IDD-MC - a model checker for bounded stochastic Petri nets

Martin Schwarick

Department of Computer Science, Brandenburg University of Technology
Postbox 10 13 44, 03013 Cottbus, Germany
ms@informatik.tu-cottbus.de

Abstract. IDD-MC is a symbolic analysis tool for bounded stochastic Petri nets with extended arcs. Its engine is based on Interval Decision Diagrams and facilitate the validation of standard Petri net properties, model checking the Computation Tree Logic (CTL) and the Stochastic Continuous Logic (CSL). In this paper we give an informal overview of the currently implemented analysis techniques and report on the most recent extension: the evaluation of rewards. We present some experimental results which show the efficiency of our implementation.

1 Introduction

Stochastic Petri Nets (SPN) are an established formalism for the modeling and analysis of systems, among them biological networks [GHL07,HGD08,HDG09]. Assuming bounded Petri nets, interesting qualitative properties as reversibility and liveness can be determined by applying graph theoretic methods to the reachability graph. Although the complexity of these methods is linear in the size of the reachability graph, its size can grow over exponentially [PW03]. This calls for dedicated techniques as partial order reduction or symbolic state space representation. Here we consider techniques based on an efficient state space encoding using Interval Decision Diagrams (IDD), a generalization of Binary Decision Diagrams (BDD).

In a biological setting tokens often represent molecules or concentration levels. Thus the state space explosion is caused both by concurrency and by a high boundedness degree. The use of IDDs addresses especially the latter issue.

The quantitative semantics of a stochastic Petri net is described by a Continuous time Markov Chain (CTMC). Assuming a net without parallel transitions, the associated CTMC is a graph isomorphic to the reachability graph, but arcs are labeled by firing rates, given by the possibly state-dependent rate functions of the Petri net transitions. In general the rates are given by a sparse matrix indexed by the reachable states. CTMC theory offers a variety of numerical methods for an exhaustive analysis [Ste94]. But in practice either an infinite or a hugh state space of the investigated models are often a very strict limitation. In these cases simulative or approximative methods represent possible alternatives [HRSS10]. However, certain properties or the demand for a high accuracy of the results require an exhaustive exploration, which in turn calls for an efficient representation of the CTMC and the adaption of the established algorithms. So did we by implementing our SPN analysis engine in IDD-MC which we will present in the next sections.

2 Related Work

There are several tools which offer similar functionality as IDD-MC as for instance the probabilistic model checker PRISM [HKNP06], the SMART tool [CJMS06], Möbius [GKL$^+$09] or GreatSPN [BBC$^+$09] . The most similarities exist regarding to PRISM. Its symbolic engine is based on Multi Terminal Binary Decision Diagrams (MTBDD) which turn out to be not the best solution when dealing with biological networks [SH09] and thus inspired the implementation of IDD-MC's stochastic analysis features.

3 IDD-MC

IDD-MC is a tool for symbolic state space based analysis; first for bounded Petri nets with extended arcs, recently also for bounded stochastic Petri nets. The symbolic engine is based on Reduced Ordered Interval Decision Diagrams (ROIDDs), IDDs for short, which represent a canonical representation for interval logic functions used to encode sets of states (markings) of bounded Petri nets; see [Tov08] for a detailed discussion. It offers a very efficient implementation of IDDs and related operations, among them dedicated operations for the firing of Petri net transitions.

3.1 Overview

Upon this IDD engine the following qualitative analysis features have been realized:

Efficient state space generation. Three state space generation algorithms have been implemented; common breath-first-search, transition-chaining and saturation. The latter algorithm is highly efficient concerning runtime and memory consumption by exploiting the locality of transition firing.

Basic Petri net properties. The tool allows to check for reversibility and liveness of transitions. Therefor efficient decomposition of the strongly connected components has been implemented.

