In M Chen and R Hofestädt (Eds.)
Approaches in Integrative Bioinformatics,
Chapter 9, Springer, pages 245–272, 2014.

Petri Nets for Modeling and Analyzing Biochemical Reaction Networks

Fei Liu and Monika Heiner

Abstract Petri nets have been widely used to model and analyze biochemical reaction networks. This chapter gives an overview of different types of Petri nets within a unifying Petri net framework that comprises the qualitative, stochastic, continuous and hybrid paradigms at both uncolored and colored levels. The Petri net framework permits to investigate one and the same biological reaction network with different modeling abstractions in various complementary ways. We describe the use of the framework to investigate biochemical reaction networks with the help of the unifying Petri net tool, Snoopy, and his close friends Charlie and Marcie. The repressilator example serves as running case study.

Keywords: Petri nets, biochemical reaction networks, unifying Petri net framework, qualitative, stochastic, continuous and hybrid Petri nets, colored Petri nets, repressilator

1 Introduction

Modeling and analysis techniques have been widely used to study biochemical reaction networks. A large variety of modeling approaches, e.g., ordinary (partial) differential equations, Boolean networks, process algebras, and Petri nets, have been applied for modeling a wide range of biochemical reaction networks (for reviews see, e.g. (Heath and Kavraki, 2009), and *chapter I in this book*). Among them, Petri nets are particularly suitable for modeling

Fei Liu
Control and Simulation Center, Harbin Institute of Technology, Postbox 3006, 150080 Harbin, China, e-mail: liufei@hit.edu.cn

Monika Heiner
Department of Computer Science, Brandenburg University of Technology, Postbox 10 13 44, 03013 Cottbus, Germany, e-mail: monika.heiner@tu-cottbus.de

the concurrent, asynchronous and dynamic behavior of biological networks. (Reddy et al., 1993; Hofestädt, 1994) were the first to pick up Carl Adam Petri's idea for a graphical representation of stoichiometric equations and applied qualitative Petri nets to model and analyse metabolic pathways. Since that time, a large variety of Petri net classes, e.g., stochastic Petri nets, continuous Petri nets, hybrid Petri nets and colored Petri nets, have been developed for modeling and analysing different types of biological networks; see e.g. (Chaouiya, 2007; Baldan et al., 2010; Heiner and Gilbert, 2011; Liu, 2012).

Petri nets offer a number of attractive advantages for investigating biological reaction networks (Heiner et al., 2008):

- intuitive graphical and directly executable modeling formalisms,
- rich and mathematically founded analysis techniques,
- coverage of structural and behavioral properties as well as their relations,
- integration of qualitative (i.e. time-free) and quantitative (i.e. time-dependent) analysis techniques and methods, including animation (the token flow),
- coverage of discrete (stochastic), continuous (deterministic) and hybrid paradigms for quantitative analysis techniques and methods,
- a wealth of computer tool support.

This chapter gives an overview of different types of Petri nets within a unifying Petri net framework and describes how they can be used to model and analyze biochemical reaction networks with the help of the unifying Petri net tool, Snoopy (Rohr et al., 2010; Heiner et al., 2012), and its close friends Charlie (Franzke, 2009; Wegener et al., 2011) and Marcie (Heiner et al., 2013).

This chapter has been deliberately written in an informal style; no formal definitions are given. We focus on an overview on the key concepts and their applications in our previous work. For formal definitions see (Heiner et al., 2008), which also provides plenty of pointers where to continue reading.

This chapter is organized as follows. Section 2 gives an overview of our unifying Petri net framework, followed by a description of each net class contained in this framework from Section 3 to 7, respectively. After a brief description of the tools we use, this chapter is concluded.

2 A Unifying Petri Net Framework

Petri nets may easily serve as a convenient umbrella formalism integrating qualitative and quantitative (i.e., stochastic, continuous, or hybrid) modeling and analysis techniques. Thus Petri nets are immediately ready to address distinctive modeling demands of systems and synthetic biology including those biochemical reaction networks that may need several modeling paradigms.

Motivated by this application scenario, a unifying Petri net framework (see Fig. 1) has been developed (Heiner et al., 2012; Heiner and Gilbert, 2012), which can be divided into two levels: uncolored (Heiner et al., 2008) and colored (Liu, 2012). Each level comprises a family of related Petri net classes, sharing structure, but being specialized by their kinetic information. Specifically, the uncolored level contains qualitative (time-free) Petri nets ($\mathcal{QPN}$) as well as quantitative (time-dependent) Petri nets such as stochastic Petri nets ($\mathcal{SPN}$), continuous Petri nets ($\mathcal{CPN}$), and generalized hybrid Petri nets ($\mathcal{GHPN}$). The colored level consists of the colored counterparts of the uncolored level, thus containing colored qualitative Petri nets ($\mathcal{QPN}^C$), colored stochastic Petri nets ($\mathcal{SPN}^C$), colored continuous Petri nets ($\mathcal{CPN}^C$), and colored generalized hybrid Petri nets ($\mathcal{GHPN}^C$).

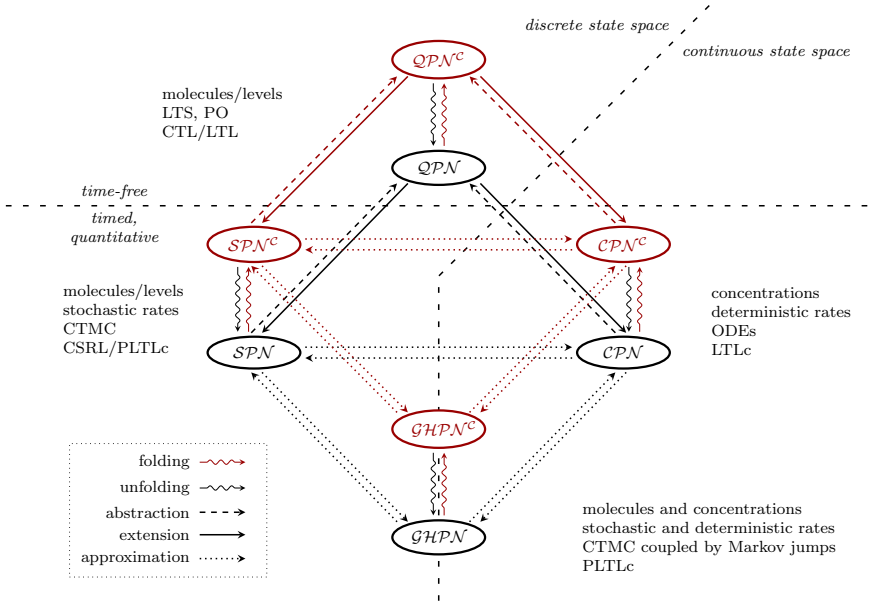

Fig. 1 A unifying Petri net framework, which has been implemented in the Petri net tool, Snoopy. Reprinted from (Heiner et al., 2012) with kind permission from Springer Science + Business Media B.V., Fig. 1, p 399.

Petri nets of these net classes can be converted into each other, see arrows in Fig. 1. Obviously, there may be a loss of information in some directions (cf. arrows labeled with "abstraction" in Fig. 1). The conversion between colored and uncolored net classes is accomplished by means of user-guided folding or automatic unfolding (cf. arrows labeled with folding and unfolding in Fig. 1). Moving between the colored and uncolored level changes the style of representation, but does not change the actual net structure of the underlying

biochemical reaction network. Therefore, all analysis techniques available for uncolored Petri nets can be applied to colored Petri nets as well.

Snoopy supports the simultaneous use of different net classes, which provides the ground to investigate one and the same case study with different modeling abstractions in various complementary ways (Heiner et al., 2008; Heiner and Gilbert, 2011; Liu, 2012).

We will address each net class in the framework in the following sections by focusing on their application for investigating biochemical reaction networks.

3 Qualitative Petri Nets ($\mathcal{QPN}$)

3.1 Modeling

$\mathcal{QPN}$ comprise – first of all – the standard *Place/Transition nets* (*P/T nets, Petri nets* for short) which basically correspond to the original ideas introduced by Carl Adam Petri in 1962 (Petri, 1962). Petri nets (see Fig. 2 for an introductory example) are bipartite directed multigraphs with two types of nodes, called places and transitions, which are connected by arcs. Places (represented as circles) and transitions (represented as boxes) model in our context biochemical species and reactions, respectively. Arcs carry stoichiometric information, called weight or multiplicity. Tokens on places represent the (discrete) quantities of species, which may be understood as the number of molecules or the level of concentration of a species, or simply the presence of, e.g., a gene. A particular arrangement of tokens over all places of a Petri net specifies the current system state (*marking*). The initial state is called the initial marking. For example, the initial marking in Fig. 2(a) consists of five tokens on place H_2 and three tokens on place O_2.

