MARCIE – Model Checking and Reachability Analysis Done Efficiently

Monika Heiner, Christian Rohr, and Martin Schwarick

Computer Science Institute, Brandenburg University of Technology Cottbus
Postbox 10 13 44, 03013 Cottbus, Germany
marcie@informatik.tu-cottbus.de
http://www-dssz.informatik.tu-cottbus.de

Abstract. MARCIE is a tool for the analysis of generalized stochastic Petri nets which can be augmented by rewards. The supported analysis methods range from qualitative and quantitative standard properties to model checking of established temporal logics. MARCIE's analysis engines for bounded Petri net models are based on Interval Decision Diagrams. They are complemented by simulative and approximative engines to allow for quantitative reasoning on unbounded models. Most of the quantitative analyses benefit from a multi-threaded implementation. This paper gives an overview on MARCIE's functionality and architecture and reports on the recently added feature of CSRL and PLTLc model checking.

Keywords: generalized stochastic Petri nets, model checking, simulation.

1 Objectives

Generalized stochastic Petri nets ($\mathcal{GSPN}$) are a widely used formalism in application fields as performance evaluation of technical systems, or synthetic and systems biology. Augmented with rewards they permit intuitive modeling and powerful analyses of inherently concurrent stochastic systems. As their semantics can be mapped to Continuous-time Markov chains (CTMC), a wide range of quantitative analysis methods up to probabilistic model checking is available.

There are several tools supporting different kinds of efficient CTMC analysis, e.g., by applying symbolic techniques or discrete event simulation. However, their use is often restricted by specific constraints. There are tools which support only the analysis of bounded models, even if discrete event simulation is used. Some tools enable the augmentation of CTMC models by rewards, but do not provide model checking of related temporal logics as the Continuous Stochastic Reward Logic (CSRL). Most tools do not support multi-threading, although this could drastically decrease the runtime of the analyses. Often tools demand for skilled users with sophisticated insights how to specify the model best and how to set the most appropriate tool parameters to configure internal data structures and algorithms. Dedicated simulation tools generally support only the simple generation of traces, although more advanced evaluation would be desirable.

MARCIE overcomes these problems and integrates all features into one tool dedicated to the analysis of $\mathcal{GSPN}$ extended by rewards.

2 Functionality

In this section we give an overview of MARCIE's functionality with special focus on its latest extensions. The numbers given in round brackets refer to Fig. 3.

2.1 Net Classes

Basically, MARCIE analyses $\mathcal{GSPN}$ augmented by rewards. However, MARCIE's internal net representation (1) distinguishes the following net classes as the range of supported analysis capabilities depends on them. The core build place/transition Petri nets extended by read and inhibitor arcs. As they do not contain any time information, we call them qualitative Petri nets ($\mathcal{QPN}$). Fig. 1 shows a very simple $\mathcal{QPN}$ for a producer and a consumer connected by an N-bounded buffer.

We speak of stochastic Petri nets ($\mathcal{SPN}$) if all transitions carry further information in terms of firing rates which govern exponentially distributed waiting times. We obtain generalized stochastic Petri nets ($\mathcal{GSPN}$) if additionally immediate transitions (no waiting time) are allowed. $\mathcal{SPN}$ and $\mathcal{GSPN}$ can be enriched by rewards which can be associated with states (rate rewards) or transitions (impulse rewards). They are specified by reward definitions in a style similar to [19]. A reward definition consists of a set of reward items – state reward items and transition reward items. A reward item specifies a set of states by means of a guard and a possibly state-dependent reward function defining the actual reward value. We call an $\mathcal{SPN}$ augmented with rate rewards a stochastic reward net ($\mathcal{SRN}$), and a $\mathcal{GSPN}$ augmented with arbitrary rewards a generalized stochastic reward net ($\mathcal{GSRN}$). Rewards do not change the state space, but prepare the ground for more convenient and powerful analyses.

2.2 Engines

IDD Engine. (3) A cornerstone of MARCIE is its efficient implementation of Interval Decision Diagrams (IDD) [29]. Three different state space generation algorithms (4) were implemented upon this. The first one is the common Breadth-First Search (BFS) algorithm. All transitions fire in one iteration once

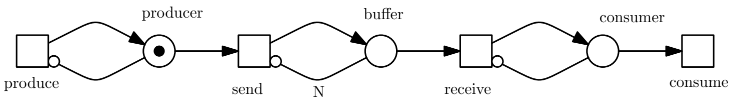

Fig. 1. A producer/consumer system with a buffer of size N

according to the transition order before adding the new states to the state space. The second algorithm is called Transition chaining. It works like BFS, but the state space is updated after the firing of each single transition. The Saturation algorithm is the last one. Transitions fire in conformance with the decision diagram. A transition is saturated if its firing does not add new states to the current state space. It should be noted that the efficiency of Chaining and Saturation depends on the transition order.