CTL model checking. Given a bounded Petri net N and a Computation Tree Logic (CTL) formula φ (see [CGP01] for an introduction), the model checking problem is to decide whether φ holds in the initial state of N. The classical CTL model checking algorithm [CES86] can be adapted to solve the problem using a symbolic representation of the reachable states of N. It determines for each subformula ψ of φ the set of states fulfilling ψ starting from the innermost formulas. Then it proceeds such that when processing a formula, the set of fulfilling states have been determined for all its subformulas. A CTL subformula can either be a state formula or a path formula containing a temporal operator. While a state formula can be evaluated locally in a state, a path formula requires to evaluate the paths starting in a state. In a symbolic setting this can be solved by fixpoint computations. The top-formula φ is *true* if the initial state is contained in its associated set.

In addition to qualitative analysis of bounded Petri nets IDD-MC offers the following quantitative analysis techniques for bounded stochastic Petri nets:

Transient analysis. Transient analysis is the computation of the probability distribution at a certain time point τ starting with a certain initial probability distribution. One of the standard techniques is uniformization. The basic idea is to embed a special discretization of the CTMC into a Poisson process which has the same probability distribution at time τ. Its computation reduces to truncate an infinite sum of matrix-vector multiplications. For a detailed description and further techniques see [Ste94]. IDD-MC realizes transient analysis by applying an on-the-fly multiplication algorithm, which does not require an explicit encoding of the CTMC's rate matrix.

Steady State analysis. Continuous time Markov Chains often reach finally a stable probability distribution, which is called the steady state. It can be interpreted as the transient probability distribution for infinite time. Computing the steady state probabilities means to solve a linear system of equations. Iterative methods as Jacobi and Gauss-Seidel are the favored techniques. IDD-MC offers Jacobi, Gauss-Seidel and Pseudo-Gauss-Seidel [Par02] solver based on our on-the-fly multiplication.

CSL model checking. The qualitative analysis techniques, transient and steady state analysis are the needed ingredients to realize model checking of the Continuous Stochastic Logic (CSL) introduced in [ASSB00]. Beeing an stochastic adaption of CTL, the basic model checking procedure is similar. The evaluation of path formulas with time-bounded temporal operators can be achieved by applying transient analysis as proposed in [BHHK00] and implemented in our tool. In [BHHK00], CSL has been extended by a special steady state operator and temporal operators without time bounds. Untimed operators require to solve a linear system of equations based on the Embedded Markov Chain of the original CTMC applying one of the iterative methods.

3.2 Efficiency issues

There are several aspects which are significant for the efficient implementation of the mentioned analysis techniques.

Variable order. It is well known that the variable order affects the size of the decision diagram (DD) and thus has a significant impact on the runtime and memory consumption of the DD implementation. What this means in practice can be seen in [SH09]. IDD-MC uses heuristics [Noa99] based on the Petri net structure to compute static variable orders which produce small-sized DDs in most cases. Furthermore, the tool makes use of so-called *Shared IDDs* which means that several IDD-instances reference to the same set of IDD nodes. This allows, for instance, to check for equality in constant time.

CTMC representation. One of the most challenging problems with quantitative analysis of an SPN is the representation of the rate matrix of its induced CTMC. There are established symbolic techniques as MTBDDs or Kronecker products implemented in existing tools [MP04]. But these techniques may fail under special conditions. For instance, a MTBDD representation suffers from an high amount of distinct non-zero values in the matrix and from a high number of BDD variables caused by an high boundedness degree of the model [SH09].

