Simulative Model Checking of Steady State and Time-Unbounded Temporal Operators

Christian Rohr

Brandenburg University of Technology Cottbus,
Chair of Data Structures and Software Dependability,
Postbox 10 13 44, D-03013 Cottbus, Germany,
rohrch@tu-cottbus.de,
http://www-dssz.informatik.tu-cottbus.de

Abstract. When working with large stochastic models simulation remains the only possible analysis technique. Therefore, simulative model checking is the way to go. While finite time horizon algorithms are well known for probabilistic linear-time temporal logic, we provide an infinite time horizon procedure as well as steady state computation, based on exact stochastic simulation algorithms. We demonstrate the approach on models of the RKIP inhibited ERK pathway and angiogenetic process.

Keywords: simulative model checking, stochastic Petri net, steady state, unbounded temporal operator, probabilistic linear-time temporal logic

1 Introduction

Stochastic modelling of biochemical reaction networks is getting more and more popular. This also increases the demand for efficient analysis of such models. While small and medium-sized models can be analysed numerically, we focus on large or unbounded models. Therefore we use stochastic simulation to overcome the problem of state space explosion.

We use stochastic Petri nets ($\mathcal{SPN}$) [1] as modelling paradigm, which gives us a complete formalised and standardised framework, as well as an intuitive way of modelling concurrent behaviour. A number of biochemical species N involved in the biological model are represented as places $p_1 \dots p_N$, and the reactions between them refer to the transitions $t_1 \dots t_M$. The kinetics of a reaction is defined as possibly state-dependent rate function h_t assigned to the transition. Places and transitions are connected via directed arcs. Each arc contains the stoichiometric value of the associated species. The semantics of such an $\mathcal{SPN}$ is defined as continuous-time Markov chain (CTMC).

The dynamic behaviour of stochastic models can be analysed in different ways. We showed in [2] that numerical analysis is currently efficient up to 1×10^9 states. Beyond this limit, stochastic simulation remains the only possible technique. Stochastic simulation may be performed with approximate or exact methods. An approximate method is τ-leaping [3], which generates an approximate realisation of the stochastic process. Its advantage is the ability to jump over

several transitions and thus be more efficient in trace generation than exact methods. But for simulative model checking, we need to know the exact occurrences of each transition. Therefore, only exact simulation algorithms are suitable for the purpose of simulative model checking, like Gillespie's direct method [4] or the next reaction method [5] by Gibson & Bruck.

In this paper, we extend the finite time horizon model checking algorithm of probabilistic linear-time temporal logic to an infinite time horizon and provide an algorithm to compute simulatively steady state formulas.

2 Stochastic Simulation

In biochemical reaction networks (with n molecular species and k reactions), the molecular reactions between the species are random processes, because it is impossible to predict the time at which the next reaction will occur. Stochastic modelling has therefore become an important tool to fully understand the system behaviour of such reaction networks.

The stochasticity can be described in a time-dependent manner by the Chemical Master Equation. In probability theory, this identifies the evolution as a continuous-time Markov chain (CTMC), with the integrated master equation obeying a Chapman-Kolmogorov equation. When working with biological systems, it may be infeasible to set up the CTMC as the state space $\mathcal{X} \subseteq \mathbb{N}^n$ can be very large or even infinite. The largeness of CTMCs makes simulation an important analysis technique: instead of computing the CTMC directly, simulation aims at imitating the CTMC by generating different paths of the CTMC, i.e., a sequence of discrete random variable $X_l(t)$. The discrete random variable $X_l(t)$ describes the number of molecules of species S_l, $l \in \{1, \ldots, n\}$ present at time t. The system state at time t is thus a discrete n-dimensional random vector $X(t) = (X_1(t), \ldots, X_n(t)) \in \mathcal{X}$. Given the system is in state $X(t)$, the probability that a transition/reaction of type $j \in \{1, \ldots, k\}$ will occur in the infinitesimal time interval $[t + \tau, t + \tau + \mathrm{d}\tau)$ is given by:

$$P(t + \tau, j | X(t))\mathrm{d}\tau = a_j(X(t)) \exp\left(-a_0(X(t))\tau\right) \mathrm{d}\tau$$

For each reaction j, the rate is given by the propensity function a_j, where $a_j(x)\mathrm{d}\tau$ is the conditional probability that a reaction of type j occurs in the infinitesimal time interval $[t, t + \mathrm{d}\tau)$, given state $X(t)$ at time t. The sum of the propensities of all possible transitions in the current state $X(t)$ is given by $a_0(X)$. Thus, the different (enabled) transitions in the net compete in a race condition and the fastest one determines next state and the time elapsed. In the new state, the race condition is started anew.