The state of the system changes by the firing of transitions. A transition is enabled to fire if all its pre-conditions are fulfilled, i.e., each of its pre-places contains at least the number of tokens specified by the weight of the corresponding arc. Upon firing of a transition, tokens from all its pre-places are removed, and tokens are added to all its post-places, each according to the corresponding arc weights. See Fig. 2 for two state changes upon firing of transition t; that is, two tokens on pre-place H_2 and one token on pre-place O_2 are removed and two tokens are added to the post-place H_2O; i.e., we reach new markings. All markings, which can be reached from a given marking by any firing sequence of arbitrary length, form the set of reachable markings. The set of markings reachable from the initial marking build the state space of a given Petri net. The *reachability graph* of a Petri net comprises these reachable markings as nodes and the transitions between them as edges. The reachability graph is finite, iff (if and only if) the state space is finite.

$\mathcal{QPN}$ do not involve any timing aspects. The firing of a transition is atomic and does not consume any time. So they allow us a purely qualitative modeling of biochemical reaction networks.

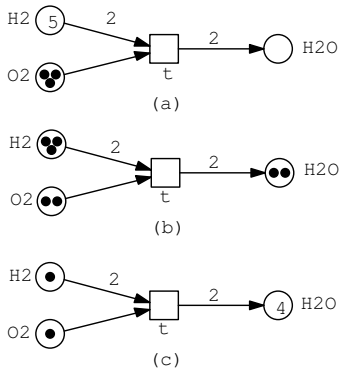

Fig. 2 A Petri net model of the chemical reaction $2H_2 + O_2 \rightarrow 2H_2O$. The places labeled with H_2 and O_2 are pre-places of the transition t, the place labeled with H_2O its post-place. (a) initial marking before t fires, (b) marking reached by firing of t once, and (c) marking reached by a second firing of t. The transition is not enabled anymore in the marking reached after these two single firing steps.

P/T nets have been enlarged to extended Petri nets ($\mathcal{XPN}$) by the provision of special arc types such as read arcs (often also called test arcs), inhibitor arcs, equal arcs, and reset arcs. All these special arcs are only allowed to go from places to transitions. Read, inhibitor, and equal arcs add constraints on the firing of a transition, but the connected places are not affected upon firing. A read arc (compare Fig. 16) allows to model that some resource is required, but not exclusively consumed upon firing. Hence the same token can be used at the same time by many transitions. An inhibitor arc (compare Fig. 10) reverses the logic of the enabling condition of a place, i.e., it imposes a precondition that a transition may only fire if the place contains less tokens than the weight of the arc indicates. An equal arc imposes the precondition that a transition may only fire if the number of tokens on the place connected by the equal arc is equal to the arc weight. A reset arc empties the place connected by this arc once the transition fires; the number of tokens does not matter.

Finally, $\mathcal{XPN}$ can be further enriched to include marking-dependent arcs, i.e., the arc multiplicities are allowed to be marking-dependent expressions of various types in terms of transitions' pre-places (Ciardo, 1994). See Fig. 3 for a technical example.

Modeling repressilator. We now use the repressilator (Blossey et al., 2008) as example to illustrate a modular and stepwise construction of a Petri net model using Snoopy.

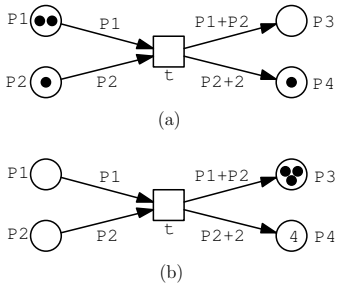

Fig. 3 A Petri net with marking-dependent arcs. Each arc may be marking-dependent, e.g., the multiplicity of the post-arc from transition t to place $P3$ is an addition expression, $P1 + P2$. (a) initial marking before t fires, (b) marking reached by firing of t.

(1) We start with designing a Petri net model of a gene, illustrated in Fig. 4(a). The presence of one gene allows the generation of proteins without consuming the gene, while generated proteins can degrade. A possible run of this model is that the transition *generate* fires twice, adding two tokens to the place *protein*, and then transition *degrade* fires once, removing one token from place *protein*. We obtain the marking where each place carries one token. It is easy to see that this Petri net has an infinite number of reachable markings.

(2) Next, we extend the basic behavior in Fig. 4(a) by allowing the gene to be blocked by the protein produced by another gene, which makes a building block called gene gate, see Fig. 4(b). The behavior of Fig. 4(b) is different from that of Fig. 4(a), as a gene may be blocked or unblocked in Fig. 4(b) while it is always unblocked in Fig. 4(a).

(3) When genes repress each other in a circular manner, we obtain a gene regulatory cycle, the repressilator (Blossey et al., 2008); see Fig. 5, which is composed of three gene gates with identical structure.

Snoopy supplies two features for the design and systematic construction of larger Petri nets – logical nodes and macro nodes. Logical nodes (i.e. logical places/transitions) serve as connectors to avoid lengthy arcs, and macro transitions (macro places) help to hide transition-bordered (place-bordered) subnets in order to design hierarchically structured Petri nets.

Using logical nodes we are able represent the repressilator model in alternative ways highlighting the modular structure of the Petri net, which are illustrated in Fig. 6 and Fig. 7, respectively, both of which are equivalent to Fig. 5.

Using macro transitions we can hide all gene-related details while keeping the protein places as interface; see Fig. 8. We obtain a hierarchical Petri net; Fig. 9 gives its top level. This Petri net is also equivalent to Fig. 5–7, it just uses a different representation style.

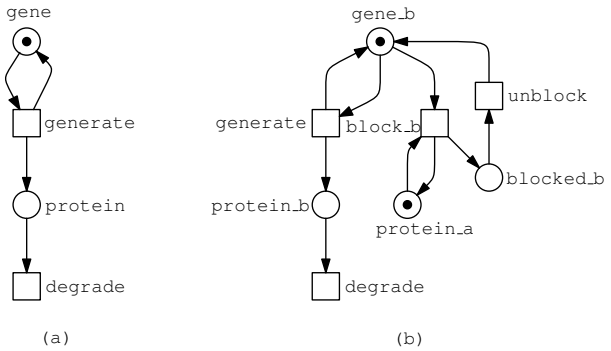

Fig. 4 (a) A Petri net model of a gene, and (b) a Petri net model of a gene gate according to (Blossey et al., 2008), who also inspired the layout: gene b may be blocked by protein a. Reprinted from ref. (Heiner and Gilbert, 2012), Copyright 2013, with permission from Elsevier

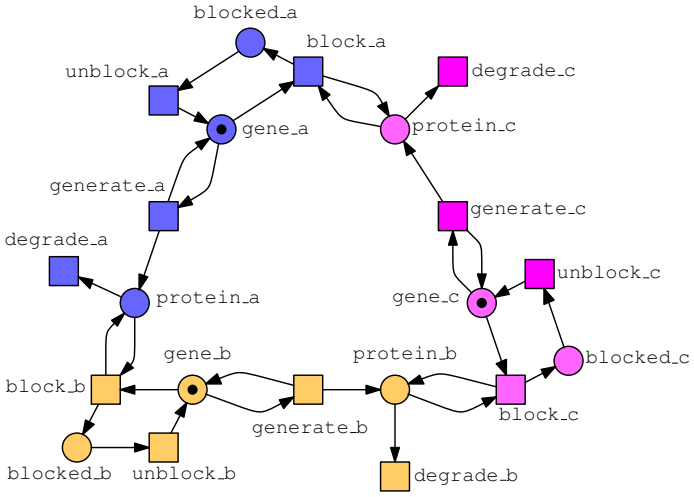

Fig. 5 The repressilator – Petri net for three genes in a regulatory cycle. Reprinted from ref. (Heiner and Gilbert, 2012), Copyright 2013, with permission from Elsevier

3.2 Analysis

$\mathcal{QPN}$ are time-free models, the qualitative analysis considers however all possible behavior of the system under any timing. Thus, the $\mathcal{QPN}$ model itself implicitly contains all possible time-dependent behaviors.

Behavioral properties. There are three orthogonal *general behavioral properties* which are usually explored first to gain some insights into the behavior of a Petri net.

