Simulative CSL model checking of Stochastic Petri nets in IDD-MC

Christian Rohr

Magdeburg Centre for Systems Biology (MaCS), Otto von Guericke University
and Max Planck Institute for Dyn. of Complex Tech. Syst. Magdeburg, Germany
rohr@mpi-magdeburg.mpg.de

Abstract. IDD-MC is a symbolic analysis tool for bounded Stochastic Petri nets. The restriction regarding the boundedness can be circumvented by a simulative approach. Besides that, the simulation is going to be capable of handling extended Stochastic Petri nets. In this paper we report on the integration of a multi-scaling stochastic simulation engine into IDD-MC. We present some experimental results which show the efficiency of our implementation.

1 Introduction

Stochastic models are becoming more and more popular in Systems Biology and their size increases while the understanding of the modelled networks grows. Often, the analysis of these systems with exact numerical algorithms is not feasible anymore. Apart from that, stochastic models with an unbounded state space can not be analyzed with such techniques at all. Both restrictions can be circumvented by stochastic simulation.

For modelling biological systems we are using stochastic Petri nets ($\mathcal{SPN}$). So we can optimize the quantitative analysis using structural information [HGD08]. The semantics of a stochastic Petri net is defined by a Continuous Time Markov Chain (CTMC).

Generalised stochastic Petri nets ($\mathcal{GSPN}$) extend $\mathcal{SPN}$ by introducing immediate transitions. These transitions have a higher priority then stochastic transitions. This means, if an immediate and a stochastic transition are enabled at the same time, the immediate transition has to fire. One can model very fast reactions or switch like behavior with such transitions. The semantics of generalised stochastic Petri nets can be reduced to CTMC.

The class of extended stochastic Petri nets ($\mathcal{XSPN}$) is based on $\mathcal{GSPN}$, but introduces deterministic and scheduled transitions [HLGM09]. Both transition types have the same priority, which is higher than the priority of the stochastic transitions but lower than the priority of immediate transitions. These transitions are used, e.g, to model external influences, taking place at certain time points. The use of deterministically timed transitions in extended stochastic Petri nets destroys the Markovian property [Ger01]. The stochastic simulation can be adapted in a way that it can handle $\mathcal{XSPN}$.

2 Stochastic Simulation

We implemented the stochastic simulation algorithm (SSA) introduced by Gillespie in [Gil77]. The algorithm creates a single finite path through the possibly infinite CTMC. The computation of such a simulation run (trajectory, path) needs only to store the current state. The basic idea is as follows.

Given the system is at time point τ in state s. The probability that a transition $t_j \in T$ will occur in the infinitesimal time interval $[\tau, \tau + \Delta\tau)$ is given by:

$$P(\tau + \Delta\tau, t_j \mid s) = h_j(s) \cdot e^{-E(s)\cdot\Delta\tau} \tag{1}$$

For each transition t_j, the rate is given by the propensity function h_j, where $h_j(s)$ is the conditional probability that transition t_j occurs in the infinitesimal time interval $[\tau, \tau + \Delta\tau)$, given state s at time τ. So, the enabled transitions in the net compete in a race condition. The fastest one determines the next state and the simulation time elapsed. In the new state, the race condition starts anew.

Algorithm 1 Stochastic simulation algorithm.

Require: $\mathcal{SPN}$ with initial state s_0, time interval $[\tau_0, \tau_{max}]$
Ensure: state s at timepoint τ_{max}

```
 1: initRand(seed)
 2: time τ := τ_0
 3: state s := s_0
 4: while τ < τ_max do
 5:     draw random numbers r_1, r_2, uniformly distributed on [0, 1)
 6:     r_1 := getURand()
 7:     r_2 := getURand()
 8:     Δτ = − ln (r_1) / E (s)
 9:     e := 0
10:     for all transitions t_j ∈ T enabled at s do
11:         e := e + h_j(s)
12:         if e > r_2 · E (s) then
13:             s := s + Δt_j
14:             break
15:         end if
16:     end for
17:     τ := τ + Δτ
18: end while
```

The SSA simulates every transition firing (basically by using Eq. (1)) one at a time, and keeps track of the current system state. To determine the time increment $\Delta\tau$ and to select the next Petri net transition to fire requires to generate two random numbers $(r1, r2)$ uniformly distributed on $(0, 1)$. Different trajectories of the CTMC are obtained by different initializations of the random number generator (line 1). Reliable conclusions about the system behaviour require many simulations due to the stochastic variance.

The simulative processing of immediate, deterministic and scheduled transitions is rather straightforward, see [Ger01]. In short, the Algorithm 1 needs to be extended in two ways.