To analyse or understand the behaviour of a biochemical reaction network, many trajectories need to be simulated for a good approximation of the underlying CTMC. Although in principle known a long time before, Gillespie was the first who developed a supporting theory for a stochastic simulation of chemical kinetics [4]. He presented the Stochastic Simulation Algorithm (SSA; often also called Gillespie's algorithm), which is a Monte Carlo procedure for numerically

generating CTMC. Since Gillespie's seminal work, several variants and different implementations and optimisations of the SSA have been proposed. Basically, each variant performs the following steps:

1. Initialise time $t = t_0$ and the system's state X at time t_0.
2. Repeat:
   (a) determine time increment $\tau \in \mathbb{R}$
   (b) select next reaction type j depending on the current state $X(t)$
   (c) perform state transition imposed by reaction of type j and update state vector X
   (d) update time $t = t + \tau$.
   until simulation time is reached.

The SSA simulates every state transition event, one at a time, and updates the system after each state transition. To determine the time increment τ and to select the next reaction requires to generate random numbers. Different realisations of the CTMC are obtained by different initialisations of the random number generator. Since reliable statements about the system behaviour (variance) can only be made based on many simulations, the usefulness of the simulation approach depends on the simulation time for each individual trajectory. Accelerating simulations is therefore desirable without changing the basic ideas of the algorithm.

Many variants of the SSA aim at reducing the computational cost of selecting the next reaction that will occur. Cao et al. [6] keep the reactions with larger propensities at the beginning of the list. The position of each reaction in the list is thereby determined after some pre-simulations. McCollum et al. [7] maintain a loosely sorted order of the reactions as the simulation proceeds. Instead of arranging the reactions in a linear list, Gibson & Bruck [5] propose to use advanced data structures (trees) to speed up the search for the next reaction that will occur. However, the time to manage the advanced data structures partially compensates the speed-up due to faster search [8].

Further performance increases of the SSA are obtained if only those propensities are recalculated that actually have changed after a state transition, whereas all others are reused (e.g., [8, 5]).

An additional speed-up to the SSA is provided by the approximate method τ-leaping [3], in which time t is advanced by a preselected amount τ and the numbers of firings of the individual transitions during the time interval $[t, t+\tau)$ are approximated by Poisson random numbers. Thus, instead of (sequentially) tracing every single state transition, several reactions are executed in parallel. With τ-leaping, it is assumed that all propensity functions are approximately constant in $[t, t + \tau)$, which is referred to as the leap condition. To ensure this, it is important to select τ sufficiently small, but also large enough to accelerate simulation.

3 Model Checking

Stochastic simulation produces traces through the state space of the model. For the specification of temporal formulas we define the probabilistic extension of

the Linear-time Temporal Logic (LTL) [9] with numerical constraints [10], which is called Probabilistic Linear-time Temporal Logic with numerical constraints (PLTLc) [11]. The grammar of all PLTLc formulas is given in Definition 1.

Definition 1. *Probabilistic Linear-time Temporal Logic with Constraints*

$$
\begin{aligned}
\psi &:= \mathcal{P}_{\bowtie x}\left[\phi\right] \mid \mathcal{P}_{=?}\left[\phi\right] \\
&\quad \bowtie \in \{<, \leq, \geq, >\}\,,\ x \in [0,1] \\
\phi &:= \mathbf{X}^{\,I}\phi \mid \mathbf{F}^{\,I}\phi \mid \mathbf{G}^{\,I}\phi \mid \phi\ \mathbf{U}^{\,I}\phi \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \sigma \\
I &:= [x_1, x_2] = \left\{x \in \mathbb{R}^{+} \mid\ x_1 \leq x \leq x_2\right\},\ \mathit{omit}\ I = [0, \infty) \\
\sigma &:= \neg\sigma \mid \sigma \wedge \sigma \mid \sigma \vee \sigma \mid \mathit{value} \trianglelefteq \mathit{value} \mid \mathit{true} \mid \mathit{false} \\
&\quad \trianglelefteq \in \{<, \leq, \geq, >, =, \neq\} \\
\mathit{value} &:= \mathit{value} \sim \mathit{value} \mid \mathit{Place} \mid \$\mathit{Variable} \mid \mathit{Int} \mid \mathit{Real} \mid \mathit{function} \\
&\quad \sim \in \{+, -, *, /\}
\end{aligned}
$$

The probability operator $\mathcal{P}$ has two different modes. If it is used with the question mark as $\mathcal{P}_{=?}\left[\phi\right]$ then it will return the probability $Pr(\phi)$ that ϕ is true. In the second case, $\mathcal{P}_{\bowtie x}\left[\phi\right]$ returns *true*, if $Pr(\phi) \bowtie x$ is fulfilled, *false* otherwise. In simulative model checking we compute a confidence interval (*c.i.*); in consequence of that, we have to introduce an additional return value in the second case. For simplicity, we assume the *c.i.* to have a lower and an upper bound $B_l, B_u \in \mathbb{R}_{\geq 0}$, such that the probability $Pr(\phi)$, which is not known in our case, is $B_l \leq Pr(\phi) \leq B_u$.

