MARCIE - Model checking And Reachability analysis done effiCIEntly

Martin Schwarick and Monika Heiner
Computer Science Department
BTU Cottbus, Germany
Email: ms@informatik.tu-cottbus.de
monika.heiner@informatik.tu-cottbus.de

Christian Rohr
Magdeburg Centre for Systems Biology (MaCS)
Magdeburg, Germany
Email: rohrch@tu-cottbus.de

***Abstract*—MARCIE is a multi-threaded tool for the analysis of Generalized Stochastic Petri Nets. Its capabilities range from standard properties of qualitative Petri nets to CTL and CSL model checking, recently extended by rewards. The core of MARCIE builds upon Interval Decision Diagrams for the symbolic representation of marking sets of bounded Petri nets (finite state space) and on-the-fly matrix computation for numerical analysis. Approximative engines supporting fast adaptive uniformization and Gillespie simulation open the door to quantitative reasoning on unbounded Petri nets (infinite state space). This paper presents MARCIE's architecture and its most important distinguishing features. Extensive computational experiments demonstrate MARCIE's strength in comparison with related tools.**

I. Motivation

Stochastic Petri Nets (SPN) [1] and Generalized Stochastic Petri Nets (GSPN) [2] are frequently used formalisms to model and analyze concurrent systems. In the last three decades they have specifically been used in technical application domains to investigate, e.g., communication protocols or manufacturing systems [3, 4]. An SPN associates firing rates to transitions of the underlying qualitative Petri net and can be seen as a high-level description of a Continuous-Time Markov Chain (CTMC). The SPN class can be generalized to GSPN by adding so-called immediate transitions, which immediately fire when becoming enabled. States with enabled immediate transitions have a zero sojourn time and are called vanishing states. The GSPN's semantics is not a CTMC anymore, but can still be reduced to it.

The structural description of (G)SPNs and their Markovian semantics offer the modeler a large variety of qualitative and quantitative analysis methods which have been implemented in numerous software tools. Many methods consider the set of reachable states and thus suffer from the state space explosion problem. Symbolic approaches relying on Decision Diagrams (DD), e.g., Binary Decision Diagrams (BDD) [5] or related generalizations, permit qualitative reasoning over reachable states in new dimensions (beyond 10^{28} states [6]). This includes also model checking of, e.g., CTL formulas.

However, quantitative analyses of the induced CTMC require efficient processing of the state transitions which are weighted with real values. Kronecker algebra [4] or generalizations of BDDs, such as Multi-Terminal Decision Diagrams (MTDD), allow for the analysis of models of remarkable state space size, but in general need some pre-conditions to be fulfilled. Often the models must exhibit a modular structure (Kronecker), or the domain of the real-valued transition rates must be relatively small (MTDD). BDD-based techniques require prior knowledge of the domain of the model variables (boundedness degree of the places) and only work for a moderate domain size. Generally, the success of DD-based approaches for qualitative and quantitative analyses relies on a suitable variable order, which may be implicitly given by a modular structure.

Recently, (G)SPNs have also been used in Systems Biology [7, 8, 9, 10] as they allow an intuitive modeling of biochemical reaction networks. Tokens on places may represent molecules or concentrations levels, and the transitions' firing rates define biochemical kinetics. Often, such models do not exhibit a modular structure and the usually state-dependent firing rates generate a large variety of individual rate values. Moreover, the state space explosion is aggravated by the combinatorial distribution of a large number of tokens in a relatively small-sized model, which usually results in a high boundedness degree of all places. Even worse, biological networks often exhibit an unbounded state space, which restricts the analysis to approximative and simulative techniques.

These aspects were our main motivations to develop an efficient and easy to use analysis tool, tailored to the particularities of biochemical reaction networks. We address high boundedness degrees, without requiring its prior knowledge, by a symbolic state space representation based on Interval Decision Diagrams (IDDs) [11], and the possibly huge number of different rate values by a symbolic "on-the-fly" [12] engine which is used for numerical CTMC analysis. Our automatic computation of suitable variable orders exploits the net structure without relying on a predefined modularization of the model. Complementary, we provide efficient approximative and simulative analysis methods in order to support unbounded nets. Many analysis algorithms are parallelized.

The resulting tool is MARCIE, which has been developed with these issues in mind. It outperforms established tools for many benchmarks [6, 12, 13, 14]. This paper gives for the first time an overview of MARCIE's architecture and its most important distinguishing features.

II. The tool

In the following we assume basic knowledge of the Petri net formalism. A general introduction can be found in [2], a tutorial tailored to Systems Biology in [8].

A. Overview

MARCIE is a tool for GSPN analysis, supporting qualitative and quantitative analyses including model checking facilities. Particular features are symbolic state space analysis including efficient saturation-based state space generation, evaluation of standard Petri net properties as well as CTL model checking. Further it offers symbolic CSL model checking and permits to compute expectations for rewards which can be added to the core GSPN. Most of MARCIE's features are realized on top of an IDD implementation [11]. IDDs are used to efficiently encode interval logic functions representing marking sets of bounded Petri nets. Thus, MARCIE falls into the category of symbolic analysis tools as SMART [15] and PRISM [16]. However, recently it has been extended by approximative and simulative engines, which work explicitly, to support also stochastic analysis of unbounded nets.

MARCIE is entirely written in C++, and uses the libraries GMP, pthreads $flex/bison$ and $boost$. It comprises about $40,000$ source lines of code. MARCIE is available for non-commercial use; we provide statically linked binaries for Mac OS X and Linux. The tool, the manual and a benchmark suite can be found on our website *http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie*. Currently, MARCIE itself comes with a textual user interface. Options and input files can also be specified by a generic Graphical User Interface (GUI) (Fig. 1), written in Java, which can be easily configured by means of an XML description. The GUI is part of our Petri net analyser Charlie [17].

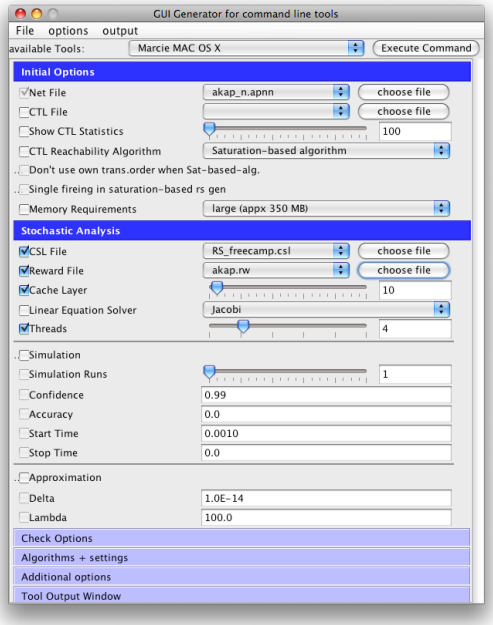

Fig. 1. Generic Graphical User Interface.