IDD-MC's numerical engine is based on an on-the-fly approach, which has also been considered for an explicit state space representation in [DSS97] and symbolically in [Sch08,SH09]. In our case the CTMC is represented by the Petri net structure, the reachable states encoded as an IDD, and the rate functions of the transitions. A multiplication of the matrix and a vector is realized by traversing the IDD for all transitions of the net. The traversation simulates the firing of all enabled transitions for all states. Therefor we consider the pre- and post conditions to compute the index of the source and target state and to collect the arguments for the possibly state-dependent rate functions. To achieve this, the IDD has been augmented with certain index informations; see [ST10] for more information. To prevent from unnecessary recomputations, we use the caching strategy from [Par02] adapted to our special settings. The multiplication operation *multiply(StateSet S, TransitionSet T, Vector argument, Vector result)* is implemented configurable concerning the set of considered states S and transitions T. This allows, for instance, to parallelize a single multiplication given a suitable state space partition. In [HRSS10] we present results were we achieved a speedup close to the number of physical cores of current multi-core work stations. Recently we extended our tool by reward structures. In the next section we briefly sketch how to incorporate rewards and their analysis.

4 Rewards

Rewards (also interpretable as costs) define additional measures for probabilistic models and can be associated to states and transitions. They are specified by possibly state-dependent reward functions. A reward for a state will be accumulated and weighted with the time the system remains in it. A transition reward is acquired each time a transition fires.

Conform to the probabilistic model checker PRISM rewards can be added to a model by specifying a reward structure, a collection of state and transition reward items.

A state reward item consists of a guard defining a set of states and the reward function. A transition reward item defines for a certain Petri net transition a reward function. Additionally a guard may restrict the set of enabling states of the transition for which the reward should be considered.

Each state reward item defines a vector which assignes to all states satisfying the associated guard the reward value computed using the reward function. Each transition reward item defines a matrix where all entries representing a state transition caused be the associated Petri net transition and starting in states satisfying the guard. Each reward structure thus defines a vector representing the sum of all reward vectors defined by the state reward items and a transition reward matrix representing the sum of the reward matrices defined by the transition reward items of the structure. These vectors and matrices are in the dimension of the reachable states and introduce the same problem as the representation of the CTMC. In PRISM reward structures are encoded by MTBDDs.

Since IDD-MC follows a matrix free strategy, we use our on-the-fly multiplication to compute the rewards when needed. A reward item is represented by a set of states and an additional implicit Petri net transition. When adding a reward structure to a stochastic Petri net, the tool adds a new transition for each defined reward item. For a state reward, the rate function of the new implicit transition is the reward function itself. For a transition reward the reward function multiplied by the rate function of the referenced Petri net transition becomes the rate function of the new implicit transition. This enables the computation of an entry-wise matrix product which is required by the analysis of transition rewards [KNP07].

For a state reward, the assigned set of states is the subset of the reachable states satisfying the specified guard. For a transition reward, the assigned set of states are the reachable enabling states of the referenced Petri net transition satisfying the specified guard. Pre- and post conditions of these new transitions are configured in that way that the transition is enabled in every reachable state and its firing does not change the system state but allows to compute the reward.

For the analysis of a reward-augmented model we extend (again conform to PRISM) CSL by the special reward operator R and four new state formulas. Given a reward structure r, a real valued reward bound b and an operator $\bowtie \in \{>, \geq, <, \leq\}$ we define the following formulas:

1. $R\{\text{“}r\text{”}\}_{\bowtie b}[C_{\leq t}]$ The expected cumulative reward within the first t time units is $\bowtie$ than b.
2. $R\{\text{“}r\text{”}\}_{\bowtie b}[I_{=t}]$ The expected state reward at time t is $\bowtie$ than b.
3. $R\{\text{“}r\text{”}\}_{\bowtie b}[F\phi]$ The expected cumulated reward until the first time a state is reached satisfying the state property ϕ is $\bowtie$ than b.
4. $R\{\text{“}r\text{”}\}_{\bowtie b}[S]$ The expected long-run average reward is $\bowtie$ than b.

Because of the given space limitations we can not discuss the evaluation of all these formulas and refer to [KNP07]. However the basic step is to compute initially a single vector of reward values in the size of the state space. In the case of the cumulative operator $C_{\leq t}$, for instance, the vector represents the sum of the state reward vector and a vector containing the row sums of a matrix achieved by an entry-wise multiplication of the CTMC rate matrix and the matrix representing the transition rewards. This pre-computation step can be simply realized using our transition-based reward representation and our multiply operation.