$$
\mathcal{P}_{\bowtie x}[\phi] = \begin{cases} \mathit{true} & \text{if } x \bowtie [B_l, B_u] \wedge x \notin [B_l, B_u] \\ \mathit{false} & \text{if } x \not\bowtie [B_l, B_u] \wedge x \notin [B_l, B_u] \\ \mathit{unknown} & \text{if } x \in [B_l, B_u] \end{cases}
$$

The operators $\neg$, $\wedge$, $\vee$ are the standard boolean operators *not*, *and*, *or*. Whereas $\mathbf{X}$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{U}$ denote the temporal operators *NEXT*, *FINALLY*, *GLOBALLY* and *UNTIL*. The *NEXT* operator ($\mathbf{X}^{\,I}\phi$) refers to *true* in the next state and within the time interval I. The *UNTIL* operator ($\phi_1\,\mathbf{U}^{\,I}\phi_2$) indicates that a state where ϕ_2 holds is eventually reached within the time interval I, while ϕ_1 continuously holds. The *FINALLY* operator ($\mathbf{F}^{\,I}\phi$) means that at some point within the time interval I a state where ϕ holds is eventually reached. Whereas the *GLOBALLY* operator ($\mathbf{G}^{\,I}\phi$) refers to the condition ϕ continuously holding true within the time interval I. The latter two are syntactic sugar, as they rely on the equivalences $\mathbf{F}\,\phi \equiv \mathit{true}\ \mathbf{U}\,\phi$ and $\mathbf{G}\,\phi \equiv \neg\mathbf{F}\,\neg\phi$.

A trace T fulfils a linear-time temporal logic formula ϕ if the following $\models$ relations hold:

$$
\begin{aligned}
T \models \mathbf{X}\,\phi &\iff T^{(1)} \models \phi \\
T \models \mathbf{F}\,\phi &\iff \exists i \in \mathbb{N} : T^{(i)} \models \phi \\
T \models \mathbf{G}\,\phi &\iff \forall i \in \mathbb{N} : T^{(i)} \models \phi \\
T \models \phi_1\,\mathbf{U}\,\phi_2 &\iff \exists i \in \mathbb{N} : T^{(i)} \models \phi_2 \text{ and } \forall j \in \mathbb{N} \wedge j < i : T^{(j)} \models \phi_1 \\
T \models \neg\phi &\iff T \not\models \phi \\
T \models \phi_1 \wedge \phi_2 &\iff T \models \phi_1 \wedge T \models \phi_2 \\
T \models \phi_1 \vee \phi_2 &\iff T \models \phi_1 \vee T \models \phi_2 \\
T \models v_1 \trianglelefteq v_2 &\iff \mathit{evalState}(v_1, T^{(i)}) \trianglelefteq \mathit{evalState}(v_2, T^{(i)})
\end{aligned}
$$

The function $\mathit{evalState}(v, T^{(i)})$ assigns a numerical value to the expression v by looking up the tokens that each place $x \in P(v)$ has in state $T^{(i)}$ of trace T.

In the next sections we present an algorithm to compute steady state formulas, and afterwards an algorithm to compute time-unbounded temporal operators in a simulative manner. Time-bounded algorithms for simulative model checking are well known, i.e., [10].

3.1 Steady State Computation

In steady state simulation, the measures of interest are defined as limits, as the length of the simulation goes to infinity. There is no natural event to terminate the simulation, so the length of the simulation is made large enough to get "good" estimates of the quantities of interest. Steady-state simulation generally poses two problems:

1. The existence of a transient phase may cause the estimate to be biased.
2. The simulation runs are long, and usually one cannot afford to carry out many independent simulations.

These are several methods that allow to cope with these problems to some extent. Among them are: the batch means method, the method of independent replicas, and the regeneration method. Each of these methods has its advantages and disadvantages. In our implementation we use a sample batch means algorithm to compute the steady state.

We choose Skart [12], which is an automated sequential procedure for on-the-fly steady state simulation output analysis, because it is specifically designed to handle observation-based statistics and usually requires a smaller initial sample size compared with other well-known simulation analysis procedures [12]. This algorithm partitions a long simulation run into batches, computes an average statistics for each batch and constructs an interval estimate using the batch means. Based on this interval estimate Skart decides whether a steady state is reached or more samples were needed. A detailed description of the algorithm is given in [12].

We extend PLTLc with the steady state operator $\mathcal{S}$. Definition 2 states the syntax of it. The return values are quite the same as for the probability operator $\mathcal{P}$. Inside of $\mathcal{S}$ are only state formulas allowed, i.e., no temporal operators.

Definition 2. *Extension of PLTLc with steady state operator $\mathcal{S}$.*

$$\psi := \mathcal{P}_{\bowtie x}\,[\phi] \mid \mathcal{P}_{=?}\,[\phi] \mid \mathcal{S}_{\bowtie x}\,[\sigma] \mid \mathcal{S}_{=?}\,[\sigma]$$
$$\bowtie \in \{<, \leq, \geq, >\}\,,\ x \in [0,1]$$

Steady-state formulas are computed with Algorithm 1. At first the simulation trace is created until the steady state is reached (line 4–8). To get an unbiased result, we cut off the first n states, which bias the steady state (line 9). The steady state probability is now the ratio T_o/T_s of the occupation time T_o of the fulfilling states and the simulation time T_s (line 11–19). But this gives correct results only for those Petri nets, where the reachability graph consists of only one strongly connected component. The complexity of this decision is the same as for constructing the reachability graph. To solve this problem, one has to make several steady state computations and average the values. In that way the individual steady state estimates are weighted according to the strongly connected components.