B. Implementation

In the following we present the basic tool architecture, which is depicted in Fig. 2. We will sketch the main ideas behind the single components which have been considered during MARCIE's development to achieve highly efficient analysis capabilities. In this paper we will concentrate on quantitative analysis aspects because this is the most challenging part.

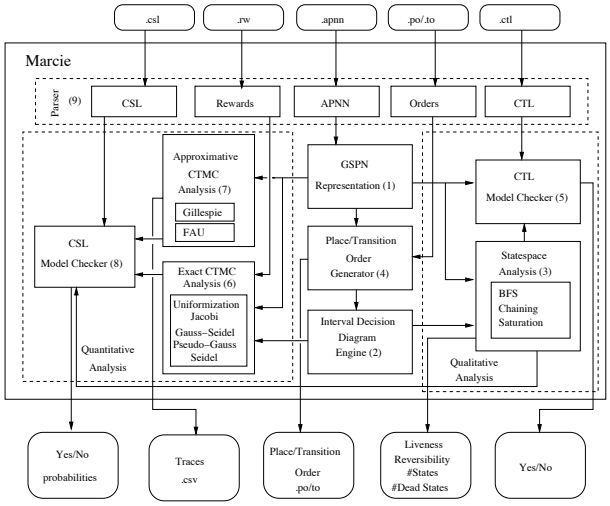

Fig. 2. MARCIE's architecture and its nine components.

1) GSPN Representation: MARCIE analyses Generalized Stochastic Petri Nets as defined in [2] and tailored to the specific needs of Systems Biology in [10]. They have to be specified in an adaption of the APNN format [18] which we call APNN* [19]. In addition to standard and inhibitor arcs, it knows read, equal and reset arcs. Currently there is no support for marking-dependent arc weights.

GSPN associates functions to transitions. For a stochastic transition, this function defines the firing rate, for an immediate transition, the function defines a weight which may be necessary to resolve conflicts between simultaneously enabled immediate transitions. MARCIE permits these functions to be marking-dependent; thus places can occur as function variables. However, for each transition, the function's domain is restricted to its pre-places. To allow arbitrary functions, the user may add so-called modifier arcs, which make places to pre-places of a transition without affecting their enabledness.

The model specification supports constants to parameterize initial markings, functions and arc weights.

The net structure is crucial for the performance of most analyses techniques in MARCIE. The GSPN Representation component is responsible for an efficient access to all structural informations and all defined rate and weight functions.

2) IDD Engine: The core of MARCIE is an efficient IDD implementation. An IDD is a directed, rooted and acyclic

graph which consists of non-terminal nodes labeled with variables and two terminal nodes labeled with 1 and 0. IDDs can be seen as generalization of the popular BDDs, but allow non-terminal nodes to have an arbitrary number of outgoing arcs. These arcs are labeled with intervals over the natural numbers. As usual, we assume a fixed order over the set of variables which must hold on every path starting at the IDD root and going to one of the two terminal nodes. A Reduced Ordered Interval Decision Diagram (ROIDD) does not contain isomorphic sub-diagrams and the labels of the outgoing arcs of a non-terminal node create a non-reducible partition of the natural numbers. It yields a canonical representation of an interval logic function; an example is given in Fig. 3.

In our setting, the set of variables is equal to the set of places in the given Petri net. Marking sets can now be represented by interval logic functions and thus encoded using IDDs.

The implemented IDD engine exhibits several features to address efficiency issues, as for instance the concept of shared DDs, fast detection of isomorphic sub-diagrams by use of an unique table, and efficient operation caches; see [11] for a detailed discussion. Furthermore, the engine offers dedicated operations for the forward and backward firing of Petri net transitions.

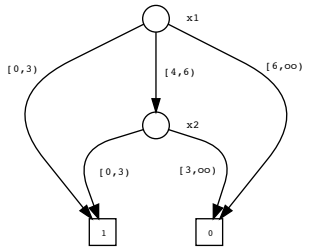

Fig. 3. An IDD representing $f = (x_1 < 3) \vee (x_1 \in [4, 6) \wedge x_2 \leq 2)$.

3) State Space Analysis: Bounded Petri nets can be analysed based on the knowledge of the states, which are reachable from the initial marking. Thus the first step is often the state space construction. MARCIE implements three different state space generation algorithms upon its IDD engine.

1) Common Breadth-First Search (BFS): an iteration fires sequentially all transitions (according to the transition ordering) before adding the new states to the state space.
2) Transitions chaining: like BFS, but the state space is updated after the firing of each single transition.
3) Saturation algorithm (SAT): transitions are fired in conformance with the decision diagram, i.e. according to an ordering, which is defined by the variable ordering. A transition is saturated if its firing does not add new states to the current state space. Transitions are bottom-up saturated (i.e. starting at the terminal nodes and going towards the root). Having fired a given transition, all preceding transitions have to be saturated again, either after a single firing (single) or the exhausted firing (fixpoint) of the current transition.

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.

4) Order Generator: It is well known that the variable order used in constructing a DD has a strong influence on its size in terms of the number of nodes, and thus on the performance of related operations. To find the optimal variable order is an NP-hard problem. MARCIE uses heuristics to pre-compute static variable orders. The simple underlying idea is to examine the structure of the given Petri net and to arrange dependent places close to each other. Two places are dependent if there are transitions whose firing affect the marking of both places [20]. In [12] we used this idea to successfully improve PRISM's performance. MARCIE offers the user seven different options to influence the generation of the place order.

Transition Chaining and Saturation are improved algorithms for the state space generation. Their efficiency depends in general on the order in which the Petri net transitions are considered. Thus MARCIE takes care of creating suitable transition orders. There are six different options to influence the used transition order.

5) CTL Model checker: The Computation Tree Logic (CTL) [21] 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. So-called path quantifiers specify whether path formulas, which can be written be 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. The basic idea is to label all LTS states with the sub-formulas they satisfy. This is done by a postorder traversal of the given CTL formula tree. In an explicit setting, the evaluation of path formulas can be done by a depth-first-search strategy to find a witness path or a counter example. In a symbolic setting, one computes for all sub-formulas the set of satisfying states. This can be achieved for path formulas by applying fixpoint computations based on backward reachability analysis as it is efficiently realized in MARCIE. For a detailed discussion of IDD-based CTL model checking we refer to [11].

