Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments

Monika Heiner[1], Sebastian Lehrack[1], David Gilbert[2], and Wolfgang Marwan[3]

[1] Department of Computer Science, Brandenburg University of Technology
Postbox 10 13 44, 03013 Cottbus, Germany
{monika.heiner,slehrack}@informatik.tu-cottbus.de
[2] School of Information Systems, Computing and Mathematics
Brunel University, Uxbridge, Middlesex UB8 3PH, UK
david.gilbert@brunel.ac.uk
[3] Otto von Guericke University & Magdeburg Centre for Systems Biology
c/o Max Planck Institute for Dynamics of Complex Technical Systems,
Sandtorstr. 1, 39106 Magdeburg, Germany
marwan@mpi-magdeburg.mpg.de

Abstract. This paper introduces extended stochastic Petri nets to model wetlab experiments. The extentions include read and inhibitor arcs, stochastic transitions with freestyle rate functions as well as several deterministically timed transition types: immediate firing, deterministic firing delay, and scheduled firing. The extensions result into non-Markovian behaviour, which precludes analytical analysis approaches. But there are adapted stochastic simulation analysis (SSA) methods, ready to deal with the extended behaviour. Having the simulation traces, we apply simulative model checking of PLTL, a linear-time temporal logic (LTL) in a probabilistic setting.

We present some typical model components, demonstrating the suitability of the introduced Petri net class for the envisaged application scenario. We conclude by looking briefly at a classical example of prokaryotic gene regulation, the lac operon case.

1 Motivation

This paper extends the Markovian stochastic Petri nets $\mathcal{SPN}_{Bio}$ as introduced in [GHL07] to model and analyse biochemical networks. Related application scenarios are discussed in [BGHO08], [GBHD09]. Case studies demonstrating a unifying framework to integrate the qualitative, stochastic and continuous paradigms can be found in [HGD08], [GHR+08], [HDG10]. Thus, $\mathcal{SPN}_{Bio}$ have been proven to be useful in systems and synthetic biology. However, there are limitations in expressivity.

Generally, biologists face the problem to design wetlab experiments to validate or contradict the current understanding of the biochemical network under investigation. In order to be better able to do so, they ask for the following advanced features:

C. Priami et al. (Eds.): Trans. on Comput. Syst. Biol. XI, LNBI 5750, pp. 138–163, 2009.

– stochastic and deterministic firing behaviour within one model,
– relative and absolute timing of the transitions' firing,
– construction of arbitrary schedules of programmed interventions.

Therefore, we are going to extend $\mathcal{SPN}_{Bio}$ belonging to the Markovian world by several features supporting the comfortable modelling of wetlab experiments. The extentions lead to the definition of biochemically interpreted Generalised Stochastic Petri nets $\mathcal{GSPN}_{Bio}$ and Deterministic and Stochastic Petri nets $\mathcal{DSPN}_{Bio}$. They include read and inhibitor arcs, stochastic transitions with freestyle rate functions as well as several deterministically timed transition types: immediate firing, deterministic firing delay, and scheduled firing.

The extension go beyond the Markov property, which precludes analytical analysis approaches; but there are adapted stochastic simulation analysis (SSA) methods, ready to deal with the extended behaviour. Having the simulation traces we apply simulative model checking of linear-time temporal logic (LTL) in a probabilistic setting (PLTL). Simulative model checking approximates the probability of a given temporal logic formula by considering finite sets of finite paths through the state space. Thus, it works even for systems with infinite state spaces.

We discuss in detail some typical model components, demonstrating the suitability of the introduced Petri net class $\mathcal{DSPN}_{Bio}$ for the envisaged application scenario. These components will be analysed by checking sets of stochastic simulation traces against PLTL properties. In doing so, a special category of properties, the so-called invariant properties, will be used to prove at the same time the plausibility of the applied simulation algorithm.

We conclude by looking briefly at a classical example of prokaryotic gene regulation, the lac operon case.

2 Stochastic Modelling

We assume basic knowledge of the standard notions of qualitative place/transition Petri nets, see e.g. [Mur89], [Rei82], [HGD08]. To be self-contained we start with recalling the fundamentals of (biochemically interpreted) stochastic Petri nets, belonging to the Markovian world, before introducing the extended notions resulting finally into non-Markovian Petri nets.

2.1 The Markovian Case - Stochastic Petri Nets ($\mathcal{SPN}_{Bio}$)

As with a qualitative Petri net, a stochastic Petri net maintains a discrete number of tokens on its places. But contrary to the time-free case, a firing rate (waiting time) is associated with each transition t, which are random variables $X_t \in [0, \infty)$, defined by probability distributions. Therefore, all reaction times can theoretically still occur, but the likelihood depends on the probability distribution. Consequently, the system behaviour is described by the same discrete state space, and all the different execution runs of the underlying qualitative

Petri net can still take place. This allows the use of the same powerful analysis techniques for stochastic Petri nets as they are applied for qualitative Petri nets.

For a better understanding we describe the general procedure of a particular simulation run for a stochastic Petri net. Each transition gets its own local timer. When a particular transition becomes enabled, meaning that sufficient tokens arrive on its preplaces, then the local timer is set to an initial value, which is computed at this time point by means of the corresponding probability distribution. In general, this value will be different for each simulation run. The local timer is then decremented at a constant speed, and the transition will fire when the timer reaches zero. If there is more than one enabled transition, a race for the next firing will take place. After the firing of the winning transition, the timers of the others still enabled transitions keep their values or are reset, depending on the specific type of the net.

Technically, various probability distributions can be chosen to determine the random values for the local timers. Biochemical systems are the prototype for exponentially distributed reactions. Thus, for our purposes, the firing rates of all transitions follow an exponential distribution, which can be described by a single parameter λ, and each transition needs only its particular, generally marking-dependent parameter λ to specify its local time behaviour. The following definition summarises this informal introduction.

Definition 1 (Stochastic Petri net, Syntax). *A* biochemically interpreted stochastic Petri net *is a quintuple* $\mathcal{SPN}_{Bio} = (P, T, f, v, m_0)$*, where*

- P *and* T *are finite, nonempty, and disjoint sets.* P *is the set of* places*, and* T *is the set of* transitions*.*
- $f : ((P \times T) \cup (T \times P)) \to \mathbb{N}_0$ *defines the set of directed* arcs*, weighted by nonnegative integer values.*
- $v : T \to H$ *is a function, which assigns a* stochastic hazard function h_t *to each transition* t*, whereby* $H := \bigcup_{t \in T} \left\{ h_t \,|\, h_t : \mathbb{N}_0^{|{}^\bullet t|} \to \mathbb{R}^+ \right\}$ *is the set of all stochastic hazard functions, and* $v(t) = h_t$ *for all transitions* $t \in T$*.*
- $m_0 : P \to \mathbb{N}_0$ *gives the* initial marking*.*

The stochastic hazard function h_t defines the marking-dependent transition rate $\lambda_t(m)$ for the transition t, i.e. $h_t = \lambda_t(m)$. The domain of h_t is restricted to the set of preplaces of t, denoted by ${}^\bullet t$ with ${}^\bullet t := \{p \in P | f\,(p,t) \neq 0\}$, to enforce a close relation between network structure and hazard functions. Therefore, $\lambda_t(m)$ actually depends on a sub-marking only.

Stochastic Petri net, Semantics. Transitions become enabled as usual, i.e. if all preplaces are sufficiently marked. However there is a time, which has to elapse, before an enabled transition $t \in T$ fires. The transition's firing delay (waiting time) is an exponentially distributed random variable X_t with the *probability density function*:

$$f_{X_t}(\tau) = \lambda_t(m) \cdot \mathrm{e}^{(-\lambda_t(m) \cdot \tau)}, \qquad \tau \geq 0.$$

The firing itself does not consume time and follows the standard firing rule of qualitative Petri nets. The semantics of a stochastic Petri net (with exponentially distributed firing delays for all transitions) is described by a continuous time Markov chain (CTMC). The CTMC of a stochastic Petri net without parallel transitions is isomorphic to the reachability graph of the underlying qualitative Petri net, while the arcs between the states are now labelled by the transition rates. For more details see [MBC+95], [BK02], [HGD08].