Algorithm 1 Steady state computation for one simulation run

Require: $\boldsymbol{trace} \leftarrow (m_0,\, t_0)$
1: **procedure** evalSteadyState(σ)
2: $steadyStateReached \leftarrow false$
3: $pos \leftarrow 0$
4: **repeat**
5: $\boldsymbol{trace} \leftarrow \boldsymbol{trace}$ + nextState($\boldsymbol{trace}^{(pos)}$)
6: $pos \leftarrow pos + 1$
7: $steadyStateReached \leftarrow$ checkSteadyState($\boldsymbol{trace}$)
8: **until** $steadyStateReached = true$
9: $cutOff \leftarrow$ getSteadyStateCutOff
10: $T_o \leftarrow 0,\ T_s \leftarrow 0$
11: **for** $i \leftarrow cutOff,\ |\boldsymbol{trace}|$ **do**
12: $(s_i, t_i) \leftarrow \boldsymbol{trace}^{(i)}$ ▷ state s_i, sojourn time t_i in s_i
13: $T_s \leftarrow T_s + t_i$
14: $res \leftarrow$ evalState(σ, s_i)
15: **if** $res = true$ **then**
16: $T_o \leftarrow T_o + t_i$
17: **end if**
18: **end for**
19: **return** T_o/T_s
20: **end procedure**

3.2 Verification of Time-Unbounded Until

In Algorithm 2 we present the algorithm for checking time-unbounded until formulas. It is nearly the same as for time-bounded formulas except that one has to stop the simulation trace at some time point. The decision of doing that is the crucial part of the algorithm. We assume that reaching the steady state is a reasonable stopping criteria (line 38). A system in a steady state has numerous properties that are unchanging in time. This implies that for any property p of the system, the partial derivative with respect to time is zero:

$$\frac{\partial p}{\partial t} = 0$$

If a system is in steady state, then the recently observed behaviour of the system will continue into the future. In stochastic systems, the probabilities that various states will be repeated will remain constant.

4 MARCIE: An Implementation

MARCIE [13] is a tool for analysing generalised stochastic Petri nets ($\mathcal{GSPN}$), supporting qualitative and quantitative analyses including model checking capabilities. Particular features are symbolic state space analysis including efficient saturation-based state space generation, evaluation of standard Petri net properties as well as Computational Tree Logic model checking. Further it offers symbolic Continuous Stochastic Logic model checking and permits to compute expectations for rewards which can be added to the core $\mathcal{GSPN}$. Most of MARCIE's features are realised on top of an Interval Decision Diagram (IDD) implementation [14]. 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. However, it additionally comprises approximative and simulative engines, which work explicitly, to support also stochastic analysis of very large and unbounded nets. It includes two exact simulation algorithms, firstly Gillespie's direct method [4], and secondly the next reaction method by Gibson & Bruck [5].

Parallelising the simulation is a good way to speed-up the computation. We use the Message Passing Interface (MPI) to develop a portable and scalable simulation tool for large-scale models. The desired number of simulations runs is equally distributed to the worker processes. Therefore the run-time decreases with the number of workers.

MARCIE is completely written in C++, and makes use of the libraries GMP, pthreads, flex/bison and boost. It comprises about 45,000 lines of source code. MARCIE is available for non-commercial use; we provide statically linked binaries for Linux and Mac OS X. The tool, the manual and a benchmark suite can be found on our website http://www-dssz.informatik.tu-cottbus.de/marcie.html. MARCIE itself comes with a textual user interface. 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 a XML description. The GUI is part of our Petri net analyser Charlie [15].