6) Exact CTMC Analysis: In addition to the qualitative analysis MARCIE supports quantitative analysis of SPNs. While qualitative analysis can be realized symbolically based on the reachable states, quantitative analysis requires the computation of probability distributions of the induced CTMC. CTMC analysis is a well studied subject and established algorithms are available [22]. In general the computation of a probability distribution must be done numerically. For this purpose, all real-valued non-zero state transitions and a number of computation vectors have to be stored in memory in addition to the state space of the CTMC. The set of state transitions can be written as a matrix, the so-called rate matrix, which is in general extremely sparse. Then, the core operation is a matrix-vector multiplication.

There are several approaches to compactly store the rate matrix either explicitly in some sparse matrix format or symbolically using Kronecker representations or some kind of decision diagrams. The latter approach is the most promising one as it does not require specific model properties (besides being bounded). The basic idea is to encode a state transition, which consists of a state pair and a real value, as a path in a decision diagram. One can distinguish two different approaches to store the real values: either in the terminal nodes (MTDD) or distributed over the edges of the diagram (Edge-Valued Decision Diagrams); see, e.g., [23, 24] for a brief overview. In any case, the extraction of the matrix entries is done by a traversal of the decision diagram. In a full symbolic approach one would also encode the computation vectors by decision diagrams, but this turned out to be not efficient. Thus, most approaches are hybrid; a symbolic matrix encoding combined with explicitly stored computation vectors.

MARCIE follows the hybrid approach in a slightly different way. It combines a symbolic state space representation with an "on-the-fly" strategy. The idea of such a "matrix-free" approach was proposed in [25] with explicitly storing the state space. MARCIE computes the matrix entries by simulating the firing of the Petri transitions. This is done by traversing the IDD which represents the state space and which has been augmented by additional information to allow the computation of state indices similar to [26, 27]. During a traversal MARCIE considers the effect of firing all transitions of the net for each state and computes the relevant state indices. Further it collects the arguments for the state-dependent rate functions of the Petri net transitions. Considering the transition firing in forward (backward) direction permits to realize a matrix-vector (vector-matrix) multiplication [12, 13]. This "on-the-fly" approach is significantly less sensitive to the number of different non-zero values in the rate matrix [12] than the MTDD approach.

A DD traversal has in general exponential effort. Thus it is important to realize an efficient caching mechanism. The usual operation caches are not suitable in our setting. Thus MARCIE follows a strategy similar to that in [27]. The traversal will stop at a predefined node layer, where the information of all possible path extensions are compactly stored. The actual matrix entries will be computed from the information of the current path and the stored cache data. The computation vectors and the entries of the matrix diagonals are explicitly stored in arrays of double precision type.

Another way to increase the efficiency is to make use of today's multi-core computers by multi-threading. The implemented matrix-vector/vector-matrix multiplication can be performed in a multi-threaded fashion.

Upon these basic capabilities to efficiently perform operations on the CTMC's rate matrix which is induced by a stochastic Petri net, MARCIE implements standard algorithms to compute transient and steady state probability distributions.

Transient probabilities. The computation of transient probabilities can be achieved by solving the system of Kolmogorov differential equations, which can be done using different methods, as for instance uniformization, Krylov subspace methods, or ordinary differential equations [22]. When dealing with very large and cyclic CTMCs, as it is often the case, the method of choice is uniformization. The basic idea is to embed a discretization of the CTMC into a Poisson process. Doing so, the computation of the transient probability distribution for a specific time point reduces to that of Discrete Time Markov Chains (DTMCs), which can be done by a repeated matrix-vector multiplication. MARCIE provides multi-threaded transient analysis based on uniformization.

Steady state probabilities. The computation of a steady state probability distribution requires to solve a homogeneous system of equations. In the light of a symbolic matrix representation, iterative methods as Jacobi and Gauss-Seidel are favoured to direct methods as Gauss elimination, because they do not change the matrix. Currently, MARCIE offers a multi-threaded Jacobi solver and a Gauss-Seidel solver. The Gauss-Seidel method requires a single computation vector and converges in many cases much faster than Jacobi, but requires a row-wise extraction of the matrix entries. MARCIE's caching approach does not allow an efficient row-wise extraction. As a compromise, MARCIE also provides a Pseudo-Gauss-Seidel solver [27].

Rewards. Often it is not sufficient to just reason about the probability to be in a certain state at a certain time point or in steady state. The expected time spent in certain states, the expected number of transition firings within a given time interval, and comparable measures can be of interest, too. For this purpose, it is possible to extend a stochastic Petri net to a stochastic reward net [28] by adding rewards, which define additional measures; they can be associated to states and transitions. A reward for a given state will be accumulated and weighted with its sojourn time. A transition reward is acquired each time a transition fires.

MARCIE supports the addition of rewards to a GSPN by loading a set of reward structures. A reward structure is a set of reward items – state reward items and transition reward items. A reward item must specify a set of states by means of an interval logic function and a possibly state-dependent reward function defining the actual reward value.

MARCIE represents rewards internally as additional, implicit Petri net transitions, which allows us to apply the "on-the-fly" approach also to compute rewards; see [14] for more details.

Immediate transitions. All quantitative analyses can be applied without any exception to GSPNs. In this case, the "on-the-fly" engine has also to consider the vanishing states, which may represent the majority of reachable states, as one can see in Table II. This will substantially increase the memory effort. Furthermore, MARCIE needs an additional computation step to propagate the probabilities from the vanishing states to the tangible states. In summary, dealing with GSPNs is much more expensive, and we recommend, if possible, to transform the GSPN in a SPN as proposed in [2].

7) Approximative CTMC Analysis: The exact quantitative analysis of GSPNs is restricted to bounded state spaces.

Approximative techniques are needed in order to analyze unbounded models or models with very huge state spaces ($\gg 1 \times 10^9$ states).

Approximative numerical analysis. To overcome the problem of an unmanageable state space size, the approximative numerical analysis prunes insignificant states. The basic idea is to combine a breadth-first variant of the state space construction with a transient analysis using uniformization. During construction, all explored states having a probability below a specified threshold will be removed from the current state space. Thus, only a finite subset of a possibly infinite state space will be considered. This "sliding window" method [29] can be further enhanced by a technique called adaptive uniformization, where the Poisson process is replaced by a birth process. This combination was first introduced in [30] as fast adaptive uniformization (FAU). In contrast to the exact numerical analysis,MARCIE's implementation of this algorithm uses an explicit state space representation.