Based on this general $\mathcal{SPN}_{Bio}$ definition, specialised biochemically interpreted stochastic Petri nets can be defined by specifying the required kind of stochastic hazard function more precisely. In this paper, we are going to use the molecule semantics with mass action transition rates. Therefore we deploy the *stochastic mass-action hazard function*, which tailors the general $\mathcal{SPN}_{Bio}$ definition to biochemical mass-action networks, where tokens correspond to molecules:

$$h_t := c_t \cdot \prod_{p \in {}^\bullet t} \binom{m(p)}{f(p,t)} .$$

The constant c_t is the transition-specific stochastic rate constant, and $m(p)$ is the current number of tokens on a preplace p of the transition t. The binomial coefficient describes the number of non-ordered combinations of the $f(p,t)$ molecules, required for the reaction, out of the $m(p)$ available ones. In the following we abbreviate this formula by *BioMassAction(c_t)*.

See [GHL07] for another example, reading the tokens as concentration levels.

2.2 The Non-markovian Case - Extended Stochastic Petri Nets

We start off with an overview and brief biochemical motivation before introducing two classes of extended stochastic Petri nets.

There are quite a number of various extensions based on the fundamental stochastic Petri net class $\mathcal{SPN}$, see e.g. [MBC+95], [Ger01]. The most important additional features concern deterministically timed transitions, or *deterministic transitions* for short, which come along in different types. The crucial point is that the firing delay (waiting time) before an enabled transition fires does not depend anymore on a random variable, but is specified by a fixed time duration. To avoid confusion, we will call the transitions with a probabilistic firing delay, as introduced in the former subsection, *stochastic transitions*, if necessary. In summary, our extended stochastic Petri nets support the following features:

- read and inhibitor arcs,
- programmed transitions (freestyle rate functions),
- deterministic firing delay,
- scheduled transitions.

Read and inhibitor arcs. are popular add-ons enhancing modelling comfort. Read arcs (often also called test arcs) allow to specify positive side-conditions, e.g., if the occurrence of a subunit depends on the conformation of a protein

complex, or if a cell's reaction to a given stimulus depends on the specific physiological conditions of the cell. Contrary, inhibitor arcs allow to specify negative side-conditions in an abstract way, e.g., if the presence of a given protein or condition inhibits a specific reaction.

Speaking in technical terms, read and inhibitor arcs are directed arcs, going always from places to transitions. The standard firing rule needs to be adapted accordingly. The enabling condition is extended in the following way: if there is an arc a with a weight $w = f(p,t)$ connecting a place p with a transition t, then t can be enabled in a marking m if the following conditions are also satisfied:

- a is a read arc $\wedge\, m(p) \geq w$,
- a is an inhibitor arc $\wedge\, m(p) < w$.

The token situation on p is not changed by the firing of t, i.e. $m'(p) = m(p)$ for $m \xrightarrow{t} m'$.

Programmed transitions are stochastic transitions with freestyle rate functions. The firing rate can be specified by arbitrary mathematical functions, stored in lookup tables, if necessary.

To give an example, a popular phenomenon in biology is cooperativity. A biochemical reaction may be controlled by an highly non-linear, cooperative mechanism. Simple versions of cooperativity may be represented by complicated Petri net structures, but there are limits. The kinetic mechanisms of a cooperative behaviour are often not completely understood. However, the acquired understanding must be included in the model to get a coherent system model.

Deterministic firing delay is the outstanding characteristics of deterministic transitions. The delay is always relative to the time point where the transition gets enabled. There is one popular special case, the zero delay, for which the *immediate transitions* are introduced. Immediate transitions have always highest priority, which creates a subtle difference between an immediate transition and a deterministic transition with zero firing delay: if there is a conflict between the two, the immediate transition gets priority.

We will use the function *TimedFiring(delay)* to assign the delay constant.

Scheduled transitions belong to the deterministic transitions. The deterministic firing occurs according to a schedule specifying absolute points of the simulation time. A schedule can specify just a single time point, or equidistant time points within a given interval, triggering the firing once or periodically. However, transitions only fire at their scheduled time points if they are enabled. Scheduled transitions can dramatically restrict the behaviour, as we will see in Section 4.3, example EX5.

Scheduled transitions allow to disturb the core model at well-defined time points as it is done experimentally with the actual biological system under investigation in the wetlab; see Section 5 for an example.

We will use two functions to assign the required values: *FixedTimedFiring_Single(time_point)*, *FixedTimedFiring_Periodic(begin_time_point, repetition, end_time_point)*.

2.3 Generalised Stochastic Petri Nets ($\mathcal{GSPN}_{Bio}$)

Generalised stochastic Petri nets ($\mathcal{GSPN}_{Bio}$) are stochastic Petri nets $\mathcal{SPN}_{Bio}$ extended by *inhibitor arcs* and *immediate transitions*.

Inhibitor arcs are a powerful modelling feature and are known to bring computational completeness. Consequently, Petri nets of the net class $\mathcal{GSPN}$ have the same expressivity as an universal Turing machine [PW03]. However, in terms of construction of the reachability graph (continuous-time Markov chain), they do not establish additional challenges for finite state spaces, i.e. bounded Petri nets.

Immediate transitions are a very special kind of deterministic transitions with zero firing delay, i.e. they fire immediately after getting enabled, and always prior to (general) deterministic and stochastic transitions. Consequently, getting enabled and the firing itself coincide for immediate transitions. A cyclic system behaviour involving only the firing of immediate transitions corresponds to an infinite behaviour without time progress; we get a *time deadlock*.

If a stochastic simulation encounters a situation with more than one immediate transition enabled, one is chosen randomly [Ger01]. However, an analysis approach will consider all possible choices.

In terms of the reachability graph (continuous-time Markov chain), induced by a $\mathcal{GSPN}_{Bio}$ Petri net, we distinguish between *transient* and *non-transient* states. A system never spends time in a transient state before changing into another state. Thus, the time spent (*sojourn time*) in transient states is always zero, and not exponentially distributed anymore.

Consequently, the underlaying semantics is not a continuous-time Markov chain anymore. However, the transient states can be removed such that the reduced reachability graph corresponds again to a continuous-time Markov chain. See [MBC+95] for a precise description of the reduction technique and related formal definitions. In summary this means that $\mathcal{GSPN}_{Bio}$ can still be analysed analytically, if the state space, i.e. the continuous-time Markov chain can be constructed.

2.4 Deterministic and Stochastic Petri Nets ($\mathcal{DSPN}_{Bio}$)

Deterministic and Stochastic Petri Nets ($\mathcal{DSPN}_{Bio}$) are generalised stochastic Petri nets ($\mathcal{GSPN}_{Bio}$) extended by deterministic transitions.

Deterministic transitions possess a deterministic firing delay (waiting time), specified by a nonnegative real value. When a deterministic transition gets enabled, a count-down timer is started, initialized with the transition's firing delay. If the transition gets disabled before the timer reaches zero, the timer is switched off, and the transition will not fire. Otherwise, the transition will fire as soon as the timer reaches zero. The firing itself does not consume time.

If we consider stochastic Petri nets without deterministic transitions, the probability of two transitions firing at the same time is practically zero. Contrary,

in stochastic Petri nets with deterministic transitions, it is possible that two transitions want to fire simultaneously. We already discussed the special case of two concurrently enabled immediate transitions. To analyse such a system, all possible choices have to be considered, while in the simulation a random choice takes place.

Definition 2 (Deterministic and stochastic Petri net). *A biochemically interpreted deterministic and stochastic Petri net is a septuple* $DSPN_{Bio} = (P, T, f, g, v, d, m_0)$*, where*

- P *und* T *are finite, nonempty, and disjoint sets.* P *is the set of places, and* T *is the set of transitions.*
- *The set* T *is the union of three disjunctive transition sets, i.e.* $T := T_{stoch} \cup T_{im} \cup T_{timed}$ *with:*
  1. T_{stoch}*, the set of stochastic transitions with exponentially distributed waiting time,*
  2. T_{im}*, the set of immediate transitions with waiting time zero, and*
  3. T_{timed}*, the set of transitions with deterministic waiting time.*
- $f : ((P \times T) \cup (T \times P)) \to \mathbb{N}_0$ *defines the set of directed arcs, weighted by nonnegative integers.*
- $g : (P \times T) \to \mathbb{N}_0$ *defines the set of directed inhibitor arcs, weighted by nonnegative integers.*
- $v : T_{stoch} \to H$ *is a function, which assigns a stochastic hazard function* h_t *to each transition* $t \in T_{stoch}$*, whereby* $H := \bigcup_{t \in T_{stoch}} \left\{ h_t \,|\, h_t : \mathbb{N}_0^{|{}^\bullet t|} \to \mathbb{R}^+ \right\}$ *is the set of all stochastic hazard functions, and* $v(t) = h_t$ *for all transitions* $t \in T_{stoch}$*.*
- $d : T_{timed} \to \mathbb{R}^+$ *assigns to each deterministic transition* $t \in T_{timed}$ *a nonnegative deterministic waiting time.*
- $m_0 : P \to \mathbb{N}_0$ *gives the initial marking.*

The stochastic transitions correspond to the transitions of the net class $\mathcal{SPN}_{Bio}$, so they have an exponentially distributed waiting time following the definitions given in Section 2.1.