Having the state space, MARCIE permits to find dead states and to decide reversibility and liveness of transitions, which involves a symbolic decomposition of the state space into strongly connected components.

The implemented IDD engine enjoys several features to address efficiency issues, as for instance the concept of shared DDs, fast detection of isomorphic sub-diagrams by use of a unique table, and efficient operation caches; see [29] for a detailed discussion. Furthermore, the engine offers dedicated operations for forward and backward firing of Petri net transitions. It is well known that the variable order used for constructing a DD may have a strong influence on its size in terms of number of nodes, and thus on the performance of all related operations. To find an optimal variable order is an NP-hard problem. MARCIE's heuristic to pre-compute static variable orders has a simple underlying idea. It examines the structure of the given Petri net and arranges dependent places close to each other. Two places are dependent if there is a transition which affects both places [20]. MARCIE's order generator (2) offers seven options to control the generation of the place order, and six options to influence the transition order.

Symbolic CTMC Engine. (6) MARCIE provides exact quantitative analyses based on the computation of various probability distributions. Its symbolic engine is responsible for a compact representation of the real-valued state transition relation, a matrix, and some efficient numerical operations which are basically matrix-vector multiplications. The engine combines IDD-based state space encoding and "on-the-fly" generation of the state transitions which are labeled with the firing rates of stochastic transitions or the firing probabilities of immediate transitions. The computation vectors and the entries of the matrix diagonals are explicitly stored in arrays of double precision type and represent the actual limitation of applicability as their size equals the number of reachable states. On current workstations this still allows us to consider models with more than 10^9 states.

MARCIE computes the instantaneous and cumulative transient probability distribution and the steady state probability distribution for $\mathcal{GSPN}$, and the joint distribution of time and accumulated reward, a special case of Meyer's performability, for $\mathcal{SRN}$. For the latter, MARCIE makes use of Markovian approximation [5] and transforms the $\mathcal{SRN}$ description into an $\mathcal{SPN}$. For more details we refer to [26,28,25].

Approximative CTMC Engine. (7) To overcome the problem of an unmanageable state space size, MARCIE provides an approximative CTMC engine. This engine dynamically restricts the number of states. The basic idea is to combine a breadth-first variant of the state space construction with a transient analysis using uniformization [7]. During construction, all explored states having a probability below a specified threshold will be removed from the current state space. The default threshold is 10^{-11}, but can be changed by the user. Thus, only a finite subset of a possibly infinite state space will be considered. Contrary to the symbolic CTMC engine, it uses an explicit state space representation.

The approximative engine can handle $\mathcal{SPN}$ and $\mathcal{GSPN}$. The results can be exported in comma separated values (CSV) format for plotting or further analysis.

Stochastic Simulation Engine. (8) If the approximative numerical analysis exceeds the available memory, the method of choice has to be simulation. MARCIE provides two stochastic simulation algorithms – the direct method introduced by Gillespie [12], and the next reaction method introduced by Gibson & Bruck [11]. Stochastic simulation generates paths of finite length of a possibly infinite CTMC. In contrast to numerical analysis, simulation has a constant memory consumption, because only the current state is hold in memory.

Generally it is necessary to perform a sufficient number of simulation runs due to the variance of the stochastic behavior. We choose the confidence interval method as described in [22] to determine the required number of simulation runs. The user can specify the confidence interval by defining the confidence level, usually 95% or 99%, and the estimated accuracy, e.g., 10^{-3} or 10^{-4}. MARCIE calculates the required number of simulation runs to achieve this confidence interval. Alternatively, the user can set the number of simulation runs manually.

The individual simulation runs are done independently from each other. Thus, it is not challenging to parallelize stochastic simulations. MARCIE provides a multi-threaded simulation engine. Stochastic simulation results can be exported in CSV format for visualization, further analyses or documentation purposes.

This engine can not only treat $\mathcal{GSPN}$, but also $\mathcal{XSPN}$; see [14] for details.