Simulation. If the approximate numerical analysis exceeds the available memory, the method of choice has to be simulation. MARCIE provides the stochastic simulation algorithm introduced by Gillespie [31]. It generates paths of finite length of a possibly infinite CTMC. Unlike the numerical analysis, the memory consumption of the simulation is constant, because only the current state is hold in memory. It is necessary to perform a reliable amount of simulation runs due to the variance of the stochastic behaviour.

We choose the confidence interval method as described in [32] 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., 1×10^{-3} or 1×10^{-4}. MARCIE calculates the required number of simulation runs to achieve this confidence interval. Besides that, the user can set the number of simulation runs manually.

Each simulation run is done independently of the others. Thus, it is not challenging to parallelize the stochastic simulation. So MARCIE provides a multi-threaded simulation engine.

Immediate Transitions. The approximative numerical analysis as well as the simulation are capable of handling immediate transitions. The approximative numerical analysis has to handle two types of states for this purpose. Besides tangible states (only stochastic transition are enabled), vanishing states can appear, where one or more immediate transitions are enabled. Vanishing states are processed as soon as they occur, because of the (per definition) higher priority of immediate transitions.

The simulative processing of immediate transitions is rather straightforward, see [33] for details. The stochastic simulation can not only deal with GSPN, but can also analyse eXtended Stochastic Petri Nets (XSPN). A detailed discussion can be found in [34].

8) CSL Model checker: The Continuous Stochastic Logic (CSL) introduced in [35] is the stochastic counterpart to CTL and permits to define complex properties. The path quantifiers of CTL are replaced by the probability operator $\mathcal{P}$. The usual temporal operators are decorated with time intervals.

In [36], 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 that for CTL, but now the evaluation of path formulas requires in general the computation of a probability distribution. Depending on the given time interval, this can be achieved by transient analysis [37] or by solving a linear system of equations using one of the iterative solvers. The evaluation of the steady state operator is of course done by steady state analysis.

Reward Extensions. CSL has been extended in [38] by special operators to trigger the computation of expectations of instantaneous and cumulative rewards.

When excluding the $\mathcal{S}$ operator, the remaining CSL fragment is a proper subset of the Continuous Stochastic Reward Logic (CSRL) [39]. Now, the temporal operators are additionally decorated with a reward interval. CSRL model checking requires to compute the joint distribution of the CTMC and the stochastic process representing the evolution of the accumulated reward. This is much more involved than computing transient probabilities, since the latter does not feature the Markov property. However, there are different algorithms to compute the joint distribution [40].

Exact. Currently, MARCIE supports model checking of CSL as it is defined in [36] extended by the reward operators given in [38], most of them in a multi-threaded way.

Furthermore, MARCIE implements the Markovian approximation algorithm [40] and allows to check a subset of CSRL for state rewards. However, this feature is not documented yet and still under test.

Approximative. In our current implementation MARCIE supports only a subset of CSL for approximative numerical analysis and simulation; unnested $\mathcal{P}$-operator with time-bounded temporal operators can be checked.

9) Parser: This component contains the parsers for the actual Petri nets, CTL and CSL formulas, reward structures, as well as place and transition orders. Currently, most of them are realized using the lexical analyzer $flex$ and the parser generator $bison$, but we are going to move to the lightweight parser generator $Spirit$ from the $boost$ library. For a detailed syntax specification of all inputs we refer to [19].

III. Related 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. For the lack of space we restrict ourselves to a very short selection from which we will choose one tool for computational comparison with MARCIE. An elaborated comparison of available CSL model checkers can be found in [41], comprising explicit, symbolic and simulative engines.

The probabilistic model checker PRISM [16] is considered to be the "most popular and advanced tool in the field" [42]. It supports the analysis of CTMCs as well as DTMCs and Markov Decision Processes (MDPs) by means of CSL, PCTL

and LTL. It further allows the use of CSL extensions to compute expectations of reward measures. PRISM is based on MTBDDs. It defines its own high level model description language which can be easily used to define SPN specifications.

Another CSL model checker is the Markov Reward Model Checker (MRMC) [42]. It also offers analysis capabilities for CTMCs and related formalisms based on temporal logics. It is the only tool which supports model checking of CSRL formulas. MRMC uses sparse representations to encode the state space and matrices. Particular features are the support for state space reduction based on bisimulation and a simulative steady state detection. MRMC supports simulative model checking of full CSL. It requires third party tools as PRISM to generate the actual Markov model, which results in handling possibly huge files.

A further popular tool is SMART [15]. It offers qualitative and quantitative analysis of GSPNs with marking-dependent arcs and defines its own high level description language. SMART supports CTL, but not CSL model checking, although it is able to efficiently 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). However, some of these storage strategies force the user to obey some modelling restrictions. The use of MDDs, which for instance permit efficient saturation-based reachability analysis, requires to specify a suitable place partition. SMART also implements discrete event simulation.

All three tools can be used for simulative as well as numerical analysis of stochastic Petri nets. None of the tools currently supports multi-threading.

For comparison with MARCIE we decided to use PRISM. It is easy to use and the case studies which we will consider in the following were either already available or easy to create in the PRISM language. To use its efficient hybrid engine does not require to consider specific modelling restrictions as it is the case when using, e.g., SMART's MDD engine. Generally it is most comparable with MARCIE. We made experiments using PRISM's hybrid and sparse engine. See also [12], where we already compared PRISM and MARCIE's predecessor *IDD-MC* concerning transient analysis of biological models.

None of the mentioned tools supports the direct numerical approximation algorithm for computing transient solutions of stochastic models as described in II-B7 and implemented in MARCIE. To the best of our knowledge, the tool Sabre [43] is besides MARCIE the only publicly available implementation. But in contrast to MARCIE, Sabre does not include any model checking capabilities which precludes it from being considered for tool comparison.

IV. Comparison

In this section we provide benchmark results to empirically prove MARCIE's efficiency. We concentrate on quantitative analyses and use CSL properties to trigger the computation of transient and steady state probabilities. We will compare our performance results with PRISM using five different stochastic Petri nets. Besides established CTMC benchmarks we will also use two models from Systems Biology.

A. Case Studies

For our experiments we will use the following technical systems.

FMS. The Flexible Manufacturing System with three machines has been published in [3]. The original model contains immediate transitions; thus it is a GSPN. As discussed, our exact CTMC analysis engine suffers from immediate transitions, and PRISM does not support immediate commands. Thus we will consider a pure SPN model which has been derived from the GSPN model by applying the elimination rules for immediate transitions given in [2]. Furthermore, the FSM model contains arcs with marking-dependent weights. MARCIE does not support such arcs as they potentially destroy the locality principle. Instead, our model simulates the marking dependencies by additional transitions, each representing a specific firing situation in the original model. The FMS is scalable concerning the number of items which can be processed by the machines. The places $P1$, $P2$ and $P3$ carry initially N tokens. The model can be easily scaled be increasing the value of N. Our SPN model is limited to $N = 15$ due to the explicit modelling of the marking-dependent arcs.