The net class $\mathcal{DSPN}_{Bio}$ is a subset of the class $e\mathcal{DSPN}$, introduced in [Ger01]. For details of the subset relation see [Leh07]. Therefore, the theory, which has been developed to analyse $e\mathcal{DSPN}$ Petri nets, see [Ger01] and [Haa03], can be deployed to analyse $\mathcal{DSPN}_{Bio}$, too.

The remaining two features *read arcs* and *scheduled transitions* are not explicitly mentioned in the definition above, because the just allow a simplified specification using the orthogonal basic concepts in $\mathcal{DSPN}_{Bio}$.

Read arcs do not extend the modelling power as long as an interleaving semantics is considered. A read arc and two opposite arcs are indistinguishable in terms of the reachability graph (continuous-time Markov chain).

Scheduled transitions can be replaced by net components consisting of immediate and deterministic transitions only; see [Leh07] for construction patterns. Thus, they do not extend the modelling power.

3 Stochastic Analysis

The non-Markovian behaviour of $DSPN_{Bio}$ precludes the standard analytical approaches belonging to the Markovian world. However, there are adapted stochastic simulation methods, ready to deal with the extended behaviour, see e.g. [Ger01], [Haa03], [Leh07], and many more. A detailed discussion of the necessary adaptions compared to the fundamental Gillespie algorithm [Gil77] is beyond the given space limitations of this paper. Having the simulation traces, we apply simulative model checking of linear-time temporal logic (LTL) in a probabilistic setting (PLTL).

Simulative model checking follows the idea of Monte Carlo sampling and handles large or even infinite state spaces through approximating results by analysing only a subset of the state space – a finite set of finite outputs (traces) from a stochastic simulation algorithm (SSA), e.g. Gillespie's exact SSA or any other suitable variations of it.

A natural choice of logic to describe properties of sets of traces is linear-time logic. A linear-time logic operates over sets of linear paths through the state space, equivalent to operating on simulation outputs. A given property holds if it holds in all possible paths. Consequently, there are no path quantifiers.

We apply PLTL, a probabilistic linear-time temporal logic [DG08], [MC208]. This logic extends standard Linear-time Temporal Logic (LTL) [Pnu81] to a stochastic setting with a probability operator and a filter construct, defining the initial state of the property. LTL is the fragment of full Computational Tree Logic (CTL*) [CGP01] without path quantifiers, implicitly quantifying universally over all paths. To be self-contained we briefly recall the PLTL basics.

Syntax. PLTL is a logic to create path formulae ϕ and to ask for their probabilities. The grammar given in Table 1 defines a PLTL formula ψ.

Semantics. The semantics is defined over finite sets of finite linear traces of temporal behaviour, in our case by stochastic simulation runs. Each trace is evaluated to a Boolean truth value, and the probability of a property holding true is computed by the fraction of true values in the set over the whole set. It goes without saying, the choice of simulator and simulation parameters used to compute the sequence of states can affect the semantics of the PLTL property and the correctness of the result.

$P_{\trianglelefteq x}$ is any inequality comparison of the probability of the property holding true, for example $P_{\geq 0.5}$. The expression $P_{=?}$ returns the value of the probability of the property holding true. Equality testing of the probability, $P_{=x}$, is not supported for obvious reasons.

PLTL allows the use of filters over top-level LTL expressions, denoted by $\{AP\}$, similar to those used in Probabilistic Computational Tree Logic (PCTL) [HJ94] and Continuous Stochastic Logic (CSL) [ASSB96]. This permits specifications to refer to the state or states that the property is checked from, rather than default to the initial state. This means that for a query of the form $\phi\,\{AP\}$, ϕ is checked from the first state that AP is satisfied. This can be a different one for each stochastic run. The temporal operators follow the standard LTL semantics:

Table 1. PLTL syntax. Please note that the square and curly brackets are part of PLTL.

$$
\begin{array}{lcl}
\psi & ::= & \mathbf{P}_{\trianglelefteq x}[\,\phi\,] \\
 & | & \mathbf{P}_{\trianglelefteq x}[\,\phi\ \{AP\}\,]\ . \\
\phi & ::= & \mathbf{X}\phi \mid \mathbf{G}\,\phi \mid \mathbf{F}\phi \mid \phi\ \mathbf{U}\ \phi \mid \phi\ \mathbf{R}\ \phi \\
 & | & \neg\ \phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \Rightarrow \phi \\
 & | & AP\ . \\
AP & ::= & \neg\ AP \mid AP \vee AP \mid AP \wedge AP \mid AP \Rightarrow AP \\
 & | & value\ comp\ value \\
 & | & true \mid false\ . \\
comp & ::= & = \mid \neq \mid \geq \mid > \mid < \mid \leq\ . \\
value & ::= & value\ op\ value \\
 & | & variable \mid max(variable) \mid d(variable) \\
 & | & Int \mid Real\ . \\
op & ::= & + \mid -\ \mid * \mid /\ ,
\end{array}
$$

with $\trianglelefteq \in \{<, \leq, \geq, >\}, x \in [0, 1]$. $\mathbf{P}_{\trianglelefteq x}$ can be replaced by $\mathbf{P}_{x=?}$.

- **Next (X)** - The property must hold true in the next time point.
- **Globally (G)** - The property must hold true always [1].
- **Finally (F)** - The property must hold true sometime in the future.
- **Until (U)** - The first property must hold true until the second property holds true.
- **Release (R)** - The second property can only ever not hold true if the first property becomes true.

The meta term *variable* stands for any variable in the model, *Int* is any integer number and *Real* is any real number. In our case of stochastic Petri net analysis, a *variable* is going to be a place name, and the formulae refer to the number of tokens on a place in a given state. Additionally, there is a predefined variable *time*, referring to the simulation time points. Thus we can, for example, express properties which occur after some simulation time has elapsed.

The function *max* operates over all the token values of a place to return the maximum in the given simulation runs, thus the peak of a species' concentration, modelled by a place, can be checked, e.g. $Protein = max(Protein)$. The function d operates on each place in each state individually to return the derivative, thus increasing token numbers can be checked, e.g. $d(Protein) > 0$.

This approach to simulative model checking incorporates two approximations. The truth value of a single trace is approximated by operating over a finite sequence of states only; and the probability of the property is approximated through sampling a finite number of traces only. Thus, a subset of the model's behaviour is considered only. However, there are two special categories

[1] To be precise, in the given setting of model checking by finite traces, globally means 'always –as far as known'.

of properties, where definitive, i.e. non-approximating answers are possible by simulative model checking.

- *Monotone properties* comply with the following condition: if the property is satisfied in any path through the state space, then it is satisfied in any extension of the path [HLMP04]. Formulae without the Globally operator are monotone properties. The Globally operator and semantically equivalent descriptions by the other operators are incompatible with the monotony property. Considering longer paths can only increase the probability.
- *Invariant Properties* have to hold true in every state in every path. Thus they comply with the following condition: if the property is satisfied in any path through the state space, then it is satisfied in any other path. Their probability is independent of the number of considered paths. They are often used as consistency checks, and so do we in this paper.

PLTL may be considered as a linear-time counterpart to CSL. It can easily be used to formalise the visual evaluation of diagrams as generated by deterministic/stochastic simulation runs or by recording experimental time series. In the following chapter we are going to use PLTL to analyse sets of stochastic simulation traces of extended stochastic Petri nets, which have been constructed to illustrate the expressiveness of $DSPN_{Bio}$.