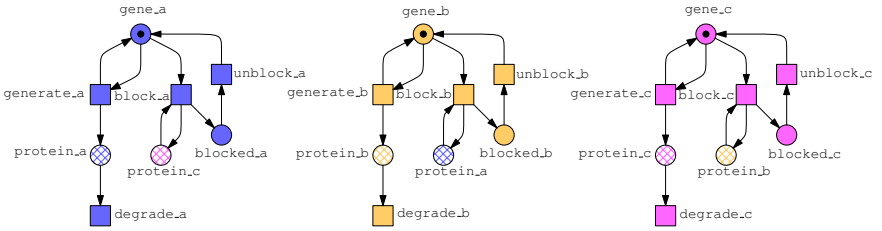

Fig. 6 The repressilator – Petri net for three genes in a regulatory cycle represented using logical nodes (here places, cross-hatched) to preserve gene-centred modules. Logical nodes with identical names serve as connectors; they are multiple representations of the same node used for layout clarity. See also Fig. 7. Reprinted from ref. (Heiner and Gilbert, 2012), Copyright 2013, with permission from Elsevier

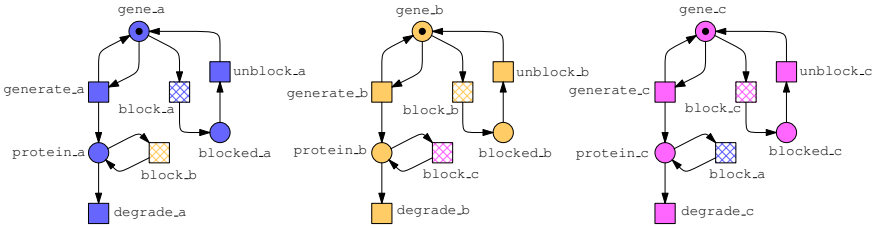

Fig. 7 The repressilator – Petri net for three genes in a regulatory cycle represented using logical transitions. See also Fig. 6.

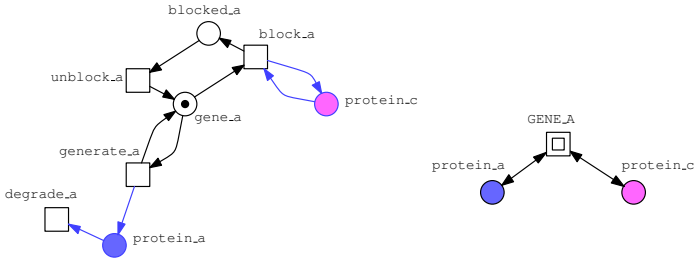

Fig. 8 Hierarchical structuring by the use of macro transitions. The uncolored nodes (left) make the contents of the macro transition GENE_A (right). The blue arcs highlight the connection to the interface places.

- *boundedness.* A place is said to be k-bounded (bounded for short) if the maximal number of tokens on this place is bounded by a constant k in all reachable markings. A Petri net is k-bounded (bounded for short) if all its places are k-bounded.
- *liveness.* A transition is said to be live if it will always be possible to reach a state (marking) where this transition gets enabled, whatever happens. A Petri net is live, if each transition is live.
- *reversibility.* A Petri net is said to be reversible if the initial marking can be reached again from each reachable marking.

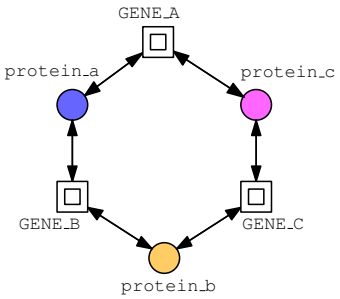

Fig. 9 Hierarchical Petri net model of the repressilator using macro transitions; compare Fig. 8. Only the top level is shown.

For example, by playing the token game for our repressilator model (take any Petri net in Fig. 5–7, 9) we can easily figure out that the places *gene_i* and *blocked_i*, with $i = \{a, b, c\}$, are 1-bounded. But the net is unbounded as all places *protein_i* are unbounded. If the generation of a protein occurs faster than its degradation, infinite many tokens (molecules) will be accumulated. Furthermore, we can argue that this Petri net is likely to be live and reversible.

When models get more complicated, it might not be obvious anymore to decide behavioral properties by reasoning only. Then we need mathematically sound analysis techniques. Petri net theory offers a rich body of such analysis techniques, most of them are implemented in our analysis tool Charlie. We sketch here only a few of them to give an impression of what kind of analysis techniques we have.

Structural properties. Structural properties (Murata, 1989; Marsan et al., 1995; Heiner et al., 2008) permit – if they hold – to deduce behavioral properties of Petri nets from their structure without constructing the complete or partial state space. If a property is proved structurally for a given Petri net, it holds for this Petri net in any initial marking. The most important structural properties can be classified as follows: elementary graph properties, siphons/traps, and place/transition invariants.

Elementary graph properties. The elementary graph properties relate to the following questions (see (Heiner et al., 2008) for explanations of all the following terms):

- Is the Petri net pure (PUR), ordinary (ORD), homogeneous (HOM), conservative (CSV), static conflict free (SCF), connected (CON), or strongly connected (SC)?
- Has the Petri net boundary nodes; i.e., input transitions (FT0), output transitions (TF0), input places (FP0), or output places (PF0))?
- Does the Petri net structure obey the constraints of a state machine (SM), synchronization graph (SG), extended free choice net (EFC), or extended simple net (ES)?

Elementary graph properties occasionally permit on their own conclusions on behavioral properties. For example, a Petri net having input transitions, i.e. transitions without pre-places, is unbounded (as the firing of input transitions does not depend on any pre-conditions), or a Petri net having input places, i.e. places without pre-transitions, is not live (as the tokens on an input place are sooner or later used up). Our repressilator Petri net has output transitions, i.e. transitions without post-places, which tells us that the model is either not live or unbounded (at least the pre-place of the output transition had to be unbounded).

Siphons/traps. A non-empty set S of places of a Petri net is called a siphon if there is no transition which has post-places in S, but no pre-places in S. Consequently, every transition, which fires tokens onto a place in S also has a pre-place in this set, i.e. the set of pre-transitions of S is contained in the set of post-transitions of S. Pre-transitions of a siphon can not fire, if the place set is clean, i.e. none of the places carries a token. Therefore, a siphon can not get tokens again, as soon as it is clean, and then all its post-transitions are dead.

Contrary, a non-empty set Q of places of a Petri net is called a trap if there is no transition which has pre-places in Q, but no post-places in Q. Consequqntly, every transition, which subtracts tokens from a place of the trap set, also has a post-place in this set, i.e. the set of post-transitions of Q is contained in the set of pre-transitions of Q. Post-transitions of a trap always return tokens to the place set. Therefore, once a trap contains tokens, it can not become clean again.

Siphon and trap are closely related, but contrasting notions. When they come on their own, we usually get deficient behavior. However, both notions have the power to perfectly complement each other. A Petri net satisfies the Siphon Trap Property (STP) if every siphon includes an initially marked trap. For certain combinations of structural properties, we can derive behavioral properties. For example, if a net is ORD and ES, and the STP holds, then the net is live. STP holds also for our repressilator model, but the net structure is beyond ES. Thus we can only conclude the absence of dead states, i.e. states, where no transition is enabled.

Place/transition invariants. Place and transition invariants (P- and T-invariants for short) play a crucial role in analyzing biological systems due to their biological interpretations. Both of them can be obtained by solving a linear equation system which describes the Petri net structure and which is independent of the initial marking. Any linear combination of P-invariants (T-invariants) yields again a P-invariant (T-invariant). Therefore, one is usually interested in minimal invariants; i.e. invariants which can not be described by a linear combination.

A P-invariant represents a set of places over which the weighted token count keeps constant whatever happens in the Petri net. So a place belonging to a P-invariant is k-bounded. We get the upper bound k by multiplying the invariant with the initial marking. In metabolic networks, P-invariants often

correspond to conservation laws in chemistry, reflecting substrate conservations, while in signal-transduction networks P-invariants often correspond to proteins and their possible states.

A T-invariant describes a multiset of transitions; it can be interpreted in two different ways. The multiset either specifies how often a transition has to fire to return to the original marking, or the multiset gives the relative firing rates required to keep the Petri net in the same state – the steady state.

Taking our repressilator model as example, Charlie yields the following results. The Petri net has three minimal P-invariants, one for each gene gate: x_i = (gene_i, blocked_i), where i=a,b,c. For each P-invariant, the constant token sum is 1, which confirms our expectations: a gene is either blocked or unblocked, it can neither disappear nor be multiplied.

The Petri net has also six minimal T-invariants, two for each gene gate: $y1_i$ = (block_i, unblock_i) and $y2_i$ = (generate_i, degrade_i), where i=a,b,c, which cover the whole Petri net, i.e., each transition belongs to a T-invariant. These T-invariants confirm our previous observations that a balanced firing of these transition sets reproduces the initial marking, and a balanced firing according to $y2_i$ makes the Petri net bounded.