Algorithm 2 Unbounded Until for one simulation run

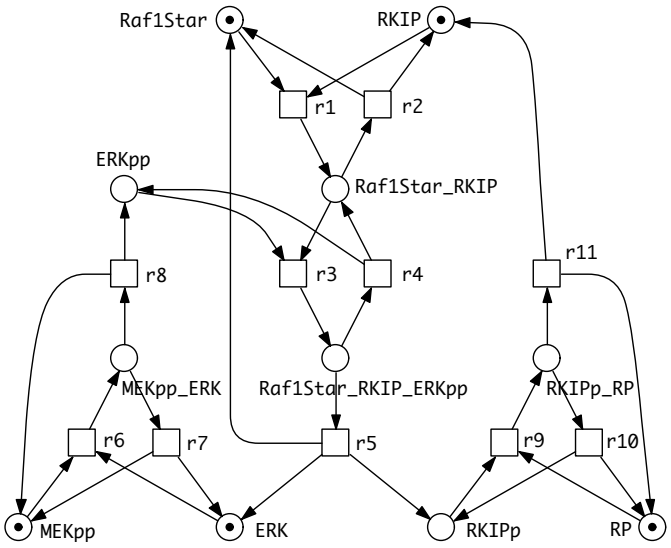
Require: $\boldsymbol{trace} \leftarrow (m_0,\, t_0)$
1: **procedure** EVALFORMULA(ϕ, pos, $\boldsymbol{trace}$)
2: $steadyStateReached \leftarrow false,\ res \leftarrow false$
3: **repeat**
4: switch ϕ
5: case σ :
6: $(s_{pos}, t_{pos}) \leftarrow \boldsymbol{trace}^{(pos)}$
7: $res \leftarrow$ EVALSTATE(σ, s_{pos})
8: **return** (pos, res)
9: case $\neg\phi_1$:
10: (pos_1, res_1) $\leftarrow$ EVALFORMULA(ϕ_1, pos, $\boldsymbol{trace}$)
11: **return** (pos_1, $\neg res_1$)
12: case $\phi_1 \wedge \phi_2$:
13: (pos_1, res_1) $\leftarrow$ EVALFORMULA(ϕ_1, pos, $\boldsymbol{trace}$)
14: (pos_2, res_2) $\leftarrow$ EVALFORMULA(ϕ_2, pos, $\boldsymbol{trace}$)
15: **return** ($min(pos_1, pos_2)$, $res_1 \wedge res_2$)
16: case $\phi_1 \vee \phi_2$:
17: (pos_1, res_1) $\leftarrow$ EVALFORMULA(ϕ_1, pos, $\boldsymbol{trace}$)
18: (pos_2, res_2) $\leftarrow$ EVALFORMULA(ϕ_2, pos, $\boldsymbol{trace}$)
19: **return** ($max(pos_1, pos_2)$, $res_1 \vee res_2$)
20: case $X\phi_1$:
21: **if** $pos = |\boldsymbol{trace}|$ **then**
22: $\boldsymbol{trace} \leftarrow \boldsymbol{trace}$ + NEXTSTATE($\boldsymbol{trace}^{(pos)}$)
23: **end if**
24: $pos \leftarrow pos + 1$
25: (pos_1, res_1) $\leftarrow$ EVALFORMULA(ϕ_1, pos, $\boldsymbol{trace}$)
26: **return** (pos_1, res_1)
27: case $\phi_1 U \phi_2$:
28: (pos_2, res_2) $\leftarrow$ EVALFORMULA(ϕ_2, pos, $\boldsymbol{trace}$)
29: **if** $res_2 = true$ **then**
30: **return** (pos_2, res_2)
31: **end if**
32: (pos_1, res_1) $\leftarrow$ EVALFORMULA(ϕ_1, pos, $\boldsymbol{trace}$)
33: **if** $res_1 = false$ **then**
34: **return** (pos_1, res_1)
35: **end if**
36: $pos \leftarrow pos_1$
37: end switch
38: $steadyStateReached \leftarrow$ CHECKSTEADYSTATE($\boldsymbol{trace}$)
39: **if** $pos = |\boldsymbol{trace}|$ **then**
40: $\boldsymbol{trace} \leftarrow \boldsymbol{trace}$ + NEXTSTATE($\boldsymbol{trace}^{(pos)}$)
41: **end if**
42: $pos \leftarrow pos + 1$
43: **until** $steadyStateReached = true$
44: **return** (pos, res)
45: **end procedure**

5 Case Studies

In this section we demonstrate our approach on the models of the RKIP inhibited ERK pathway and angiogenetic process. All Petri nets were modelled with Snoopy [16] and analysed with MARCIE [13]. The experiments were carried out on a machine with 4x AMD Opteron™ 6276 with 2.3 GHz and 256GB RAM running CentOS 6.

5.1 RKIP inhibited ERK pathway

This model shows the influence of the Raf Kinase Inhibitor Protein (RKIP) on the Extracellular signal Regulated Kinase (ERK) signalling pathway. A model of non-linear ordinary differential equations was originally published in [17]. Later on, it was discussed as qualitative and continuous Petri nets in [18], and as three related Petri net models in [19]. The stochastic Petri net $\mathcal{SPN}_{ERK}$ comprises 11 places and 11 transitions connected by 34 arcs and is shown in Fig. 1. All



$$\text{Raf1Star} + \text{RKIP} \overset{r1,r2}{\leftrightarrow} \text{Raf1Star\_RKIP}$$
$$\text{Raf1Star\_RKIP} + \text{ERKpp} \overset{r3,r4}{\leftrightarrow} \text{Raf1Star\_RKIP\_ERKpp}$$
$$\text{Raf1Star\_RKIP\_ERKpp} \overset{r5}{\rightarrow} \text{Raf1Star} + \text{ERK} + \text{RKIPp}$$
$$\text{ERK} + \text{MEKpp} \overset{r6,r7}{\leftrightarrow} \text{MEKpp\_ERK} \overset{r8}{\rightarrow} \text{ERKpp} + \text{MEKpp}$$
$$\text{RKIPp} + \text{RP} \overset{r9,r10}{\leftrightarrow} \text{RKIPp\_RP} \overset{r11}{\rightarrow} \text{RKIP} + \text{RP}$$

Fig. 1. Stochastic Petri net of the RKIP inhibited ERK pathway, including textual representation of the chemical reactions.

transition rate functions use mass action kinetics with the original parameter

values from [17]. The model is scalable by the initial amount of tokens in the places RKIP, MEKpp, ERK and RP. The more initial tokens on each of these places, the bigger the state space of the Petri net. Table 1 shows the number of reachable states for different initial markings.