4 Typical Components

We present some typical model components, controlling a network's inflow and outflow, and thus demonstrating the suitability of the introduced Petri net class $DSPN_{Bio}$ for the envisaged application scenarios of model-based design of wet-lab experiments. We use the following abbreviations introduced in Section 2:

- *BioMassAction*(c_t),
- *TimedFiring(delay)*,
- *FixedTimedFiring_Single(time_point)*,
- *FixedTimedFiring_Periodic(begin_time_point, repetition, end_time_point)*;

and we apply the following drawing conventions:

- read arcs: identified by a black dot,
- inhibitor arcs: identified by a hollow dot,
- stochastic transition: hollow square,
- deterministically timed transition: black square,
- immediate transition: black rectangle.

We are going to examine the behaviour of each component by simulative PLTL model checking over 100 (1,000) simulation runs. The individual runs are independent, so generally different. We confine ourselves deliberately on introductory formulae to illustrate the key ideas, increasing at the same time our confidence in the accuracy of our simulation algorithm for the non-Markovian setting.

4.1 Time-Controlled Inflow/Outflow

EX1. In our first example we consider a closed system, consisting of one reversible reaction $A \leftrightarrow B$, modelled by the two transitions *t1* (*BioMassAction(0.11)*) and *t2* (*BioMassAction(0.1)*). The two deterministically timed transitions *input* (*FixedTimedFiring_Periodic(11,1,20)*) and *output* (*FixedTimedFiring_Periodic(31,1,40)*) are responsible for the absolutely timed inflow and outflow of tokens, see Figure 1.

The transition *input* does not have preplaces, thus it fires for sure at the time points $11, 12, \ldots, 20$, producing each time 1,000 additional tokens on place A. Contrary, the transition *output* removes 1,000 tokens from place B at the time points $31, 32, \ldots, 40$, provided there are enough tokens to enable the firing. Figure 2 shows the first 100 time units of a single simulation run.

We give some introductory samples of temporal-logic formulae (queries), formalising the visual inspection of the simulation output as it might be done by the expert evaluating former or designing the next wetlab experiments. We apply these queries to a set of 100 stochastic (single) simulation traces. The ratio

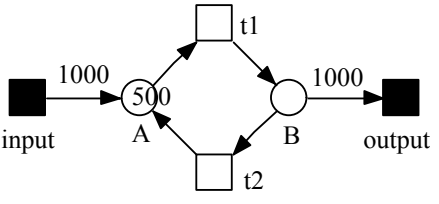

Fig. 1. First example of time-controlled inflow/outflow (EX1)

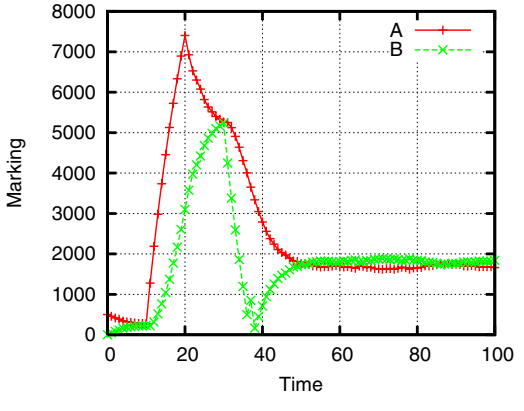

Fig. 2. Simulation result of the network given in Figure 1 (single run) (EX1)

of traces where the formula holds to the total number gives us a rough estimate of a formula's probability.

We check over exact Gillespie traces, i.e. all single events are logged. There are generally no "even" time points (like 30.000000 for 30). However, the firing of scheduled transitions at absolute time points (e.g. 20 in this example) causes exact time points in the simulation traces. We have to keep this in mind when refering to absolute time points in the following queries.

Please remember, all place names are read as integer variables in the following formulae; and the predefined variable *time* relates to the simulation time. The probabilities as computed by simulative model checking are given in brackets.

- Maxima *(probabilities: 1.0, 0.95).*
  $\mathbf{P}_{=?}$ [$\mathbf{G}(A < 7550)$]
  $\mathbf{P}_{=?}$ [$\mathbf{G}(B < 5350)$]
- Peaks *(probabilities: 0.9, 1.0).*
  $\mathbf{P}_{=?}$ [$\mathbf{F}(time = 20 \wedge A > 0.9 \cdot max(A) \wedge (3000 < B \wedge B < 3500))$]
  $\mathbf{P}_{=?}$ [$\mathbf{F}((29 < time \wedge time < 30) \wedge (5000 < A \wedge A < 5400) \wedge B > 0.9 max(B))$]
- Steady state, relative statements *(probabilities: 0.03, 0.59, 0.8, 0.91).*
  $\mathbf{P}_{=?}$ [$time \geq 50 \Rightarrow \ \mathbf{G}(A < B)$]
  $\mathbf{P}_{=?}$ [$time \geq 55 \Rightarrow \ \mathbf{G}(A < B)$]
  $\mathbf{P}_{=?}$ [$time \geq 60 \Rightarrow \ \mathbf{G}(A < B)$]
  $\mathbf{P}_{=?}$ [$time \geq 70 \Rightarrow \ \mathbf{G}(A < B)$]
- Steady state, absolute statements *(probabilities: 0.39, 1.0).*
  $\mathbf{P}_{=?}$ [$time \geq 50 \Rightarrow \ \mathbf{G}((1500 < A \wedge A < 1800) \wedge (1600 < B \wedge B < 2000))$]
  $\mathbf{P}_{=?}$ [$time \geq 60 \Rightarrow \ \mathbf{G}((1500 < A \wedge A < 1800) \wedge (1600 < B \wedge B < 2000))$]

EX2. We vary the pattern of our first example to remove repeatedly all currently available tokens on place B at equidistant time points, see Figure 3.

The immediate transition *output* consumes all tokens on place B, while there is a token on place *output_on*. The token on place *output_on* is controlled by the deterministically timed transition *switch_output_on* (*FixedTimedFiring_Periodic(20,20,_SimEnd)*) and the immediate transition *switch_output_off*. The transition *switch_output_on* initiates every 20 time units the cleaning process. The immediate transition *switch_output_off* switches off the outflow as soon as the place B is clean; otherwise each token arriving on B would be instantly removed and no token accumulation would be possible anymore. A single simulation run is given in Figure 4. We analyse a set of 100 of such stochastic traces by the following temporal-logic queries *(all yield probability 1.0).*

- If the output is switched on, B is cleaned immediately.
  $\mathbf{P}_{=?}$ [$\mathbf{G}(output\_on = 1 \Rightarrow B = 0)$]
- Cleaning of B at time point 20.
  $\mathbf{P}_{=?}$ [$\mathbf{F}(time = 20 \wedge B = 0)$]

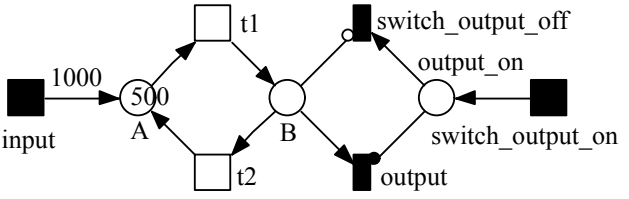

Fig. 3. Second example of time-controlled inflow/outflow (EX2)

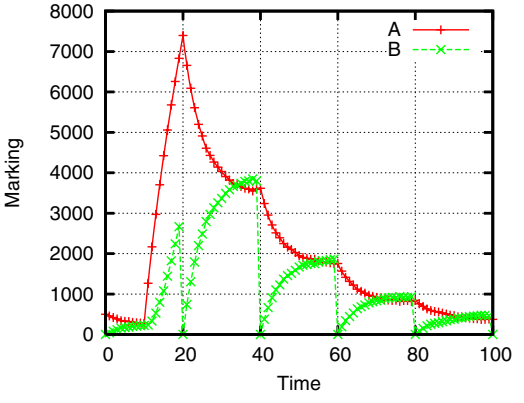

Fig. 4. Simulation result of the network given in Figure 3 (single run) (EX2)

- Cleaning of B at time point 20, ensuring that B does not get cleaned earlier.
  $\mathbf{P}_{=?}$ [$\mathbf{F}(B > 0 \wedge (B > 0\ \mathbf{U}\ (time = 20 \wedge B = 0)))$]
- Cleaning of B at time point 40, ensuring that B remains marked inbetween as soon as it got a token.
  $\mathbf{P}_{=?}$ [$\mathbf{F}(time = 20 \wedge B = 0) \wedge \mathbf{F}(B > 0 \wedge (B > 0\ \mathbf{U}\ (time = 40 \wedge B = 0)))$]

4.2 Token-Controlled Inflow

We discuss two examples and start again with a reversible reaction $A \leftrightarrow B$, modelled by the two stochastic transitions *t1* (*BioMassAction(0.1)*) and *t2* (*BioMassAction(0.005)*), which we consider as a closed system, challenged by experimental interventions.