Model checking. If the state space is finite and of manageable size, analytical model checking can be used to analyze $\mathcal{QPN}$, otherwise simulative model checking may help to obtain an approximative answer. In any case, the behavioral properties of interest have to be expressed in temporal logics, e.g., in a branching time temporal logic, one instance of which is Computational Tree Logic (CTL) (Clarke et al., 2001), or in a linear-time logic (LTL) (Pnueli, 1981). Both logics are supported by Marcie.

To be able to deploy CTL model checking, we introduce a bounded version of our repressilator example, see Fig. 10. Its state space is finite, but explosively grows as illustrated in Table 1.

Table 1 State space growth for increasing K (maximum number of each protein), compare Fig 10, computed with Marcie's symbolic state space representation. For the very specific case of our repressilator example we are able to specify a general formula for the state space growth: $2^n * (K+1)^n$, with n being the number of genes in the regulatory circle (in our running example we use n=3).

K	number of states	K	number of states
1	64 (1)	1000	8,024,024,008 (9)
50	1,061,208 (6)	5000	1,000,600,120,008 (12)
100	8,242,408 (6)	10,000	8,002,400,240,008 (12)
150	27,543,608 (7)	50,000	1,000,060,001,200,008 (15)
500	1,006,012,008 (9)	100,000	8,000,240,002,400,008 (15)

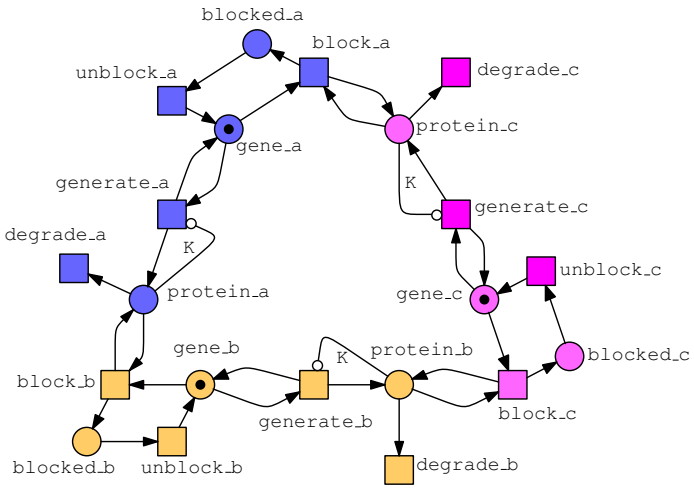

Fig. 10 The repressilator in a bounded version – maximal K tokens can be accumulated on each protein place. Inhibitor arcs (hollow circle as arc head) with arc weight K limit the generation of proteins. K is a constant which is used to conveniently parameterize the model; compare Table 1.

Having a bounded repressilator model, we can check behavioral properties expressed as CTL properties. We give three examples for *special behavioral properties*.

- Forever it holds, gene b is either unblocked or blocked.
  AG [$(gene\_b = 1$ & $blocked\_b = 0)$ | $(gene\_b = 0$ & $blocked\_b = 1)$]
- It is forever possible that there at least k molecules of protein b; i.e. there will be new proteins b forever, which includes liveness of transition *degrade_b*.
  AG EF [$protein\_b \geq k$]
- It is possible that there are at least k molecules of each protein at the same time.
  EF [$protein\_a \geq k$ & $protein\_b \geq k$ & $protein\_c \geq k$]

Here, **A** (for all paths) and **E** (there is one path) are path quantifiers, and **G** (globally) and **F** (finally) are temporal operators. CTL can also be used to query the general behavioral properties. For more examples of temporal formulae the reader may wish to check, e.g., (Heiner et al., 2008) where model checking has been explored for $\mathcal{QPN}$.

3.3 Applications

There are quite a number of applications of qualitative Petri nets for modeling of biochemical systems. In this chapter we do not wish to give a review, but just give some examples.

Model validation by means of P/T-invariant analysis is discussed in (Heiner and Koch, 2004) for three case studies: apoptosis, carbon metabolism in potato tuber, and the glycolysis and pentose phosphate metabolism. Structural analysis has also been used in (Heiner, 2009) to derive coarse network structures highlighting the structural principles inherent in the functional modules identified by T-invariants, and in (Heiner and Sriram, 2010) to determine the core of a Hypoxia response network and to identify its fragile node.

$\mathcal{QPN}$ have been deployed in (Heiner et al., 2008, 2010) to model signal transduction pathways, and their detailed analysis is exercised step by step.

In (Blätke et al., 2013a), IL-6 signalling in the JAK/STAT signal transduction pathway serves as case study to illustrate a modular protein-centred modeling approach.

4 Stochastic Petri Nets ($\mathcal{SPN}$)

4.1 Modeling

$\mathcal{SPN}$ extend $\mathcal{QPN}$ by assigning to transitions exponentially distributed waiting times, which are specified by firing rate functions (stochastic rates, compare Fig. 1). The underlying semantics of $\mathcal{SPN}$ is a Continuous-Time Markov Chain (CTMC). $\mathcal{SPN}$ have been previously extended to *generalized stochastic Petri nets* ($\mathcal{GSPN}$) (Marsan et al., 1995) and later to *deterministic and stochastic Petri nets* ($\mathcal{DSPN}$) (German, 2001).

Our *extended stochastic Petri nets* ($\mathcal{XSPN}$) (Heiner et al., 2009), which comprise $\mathcal{GSPN}$ and $\mathcal{DSPN}$, provide the four special arcs types and marking-dependent arcs as available for $\mathcal{XPN}$, and furthermore three special transition types: immediate transitions (zero waiting time), deterministic transitions (deterministic waiting time, relative to the time point where the transition gets enabled), and scheduled transitions (scheduled to fire, if any, at single or equidistant, absolute points of the simulation time). In Snoopy, we do not distinguish between these three classes of stochastic Petri nets. Thus, we usually call our extended stochastic Petri nets simply $\mathcal{SPN}$ if confusion is precluded.

In biological reaction networks, rate functions are often marking-dependent. In Snoopy, popular kinetics like mass action semantics (Lund, 1965) and

level semantics (Heiner et al., 2008) are supported by pre-defined function patterns.

Modeling repressilator. Let us return to our repressilator model. If we associate a rate function with each transition, e.g., the rate functions given in Table 2, we can consider it as a stochastic repressilator model.

Table 2 Rate functions for the $\mathcal{SPN}$ repressilator model. $MA(c)$ denotes the mass action function, where c is a kinetic parameter. See last column for the explicit rate functions for gene a.

transition class	kinetic parameter c	rate function pattern	example: gene a
generate	0.1	$MA(0.1)$	$0.1 * gene\_a$
block	1.0	$MA(1.0)$	$1.0 * gene\_a * protein\_c$
unblock	0.0001	$MA(0.0001)$	$0.0001 * blocked\_a$
degrade	0.001	$MA(0.001)$	$0.001 * protein\_a$

4.2 Analysis

The CTMC for a given $\mathcal{SPN}$ is isomorphic to the reachability graph of its corresponding $\mathcal{QPN}$, but edges are enriched by the transition rates. Thus, all $\mathcal{QPN}$ analysis techniques can still be applied, and all behavioral properties which hold for a $\mathcal{QPN}$ are still valid for the $\mathcal{SPN}$. Additionally, we have the following techniques to explore stochasticity.

Stochastic simulation. Stochastic simulation like the Gillespie stochastic simulation algorithm (SSA) (Gillespie, 1977) generates random walks through the CTMC. Approximated traces can be obtained by averaging a number of simulation runs. Besides, the unrestricted use of special (immediate, deterministic, scheduled) transitions destroys the Markov property. But the adaptation of the Gillespie stochastic simulation algorithm is rather straightforward and supported in Snoopy.

For example, assigning rates with the given kinetic parameters to any of our repressilator Petri nets generates sustained oscillation for all proteins, with each single run behaving differently. See Fig. 11 for a plot with the rates given in Table 2.

Simulative model checking. To systematically explore simulation traces, we use PLTLc (Donaldson and Gilbert, 2008), a probabilistic extension of LTL with constraints, to express our behavioral properties of interest. Simulative model checking considers a finite set of finite outputs from Gillespie's

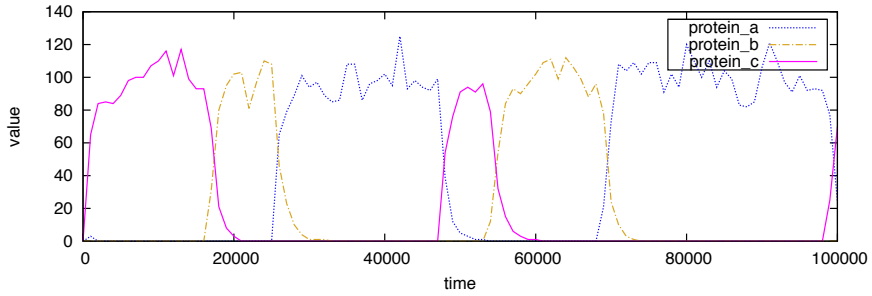

Fig. 11 Plot of one stochastic simulation run for the $\mathcal{SPN}$ repressilator model; for rate functions, see Table 2. Each single run looks differently in terms of oscillation, e.g., which gene starts rising.

exact SSA, i.e. a finite subset of the state space. This permits to explore very big or even infinite state spaces in reasonable time, or just to obtain a first rough estimate. Each trace is evaluated to a Boolean truth value and the probability of a behavioral property holding true is approximated by the number of traces with true values over the whole sample set. One has to consider a sufficient amount of simulation traces to obtain reliable approximations. The number of traces required increases with the expected confidence in the numerical results. Rare events may dramatically increase the required size of the sample set.