Kanban. The kanban system has been used in [4] to demonstrate efficient CTMC encoding and solution using the Kronecker approach. The model is scalable by the initial marking.

PSS. The SPN model of a polling server system has been derived from that given in the PRISM case study collection which is based on [44]. The model is scalable by the number of stations communicating with the server. Thus scalability is here given by the net structure, not only by the initial marking.

Additionally, we use the following models from Systems Biology.

AKAP. This model of the AKAP scaffold-mediated crosstalk between the cAMP and the Raf-1/MEK/ERK pathways has been specified and analysed using PRISM in [45]. The AKAP model is scalable by the initial marking and the arc weights.

ANG. This model of signal transduction events involved in the angiogenetic process has been published in [9]. The model is scalable by the initial marking.

All Petri net models have been done with our Petri net editor Snoopy [46] and are available on our website *http://www-dssz.informatik.tu-cottbus.de/DSSZ/Examples*. There we also provide Petri nets for further biochemical models which we have used as benchmarks in [12, 34].

B. Experiments

In this section we will sketch the setup for the computational experiments we made, and we will present the experimental

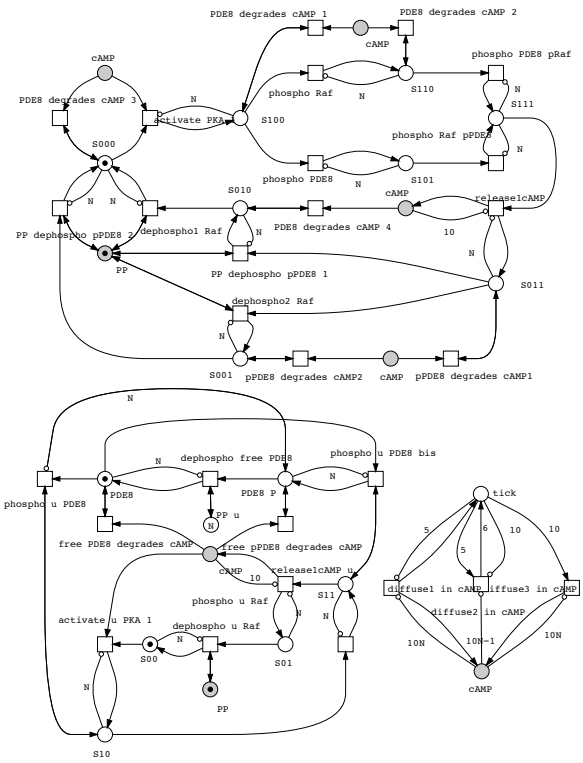

Fig. 4. A Petri net model of the AKAP scaffold-mediated crosstalk. The net consists of 18 places and 28 transitions which are connected by 108 arcs. The grey colored places are fusion places (multiple logical occurrences of the same place) used to achieve a clear layout.

TABLE I
SIZE OF THE REACHABILITY GRAPHS FOR DIFFERENT MODEL CONFIGURATIONS. THE REACHABILITY GRAPH AND THE CTMC ARE ISOMORPHIC IF THERE ARE NO PARALLEL TRANSITIONS. THE REACHABILITY GRAPHS OF THE AKAP MODEL CONTAIN SUCH PARALLEL TRANSITIONS.

model	N	states	transitions
FMS	2	810	3,699
	4	35,910	237,120
	6	537,768	4,205,670
	8	4,459,455	38,533,968
	10	25,397,658	234,523,289
	12	111,414,940	1,078,917,632
	14	403,259,040	4,047,471,180
$Kanban$	2	4,600	28,120
	4	454,475	3,979,850
	6	11,261,376	115,708,992
	8	133,865,325	1,507,898,700
PSS	5	240	800
	10	15,360	89,600
	15	737,280	6,144,000
	20	31,457,280	340,787,200
$AKAP$	3	1,632,240	12,691,360
	4	15,611,175	141,398,580
	5	74,612,328	734,259,344
	6	386,805,104	4,116,788,172
ANG	2	5,384	26,193
	4	2,413,480	21,810,412
	6	277,789,578	24,813,347,031

TABLE II
THE EVOLUTION OF THE STATE SPACE OF THE FMS MODEL GIVEN AS SPN AND GSPN. $|S_q|$ DENOTES THE NUMBER OF STATES WHEN CONSIDERED COMPLETELY QUALITATIVE, I.E. IMMEDIATE TRANSITIONS ARE NOT PRIORIZED OVER STOCHASTIC TRANSITIONS. A PRIORIZATION OF IMMEDIATE TRANSITIONS AS IT IS THE CASE IN THE GSPN SEMANTICS YIELDS THE SET OF REACHABLE STATES S WHICH CONSISTS OF THE VANISHING S_v AND THE TANGIBLE S_t STATES.

N	$\|S_q\|$	$\|S\|$	$\|S_v\|$	$\|S_t\|$
2	3,444	2,202	1,392	810
4	438,600	138,060	102,150	35,910
6	15,126,440	2,519,580	1,981,812	537,768
8	248,002,785	–	–	4,459,455
10	2,501,413,200	–	–	25,397,658

results. All experiments were done on a 8×2.26 Mac Pro workstation with 32GB RAM running Cent OS 5.5. We restricted all experiments to a run-time of at most 12 hours.

We considered for all models different state spaces which are shown in Table I. MARCIE's capabilities concerning state space generation and CTL model checking are not appropriately reflected by these figures. For instance, MARCIE is able to compute the state space for the FMS and Kanban model up to $N = 300$ which means to have about 3.65×10^{28} or 2.65×10^{24} states [6].

However, in this paper we focus on the quantitative analysis; thus the feasible model size is limited by the memory effort for the probability vectors. On our test system this would permit to consider models as Kanban, N=10, with about 1×10^9 states, depending on the used solver. For the FSM we use the SPN model which permits a more efficient analysis as explained in section II-B6.

All experiments were triggered by CSL formulas, which possibly refer to model-specific reward structures. The CSL formulas and reward structures are available on our website.

Our numerical experiments can be divided into three categories.

