 1–23 (2015)
34. Dehnert, C., Jansen, N., Wimmer, R., Ábrahám, E., Katoen, J.-P.: Fast debugging of PRISM models. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 146–162. Springer, Cham (2014). doi:10.1007/978-3-319-11936-6_11
35. Aljazzar, H., Leitner-Fischer, F., Leue, S., Simeonov, D.: DiPro - a tool for probabilistic counterexample generation. In: Groce, A., Musuvathi, M. (eds.) SPIN 2011. LNCS, vol. 6823, pp. 183–187. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22306-8_13
36. Jansen, N., Ábrahám, E., Volk, M., Wimmer, R., Katoen, J.P., Becker, B.: The COMICS tool - Computing minimal counterexamples for DTMCs. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012, vol. 7561, pp. 349–353. Springer, Heidelberg (2012)
37. Guennebaud, G., Jacob, B., et al.: Eigen v3. (2017). http://eigen.tuxfamily.org
38. GMM++ Website: (2015). http://getfem.org/gmm/index.html
39. Daws, C.: Symbolic and parametric model checking of discrete-time Markov chains. In: Liu, Z., Araki, K. (eds.) ICTAC 2004. LNCS, vol. 3407, pp. 280–294. Springer, Heidelberg (2005). doi:10.1007/978-3-540-31862-0_21
40. Gurobi Optimization Inc.: Gurobi optimizer reference manual (2015). http://www.gurobi.com
41. GNU project: Linear programming kit, version 4.6 (2016). http://www.gnu.org/software/glpk/glpk.html
42. Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78800-3_24
43. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The mathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36742-7_7
44. Dräger, K., Forejt, V., Kwiatkowska, M., Parker, D., Ujma, M.: Permissive controller synthesis for probabilistic systems. LMCS **11**(2), 1–34 (2015)
45. Junges, S., Jansen, N., Dehnert, C., Topcu, U., Katoen, J.-P.: Safety-constrained reinforcement learning for MDPs. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 130–146. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_8
46. Kwiatkowska, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: Proceedings of QEST, pp. 203–204. IEEE CS (2012)