Let us return to our running example. We use a PLTLc-specific feature to explore the value range for the proteins – the *free variables*, which are specified by a leading \$.

- What is the probability that up to time point τ one of the proteins rises above v? We do not know which protein will start rising, so we use the disjunction.

  $\mathbf{P}_{=?}[\, \mathbf{F}_{[0,\tau]}\ protein\_a > \$v \mid protein\_b > \$v \mid protein\_c > \$v \,]$

Simulative model checking yields the domain of the free variable v and the probability of each interval, see Fig. 12. We observe that values beyond 150 are increasingly unlikely. Thus, we take $K = 160$ as upper bound (see Fig. 10), which cuts the infinite state space down to 33,386,248 states.

Analytical model checking. As long as the underlying semantics of a stochastic Petri net is described by a finite CTMC of manageable size, it can be analyzed using such standard stochastic analysis techniques as transient analysis, steady state analysis or analytical model checking (Schwarick and Tovchigrechko, 2010; Schwarick et al., 2011).

Transient analysis means to compute the transient probabilities to be in a certain state at a specific time point using, e.g., the uniformization or Jensen method (Stewart, 1994). Steady state analysis computes the steady state probabilities using, e.g., Gaussian elimination or Jacobi iteration (Parker,

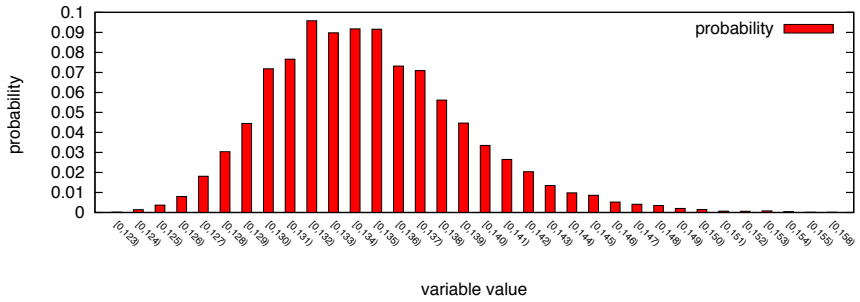

Fig. 12 Probability distribution of the value range for the protein places, determined by 10,000 stochastic simulation runs with $\tau = 200,000$ for the $\mathcal{SPN}$ repressilator model. Increasing the number of runs smoothes the bell shape, but does not shift the value range. Values beyond 158 are most unlikely.

2002). In analytical model checking, special behavioral properties can be checked, which have been expressed in, e.g., Continuous Stochastic Logic (CSL), a stochastic counterpart of CTL which was originally introduced in (Aziz et al., 2000).

For illustration, we compute for the bounded version of our repressilator model ($K = 160$) the probability that in the steady state there are k molecules of protein b, see Fig. 13. The obtained probability distribution tells us that we have oscillations with very sharp rise and fall, with peaks around 100. However, most of the time there are only a few proteins. The expectation value $E[protein\_b] = 33.18$ corresponds to the steady state value which we observe when averaging over a sufficient amount of simulation traces, see Fig. 14.

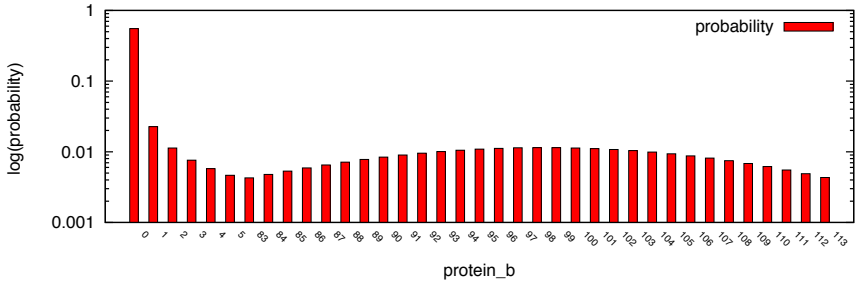

Fig. 13 Probability distribution for having k molecules of protein b in the steady state; x-axis: values with a probability below the (arbitrarily chosen) threshold (0.004) have been omitted, y-axis: given in log scale.

Likewise, we could use transient analysis to evaluate the following CSL formulae.

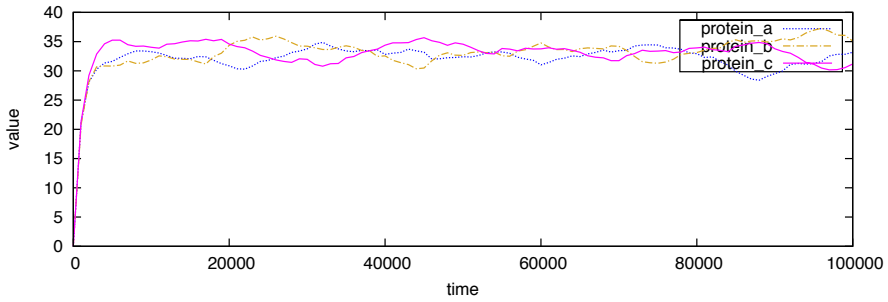

Fig. 14 Average of 1,000 simulation traces for the $\mathcal{SPN}$ repressilator model. Increasing the number of averaged traces smoothes the curves.

- What is the probability that at time point τ there are at least k molecules of protein b.
  $\mathbf{P}_{=?}$ [$\mathbf{F}_{[\tau,\tau]}$ ($protein\_b \geq k$)]
  Increasing τ will finally approach the steady state values for any $k \leq K$.
- What is the probability that up to time point τ there are at least k molecules of each protein at the same time.
  $\mathbf{P}_{=?}$ [$\mathbf{F}_{[0,\tau]}$ ($protein\_a \geq k$ & $protein\_b \geq k$ & $protein\_c \geq k$)]
  The probability is technically larger than 0 for any $k \leq K$ (as the property holds for the $\mathcal{QPN}$), but drops dramatically with increasing k, and reaches very fast insignificant values which are practically 0.

Marcie supports standard stochastic analysis techniques and analytical model checking for (Markovian) $\mathcal{SPN}$ and simulative model checking for $\mathcal{XSPN}$. We recommend (Heiner et al., 2009; Schwarick et al., 2011) for more illustrative examples.

4.3 Applications

$\mathcal{SPN}$ have been used in (Heiner et al., 2008, 2010) to model and analyse signal transduction pathways; the detailed analysis exploits analytical and simulative model checking.

A classical example of prokaryotic gene regulation, the lac operon, is taken in (Heiner et al., 2009) to demonstrate the power of $\mathcal{XSPN}$ for model-based design of wet lab experiments. This paper may also serve as a gentle introduction into the use of simulative model checking.

In (Marwan et al., 2012), $\mathcal{XSPN}$ have been applied to investigate phosphate regulation in enteric bacteria; modeling details and stochastic simulation runs are given. There one also finds more information of how to control stochastic simulation experiments in Snoopy.

5 Continuous Petri Nets ($\mathcal{CPN}$)

5.1 Modeling

Continuous Petri nets offer a graphical way to specify systems of ordinary differential equations (ODEs) (Heiner et al., 2008). The discrete tokens, which we had so far in $\mathcal{QPN}$ and $\mathcal{SPN}$, are exchanged in $\mathcal{CPN}$ by real-valued tokens, one token for each place. The instantaneous firing of a transition is carried out like a continuous flow. Its strength is determined by the continuous rate functions (deterministic rates, compare Fig. 1), which are assigned to each transition. $\mathcal{CPN}$ and $\mathcal{SPN}$ have the power to approximate each other as it is depicted in Fig. 1.

Modeling repressilator. Let us return to the repressilator model in Fig. 5. If we read tokens on each place as (real-valued) concentrations of species and associate a deterministic rate function with each transition, e.g., the same rate functions as given in Table 2, we can consider it as a continuous repressilator model. The underlying ODEs of the continuous model as generated by Snoopy are given in Table 3.