1) Steady State Analysis: To trigger the computation of the steady state distribution we used either the CSL template $S_{=?}[sp]$ with a model-specific state property sp or the CSL reward template $R\{rs\}_{=?}[S]$ with a model-specific reward structure rs. We used the Jacobi solver which is the default steady state solver both in MARCIE and PRISM. Only for the kanban system and the angiogenetic process we had to move to Gauss-Seidel, because the Jacobi solver did not converge within 10,000 iterations. Steady state analysis requires to determine the reversibility of the SPN (irreducibility of the CTMC) and, if necessary, a SCC decomposition. It turned out that for the PSS model the computed variable order was not the best choice for this precomputation step. The SCC decomposition required much more time than the actual analysis. Thus we used the plain place order.

FMS. We used transition rewards to specify and compute the productivity of the system; see [3] for more details.

Kanban. We used transitions rewards to specify and compute the throughput of the system.

PSS. We computed the steady state probability that station 1 is waiting, which gives the state property $(s1 = 1) \wedge \neg(s = 1 \wedge a = 1)$.

TABLE III

TOTAL RUN-TIME FOR STEADY STATE ANALYSIS INCLUDING STATE SPACE GENERATION, INITIALIZATION, REWARD COMPUTATION - IF NECESSARY, AND PROBABILITY COMPUTATION. THE NUMBER OF REQUIRED ITERATIONS IS GIVEN IN THE COLUMNS $iter$. THE LAST TWO COLUMNS SHOW RESULTS OF PRISM USING ITS HYBRID AND SPARSE ENGINE. IF POSSIBLE WE USED THE JACOBI SOLVERS. FOR THE KANBAN AND ANG MODELS WE USED THE GAUSS-SEIDEL SOLVERS. THE NUMBER OF ITERATIONS REQUIRED BY THE GAUSS-SEIDEL SOLVERS DIFFER BETWEEN MARCIE AND PRISM BECAUSE OF DIFFERENT VARIABLE ORDERS.

model		MARCIE					PRISM		
	N	iter	1	2	4	8	iter	hybrid	sparse
FMS	2	378	$\ll$1s	$\ll$1s	$\ll$1s	$\ll$1	378	2s	3s
	6	1,084	59s	42s	57s	34s	1,084	41s	30s
	10	1,812	1h11m15s	45m07s	30m31s	24m47s	1,812	2h25m42s	37m15s
	12	2,193	6h13m53s	3h52m32s	2h26m17s	2h04m45s	2,193	–	3h32m11s
	14	2,589	–	–	–	9h10m04s	†	†	†
$Kanban$	2	48	$\ll$1s				189	2	2s
	4	122	15s		Gauss-Seidel		323	12s	10s
	6	224	11m07s		no multi-threading support		622	9m48s	6m18s
	8	356	3h57m23s				999	3h27m27s	1h59m59s
PSS	10	406	$\ll$1s	$\ll$1s	$\ll$1s	$\ll$1s	406	2s	2s
	15	657	14s	9s	5s	3s	657	24s	21s
	20	920	16m16s	10m58s	6m21s	3m08s	920	31m28s	26m31s
$AKAP$	4	1,433	21m17s	13m07s	7m04s	4m40s	1,433	17m37s	17m08s
	5	1,348	1h46m52s	1h05m12s	37m39s	23m03s	–	–	–
	6	1,659	–	8h00m02s	3h53m29s	2h43m48s	–	–	†
ANG	2	$n.a.$	48s				$n.a.$	4m05s	28s
	3	$n.a.$	1s24m55s		Gauss-Seidel		†	†	†
	4	$n.a.$	–		no multi-threading support		†	†	†

– means that the experiment was canceled after 12 hours

† means that the CTMC could not be created using the default tool settings

AKAP. We used state rewards to specify and compute the average number of tokens on place $cAMP$.

ANG. We used state rewards to specify and compute the average number of tokens on place DAG.

2) Transient Analysis: We use either the CSL template $P_{=?}[F[t,t]sp]$ which computes the probability to be in a state satisfying the model-dependent state property sp at time point t when starting in the initial state, or the CSL reward template $R\{rs\}_{=?}[I = t]$ which computes the expected instantaneous reward at time point t when starting in the initial state and considering the model specific reward structure rs. The time parameter t was set to 1.0 for all experiments.

FMS. We computed the probability for time t to have N tokens on place $P1$.

Kanban. We computed the probability for time t to have N tokens on place $x1$.

PSS. We computed the probability at time t that station 1 is waiting.

AKAP. We used state rewards to specify and compute the expected number of tokens on place $cAMP$ at time point t.

ANG. We used state rewards to specify and compute the expected number of tokens on place DAG at time point t.

3) Results: In Table III and IV we provide results of our numerical experiments. For the transient analysis, MARCIE's IDD engine outperforms PRISM's hybrid engine as well as its sparse engine for increasing values of N even in the single-threaded mode. For the steady state analysis, MARCIE can not compete with PRISM's Gauss-Seidel solver, while the performance of MARCIE's Jacobi solver lies between PRISM's sparse and hybrid engine, as expected without multi-threading. An exception is the PSS model, where MARCIE performs better in all cases.

MARCIE's multi-threading feature reduces the computation time for all models and all supported analysis methods, although the actual speedup varies.

Because of space limitations we just give the total run-time and ignore the memory consumption. A closer look at the obtained results reveals that there are significant differences regarding the time needed to construct the state space, to compute the rewards, and the probability distribution or to initialize cache data. Thus our tables only represent a very rough comparison.

Both tools can be used more efficiently when using dedicated options. For instance, the computed cache layer for the FMS, which is 8, is not the best choice. A value of 12 would drastically reduce initialization time and memory consumption. The cache initialization for N=12 takes 866 seconds when using 4 threads, while the computation of transient probabilities takes just 583 seconds. Thus the total run-time increases with 8 threads, although the computation time decreases. In this case the overhead for initialization is accompanied by too few required iterations.

4) Approximative Analysis: For the fast adaptive uniformization as well as the simulation we use the CSL template $P_{=?}[F[t,t]\, place = N]$ which computes the probability to be in a state satisfying the model-dependent state property that $place$ has N token at time point t when starting in the initial state. We used the same places as for the transient analysis.