Table 1. The size of the state space for different initial markings of $\mathcal{SPN}_{ERK}$ computed with MARCIE's symbolic state space generation. All places which carry one token in Fig. 1 have now initially N tokens.

N	|states|	N	|states|	N	|states|	N	|states|
5	1,974	20	1,696,618	40	79,414,335	100	1.591×10^{10}
10	47,047	25	5,723,991	50	2.834×10^{8}	250	3.582×10^{12}
15	368,220	30	15,721,464	60	8.114×10^{8}	500	2.231×10^{14}

We first check the reachability of a state at some time in the future, such that the number of tokens on place *MEKpp* is between 60% and 80% of N:

$$\mathcal{P}_{=?}\left[\mathbf{F}\ \left[MEKpp \geq N \cdot 0.6 \wedge MEKpp \leq N \cdot 0.8\right]\right].$$

In any case such a state was reached, therefore the probability of the formula is 1, see Table 2.

Table 2. Reachability analysis for different initial markings N of $\mathcal{SPN}_{ERK}$. The total time is given for different numbers of workers.

N	1	2	4	8	16	32	64	result
20	0m56s	0m28s	0m14s	0m7s	0m3s	0m2s	0m0s	[1,1]
30	1m17s	0m38s	0m19s	0m10s	0m4s	0m3s	0m1s	[1,1]
40	1m38s	0m48s	0m24s	0m12s	0m6s	0m3s	0m1s	[1,1]
50	1m57s	0m59s	0m30s	0m15s	0m7s	0m4s	0m2s	[1,1]
60	2m23s	1m9s	0m35s	0m17s	0m8s	0m5s	0m3s	[1,1]

Since we know now that such a state is eventually reached, we want to compute the steady state probability of being in such a state, where the number of tokens on place *MEKpp* is between 60% and 80% of N:

$$\mathcal{S}_{=?}\left[MEKpp \geq N \cdot 0.6 \wedge MEKpp \leq N \cdot 0.8\right].$$

This is the same property as in [2], so we can verify the correctness of the results. The results in Table 3 show first that the resulting confidence interval covers the probability computed by the Jacobi method in [2]. Second the algorithm scales nearly linear with the number of worker processes. A very interesting behaviour regards the relationship between the state space size and the total run-time of the computation. One could expect an increase of the run-time, but it stays the same. This is a result of the level semantics described in [20].

Table 3. Steady state analysis for different initial markings N of $\mathcal{SPN}_{ERK}$. The total time is given for different numbers of workers.

N	1	2	4	8	16	32	64	result
20	7m31s	3m46s	1m53s	0m57s	0m27s	0m17s	0m10s	[0.77482, 0.77534]
30	7m34s	3m43s	1m51s	0m57s	0m28s	0m17s	0m11s	[0.83277, 0.83325]
40	7m40s	3m43s	1m56s	0m57s	0m28s	0m18s	0m11s	[0.87416, 0.87470]
50	7m43s	3m57s	1m57s	1m0s	0m30s	0m17s	0m11s	[0.90437, 0.90486]
60	7m53s	3m59s	1m56s	1m1s	0m28s	0m17s	0m12s	[0.92641, 0.92696]

5.2 Angiogenesis

Angiogenesis is a complex phenomenon that goes from a molecular level to macroscopic events. This Petri net models a part of the signal transduction pathway involved in the angiogenetic process and was originally published in [21]. The stochastic Petri net $\mathcal{SPN}_{ANG}$ comprises 39 places and 64 transitions connected by 185 arcs.

The model is scalable by the initial amount of tokens in the places *Akt*, *DAG*, *Gab1*, *KdStar*, *Pip2*, *P3k*, *Pg* and *Pten*. The more initial tokens on each of these places, the bigger the state space of the Petri net. The number of reachable states for different initial markings are shown in Table 4. As in the previous case

Table 4. The size of the state space for different initial markings of $\mathcal{SPN}_{ANG}$ computed with MARCIE's symbolic state space generation. The places *Akt*, *DAG*, *Gab*1, *KdStar*, *Pip*2, *P*3*k*, *Pg* and *Pten* carry initially N tokens.