From these ODEs, we can see that each place in the Petri net model is mapped to a variable in the ODEs, and each variable gets its own equation. A place's pre-transitions increase the token value; thus, their rate functions appear as plus terms in the equation. Contrary, post-transitions decrease the token value; thus, their rate functions appear as minus terms. A transition which is pre- and post-transition yields two terms, which can be reduced by algebraically transforming the right-hand side of the equation.

5.2 Analysis

Continuous simulation. The ODEs induced by a given $\mathcal{CPN}$ are usually not linear, which calls for numerical integration algorithms. Snoopy supports 14 different stiff/unstiff ODE integrators to numerically solve the ODEs. These ODE solvers range from simple fixed-step-size solvers (e.g., Euler), which are suitable for unstiff $\mathcal{CPN}$ models, to more sophisticated variable-order, variable-step, multi-step solvers (e.g., Backward Differentiation Formulas (BDFs)), which have to be used for stiff $\mathcal{CPN}$ models. Snoopy's implementation of the latter solvers build on the library SUNDIALS CVODE (Hindmarsh et al., 2005).

Running continuous simulation for the $\mathcal{CPN}$ repressilator model yields plots as illustrated in Fig. 15.

Continuous model checking. The behavior of a $\mathcal{CPN}$ model is deterministic, i.e. each run with the same parameters yields the same results. Thus, the state space can be considered as being continuous and linear. It can be

Table 3 The unreduced ODEs induced by the $\mathcal{CPN}$ repressilator model, as generated by Snoopy. We deliberately give the unreduced ODEs to highlight the relation to the generating $\mathcal{CPN}$.

$$
\begin{aligned}
\frac{d\,gene\_a}{dt} &= (0.1 * gene\_a) + (0.0001 * blocked\_a) \\
&\quad - (0.1 * gene\_a) - (1 * protein\_c * gene\_a) \\
\frac{d\,protein\_a}{dt} &= (0.1 * gene\_a) + (1 * gene\_b * protein\_a) \\
&\quad - (0.001 * protein\_a) - (1 * gene\_b * protein\_a) \\
\frac{d\,blocked\_a}{dt} &= (1 * protein\_c * gene\_a) - (0.0001 * blocked\_a) \\
\frac{d\,gene\_b}{dt} &= (0.1 * gene\_b) + (0.0001 * blocked\_b) \\
&\quad - (0.1 * gene\_b) - (1 * protein\_a * gene\_b) \\
\frac{d\,protein\_b}{dt} &= (0.1 * gene\_b) + (1 * gene\_c * protein\_b) \\
&\quad - (0.001 * protein\_b) - (1 * gene\_c * protein\_b) \\
\frac{d\,blocked\_b}{dt} &= (1 * protein\_a * gene\_b) - (0.0001 * blocked\_b) \\
\frac{d\,gene\_c}{dt} &= (0.1 * gene\_c) + (0.0001 * blocked\_c) \\
&\quad - (0.1 * gene\_c) - (1 * protein\_b * gene\_c) \\
\frac{d\,protein\_c}{dt} &= (0.1 * gene\_c) + (1 * gene\_a * protein\_c) \\
&\quad - (0.001 * protein\_c) - (1 * gene\_a * protein\_b) \\
\frac{d\,blocked\_c}{dt} &= (1 * protein\_b * gene\_c) - (0.0001 * blocked\_c)
\end{aligned}
$$

explored by using, for example, continuous Linear Temporal Logic with constraints (LTLc) (Calzone et al., 2006), or PLTLc (Donaldson and Gilbert, 2008) in a deterministic setting. Both are interpreted over the continuous simulation trace generated by numerically integrating ODEs. Please refer to (Donaldson and Gilbert, 2008; Heiner et al., 2008) for details about how to use MC2 tools to do simulative model checking and a couple of biological examples.

For illustration we specify the following PLTLc property for the $\mathcal{CPN}$ repressilator model.

- Does finally the value of protein b first rise and then fall; with other words: does there exist a peak in the trace?
  $\mathbf{P}_{=?}$ [$\mathbf{F}$ [$(d(protein\_b) > 0)$ & $\mathbf{F}$ [$(d(protein\_b) < 0)$]]]

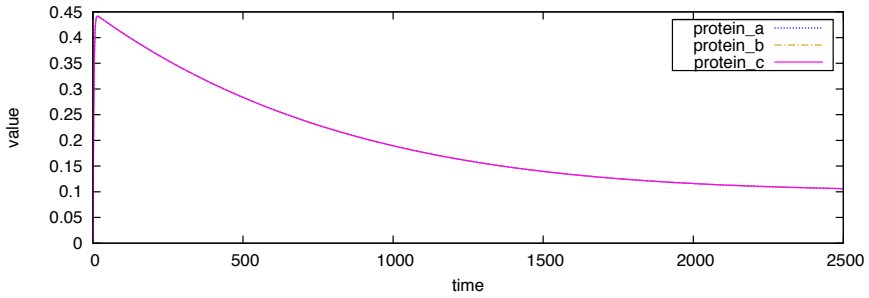

Fig. 15 Plot of a continuous simulation run for the $\mathcal{CPN}$ repressilator model. For rate functions, see Table 2. This plot suggests that the repressilator quickly reaches a steady state. The three curves for the three proteins coincide, so we see only one of them. Contrary, Fig. 11 suggests that the stochastic repressilator fluctuates around a steady state value.

The function $d(species)$ returns the derivative of the concentration of the species at each time point. The probability is 1, i.e., there is a peak, see Fig. 15.

Other analysis techniques. Besides, all standard ODEs analysis techniques e.g., bifurcation analysis, sensitivity analysis, and parameter scanning (Segel, 1980), are applicable when the ODEs are exported to suitable tools, e.g., Matlab (Matlab, 2013).

5.3 Applications

(Gilbert and Heiner, 2006) present results of an investigation to integrate Petri nets and ODEs for the modeling and analysis of biochemical networks, and apply their approach to a model of the influence of the Raf Kinase Inhibitor Protein (RKIP) on the Extracellular signal Regulated Kinase (ERK) signaling pathway.

A novel methodology for the engineering of biochemical network models is proposed in (Breitling et al., 2008) and illustrated for signalling pathways. It includes the structured design of ODEs using $\mathcal{CPN}$ and their systematic composition.

(Soliman and Heiner, 2010) discuss sufficient conditions for the unique construction of $\mathcal{CPN}$ models from ODEs. The challenge is to reveal the network structure which is hidden in a given ODE. Generally, this reverse problem does not have a unique solution, while the ODEs induced by a given $\mathcal{CPN}$ are uniquely defined.

6 Generalized Hybrid Petri Nets ($\mathcal{GHPN}$)

6.1 Modeling

Snoopy integrates all functionalities of its stochastic and continuous Petri nets ($\mathcal{SPN}$ and $\mathcal{CPN}$) into one net class, yielding generalized hybrid Petri nets ($\mathcal{GHPN}$) (Herajy and Heiner, 2012; Herajy, 2013). $\mathcal{GHPN}$ are specifically tailored (but not limited) to models that require an interplay between stochastic and continuous behavior. They provide a trade-off between accuracy and runtime of model simulation by adjusting the number of stochastic transitions appropriately, which can be done either statically (by the user) or dynamically (by the simulation algorithms). A typical application of $\mathcal{GHPN}$ is the hybrid representation of biochemical reactions at different scales (also called stiff systems), where slow reactions are represented by stochastic transitions and fast reactions by continuous transitions.

Modeling repressilator. For illustration, we now interpret our repressilator Petri net, re-using the rate functions given in Table 2, as a $\mathcal{GHPN}$ model. The 1-bounded places as determined by P-invariant analysis and the related transitions as determined by T-invariant analysis are kept discrete. The unbounded places and related transitions are approximated by continuous places and transitions, respectively. That is, places *gene_i* and *blocked_i*, and transitions *block_i* and *unblock_i* are treated as discrete, and all other nodes as continuous. To distinguish between discrete and continuous nodes, we choose different graphical representations, see Fig. 16.

6.2 Analysis

Hybrid simulation. Snoopy's hybrid simulation builds on Gillespie's direct method (Gillespie, 1977) to simulate stochastic transitions, and on continuous simulation to integrate the ODEs induced by the continuous transitions using SUNDIALS CVODE (Hindmarsh et al., 2005).

For example, if we still assign the rates in Table 2 to the hybrid model and consider them as stochastic or deterministic rates, depending on the transition type, hybrid simulation yields plots as illustrated in Fig. 17.

Simulative model checking. Likewise, we can use PLTLc (Donaldson and Gilbert, 2008) to do simulative model checking of a $\mathcal{GHPN}$ model, which also handles a subset of the state space, e.g., a set of finite outputs from hybrid simulation in Snoopy.