Results. Table V shows the total run-time for transient analysis using simulation in MARCIE and PRISM. The stochastic simulation in PRISM seems not to be optimized for simulative model checking. The stochastic simulation greatly benefits from multi-threading. The speedup is almost linear with the

TABLE IV

TOTAL RUN-TIME FOR TRANSIENT ANALYSIS INCLUDING STATE SPACE GENERATION, INITIALIZATION AND PROBABILITY COMPUTATION. THE NUMBER OF REQUIRED ITERATIONS IS GIVEN IN THE COLUMN $iter$. THE LAST TWO COLUMNS SHOW RESULTS OF PRISM USING ITS HYBRID AND SPARSE ENGINE.

model			Marcie				PRISM	
	N	iter	1	2	4	8	hybrid	sparse
FMS	8	202	2m27s	2m28s	1m51s	1m28s	3m47s	2m50s
	10	202	13m11s	9m44s	7m51s	8m22s	27m39s	15m57s
	12	208	55m54s	42m41s	31m56s	29m57s	2h10m47s	1h39m05s
	14	208	4h07m51s	3hm35ms07s	2h48m59s	2h41m41s	8h26m38s	OutOfMemory
$Kanban$	4	181	6s	3s	2s	2s	12s	8s
	6	181	1m57s	1m34s	48s	34s	5m16s	3m09s
	8	181	23m59	17m53s	9m18s	6m41s	1h08m15s	43m23s
PSS	10	377	$\ll$1s	$\ll$1s	$\ll$1s	$\ll$1s	2s	2s
	15	377	17s	7s	4s	3s	33s	17s
	20	377	9m02s	5m32s	2m59s	2m06s	29m18s	14m22s
$AKAP$	4	1,532	25m15s	17m09s	7m49s	5m13s	52m20s	24m12s
	5	1,850	2h40m14s	1h34m36s	47m54s	30m26s	–	–
	6	2,218	–	–	5h44m53s	3h49m45s	†	†
ANG	4	547	1m53s	1m18s	54s	44s	2m27s	7m24s
	5	794	31m08s	20m50s	13m45s	11m46s	34m06s	4h25m34s
	6	1,093	7h36m57s	4h39m52s	3h17m47s	3h00m16s	†	†

– means that the experiment was canceled after 12 hours

† means that the CTMC could not be created using the default tool settings

TABLE V

TOTAL RUN-TIME FOR SIMULATIONS DONE WITH MARCIE AND PRISM. THE CONFIDENCE LEVEL IS 99%, THE DESIRED ACCURACY 1×10^{-5}, WHICH LEADS TO $66,348,303$ SIMULATION RUNS.

model	N	t	Threads	MARCIE	PRISM
FMS	14	1	1	40m43s	3h24m13s
			8	5m4s	$n.a.$
$Kanban$	10	10	1	13m48s	2h23m58s
			8	1m58s	$n.a.$
PSS	20	1	1	4h10m4s	62h32m33s
			8	31m53s	$n.a.$
$AKAP$	6	1	1	2m37s	10m23s
			8	25s	$n.a.$
ANG	6	1	1	38m48s	8h26m57s
			8	5m20s	$n.a.$

number of threads.

TABLE VI

TOTAL RUN-TIME AND NUMBER OF STATES OF THE APPROXIMATED CTMCS OF MARCIE'S FAST ADAPTIVE UNIFORMIZATION ALGORITHM.

model	N	t	time	$\lvert S_{appr}\rvert$	$\lvert S\rvert$
FMS	14	1	29s	821,199	403,259,040
$Kanban$	10	10	8m46s	20,444,464	1,005,927,208
PSS	20	1	33s	296,991	31,457,280
$AKAP$	6	1	3s	81,657	386,805,104
ANG	6	1	42m21s	12,619,037	277,789,578

Table VI shows the size of the approximated CTMC and the total run-times using MARCIE's fast adaptive uniformization. The approximation of the size of the CTMC lies between $\ll 1\%$ and 4.5%.

V. CONCLUSION

In this paper we presented MARCIE, a tool for qualitative and quantitative analysis of Generalized Stochastic Petri Nets. We gave an overview of its current features, which particularly include symbolic CTL and CSL model checking for bounded nets and approximative model checking of unnested time-bounded formulas, which also works for unbounded nets. The quantitative analyses are based on efficient solvers to compute transient and steady state probabilities. To the best of our knowledge, MARCIE is the first available tool offering these features symbolically and multi-threaded.

We demonstrated MARCIE's efficiency by presenting a bulk of experimental results and a comparison with PRISM. The experimental results which we presented in this paper and in [12, 13] suggest that especially for biological networks MARCIE is the most efficient model checking tool.

Currently we are going to extend the model checking capabilities by full CSRL model checking based on our "on-the-fly" engine and by simulative PLTLc [47] model checking. Furthermore we are preparing out-of-core support and distributed symbolic computation of probability distributions.

Acknowledgements.: We would like to thank Verena Wolf for fruitful discussions and for the support in MARCIE's FAU implementation.

REFERENCES

[1] P. M. Merlin and D. J. Farber, "Recoverability of communication protocols - implications of a theoretical study," *IEEE Transactions on Communications*, vol. 24, no. 9, pp. 1036–1043, 1976.

[2] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, John Wiley and Sons, 1995, 2nd Edition.

[3] G. Ciardo and K. S. Trivedi, "A Decomposition Approach for Stochastic Reward Net Models," *Performance Evaluation*, vol. 18, no. 1, pp. 37–59, 1993.

[4] G. Ciardo and M. Tilgner, "On the use of Kronecker operators for the solution of generalized stocastic Petri nets," Institute for Computer Applications in Science and Engineering, ICASE Report 96-35, 1996.

[5] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," vol. C-35, no. 8, pp. 677–691, 1986.

[6] M. Heiner, M. Schwarick, and A. Tovchigrechko, "DSSZ-MC - A Tool for Symbolic Analysis of Extended Petri Nets," in *Proc. Petri Nets 2009*. LNCS 5606, Springer, 2009, pp. 323–332.

[7] D. Gilbert, M. Heiner, and S. Lehrack, "A unifying framework for modelling and analysing biochemical pathways using Petri nets," in *Proc. CMSB 2007*. LNCS/LNBI 4695, Springer, 2007, pp. 200–216.

[8] M. Heiner, D. Gilbert, and R. Donaldson, "Petri nets in systems and synthetic biology," in *SFM*. LNCS 5016, Springer, 2008, pp. 215–264.

[9] L. Napione, D. Manini, F. Cordero, A. Horvath, A. Picco, M. D. Pierro, S. Pavan, M. Sereno, A. Veglio, F. Bussolino, and G. Balbo, "On the Use of Stochastic Petri Nets in the Analysis of Signal Transduction Pathways for Angiogenesis Process," in *Proc. CMSB 2009*. LNCS/LNBI 5688, Springer, 2009, pp. 281–295.

[10] M. Heiner, S. Lehrack, D. Gilbert, and W. Marwan, "Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments." LNCS/LNBI 5750, Springer, 2009, pp. 138–163.