EX3. In our first example of token-controlled inflow, the tokens on place A are raised by 50 as soon as the token amount drops below the threshold 30, see Figure 5. This behaviour is implemented by the immediate transition *input*, the firing of which is prevented by an inhibitor arc testing A. The weight 30

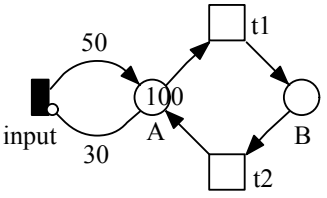

Fig. 5. First example of token-controlled inflow (EX3)

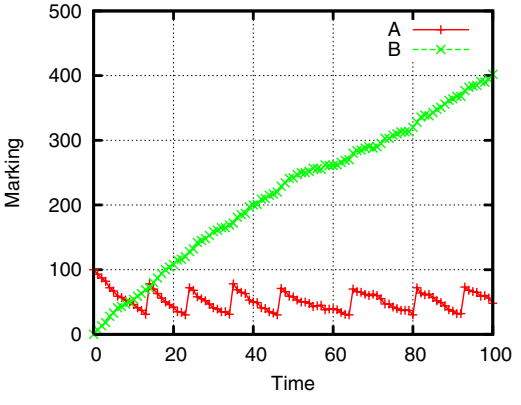

Fig. 6. Simulation result of the network given in Figure 5 (single run) (EX3)

of the inhibitor arc prevents the firing of *input* until the token amount drops below 30. 50 tokens are added to place A as soon as the inhibition condition becomes invalid, preventing again further inflow until the next drop occurs. Figure 6 shows a single simulation run. We analyse a set of 1,000 runs against the following formulae.

- The tokens on A never fall below the threshold 30 *(probability: 1.0)*.
  $\mathbf{P}_{=?}\ [\ \neg\ \mathbf{F}(A < 30)\ ]$
- The transition *input* tries to keep the tokens on A between 30 and 80. But there are always some tokens on place B, which may return to A *(probabilities: 0.946, 0.996, 0.999)*.
  $\mathbf{P}_{=?}\ [\ \mathbf{F}(A = 30 \wedge\ \mathbf{G}(30 \leq A \wedge A \leq 80))\ ]$
  $\mathbf{P}_{=?}\ [\ \mathbf{F}(A = 30 \wedge\ \mathbf{G}(30 \leq A \wedge A \leq 82))\ ]$
  $\mathbf{P}_{=?}\ [\ \mathbf{F}(A = 30 \wedge\ \mathbf{G}(30 \leq A \wedge A \leq 84))\ ]$
- There is a constant inflow due to the transition *input*, and the rate of $t1$ is (significantly) higher than of $t2$. Therefore, B increases permanently and

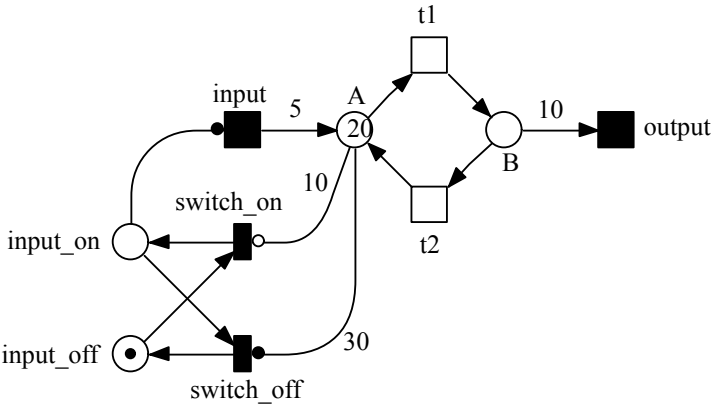

Fig. 7. Second example of token-controlled inflow (EX4)

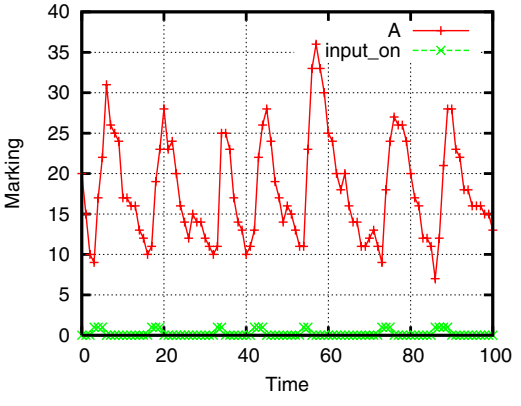

Fig. 8. Simulation results of the network given in Figure 7 (single run). The input is switched on/off (place *input_on*) in dependence on the token situation on A (EX4).

without limits. This is true in the averaged case only, e.g. 100 runs. $d(B)$ specifies the derivative.

$\mathbf{P}_{=?}$ [$\mathbf{G}(d(B) \geq 0)$]

EX4. Our second example of token-controlled inflow is given in Figure 7. The transitions *t1* (*BioMassAction(0.2)*) and *t2* (*BioMassAction(0.1)*) form again the reversible reaction $A \leftrightarrow B$. We add the deterministically timed transition *output* (*FixedTimedFiring_Periodic(5,5,_SimEnd)*) to get a significant consumption of tokens. Each time *output* gets activated, it removes 10 tokens from B.

If the token amount on place A drops below 10, the deterministically timed transition *input* (*TimedFiring(0.5)*) starts working and adds by each firing 5

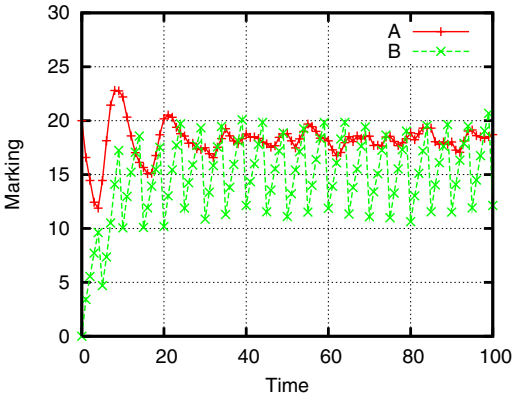

Fig. 9. Simulation results of the network given in Figure 7 (100 runs). A and B oscillate due to the repeated switching between inflow on/off (EX4).

tokens with 0.5 units waiting time inbetween, until there are at least 30 tokens on A. This behaviour is controlled by the immediate transitions *switch_on* and *switch_off*, and the two places *input_on* and *input_off*, forming a 1-P-invariant[1]. *Switch_on* can only fire if there are less than 10 tokens on A, and *switch_off* can only fire, if there are at least 30 tokens on A.

We give two related diagrams. The single run in Figure 8 shows how the input is switched on/off (place *input_on*) in dependence on the token situation on A. Figure 9 gives the average of 100 runs. It highlights the oscillation of A and B, caused by the repeated switching between inflow on/off. We analyse the token-controlled inflow component by the following formulae (1,000 runs) (*the first three yield probability 1.0*).

- The two places *input_on* and *input_off* form a 1-P-invariant.
  $\mathbf{P}_{=?}$ [$\mathbf{G}((input\_on = 1 \wedge input\_off = 0) \vee (input\_on = 0 \wedge input\_off = 1))$]
- The transition *input* is switched on/off if the token amount on A crosses the threshold 10 or 30, respectively.
  $\mathbf{P}_{=?}$ [$\mathbf{G}(A < 10 \Rightarrow input\_on = 1)$]
  $\mathbf{P}_{=?}$ [$\mathbf{G}(A \geq 30 \Rightarrow input\_off = 1)$]
- There is a delay of 0.5 time units between the on/off switch and the reaction of the actual inflow transition. E.g., after having switched off the input, 5 additional tokens will arrive by the already triggered firing of the transition *input*. Thus, even a weaker range than specified by the threshold values does not get probability 1 *(probability: 0.995)*.
  $\mathbf{P}_{=?}$ [$\mathbf{G}(5 \leq A \wedge A \leq 40)$]

[1] Exactly one of both places carries a token at any point of time.