To compute the state reward vector of a reward structure we apply the multiplication successively to all implicit transitions representing state reward items. The argument vector is initialized with 1 in every entry. Since transitions do not change the state by firing, this means to extract the elements of the diagonal of the matrix, belonging to rows defined by a state set satisfying the guard. The computed rate of these implicit state transitions are the actual state rewards and are added to the result vector. Actually we are computing the row sums of a matrix where only one element per row is non-zero and add these sums to our result vector.

Obviuosly, we will not set the result vector to zero after a multiplication. We similarely compute the entries of transitions reward matrix just considering the transitions representing transition reward items.

For the mentioned case of the cumulative operator, we start off with a zero result vector and apply the multiplication sequentially for all implicit transitions defined by the reward structure, state rewards as well transition rewards.

4.1 Experimental results

To demonstrate the efficiency of our tool we present some experimental results. We run IDD-MC and PRISM−3.3.1 on a 8×2.26 GHz MAC Pro with 32 GB RAM with eight physical cores (with hyper-threading 16 logical cores). We consider a stochastic Petri net model of the Mitogen activated protein kinase cascade (MAPK) [HF96]. The Petri net model has been created with our tool Snoopy [RMH10]. The PRISM model and the reward structure were taken from the PRISM case study suite. Snoopy implements an export mechanism to create PRISM models including the computation of the same static variable orders as IDD-MC. Thus we used for our experiments Snoopy also to create another model description in the PRISM language. The used CSL formula $\mathbf{R}\{\text{“}time\text{”}\}_{=?}[\mathbf{F}(kpp = N)]$[1] was taken from [KNP08]. This formula requires the generation of the Embedded Markov Chain. To solve the linear system of equations we used the Jacobi method, the default in both tools. In our experiments we varied the number of tokens on certain places using parameter N of the scalable model, which comprises 22 places and 30 transitions. The number of reachable states represents the dimension of the rate matrix, the number of transitions represents the number of non-zero matrix entries. We used IDD-MC with one and with 16 threads.

The figures in Table 1 show that a good variable order reduces the analysis time significantly. Using all logical cores of our test system allows a speed up about factor seven. Except the very small state space for $N = 2$ and $N = 4$ IDD-MC outperforms PRISM for the given model and formula also without multi-threading, and using the same variable order.

5 Conclusion

We have presented IDD-MC, a symbolic tool for qualitative and quantitative analysis of bounded stochastic Petri nets. We summarized its main features, especially the analysis of reward structures. Experiments show that the tool

[1] To use '$_{=?}$' means to compute the reward for the initial state

N	states	transitions	IDD-MC$_1$	IDD-MC$_{16}$	PRISM$_{org}$	PRISM$_{go}$
2	2,172	13,608	1s	2s	0.7s	0.2s
4	99,535	910,872	23s	5s	10min15s	18s
6	1,373,026	15,015,264	4m26s	40s	8h48m49s	33m41s
8	10,276,461	125,012,862	30m18s	4m30s	−	6h26m32s
10	52,820,416	690,183,846	164m22s	22m30s	†	−

− means, that we aborted the experiment after 24h.

† menas, that we skipped the experiment.

Table 1. CTMC sizes for different initial markings of the MAPK cascade and the model checking times to evauluate the CSL formula $\mathbf{R}\{\text{“}time\text{”}\}_{=?}[\mathbf{F}(kpp = N)]$ with IDD-MC and PRISM.

outperforms the PRISM model checker for the considered case study. We intend to support Generalized stochastic Petri nets (GSPN) and to realize out-of-core techniques. Furthermore there are several aspects for performance optimization as improving variable ordering and parallelization.

References

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous-time Markov chains. *ACM Trans. on Computational Logic*, 1(1), 2000.