2.3 Model Checkers

CTL (5) The Computation Tree Logic (CTL) [4] is a widely used branching time logic. It permits to specify properties over states and paths of a labeled transition system (LTS), the Kripke structure. Path quantifiers specify whether path formulas, which can be written by means of temporal operators, should be fulfilled on all paths or at least on one path starting in some state. One can interpret the reachability graph of a Petri net as a Kripke structure and thus apply CTL model checking algorithms. MARCIE supports symbolic CTL model checking for $\mathcal{QPN}$ based on its IDD engine, for details see [29].

CSL (9) The Continuous Stochastic Logic (CSL) [1] is the stochastic counterpart to CTL. The path quantifiers of CTL are replaced by the probability operator $\mathcal{P}$. The usual temporal operators are decorated with time intervals. In [2], CSL has been extended by the steady state operator $\mathcal{S}$ and by time-unbounded versions of the temporal operators. The basic CSL model checking algorithm is similar to the one for CTL, but additionally requires to compute steady state and transient probabilities. MARCIE supports CSL model checking of $\mathcal{GSPN}$ based on its exact symbolic engine. Unnested formulas can also be checked with the simulative engine. If CSL formulas are unnested and time bounded, it is also possible to use the approximative engine.

Reward Measures. In addition to CSL, special operators for the computation of expectations of instantaneous and cumulative state and transition rewards have been introduced [19]. MARCIE's symbolic CTMC and simulation engines support these measures, too. A genuine extension of CSL by rewards is presented in the next section.

2.4 New Functionalities

We now discuss in more details the latest features integrated into MARCIE.

Abstract Net Definition Language. MARCIE reads Petri net models defined in the Abstract Net Description Language (ANDL) [24]. ANDL is a lightweight and human readable description language with semantical and syntactical similarities to a guarded command language. However, contrary to, e.g., the PRISM language [16], ANDL enjoys an explicit Petri net semantics and defines additional transition types and rate function patterns.

ANDL complements model specification with bloated XML-based languages like PNML [15] (which MARCIE does support for $\mathcal{QPN}$) and serves as exchange format between MARCIE and its friend Snoopy [13], which can be used to construct $\mathcal{QPN}$, $\mathcal{SPN}$, $\mathcal{GSPN}$, and $\mathcal{XSPN}$.

The ANDL specification of the running $\mathcal{SPN}$ example in Fig. 1 and an additional reward definition are given in Fig. 2.

```
spn [procon] {
constants:
  int       cap; // buffer capacity
  double p_rate; // production rate
  double c_rate; // consumption rate
places:
  producer = 1;
  consumer = 0;
  buffer   = 0;
transitions:
  receive : [consumer < 1] : [consumer + 1] & [buffer - 1]   : 1;
  send    : [buffer < cap] : [buffer + 1]   & [producer - 1] : 1;
  produce : [producer < 1] : [producer + 1]                  : p_rate;
  consume :                : [consumer - 1]                  : c_rate;
}
```

```
rewards [ r2r ] {
  /* states where the
 consumer is ready to
 receive, have
 a reward of 1
  */
  consumer > 0 : 1 ;
}
```

Fig. 2. The ANDL specification of a scalable $\mathcal{SPN}$ for the producer/consumer model. The buffer capacity and the rates of item production and consumption are defined by constants. The $\mathcal{SPN}$ is augmented by the separate reward definition $r2r$ which associates a reward of one to states where the consumer can receive an item.

Continuous Stochastic Reward Logic. (9) As a new feature MARCIE supports the Continuous Stochastic Reward Logic (CSRL) [5]. CSRL is a superset of CSL and augments the temporal operators with additional reward intervals. Re our example and the reward definition *r2r*, we can ask for instance for the probability to reach within t time units a state where the buffer is full. We may also expect that the consumer is ready to receive for at least half of the time period. Thus we specify the reward interval $[t/2, t]$, and obtain the CSRL formula

$$\mathcal{P}^{[r2r]}_{=?}[\mathbf{F}^{[0,t]}_{[t/2,t]}\, \mathit{buffer} = \mathit{cap}]\ .$$

The extent of MARCIE's support of CSRL model checking depends on the engines. The simulative engine supports unnested CSRL formulas for $\mathcal{GSRN}$. The symbolic engine currently supports full CSRL for $\mathcal{SRN}$. It uses Markovian approximation to transform the $\mathcal{SRN}$ into an $\mathcal{SPN}$ and maps the CSRL formula to a CSL formula with the reward bounds encoded as state properties. For more details we refer to [25].