4.3 Switch between Deterministic and Stochastic Transitions

The following two networks demonstrate how to switch between deterministic and stochastic transitions. We start off with a time-controlled switch, before discussing a token-controlled switch.

In both cases we consider a non-reversible reaction $A \rightarrow B$, which is nevertheless modelled by two transitions: the stochastic transition *t_stoch* (*BioMassAction(0.1)*) and the deterministically timed transition *t_det* (*TimedFiring(0.25)*). The chosen net structure ensures that always one of these two transitions only is able to transfer tokens from place A to place B; with other words: the token flow occurs either stochastically or deterministically. The mutually exclusive firing is implemented by the two places *stochastic_on* and *det_on*, forming a 1-P-invariant and establishing side-conditions for *t_stoch* or *t_det*, respectively.

EX5. The actual time-controlled switch is performed by two deterministically timed transitions: *switch_to_det* (*FixedTimedFiring_Single(10)*) and *switch_to_stoch* (*FixedTimedFiring_Single(30)*), which fire (each once!) at the absolute time points 10 or 30, respectively, causing a switch in the other operation mode, see Figure 10. In summary, the modelled reaction $A \rightarrow B$ behaves deterministically between the time points 10 and 30, and stochastically else.

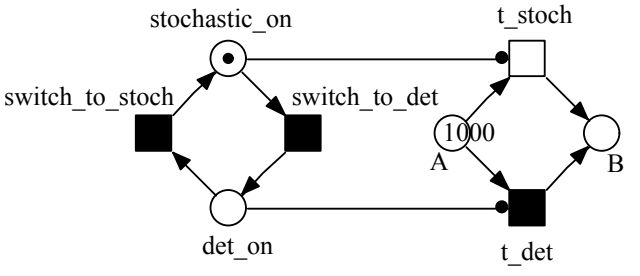

Fig. 10. Example of time-controlled switch between deterministic and stochastic behaviour. The semantic functions assigned to the transitions *switch_to_det* and *switch_to_stoch* allow them to fire only once (EX5).

EX6. We keep the basic principle for the token-controlled switch, but replace the transitions switching between the operation modi by immediate transitions, which depend on the token situation in place A. The immediate transitions *switch_to_det* and *switch_to_stoch* fire each once as soon as the token amount on place A drops below 700 or 500, see Figure 12. In summary, the modelled reaction $A \rightarrow B$ behaves deterministically for token amount between 500 and 700, and stochastically else.

The diagrams in Figure 11 and 13 show the behaviour of the two patterns for a single run each. For both we confirm the mutually exclusive operation mode of the stochastic and deterministic behaviour by the following query.

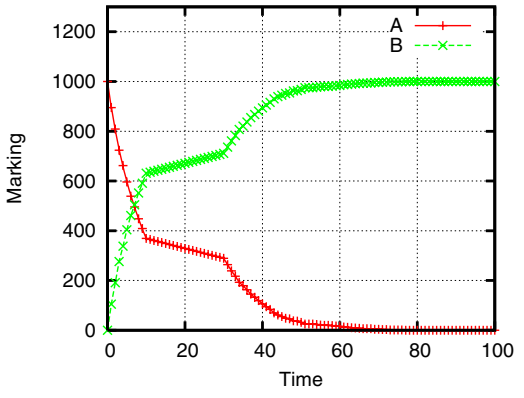

Fig. 11. Simulation result of the network given in Figure 10 (single run). There is a deterministic token flow from A to B between time points 10 and 30, and stochastic flow else (EX5).

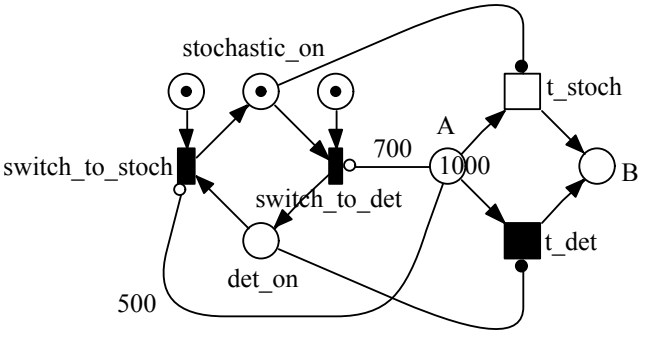

Fig. 12. Example of token-controlled switch between deterministic and stochastic behaviour. The additional preplaces of the immediate transitions bring the equivalence to the net component in Figure 10; i.e. the immediate transitions fire only once (EX6).

– The two places *stochastic_on* and *det_on* form a 1-P-invariant.
$\mathbf{P}_{=?}$ [$\mathbf{G}((stochastic\_on = 1 \wedge det\_on = 0) \vee (stochastic\_on = 0 \wedge det\_on = 1))$]

We conclude the analyses with checking the range of deterministic versus stochastic behaviour for the two discussed patterns.

– Deterministic token flow from A to B between time points 10 and 30.
$\mathbf{P}_{=?}$ [$(10 \leq time \wedge time < 30) \Rightarrow det\_on = 1$]

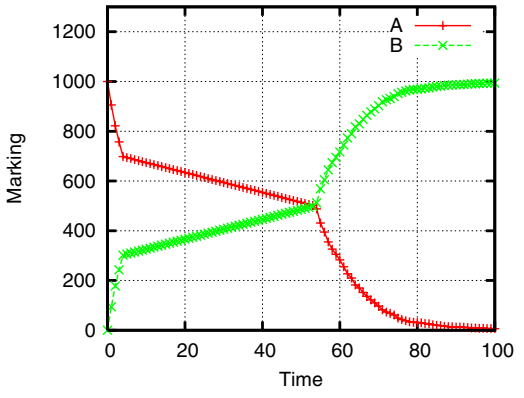

Fig. 13. Simulation result of the network given in Figure 12 (single run). There is a deterministic token flow from A to B for a token amount on A between 500 and 700, and stochastic flow else (EX6).

- Stochastic token flow from A to B from 0 up to time point 10, and starting at time 30 again.
  $\mathbf{P}_{=?}$ [$(time < 10 \vee 30 \leq time) \Rightarrow stochastic\_on = 1$]
- Deterministic token flow from A to B for a token amount on A between 500 and 700.
  $\mathbf{P}_{=?}$ [$(500 \leq A \wedge A < 700) \Rightarrow det\_on = 1$]
- Stochastic token flow from A to B for a token amount on A less than 500 or greater or equal 700.
  $\mathbf{P}_{=?}$ [$(A < 500 \vee 700 \leq A) \Rightarrow stochastic\_on = 1$]

All these properties are invariant properties, i.e. they yield probability 1.0, independently of the number and the length of considered simulation traces.

5 Lac Operon Model

We conclude by looking briefly at a classical example of prokaryotic gene regulation, the lac operon case. We follow the simplified version discussed in [Wil06] and specified there by a set of reaction equations and in an SBML-shorthand notation. We keep all naming conventions and the initial conditions, and translate the textual representation into a (qualitative) Petri net, reflecting explicitly the inherent structure of the regulatory network, compare Figure 14. Finally, we assign the rate equations as specified in the SBML code, and we get a stochastic Petri net.