[11] A. Tovchigrechko, "Model checking using interval decision diagrams," Ph.D. dissertation, BTU Cottbus, Dep. of CS, 2008.

[12] M. Schwarick and M. Heiner, "CSL model checking of biochemical networks with interval decision diagrams," in *Proc. CMSB 2009*. LNCS/LNBI 5688, Springer, 2009, pp. 296–312.

[13] M. Schwarick and A. Tovchigrechko, "IDD-based model validation of biochemical networks," *Theoretical Computer Sience*, 2010.

[14] M. Schwarick, "IDD-MC - a model checker for bounded Stochastic Petri nets," in *Proc. AWPN 2010*, ser. CEUR Workshop Proceedings, vol. 643. CEUR-WS.org, Sep. 2010, pp. 80–87.

[15] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu, "Logical and stochastic modeling with SMART," *Performance Evaluation*, vol. 63, no. 1, 2006.

[16] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *Proc. TACAS 2006*. Springer, LNCS 3920, 2006, pp. 441–444.

[17] A. Franzke, "Charlie 2.0 - a multi-threaded Petri net analyzer," Diploma Thesis, 2009.

[18] F. Bause, P. Kemper, and P. Kritzinger, "Abstract Petri Net Notation," Univ. Dortmund, CS Dep., Tech. Rep., 1994.

[19] M. Schwarick, *Manual: Marcie - An analysis tool for Generalized Stochastic Petri nets*, BTU Cottbus, Dep. of CS, 2010.

[20] A. Noack, "A ZBDD Package for Efficient Model Checking of Petri Nets (in German)," BTU Cottbus, Dep. of CS, Tech. Rep., 1999.

[21] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite state concurrent systems using temporal logic specifications," *ACM TOPLAS*, vol. 8, no. 2, pp. 244–263, 1986.

[22] W. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton Univ. Press, 1994.

[23] A. Miner and D. Parker, *Validation of Stochastic Systems: A Guide to Current Research*, ser. LNCS 2925. Springer, 2004, ch. Symbolic Representations and Analysis of Large Probabilistic Systems, pp. 296–338.

[24] G. Ciardo, *Data Representation and Efficient Solution: A Decision Diagram Approach*, ser. LNCS, M. Bernardo and J. Hillston, Eds. Springer Berlin / Heidelberg, 2007, vol. 4486.

[25] D. Deavours and W. H. Sanders, "On-the-Fly solution techniques for Stochastic Petri nets and extensions," in *IEEE Transactions on Software Engineering*, 1997, pp. 132–141.

[26] A. S. Miner and G. Ciardo, "Efficient Reachability Set Generation and Storage Using Decision Diagrams," in *Proc. 20th ICATPN*, ser. LNCS 1639. Springer, 1999, pp. 6–25.

[27] D. Parker, "Implementation of symbolic model checking for probabilistic systems," Ph.D. dissertation, University of Birmingham, 2002.

[28] G. Ciardo, A. Blakemore, P. F. Chimento, J. K. Muppala, and K. S. Trivedi, "Automated generation and analysis of markov reward models using stochastic reward nets." *IMA Volumes in Mathematics and its Applications: Linear Algebra, Markov Chains, and Queueing Models / Meyer, C.; Plemmons, R.J*, vol. 48, pp. 145–191, 1993.

[29] T. Henzinger, M. Mateescu, and V. Wolf, "Sliding window abstraction for infinite Markov chains," in *Proc. CAV*. Springer, LNCS 5643, 2009, pp. 337–352.

[30] F. Didier, T. A. Henzinger, M. Mateescu, and V. Wolf, "Fast Adaptive Uniformization for the Chemical Master Equation," in *HiBi*, 2009.

[31] D. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *The Journal of Physical Chemistry*, vol. 81(25), pp. 2340–2361, 1977.

[32] W. Sandmann and C. Maier, "On the statistical accuracy of stochastic simulation algorithms implemented in Dizzy," in *Proc. WCSB 2008*, 2008, pp. 153–156.

[33] R. German, *Performance analysis of communication systems with non-Markovian stochastic Petri nets*. Wiley, 2001.

[34] M. Heiner, C. Rohr, M. Schwarick, and S. Streif, "A comparative study of stochastic analysis techniques," in *Proc. CMSB 2010*. Springer, 2010.

[35] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Model checking continuous-time Markov chains," *ACM Trans. on Computational Logic*, vol. 1, no. 1, 2000.

[36] C. Baier, H. Hermanns, and J.-P. Katoen, "Approximative Symbolic Model checking of Contiuous-Time Markov Chains," in *Proc. 10th CONCUR*, 1999, pp. 163–175.

[37] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model checking continuous-time Markov chains by transient analysis," in *Proc. CAV 2000*. LNCS 1855, Springer, 2000, pp. 358–372.

[38] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *SFM*, 2007, pp. 220–270.

[39] B. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier, "Model Checking Performability Properties," *IEEE CS Press*, pp. 103–112, 2002.

[40] L. Cloth and B. R. H. M. Haverkort, "Five performability algorithms. a comparison," in *MAM 2006: Markov Anniversary Meeting, Charleston, SC, USA*. Boson Books, 2006, pp. 39–54.

[41] D. N. Jansen, J.-P. Katoen, M. Oldenkamp, M. Stoelingan, and I. Zapreev, "How fast and fat is your probabilistic model checker?" in *HVC 2007*. Springer, LNCS 4899, 2008, pp. 69–85.

[42] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen, "The Ins and Outs of The Probabilistic Model Checker MRMC," in *Proc. 6th QEST*. IEEE Computer Society, 2009, pp. 167–176.

[43] F. Didier, T. A. Henzinger, M. Mateescu, and V. Wolf, "Sabre: A tool for stochastic analysis of biochemical reaction networks." in *QEST*, 2010, pp. 217–218.

[44] O. Ibe and K. Trivedi, "Stochastic Petri net models of polling systems," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 9, pp. 1649–1657, 1990.

[45] O. Andrei and M. Calder, "A model and analysis of the AKAP scaffold," *Electron. Notes Theor. Comput. Sci.*, vol. 268, pp. 3–15, December 2010.

[46] C. Rohr, W. Marwan, and M. Heiner, "Snoopy–a unifying Petri net framework to investigate biomolecular networks," *Bioinformatics*, vol. 26, no. 7, pp. 974–975, 2010.

[47] R. Donaldson and D. Gilbert, "A Monte Carlo model checker for probabilistic LTL with numerical constraints," University of Glasgow, Department of Computing Science, Tech. Rep., 2008.