Probabilistic Linear-Time Temporal Logic. (10) Besides the branching time temporal logics CTL and CSRL, MARCIE supports the Probabilistic Linear-time Temporal Logic with numerical constraints (PLTLc) [9]. In PLTLc, one can encode formulas on the future of paths through the state space of the model under study. So, it is quite obvious to deploy stochastic simulation to verify PLTLc formulas, because that is what stochastic simulation does – to compute paths through the model's state space. In contrast to branching time logics, PLTLc can not compute the probabilities of a given state, because it operates on paths, not on distributions. Therefore, it is impossible to nest the probability operator $\mathcal{P}$ in PLTLc. We recently extended PLTLc to check time-unbounded temporal operators, see [21].

Unlike symbolic model checking, simulative model checking computes a confidence interval of the expected probability rather than the concrete value, i.e., simulative model checking calculates probabilities up to a certain accuracy, which is the width of the confidence interval. Besides the standard return value of the $\mathcal{P}$ operator, the PLTLc model checker yields the expected probabilities of the domains of free variables (denoted with \$) for which the formula holds. Back to our example, the maximum number of tokens on the place *buffer* up to time point t and their probabilities can be determined with the following formula

$$\mathcal{P}_{=?}[\mathbf{F}^{[0,t]}\, \mathit{buffer} > \$x]\ .$$

The PLTLc model checker works with any exact stochastic simulation algorithm, e.g., the direct method and the next reaction method, both are implemented in MARCIE. The model checking procedure is done on-the-fly, i.e., the formula is checked while the trace is generated. Furthermore, pre-computed integer traces given in CSV format can be verified.

3 Architecture

In the following we present the basic tool architecture, which is depicted in Fig. 3. We sketch the main ideas which we took into consideration during the development of MARCIE's components to achieve highly efficient analysis capabilities.

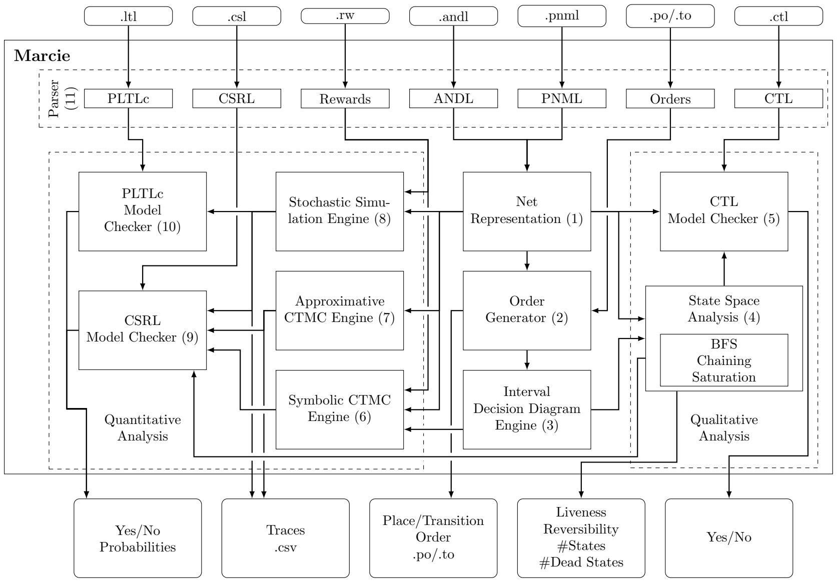

Fig. 3. MARCIE's architecture and its eleven components

MARCIE is entirely written in the programming language C++ with intensive use of template programming. It builds on the GNU multiple precision library and several parts of the boost library.

Currently, all parsers (11) for the actual Petri nets, CTL and CSRL formulas, reward definitions, as well as place and transition orders are built on the aging lexical analyzer Flex and parser generator Bison. We are about to move to the lightweight parser generator Spirit from the boost library, as it has already been done for the PLTLc parser. See [24] for detailed input syntax specifications.

4 Comparison with Other Tools

One could create a long list of tools, supporting the analysis of CTMCs and related formalisms and, thus, indirectly stochastic Petri nets as well. Due to the lack of space we confine ourselves to a very brief shortlist. Table 1 compares the main features. An elaborated comparison of CSL model checkers can be found in [17], comprising explicit, symbolic and simulative engines.

The probabilistic model checker PRISM [16] supports analysis of CTMCs, DTMCs and Markov Decision Processes by means of CSL, PCTL, and LTL, and exploits Multi-Terminal BDDs. It also permits the computation of expectations of reward measures and defines its own model description language in the style of guarded commands which can be easily used to specify bounded $\mathcal{SPN}$. An extensive performance comparison of MARCIE and PRISM was done in [27].