- After every firing of a Petri net transition (line 13), it needs to be checked whether immediate transitions got enabled. If so, these have to be processed until no more immediate transitions are enabled. This possibly leads to a time deadlock, if there exists a cyclic path of immediate transitions.
- Having calculated the next time step (line 8), it needs to be checked whether a deterministic or scheduled transition gets enabled in the time interval $[\tau, \tau + \Delta\tau]$. If yes, the one closest to τ is processed and the simulation time will be set to the value of this transition.

2.1 Model checking.

We use the Continuous Stochastic Logic (CSL) introduced by [ASSB00]. In principle the stochastic simulation algorithm allows to check any unnested, time-bounded CSL formula without the steady state operator [YS02]. The ratio of the number of fulfilling and total number of runs leads to an approximation of the desired probability.

To achieve an appropriate accuracy of the results, one has to determine the required amount of simulation runs. The method of our choice is the confidence interval as described in [SM08]. The confidence interval contains the property of interest with some predefined probability, called confidence level. This confidence level has usually values of 90%, 95%, or 99%. Assuming 95% and an accuracy of the results of 10^{-5} leads to $\approx 38,000,000$ runs.

2.2 Parallelization.

Such a high amount of independent simulation runs requires parallelization. Because of the independence of the individual simulations runs, parallelization is straightforward. It basically requires a master, which distributes the work load on n identical slaves and collects the results.

This scenario can be realized in two different ways:

Multithreading is the method of choice, if the program is meant to run on a symmetric multiprocessing (SMP) computer, where all processors have access to the main memory. The master thread creates n worker threads and sets the required work load for each of them. In the end each worker sends the results back to the master thread.

Multitasking plays its strength in a distributed memory environment like computer clusters, where not all memory is available to all processors. In our implementation we use the Message Passing Interface (MPI). That provides special communication patterns. In the beginning, n processes are created and the master uses the "broadcast" operation to distribute the work load on the processes. When the simulation is finished, the results are collected from the processes. For that purpose the "gather" operation is used.

3 Case Study

We use the abstract circadian clock model of Barkei and Leiber, introduced in [BL00]. It shows circadian rhythms which are widely used in organisms to keep a sense of daily time. More background information can be found in [VKBL02].

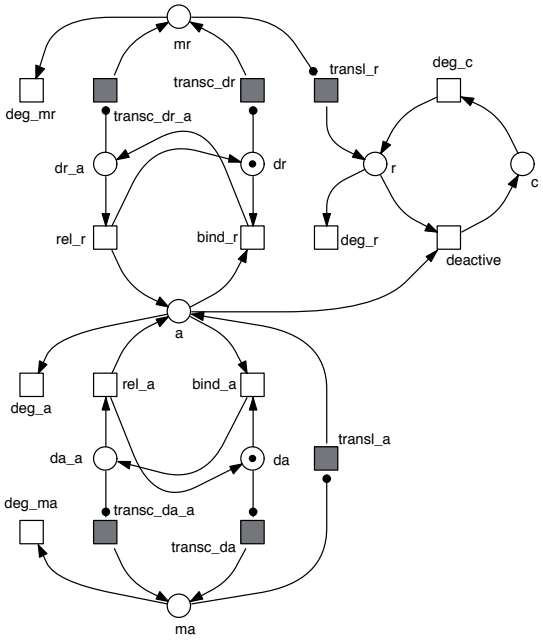

Fig. 1. $\mathcal{SPN}$ of the abstract circadian clock model

We modeled it as a stochastic Petri net (Fig. 1) containing 9 places and 16 transitions. All transitions use mass-action kinetics and the parameters were taken from [VKBL02]. The Petri net is unbounded, because once a transition with prefix "trans" got enabled (there are 6 of them) it could create an endless amount of token on its post place.

The second case study is a gene-regulation network. The Lactose-Operon model [Wil06] models the transport and metabolism of lactose in bacteria. This $\mathcal{XSPN}$ (Fig. 2) taken from [HLGM09] contains the scheduled transition "Intervention", which increases the amount of "Lactose" by 10,000 each 50,000 time units. The Petri net contains 11 places and 17 transitions and is unbounded too. All stochastic transitions use mass-action kinetics with parameters from [Wil06].

Both case studies are modeled with the generic graph editor Snoopy [RMH10]. In order to show the scalability of our implementation we decided to generate averaged traces until time point $\tau = 100$ for the circadian clock model and $\tau = 130,000$ for the lac-operon model.