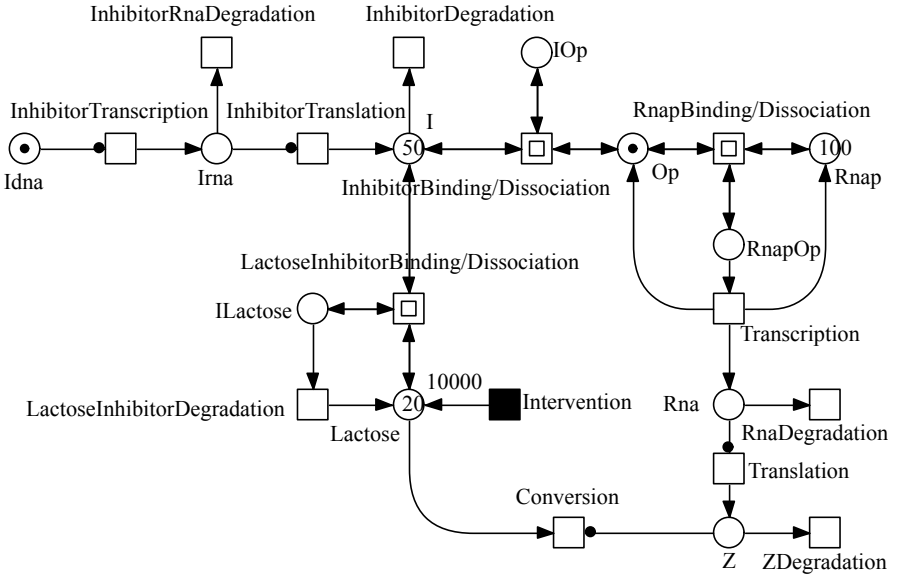

Fig. 14. Lac operon model according to [Wil06]. Macro transitions (drawn as two centric squares) indicate reversible reactions.

The core model of the network under consideration is extended by a special transition – an event in SBML terminology – modelling a timed intervention in a wetlab experiment. The transition *Intervention* (*FixedTimedFiring_Periodic(50000,50000,_SimEnd)*[2]) introduces 10,000 molecules of Lactose every 50,000 time units, compare Figure 15.

To increase our confidence in the model we start with a preliminary structural analysis and compute the P-invariants and T-invariants[3]. There are input transitions, so the net can not be covered by P-invariants. However, there are three P-invariants, inducing mass-conserving subnetworks (modules) and enjoying obvious biological meaning. The preserved species is given first in the following short-hand notation:

- $pi_1 = \{Idna\}$,
- $pi_2 = \{Rnap, RnapOp\}$,
- $pi_3 = \{Op, IOp, RnapOp\}$.

Contrary, T-invariants do cover the net, which is a common consistency criteria for well-formed net structures, allowing e.g. a steady state behaviour. Each T-invariant induces a self-contained, state-repeating subnetwork (module). Besides the expected three trivial T-invariants for the three reversible reactions:

[2] Here we differ from the model given in [Wil06], where the modelled intervention occurs only once at a specified point of time.

[3] For all notions used in this section, but note introduced in this paper, see [HGD08].

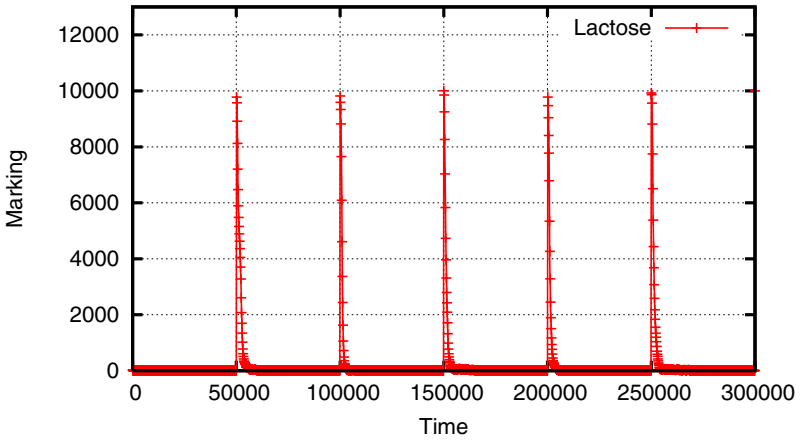

Fig. 15. Simulation result of the lac operon model: Lactose

- $ti_1 = \{LactoseInhibitorBinding, \underline{LactoseInhibitorDissociation}\}$,
- $ti_2 = \{\underline{InhibitorBinding}, \underline{InhibitorDissociation}\}$,
- $ti_3 = \{\underline{RnapBinding}, \underline{RnapDissociation}\}$,

we get the following six non-trivial T-invariants, each input/output behaviour is made of:

- $ti_4 = \{InhibitorTranscription, InhibitorRnaDegradation\}$,
- $ti_5 = \{InhibitorTranslation, InhibitorDegradation\}$,
- $ti_6 = \{InhibitorTranslation, LactoseInhibitorBinding,$
  $LactoseInhibitorDegradation\}$,
- $ti_7 = \{Intervention, Conversion\}$,
- $ti_8 = \{RnapBinding, Transcription, RnaDegradation\}$,
- $ti_9 = \{Translation, ZDegradation\}$.

There are four transitions (underlined), which are not involved in non-trivial T-invariants. However, they are crucial for the regulation mechanism between Z and Lactose. Please note, each T-invariant is given in a short-hand notation, enumerating the T-invariants' transitions in an order, which they may follow to reproduce a state, or what has to happen to get the system back in the steady state after some disturbences.

Remarkably, the net fulfills the Deadlock Trap Property (DTP), however is beyond the structural net class *extended simple*. In summary this allows the conclusion that there is no reachable dead state, in which any further system activities would be prevented. Actually, we expect the model to be live, which can not be proven with the analysis techniques available for (qualitatively) unbounded Petri nets.

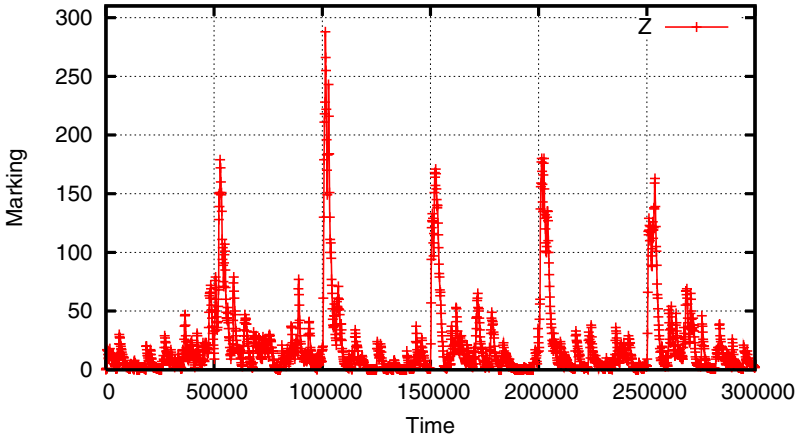

Fig. 16. Simulation result of the lac operon model: Z

However, there are property-preserving reduction rules downsizing the net structure, which are supported by the Integrated Net Analyser INA [SR99]. Applying these structural reduction rules, we get a smaller network, consisting of 2 places and 4 transitions. Liveness becomes obvious for this reduced network; see the supplementary material for more details.

The place Z models the enzyme β-Galactosidase; its reaction to the repeatedly sudden increase of Lactose molecules is shown in Figure 16. We analyse for the first intervention how a peak of *Lactose* triggers a peak of Z.

- The intervention causes *Lactose* to peak at time point 50,000 *(probabilities 1.0, 1.0, 0.65)*.
  $\mathbf{P}_{=?}$ [$(49,999 \leq time \wedge time < 50,000) \Rightarrow Lactose \leq 0.01 \cdot max(Lactose)$]
  $\mathbf{P}_{=?}$ [$time = 50,000 \Rightarrow Lactose \geq 0.99 \cdot max(Lactose)$]
  $\mathbf{P}_{=?}$ [$(52,000 \leq time \wedge time < 52,001) \Rightarrow Lactose \leq 0.1 \cdot max(Lactose)$]
- Z is highly likely to be at low concentration at time point 50,000 *(probability 0.9)*.
  $\mathbf{P}_{=?}$ [$time = 50,000 \Rightarrow Z \leq 0.1 \cdot max(Z)$]
- Z will rise to at least 80% of its maximal value within 2,000 time units *(probability 0.925)*.
  $\mathbf{P}_{=?}$ [$F($ $(50,000 < time \wedge time < 52,000) \wedge Z \geq 0.6 \cdot max(Z)$ $)$]
- In summary, a peak of *Lactose* triggers a peak of Z within 2,000 time units *(probability 0.925)*.
  $\mathbf{P}_{=?}$ [$time = 50,000 \wedge Lactose \geq 0.99 \cdot max(Lactose) \wedge Z \leq 0.1 \cdot max(Z)$
  $\Rightarrow F(Z \geq 0.8 \cdot max(Z) \wedge time < 52,000)$]

6 Tools

The Petri net components and the lac operon model have been constructed using Snoopy [Sno08], [HRS08], a tool to design and animate or simulate hierarchical graphs, among them qualitative and continuous Petri nets, and the extended stochastic Petri nets as used in this paper. Snoopy provides export to various analysis tools as well as import and export of the Systems Biology Markup Language (SBML) [HFS+03].