Table 1. Feature comparison of MARCIE and related tools. Entries in round brackets suggest a look in the tool's manual for further details.

		MARCIE	Prism	MRMC	Smart	Möbius
Qualitative	Net classes	$\mathcal{XPN}$	$\mathcal{SPN}$	—	$\mathcal{GSPN}$	$(\mathcal{XSPN})$
	State space generation	BFS, Chaining, Saturation	BFS	—	BFS, Chaining, Saturation	BFS
	Orders	heuristics	plain	—	plain	plain
	Standard properties	✓	(✓)	—	(✓)	—
	Model checker	CTL	—	—	CTL	—
Numerical	Net classes	$\mathcal{GSPN}$	$\mathcal{SPN}$	$(\mathcal{SPN})$	$\mathcal{GSPN}$	$(\mathcal{XSPN})$
	Transient	✓	✓	✓	✓	✓
	Steady state	✓	✓	✓	✓	✓
	Rewards	✓	✓	✓	—	✓
	Model checker	CSRL	CSL	(CSRL)	—	—
	Multi threading	(✓)	—	—	—	—
Simulative	Net classes	$\mathcal{XSPN}$	$\mathcal{SPN}$	$(\mathcal{SPN})$	—	$\mathcal{XSPN}$
	Transient	✓	✓	✓	—	✓
	Steady state	✓	—	✓	—	✓
	Rewards	✓	✓	—	—	✓
	Model checker	(CSRL), PLTLc	(CSL)	(CSL)	—	—
	Multi threading	✓	—	—	—	✓

See also [26], where we compared PRISM and MARCIE's predecessor *IDD-MC* concerning transient analysis of biological models.

Another CSL model checker is the Markov Reward Model Checker (MRMC) [18]. It also offers analysis capabilities for CTMCs and related formalisms based on temporal logics. Besides MARCIE, it is the only tool supporting model checking of CSRL formulas. MRMC uses sparse representations to encode state space and matrices. Special features are bisimulation-based state space reduction and simulative steady state detection. MRMC provides simulative model checking of unnested CSL. It requires third party tools to generate the actual Markov model, which becomes prohibitive with increasing file size.

A further popular tool is SMART [3]. It offers qualitative and quantitative analysis of $\mathcal{GSPN}$ with marking-dependent arcs and defines its own model description language. SMART supports CTL, but not CSL model checking, in spite of its ability to compute transient and steady state probabilities. The user can choose between various explicit and symbolic storage strategies for the state space (e.g., AVL trees, Multi-valued Decision Diagrams (MDDs)) and for the rate matrix (e.g., Kronecker representations, Multi-Terminal MDDs, Edge-Valued MDDs, Matrix Diagrams (MxD)). However, some of these storage strategies force the user to obey some modeling restrictions. The use of MDDs, which, e.g., allow for saturation-based reachability analysis, requires to specify a suitable place partition.

A tool which offers explicit, symbolic (MDD, MxD, MTBDD, Lumping) and multi-threaded simulative analysis is Möbius [6].

None of these tools supports the numerical approximation algorithm for computing transient solutions of stochastic models as implemented in MARCIE. To the best of our knowledge, the tool Sabre [8] is besides MARCIE the only publicly available implementation. But in contrast to MARCIE, Sabre does not include any model checking capabilities.

The Monte Carlo Model Checker MC2 [9] validates PLTLc formulas, but does not include any simulation engine. MC2 works offline by reading a set of sampled trajectories, generated by any simulation or ODE solver software.

Furthermore, there exist a great variety of dedicated simulation tools, e.g., StochKit2 [23], but all lack advanced analysis methods.

5 Installation

MARCIE is available for non-commercial use. We provide statically linked, self-contained binaries for Mac OS X, and Linux. The tool, its manual and a benchmark suite can be found on our website `http://www-dssz.informatik.tu-cottbus.de/marcie.html`. Currently, MARCIE itself comes with a textual user interface. Tool options and input files can also be specified by a generic Graphical User Interface (GUI), written in Java, which can be easily configured by means of an XML description. The GUI is part of our Petri net analyzer Charlie [10].