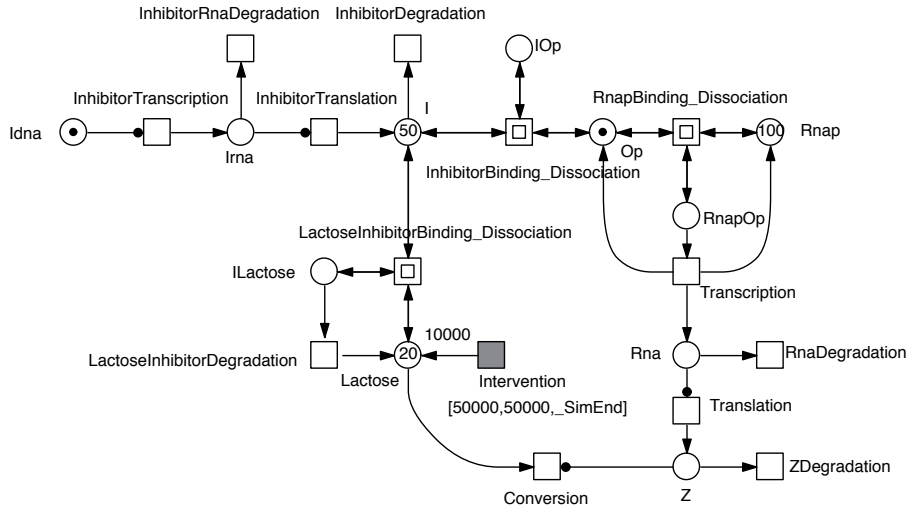

Fig. 2. $\mathcal{XSPN}$ of the lac operon model

Table 1. Comparison of the runtime for n different workers for the multi-threaded (MT) and the MPI version. The speedup is given in braces behind the time value. We created 10,000 simulation runs until $\tau = 100$ for the circadian clock model (index 1) and 1,000 simulation runs until $\tau = 130,000$ for the lac operon model (index 2).

	Circadian clock		Lac operon	
n	MT_1	MPI_1	MT_2	MPI_2
1	15m43s (1×)	20m27s (1×)	5m34s (1×)	7m12s (1×)
2	7m52s (2×)	9m55s (2.1×)	2m49s (2×)	3m33s (2×)
4	4m05s (3.8×)	5m08s (4×)	1m32s (3.6×)	1m50s (3.9×)
8	2m50s (5.5×)	2m33s (8×)	59s (5.7×)	56s (7.7×)
12	2m05s (7.5×)	1m42s (12×)	45s (7.4×)	37s (11.7×)
16	1m42s (9.2×)	1m18s (15.7×)	40s (8.4×)	31s (13.9×)

The experiments considering the multi-threaded version of the simulation engine were done on a 2.26 GHz Apple Mac Pro with 32 GB RAM and eight physical (with hyper-threading 16 logical) cores. IDD-MC was build on Mac OS X 10.5.8 as 64-bit application. The experiments with the MPI version were done on a cluster with 96 cores distributed on 24 nodes. It was build on CentOS 5.5 as 64-bit application.

Table 1 shows the result of our experiments. The runtime decreases well with an increasing number of workers. The scaling of the multi-threaded version is correlating with the number of workers up to $n = 4$, after that it goes down a bit. This is due to the 2 processors, each with 4 cores and 8 threads. The MPI version scales linearly over all settings.

4 Conclusion

We extended the analysis capabilities of IDD-MC [SH09] by implementing a stochastic simulation engine. We verified the scalability of the parallelized versions, using multithreading; and multitasking.

For the time being the stochastic simulation only provides transient analysis and generation of averaged traces. In the future we intend to support unnested time-bounded CSL formulas without steady state operator. We plan to closer investigate the possibilities of computing the steady state using stochastic simulation.

IDD-MC is available for non-commercial use [IDD10]; it includes the MT version. The MPI version is available on request.

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.

[BL00] N. Barkai and S. Leibler. Biological rhythms: Circadian clocks limited by noise. *Nature*, 403:267–268, 2000.

[Ger01] R. German. *Performance analysis of communication systems with non-Markovian stochastic Petri nets*. Wiley, 2001.

[Gil77] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340 – 2361, December 1977.

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

[HLGM09] M. Heiner, S. Lehrack, D. Gilbert, and W. Marwan. Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments. pages 138–163. LNCS/LNBI 5750, Springer, 2009.

[IDD10] IDD-MC Website. A model checker for the Continuous Stochastic Logic (CSL) of stochastic Petri nets. BTU Cottbus, http://www-dssz.informatik.tu-cottbus.de/software/iddcsl/latest/iddcsl.html, 2010.

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

[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.

[SM08] W. Sandmann and C. Maier. On the statistical accuracy of stochastic simulation algorithms implemented in Dizzy. In *Proc. WCSB 2008*, pages 153–156, 2008.

[VKBL02] J. Vilar, H.-Y. Kueh, N. Barkai, and S. Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proc. National Academy of Sciences of the United States of America*, 99(9):5988–5992, 2002.

[Wil06] D.J. Wilkinson. *Stochastic Modelling for System Biology*. CRC Press, New York, 1st Edition, 2006.

[YS02] H. Younes and R. Simmons. Probabilistic verification of descrete event systems using acceptance sampling. In *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 223–235. LNCS 2404, Springer, 2002.