For the hybrid repressilator model, we specify the following property using PLTLc.

- What is the probability of *protein_i* ($i = a, b, c$) to be sometimes (finally) greater than 120?

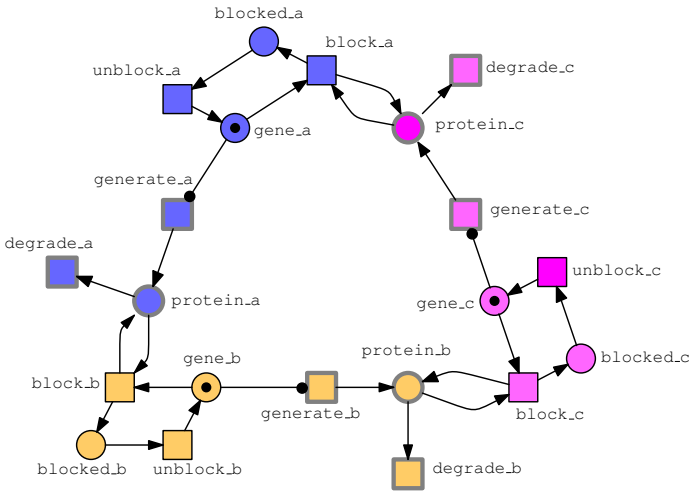

Fig. 16 The $\mathcal{GHPN}$ model of the repressilator, where the continuous places (transitions) are represented by shaded line circles (squares). Besides, the bidirectional arcs between places *gene_i* and transitions *generate_i* are replaced by read arcs (black dots as arc heads) in order to comply with the connection rules (a continuous transitions is not allowed to remove from or write to a discrete place).

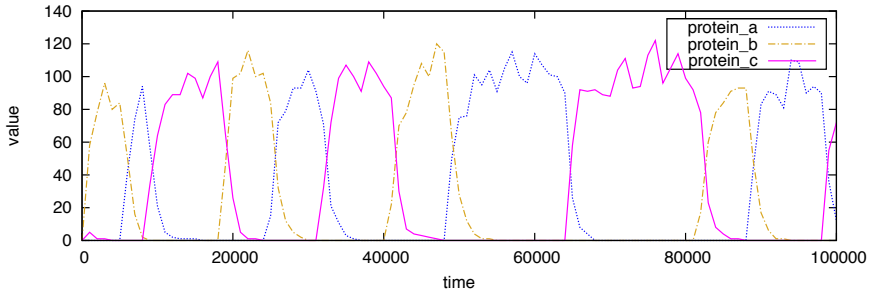

Fig. 17 Plot of one hybrid simulation run for the repressilator. For rate functions, see Table 2. This plot suggests that $\mathcal{GHPN}$ are able to capture the oscillation. Repeated runs look differently; thus stochasticity is captured as well.

$\mathbf{P}_{=?}$ [$\mathbf{F}$ [$protein\_i > 120$]]
With the same data as used for Fig. 17, it is evaluated for this single trace to 1 for *protein_c*, and 0 for the other proteins.

- What is the probability that there are k molecules of two proteins (here *protein_a* and *protein_b*) at the same time?
  $\mathbf{P}_{=?}$ [$\mathbf{F}$ [$(protein\_a \geq k)$ & $(protein\_b \geq k)$]]
  With the same data as used for Fig. 17, it is evaluated for this single run to 1 for $k = 1, 10, ..., 40$, and 0 for $k \geq 50$.

6.3 Applications

In (Herajy, 2013), $\mathcal{GHPN}$ are used to model and analyze three case studies: the intracellular growth of bacteriophage T7, the eukaryotic cell cycle, and the circadian rhythm. In all three cases, chemical reactions are divided into two groups: fast and slow.

7 Colored Extensions

7.1 Modeling

Colored Petri nets (Genrich and Lautenbach, 1979; Jensen, 1981) are a high-level extension of standard Petri nets, where a group of similar model components are represented by one component, each of which is defined as and thus distinguished by a color.

Colored Petri nets consist, as standard Petri nets, of places, transitions and arcs. Additionally, a colored Petri net model is characterized by a set of data types (Cardelli and Wegner, 1985), called color sets. Each place gets assigned a color set and may contain distinguishable tokens colored with a color of this color set.

Modeling repressilator. Fig. 18 gives a colored Petri net model for the repressilator model in Fig. 6. A color set *GeneSet* is defined with three colors, a, b and c to distinguish three genes. Each place gets assigned this color set *GeneSet*. By this way, we use one place to represent three similar objects, e.g. representing three protein objects as one colored place *protein*.

As there can be several tokens of the same color on a given place, the tokens on a place define a multiset over the place's color set. For example, in Fig. 18, we denote the initial marking for the place *protein* by a multiset expression, 1‘a++1‘b++1‘c, which means one token of each color of *GeneSet*.

Each transition gets a guard, which is a Boolean expression over variables, constants, and etc. The guard must be evaluated to true for the enabling of the transition. The trivial guard “true” is usually not explicitly given. For example, in Figure 18, all colored transitions have the trivial guard ”true”.

Each arc gets assigned an expression; the result type of this expression is a multiset over the color set of the connected place. For example, in Fig. 18, we define a variable x of *GeneSet*, which is used in arc expressions. The predecessor operator “-” in the arc expression $-x$ returns the predecessor of x in an ordered finite color set. For example, if $x = b$, then $-x$ returns a. If x is the first color, then it returns the last color. For example, if $x = a$, then $-x$ returns c. The result type of each arc expression is a multiset over *GeneSet*.

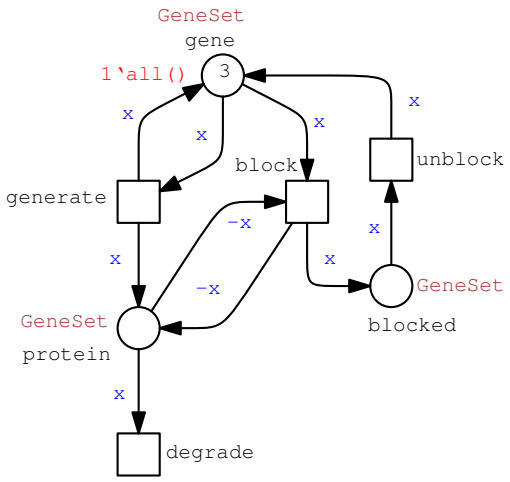

Fig. 18 A colored Petri net model for the repressilator. The declarations: colorset GeneSet = enum with a,b,c, and variable x: GeneSet. With this color set, this model corresponds exactly to the Petri nets in Fig. 5–7, 9. Reprinted from ref. (Heiner and Gilbert, 2012), Copyright 2013, with permission from Elsevier

Each uncolored net class has a colored counterpart (Liu, 2012), which inherits all features of its corresponding uncolored net class, e.g., $\mathcal{SPN}^{\mathcal{C}}$ enjoy all special arcs types and transition types of $\mathcal{SPN}$.

Snoopy provides various flexible ways to define declarations to be used in the annotations of colored Petri nets. For example, rich data types for color set definitions are supported: (1) simple types: dot, integer, string, boolean, enumeration and index, and (2) compound types: product and union. Concise initial marking specifications for larger color sets and individual rate function definitions for each transition instance are supported. Syntax checking ensures the syntactical correctness of constructed models.

A Petri net can be folded into a colored Petri net if the partition of place and transition nodes is given. After that, colored Petri nets can show their attractive advantage, scalability, e.g., changing the number of genes involved in the regulatory cycle just requires to adapt the color set GeneSet appropriately. For example, if we set GeneSet to, let's say, nine colors, a-i, a stochastic simulation plot for nine genes can be produced, illustrated in Fig. 19.

Vice versa, colored Petri nets with finite color sets can be automatically unfolded into uncolored Petri nets, which then allows the application of all of the existing powerful standard Petri net analysis techniques. For example, unfolding the colored Petri net in Fig. 18 generates the Petri net given in Fig. 6.

Modeling procedure. Usually, modeling with colored Petri nets follows the following procedure:

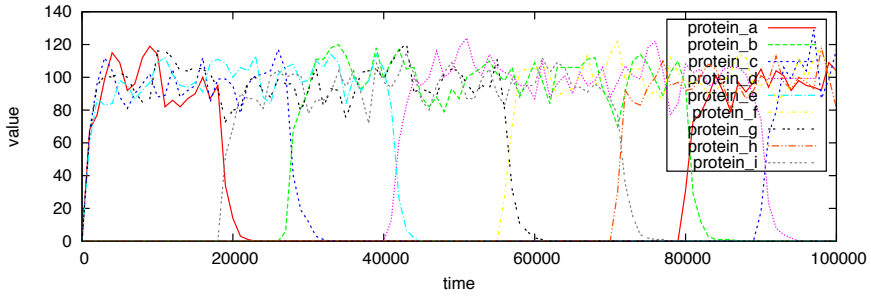

Fig. 19 Plot of a stochastic simulation run for the repressilator with nine genes. For rate functions, see Table 2.