N	|states|	N	|states|	N	|states|	N	|states|
1	96	4	2,413,480	7	2.181×10^{9}	10	4.537×10^{11}
2	5,384	5	29,224,050	8	1.464×10^{10}	15	5.207×10^{14}
3	144,188	6	277,789,578	9	8.623×10^{10}	20	1.428×10^{17}

study we check for reachability first. Now we want to know the probability of eventually reaching a state where no tokens reside on place *Akt*:

$$\mathcal{P}_{=?}\left[\mathbf{F}\ [Akt = 0]\right].$$

In contrast to $\mathcal{SPN}_{ERK}$, Table 5 shows that the probability ranges from about $0.44\,(N = 1)$ to $0.9\,(N = 6)$. That means a state where no tokens lay on place *Akt* is not always reached, because the CTMC consists of several strongly connected components and in some of them such a state does not exist. Secondly we compute the steady state probability of being in a state that has no tokens on place *Akt*:

$$\mathcal{S}_{=?}\left[Akt = 0\right].$$

The results in Table 6 show that the steady state probability is nearly the same as in the reachability case as the overall steady state probability consists of

Table 5. Reachability analysis for different initial markings N of $\mathcal{SPN}_{ANG}$. The total time is given for different numbers of workers.

N	1	2	4	8	16	32	64	result
1	12m10s	6m13s	3m3s	1m29s	0m49s	0m25s	0m13s	[0.44542, 0.44642]
2	60m19s	30m18s	14m47s	7m14s	3m36s	1m52s	1m17s	[0.81292, 0.81370]
3	65m37s	35m41s	18m31s	8m20s	4m2s	2m1s	1m55s	[0.94319, 0.94365]
4	74m43s	37m14s	19m52s	9m45s	4m58s	2m18s	3m0s	[0.98189, 0.98216]
5	79m45s	38m37s	18m54s	9m20s	4m37s	2m22s	3m35s	[0.99380, 0.99396]
6	79m37s	39m57s	19m50s	9m14s	4m34s	2m11s	3m3s	[0.99760, 0.99770]

two parts, first the probability of reaching a strongly connected component and second the steady state probability inside these component. The result means the steady state probability inside a strongly connected component, where a state exists with $Akt = 0$, is almost 1. That's why the overall steady state probability almost coincides with the reachability probability.

Table 6. Steady state analysis for different initial markings N of $\mathcal{SPN}_{ANG}$. The total time is given for different numbers of workers.

N	1	2	4	8	16	32	64	result
1	7m1s	3m17s	1m42s	0m47s	0m26s	0m13s	0m9s	[0.43773, 0.44771]
2	28m25s	14m10s	6m54s	3m33s	1m34s	0m51s	0m36s	[0.80446, 0.81237]
3	58m42s	30m19s	17m54s	7m32s	3m46s	2m30s	1m20s	[0.92772, 0.93284]
4	94m27s	49m59s	24m39s	11m53s	6m11s	3m25s	2m11s	[0.97859, 0.98140]
5	133m56s	66m44s	33m24s	17m22s	8mm44s	4m49s	3m12s	[0.98923, 0.99121]
6	170m57s	85m25s	42m54s	22m12s	11m13s	6m57s	3m30s	[0.99649, 0.99758]

6 Related work

To compute the transient probability of the formula $\mathcal{P}_{=?}[\phi_1 \,\mathbf{U}\, \phi_2]$ in state s means to compute the probability distribution starting in s and making states absorbing, which satisfy $\neg\phi_1 \vee \phi_2$. The resulting linear system of equations can be solved numerically by iterative methods like Gauss-Seidel or Jacobi. There are several tools available that support such solvers, among them MARCIE [13]. The drawback of numerical solvers is their restriction to bounded CTMCs and the complexity is typically $\mathrm{O}(n)$ and in worst case $\mathrm{O}(n^2)$. On the other hand they compute an "exact" result. The same methods were used to compute the steady state distribution of bounded CTMCs for computing the steady state probability of formulas like $\mathcal{S}_{=?}[\phi]$.

Statistical model checking [22–24] is a quite similar approach to simulative model checking, but differs in some details. Hypothesis testing, i.e., sequential

probability ratio test (SPRT), has good performance compared to the computation of point estimates, but it can only check formulas like $\mathcal{P}_{\bowtie x}$. In the end, the user gets a result of *true* or *false* and has no idea of the scale of the estimated probability.

The Monte Carlo Model Checker MC2 [11] computes a point estimate of a Probabilistic LTL logic (with numerical constraints) formula to hold for a model. MC2 does not include any simulation engine but works offline by taking a set of sampled trajectories generated by any simulation or ODE solver software.

Last not least, a combination of simulation and reachability analysis were used to compute time-unbounded formulas in [22, 25]. But this approach suffers from the same restrictions of bounded state spaces as the numerical methods.

7 Conclusions

In this paper we presented an infinite time horizon model checking algorithm plus steady state operator for probabilistic linear-time temporal logic. We verified the results of the simulative approach against the numerical solutions of the Jacobi and Gauss-Seidel methods. We proved the efficiency of our algorithm and the scalability by using several worker processes through MPI.