The qualitative analyses of the lac operon model have been made with the Petri net analysis tool Charlie [Cha08], complemented by the structural reduction rules supported by the Integrated Net Analyser INA [SR99]; see the corresponding log files in the supplementary material.

The quantitative analyses have been done by the cooperation of two tools: Snoopy's build-in simulation algorithm for extended stochastic Petri nets to generate the sets of simulation traces, and MC2 [MC208], a model checker by Monte Carlo sampling, for the simulative PLTL model checking. MC2 reads sets of simulation traces as, e.g., generated by Snoopy and expects additionally a file with the temporal-logical formulae.

As a proof of concept, we confined ourselves to rather small sets of 100 (1,000) runs only, allowing at the same time an affordable repetition of all computational experiments by the reader. A general recommendation is to start with smaller sets of simulation runs, just to check whether one got a formula right, before analysing larger sets, which could actually be done in parallel. None of the computational experiments for the typical components required more than 6 minutes per net example on a standard machine. Simulative model checking of the lac operon model is slightly more expensive. The traces have been generated on a workstation (2.83 GHz, 64 bit). The 100 exact traces (simulation time interval: 300,000) require about 5 GB. The model checking itself consumes less than 30 minutes on a standard machine.

Snoopy, Charlie as well as the data and analysis files of the discussed Petri net examples are available at

www-dssz.informatik.tu-cottbus.de/examples/xspn-components.

7 Summary

This paper extends the Markovian stochastic Petri nets $\mathcal{SPN}_{Bio}$ as introduced in [GHL07] to model and analyse biochemical networks. The extensions lead to the definition of Generalised Stochastic Petri nets $\mathcal{GSPN}_{Bio}$ and deterministic and stochastic Petri nets $\mathcal{DSPN}_{Bio}$. They include read and inhibitor arcs as well as several time-dependent transition types, which in summary preclude standard Markovian analysis approaches. Therefore we applied simulative model checking, approximating the probability of a given temporal logic formula by considering finite sets of finite paths through the state space. These paths are generated by

stochastic simulation algorithms, adjusted to deal with the extended modelling features.

We discussed some typical net components demonstrating the usability of $\mathcal{DSPN}_{Bio}$ for the envisaged application scenario of model-based experiment design and evaluation. These components have been analysed by checking sets of stochastic simulation traces against PLTL properties. Invariant properties have been used to prove at the same time the plausibility of the applied simulation algorithm. We concluded with briefly looking at the lac operon case study, one of the classical examples of prokaryotic gene regulation.

Currently we consider some further extensions of our modelling formalism; among them are variable deterministic firing delays specified by an interval or an arbitrary marking-dependent function, reset and equal arcs as well as marking-dependent arc weights.

Simulative model checking is an extremely powerful tool. By way of introduction we have deliberately deployed some basic features of PLTL only. There is an interesting extension, PLTLc [DG08], supporting free variables, and thus allowing richer and more elegant properties, which however are also more complicated to write and to interpret. Thus, demonstrating these advanced features to more sophisticated users is beyond the scope and space limits of this paper.

Acknowledgements. The stochastic features of the Snoopy tool have been developed by Sebastian Lehrack, which cumulated in his Master thesis [Leh07]. This work has been financially supported by MPI Martinsried and MPI Madgeburg. Snoopy's quality improvements by Christian Rohr are crucial for the computational experiments presented in this paper. We would like to thank Robin Donaldson for his responsive assistance in MC2 issues.

References

[ASSB96] Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Verifying Continuous-time Markov Chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996)

[BGHO08] Breitling, R., Gilbert, D., Heiner, M., Orton, R.: A structured approach for the engineering of biochemical network models, illustrated for signalling pathways. Briefings in Bioinformatics 9(5), 404–421 (2008)

[BK02] Bause, F., Kritzinger, P.S.: Stochastic Petri Nets. Vieweg (2002)

[CGP01] Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge (2001) (third printing)

[Cha08] Charlie Website. A Tool for the Analysis of Place/Transition Nets. BTU Cottbus (2008), `http://www-dssz.informatik.tu-cottbus.de/software/charlie/charlie.html`

[DG08] Donaldson, R., Gilbert, D.: A model checking approach to the parameter estimation of biochemical pathways. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 269–287. Springer, Heidelberg (2008)

[GBHD09] Gilbert, D., Breitling, R., Heiner, M., Donaldson, R.: An introduction to biomodel engineering, illustrated for signal transduction pathways. In: Corne, D.W., Frisco, P., Paun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 13–28. Springer, Heidelberg (2009)

[Ger01] German, R.: Performance analysis of communication systems with non-Markovian stochastic Petri nets. John Wiley and Sons Ltd., Chichester (2001)

[GHL07] Gilbert, D., Heiner, M., Lehrack, S.: A unifying framework for modelling and analysing biochemical pathways using Petri nets. In: Calder, M., Gilmore, S. (eds.) CMSB 2007. LNCS (LNBI), vol. 4695, pp. 200–216. Springer, Heidelberg (2007)

[GHR+08] Gilbert, D., Heiner, M., Rosser, S., Fulton, R., Gu, X., Trybiło, M.: A Case Study in Model-driven Synthetic Biology. In: Proc. 2nd IFIP Conference on Biologically Inspired Collaborative Computing (BICC), IFIP WCC 2008, Milano, pp. 163–175 (2008)

[Gil77] Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry 81(25), 2340–2361 (1977)

[Haa03] Haas, P.J.: Stochastic Petri nets: Modelling, Stability, Simulation. Springer, Heidelberg (2003)

[HDG10] Heiner, M., Donaldson, R., Gilbert, D.: Petri Nets for Systems Biology. In: Iyengar, M.S. (ed.) Symbolic Systems Biology: Theory and Methods. Jones and Bartlett Publishers, Inc., USA (in Press, 2010)

[HFS+03] Hucka, M., Finney, A., Sauro, H.M., Bolouri, H., Doyle, J.C., Kitano, H., et al.: The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models. J. Bioinformatics 19, 524–531 (2003)

[HGD08] Heiner, M., Gilbert, D., Donaldson, R.: Petri nets in systems and synthetic biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 215–264. Springer, Heidelberg (2008)

[HJ94] Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)

[HLMP04] Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 307–329. Springer, Heidelberg (2004)

[HRS08] Heiner, M., Richter, R., Schwarick, M.: Snoopy - a tool to design and animate/simulate graph-based formalisms. In: Proc. PNTAP 2008, associated to SIMUTools 2008. ACM digital library (2008)

[Leh07] Lehrack, S.: A tool to model and simulate stochastic Petri nets in the setting of biochemical networks (in German). Master thesis, BTU Cottbus, Dep. of CS (2007)

[MBC+95] Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets, 2nd edn. Wiley Series in Parallel Computing. John Wiley and Sons, Chichester (1995)

[MC208] MC2 Website. MC2 - PLTL model checker. University of Glasgow (2008), `http://www.brc.dcs.gla.ac.uk/software/mc2/`

[Mur89] Murata, T.: Petri Nets: Properties, Analysis and Applications. Proc.of the IEEE 77(4), 541–580 (1989)

[Pnu81] Pnueli, A.: The temporal semantics of concurrent programs. Theor. Comput. Sci. 13, 45–60 (1981)

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

[Rei82] Reisig, W.: Petri nets; An introduction. Springer, Heidelberg (1982)

[Sno08] Snoopy Website. A Tool to Design and Animate/Simulate Graphs. BTU Cottbus (2008), http://www-dssz.informatik.tu-cottbus.de/software/snoopy.html

[SR99] Starke, P.H., Roch, S.: INA - The Intergrated Net Analyzer. Humboldt University Berlin (1999), http://www.informatik.hu-berlin.de/~starke/ina.html

[Wil06] Wilkinson, D.J.: Stochastic Modelling for System Biology, 1st edn. CRC Press, New York (2006)