[BBC$^+$09] S. Baarir, M. Beccuti, D. Cerotti, M. De Pierro, S. Donatelli, and G. Franceschinis. The GreatSPN tool: recent enhancements. *SIGMETRICS Perform. Eval. Rev.*, 36(4):4–9, 2009.

[BHHK00] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Proc. CAV 2000*, pages 358–372. LNCS 1855, Springer, 2000.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGP01] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 2001.

[CJMS06] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. Logical and stochastic modeling with SMART. *Performance Evaluation*, 63(1), 2006.

[DSS97] Daniel Deavours, William H. Sanders, and William H. S. On-the-Fly solution techniques for stochastic petri nets and extensions. In *IEEE Transactions on Software Engineering*, pages 132–141, 1997.

[GHL07] D. Gilbert, M. Heiner, and S. Lehrack. A unifying framework for modelling and analysing biochemical pathways using Petri nets. In *Proc. CMSB 2007*, pages 200–216. LNCS/LNBI 4695, Springer, 2007.

[GKL$^+$09] S. Gaonkar, K. Keefe, R. Lamprecht, E. Rozier, P. Kemper, and W. H. Sanders. Performance and dependability modeling with Möbius. *SIGMETRICS Perform. Eval. Rev.*, 36(4):16–21, 2009.

[HDG09] M. Heiner, R. Donaldson, and D. Gilbert. *Petri Nets for Systems Biology, in Iyengar, M.S. (ed.), Symbolic Systems Biology: Theory and Methods.* Jones and Bartlett Publishers, Inc., to appear, 2009.

[HF96] C. Huang and J. Ferrell. Ultrasensitivity in the mitogen-activated protein kinase cascade. *Proc. Natl. Acad. Sci.*, 93:10078–10083, 1996.

[HGD08] M. Heiner, D. Gilbert, and R. Donaldson. Petri nets in systems and synthetic biology. In *SFM*, pages 215–264. LNCS 5016, Springer, 2008.

[HKNP06] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS 2006*, pages 441–444. Springer, LNCS 3920, 2006.

[HRSS10] M. Heiner, C. Rohr, M. Schwarick, and S. Streif. A comparative study of stochastic analysis techniques. In *Proc. CMSB 2010*. Springer, 2010.

[KNP07] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

[KNP08] M. Kwiatkowska, G. Norman, and D. Parker. Using probabilistic model checking in systems biology. *ACM SIGMETRICS Performance Evaluation Review*, 35(4):14–21, 2008.

[MP04] A. Miner and D. Parker. *Validation of Stochastic Systems: A Guide to Current Research*, chapter Symbolic Representations and Analysis of Large Probabilistic Systems, pages 296–338. LNCS 2925. Springer, 2004.

[Noa99] A. Noack. A ZBDD Package for Efficient Model Checking of Petri Nets (in German). Technical report, BTU Cottbus, Dep. of CS, 1999.

[Par02] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[PW03] L. Priese and H. Wimmel. *Theoretical Informatics - Petri Nets (in German)*. Springer, 2003.

[RMH10] C. Rohr, W. Marwan, and M. Heiner. Snoopy–a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics*, 26(7):974–975, 2010.

[Sch08] M. Schwarick. Transient Analysis of Stochastic Petri Nets with Interval Decision Diagrams. In *Proc. 15th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2008)*, volume 380 of *CEUR Workshop Proceedings*, pages 43–48. CEUR-WS.org, September 2008.

[SH09] M. Schwarick and M. Heiner. CSL model checking of biochemical networks with interval decision diagrams. In *Proc. CMSB 2009*, pages 296–312. LNCS/LNBI 5688, Springer, 2009.

[ST10] M. Schwarick and A. Tovchigrechko. IDD-based model validation of biochemical networks. *Theoretical Computer Sience*, 2010.

[Ste94] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton Univ. Press, 1994.

[Tov08] A. Tovchigrechko. *Model Checking Using Interval Decision Diagrams*. PhD thesis, BTU Cottbus, Dep. of CS, 2008.