As our algorithm is based on stochastic simulation, its run-time does not directly depend on the size of the state space, as for the numerical methods, but on the rate functions of the transitions and the structural size of the Petri net. That is the greater the sum of the transitions rates, the smaller the time steps are, and the more simulation steps need to be done to reach a certain time point.

The main drawback of simulation-based methods remains. The achieved accuracy depends on the number of simulation runs and grows exponentially with the expected accuracy. Therefore methods of choice for bounded and medium-sized models are numerical, otherwise simulation plays out its strength.

References

1. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley Series in Parallel Computing, John Wiley and Sons (1995) 2nd Edition.
2. Heiner, M., Rohr, C., Schwarick, M., Streif, S.: A comparative study of stochastic analysis techniques. In: Proc. CMSB 2010, ACM (2010) 96–106
3. Gillespie, D.T.: Approximate accelerated stochastic simulation of chemically reacting systems. The Journal of Chemical Physics **115**(4) (2001) 1716–1733
4. Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry **81(25)** (1977) 2340–2361
5. Gibson, M.A., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. The Journal of Physical Chemistry **A 104** (2000) 1876–1889
6. Cao, Y., Gillespie, D.T., Petzold, L.R.: Adaptive explicit-implicit tau-leaping method with automatic tau selection. J Chem Phys **126**(22) (Jun 2007) 224101

7. McCollum, J.M., Peterson, G.D., Cox, C.D., Simpson, M.L., Samatova, N.F.: The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. Comput. Biol. Chem. **30**(1) (2006) 39–49
8. Cao, Y., Li, H., Petzold, L.: Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. J Chem Phys **121**(9) (Sep 2004) 4059–4067
9. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press (1977) 46–57
10. Fages, F., Rizk, A.: On the analysis of numerical data time series in temporal logic. In: Proc. CMSB 2007, LNCS/LNBI 4695, Springer (2007) 48–63
11. Donaldson, R., Gilbert, D.: A Monte Carlo model checker for probabilistic LTL with numerical constraints. Technical report, University of Glasgow, Dep. of CS (2008)
12. Tafazzoli, A., Wilson, J.R., Lada, E.K., Steiger, N.M.: Skart: A skewness- and autoregression-adjusted batch-means procedure for simulation analysis. In: Winter Simulation Conference. (2008) 387–395
13. Schwarick, M., Rohr, C., Heiner, M.: Marcie - model checking and reachability analysis done efficiently. In: Proc. 8th International Conference on Quantitative Evaluation of SysTems (QEST 2011), IEEE CS Press (September 2011) 91–100
14. Tovchigrechko, A.: Model Checking Using Interval Decision Diagrams. PhD thesis, BTU Cottbus, Dep. of CS (2008)
15. Franzke, A.: A concept for redesigning Charlie. Technical report, BTU Cottbus, Dep. of CS (2008)
16. Rohr, C., Marwan, W., Heiner, M.: Snoopy–a unifying Petri net framework to investigate biomolecular networks. Bioinformatics **26**(7) (2010) 974–975
17. Cho, K.H., Shin, S.Y., Kim, H.W., Wolkenhauer, O., McFerran, B., Kolch, W.: Mathematical modeling of the influence of RKIP on the ERK signaling pathway. In: Proc. CMSB 2003, LNCS 2602, Springer (2003) 127–141
18. Gilbert, D., Heiner, M.: From Petri nets to differential equations - an integrative approach for biochemical network analysis. In: Proc. ICATPN 2006, LNCS 4024, Springer (2006) 181–200
19. Heiner, M., Donaldson, R., Gilbert, D. In: Petri Nets for Systems Biology, in Iyengar, M.S. (ed.), Symbolic Systems Biology: Theory and Methods. Jones and Bartlett Publishers, Inc. (2010)
20. Calder, M., Duguid, A., Gilmore, S., Hillston, J.: Stronger computational modelling of signalling pathways using both continuous and discrete-state methods. In: Proc. CMSB 2006, LNBI 4210, Springer (2006) 63–78
21. Napione, L., Manini, D., Cordero, F., Horvath, A., Picco, A., Pierro, M.D., Pavan, S., Sereno, M., Veglio, A., Bussolino, F., Balbo, G.: 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) 281–295
22. Younes, H., Clarke, E., Zuliani, P.: Statistical verification of probabilistic properties with unbounded until. In: Formal Methods: Foundations and Applications. Volume 6527 of Lecture Notes in Computer Science. Springer (2011) 144–160
23. Basu, S., Ghosh, A.P., He, R.: Approximate model checking of PCTL involving unbounded path properties. In: ICFEM. (2009) 326–346
24. Ballarini, P., Forlin, M., Mazza, T., Prandi, D.: Efficient parallel statistical model checking of biochemical networks. In: Proc. PDMC. (2009) 47–61
25. Zapreev, I.S.: Model checking Markov chains : techniques and tools. PhD thesis, University of Twente, Enschede (March 2008)