References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model checking continuous-time Markov chains. ACM Trans. on Computational Logic 1(1), 162–170 (2000)
2. Baier, C., Katoen, J.-P., Hermanns, H.: Approximate Symbolic Model Checking of Continuous-Time Markov Chains (Extended Abstract). In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 146–161. Springer, Heidelberg (1999)
3. Ciardo, G., Jones, R.L., Miner, A.S., Siminiceanu, R.: Logical and stochastic modeling with SMART. Perform. Eval. 63(1), 578–608 (2006)
4. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite state concurrent systems using temporal logic specifications. ACM TOPLAS 8(2), 244–263 (1986)
5. Cloth, L., Haverkort, B.R.: Five performability algorithms. A comparison. In: MAM 2006, pp. 39–54. Boson Books, Raleigh (2006)
6. Courtney, T., Gaonkar, S., Keefe, K., Rozier, E., Sanders, W.H.: Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In: DSN, pp. 353–358 (2009)

7. Didier, F., Henzinger, T.A., Mateescu, M., Wolf, V.: Fast Adaptive Uniformization for the Chemical Master Equation. In: HIBI, pp. 118–127. IEEE Comp. Soc. (2009)
8. Didier, F., Henzinger, T.A., Mateescu, M., Wolf, V.: Sabre: A tool for stochastic analysis of biochemical reaction networks. In: Proc. QEST, pp. 217–218. IEEE Computer Society (2010)
9. Donaldson, R., Gilbert, D.: A Monte Carlo model checker for probabilistic LTL with numerical constraints. Tech. rep., University of Glasgow, Dep. of CS (2008)
10. Franzke, A.: Charlie 2.0 - a multi-threaded Petri net analyzer. Diploma Thesis (in German), BTU Cottbus, Dep. of CS (2009)
11. Gibson, M.A., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. J. Phys. Chem. A 104, 1876–1889 (2000)
12. Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. 81(25), 2340–2361 (1977)
13. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy – a unifying Petri net tool. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 398–407. Springer, Heidelberg (2012)
14. Heiner, M., Rohr, C., Schwarick, M., Streif, S.: A comparative study of stochastic analysis techniques. In: Proc. CMSB 2010, pp. 96–106. ACM (2010)
15. Hillah, L., Kindler, E., Kordon, F., Petrucci, L., Trèves, N.: A primer on the Petri Net Markup Language and ISO/IEC 15909-2. PNNL 76, 9–28 (2009)
16. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
17. Jansen, D.N., Katoen, J.-P., Oldenkamp, M., Stoelinga, M., Zapreev, I.: How fast and fat is your probabilistic model checker? An experimental performance comparison. In: Yorav, K. (ed.) HVC 2007. LNCS, vol. 4899, pp. 69–85. Springer, Heidelberg (2008)
18. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Performance Evaluation 68(2), 90–104 (2011)
19. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007)
20. Noack, A.: A ZBDD Package for Efficient Model Checking of Petri Nets (in German). Tech. rep., BTU Cottbus, Dep. of CS (1999)
21. Rohr, C.: Simulative Model Checking of Steady-State and Time-Unbounded Temporal Operators. In: Proc. BioPPN 2012, CEUR Workshop Proceedings, vol. 852, pp. 62–75. CEUR-WS.org (June 2012)
22. Sandmann, W., Maier, C.: On the statistical accuracy of stochastic simulation algorithms implemented in Dizzy. In: Proc. WCSB 2008, pp. 153–156 (2008)
23. Sanft, K.R., Wu, S., Roh, M., Fu, J., Lim, R.K., Petzold, L.R.: Stochkit2: software for discrete stochastic simulation of biochemical systems with events. Bioinformatics 27(17), 2457–2458 (2011)
24. Schwarick, M.: Manual: Marcie - An analysis tool for Generalized Stochastic Petri nets. BTU Cottbus, Dep. of CS (2010)
25. Schwarick, M.: Symbolic model checking of stochastic reward nets. In: Proc. CS&P 2012, CEUR Workshop Proceedings, vol. 928, pp. 343–357. CEUR-WS.org (2012)

26. Schwarick, M., Heiner, M.: CSL model checking of biochemical networks with interval decision diagrams. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 296–312. Springer, Heidelberg (2009)
27. Schwarick, M., Rohr, C., Heiner, M.: Marcie - model checking and reachability analysis done efficiently. In: Proc. QEST 2011, pp. 91–100. IEEE CS Press (2011)
28. Schwarick, M., Tovchigrechko, A.: IDD-based model validation of biochemical networks. TCS 412, 2884–2908 (2010)
29. Tovchigrechko, A.: Model Checking Using Interval Decision Diagrams. Ph.D. thesis, BTU Cottbus, Dep. of CS (2008)