1. Convert an uncolored Petri net into a colored Petri net by using the predefined color set *Dot*, which contains only one colour called *dot*.
2. Identify similar subnets in the uncolored Petri net model. For example, we can identify three similar subnets in Fig. 6, the three gene gates.
3. Define declarations, e.g., color sets, variables or constants. For example, we define a color set with three colors a,b,c to distinguish three similar subnets in Fig. 6.
4. Assign color sets to places and define the initial marking.
5. Write and assign arc expressions.
6. Define guards for transitions.
7. Check the syntax of inscriptions of the net, e.g., arc expressions and guards.

7.2 Analysis

In Snoopy, we support automatic unfolding of colored Petri nets into uncolored Petri nets, so all the analysis techniques mentioned above for each uncolored Petri net class can be equally applied to the corresponding colored Petri net class.

The key challenge when unfolding colored Petri nets is the computation of all transition instances, which in fact is a combinatorial problem, suffering from combinatorial explosion. For overcoming this, a constraint satisfaction approach (Tsang, 1993) has been employed. Specifically, the efficient search strategies of Gecode (Gecode, 2013) have been used to greatly improve the unfolding efficiency of colored Petri nets; see (Liu, 2012) for details.

7.3 Applications

In (Liu et al., 2012), colored Petri nets have been used to model cooperative ligand binding and the repressilator, with the aim to illustrate how to use colored Petri nets to model biological systems.

In (Liu, 2012), three case studies have been given. The first case study illustrates how to model a multi-cellular C. elegans vulval development system, where each cell is encoded as color. The second describes the modeling of coupled Ca^{2+} channels that are arranged in two-dimensional space, where each channel is encoded as a color. The third explores the use of colored Petri nets to model membrane systems, where a membrane system composed of compartments is modeled as a colored Petri net model by encoding each compartment as a color.

In (Gao et al., 2012), colored Petri nets are applied to model a tissue comprising multiple cells hexagonally packed in a honeycomb formation in order to describe the phenomenon of Planar Cell Polarity (PCP) signalling in Drosophila wing, which illustrate how to use colored Petri nets to address the multiscaleness problem in systems biology.

In (Gilbert et al., 2013), phase variation in bacterial colony growth has been studied using colored stochastic Petri nets, and continuous diffusion using colored continuous Petri nets.

8 Tools

BioModel Engineering of non-trivial case studies requires adequate tool support. We deploy a sophisticated toolkit covering the whole framework:

- **Snoopy** (Rohr et al., 2010; Heiner et al., 2012) is a platform to support the construction and animation/simulation of all the types of Petri nets discussed in this chapter, with an automatic conversion between them. Obviously, there may be a loss of information in some directions (cf. arrows labeled with abstraction in Fig. 1). The conversion between colored and uncolored net classes involves user-guided folding or automatic unfolding. Snoopy supports several data exchange formats, among them the following analysis tools in this list, as well as SBML import/export, which opens the door to a bunch of tools popular in Systems and Synthetic Biology.
- **Charlie** (Franzke, 2009; Wegener et al., 2011) permits the analysis of standard properties and applies standard techniques of Petri net theory, expanded by explicit CTL and LTL model checking.
- **Marcie** (Schwarick et al., 2011; Heiner et al., 2013) is a symbolic analysis tool of standard Petri net properties, and CTL model checking for $\mathcal{QPN}$ and CSL model checking for $\mathcal{SPN}$. Exact analyses are comple-

mented by approximative PLTLc model checking built on fast adaptive uniformization and distributed Gillespie simulation.
- **MC2(PLTLc)** (Donaldson and Gilbert, 2008) is a Monte Carlo Model Checker for properties written in (PLTLc). MC2(PLTLc) can operate with stochastic/deterministic simulation output, deterministic parameter scan output or even wet lab data.

The Petri net tools are publicly available at http://www-dssz. informatik.tu-cottbus.de, and MC2(PLTLc) at http://www.brc.dcs. gla.ac.uk/software/mc2/. All the Petri net models of the repressilator used in this chapter can be downloaded at http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Examples.

9 Conclusions

Petri nets have been widely used to model and simulate biochemical reaction networks and a variety of extensions of standard Petri nets have been developed. In this chapter we have first described a unifying Petri net framework that comprises a structured family of Petri net classes, and then given a brief introduction of each net class in this framework.

Our running case study illustrates how to easily move between these Petri net classes by help of our Petri net tool Snoopy. An elaborated treatment of another version of the repressilator (Elowitz and Leibler, 2000) in the various paradigms can be found in (Blätke et al., 2013b).

The chapter is meant to give a general idea of how to use Petri nets for modeling and analyzing biochemical reaction networks of various types with different modeling paradigms. We gave plenty of pointers to related literature where the interested reader may continue reading.

Acknowledgements

This work has been supported by Germany Federal Ministry of Education and Research (0315449H), Natural Scientific Research Innovation Foundation in Harbin Institute of Technology (HIT.NSRIF.2009005), and National Natural Science Foundation of China (61273226). We would like to thank David Gilbert and Wolfgang Marwan for many fruitful discussions, and Mary Ann Blätke, Mostafa Herajy, Christian Rohr, and Martin Schwarick for their assistance in model construction, software development and model checking.

WWW-List of tools

Snoopy http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy
Charlie http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Charlie
Marcie http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie
MC2 tool http://www.brc.dcs. gla.ac.uk/software/mc2/
Examples http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Examples

References

Aziz, A., Sanwal, K., Singhal, V., and Brayton, R. (2000). Model Checking Continuous-time Markov Chains. *ACM Transaction on Computational Logic*, 1(1):162–170.

Baldan, P., Cocco, N., Marin, A., and Simeoni, M. (2010). Petri nets for modelling metabolic pathways: a survey. *Natural Computing*, 9(4):955–989.

Blätke, M., Dittrich, A., Rohr, C., Heiner, M., Schaper, F., and Marwan, W. (2013a). JAK/STAT signalling - an executable model assembled from molecule-centred modules demonstrating a module-oriented database concept for systems and synthetic biology. *Molecular BioSystem*.

Blätke, M., Rohr, C., Heiner, M., and Marwan, W. (2013b). *A Petri Net based Framework for Biomodel Engineering; submitted* .

Blossey, R., Cardelli, L., and Phillips, A. (2008). Compositionality, stochasticity and cooperativity in dynamic models of gene regulation. *HFSP Journal*, 2(1):17–28.

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

Calzone, L., Chabrier-Rivier, N., Fages, F., and Soliman, S. (2006). Machine learning biochemical networks from temporal logic properties. In *Transactions on Computational Systems Biology*, LNCS 4220, pages 68–94. Springer.

Cardelli, L. and Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. *Computing Survey*, 17(4):471–522.

Chaouiya, C. (2007). Petri Net Modelling of Biological Networks. *Briefings in Bioinformatics*, 8(4):210–219.

Ciardo, G. (1994). Petri nets with marking-dependent arc cardinality. In *Properties and Analysis, Advances in Petri Nets, LNCS*, pages 179–198. Springer-Verlag.

Clarke, E. M., Grumberg, O., and Peled, D. A. (2001). *Model Checking*. MIT Press, Cambridge.

Donaldson, R. and Gilbert, D. (2008). A model checking approach to the parameter estimation of biochemical pathways. In *Proc. of the 6th Inter-*

national Conference on Computational Methods in Systems Biology, LNCS 5307, pages 269–287. Springer.

Elowitz, M. B. and Leibler, S. (2000). A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338.

Franzke, A. (2009). Charlie 2.0 – a multi-threaded Petri net analyzer. Master's thesis, Computer Science Department, Brandenburg University of Technology Cottbus.

Gao, Q., Gilbert, D., Heiner, M., Liu, F., Maccagnola, D., and Tree, D. (2012). Multiscale Modelling and Analysis of Planar Cell Polarity in the Drosophila Wing. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 99(PrePrints):1–1.

Gecode (2013). Gecode: An open constraint solving library.

Genrich, H. J. and Lautenbach, K. (1979). The analysis of distributed systems by means of predicate/transition-nets. In *Proc. of the International Symposium on Semantics of Concurrent Computation*, LNCS 70, pages 123–146. Springer.

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

Gilbert, D. and Heiner, M. (2006). From Petri nets to differential equations - an integrative approach for biochemical network analysis. In *Proc. of the 27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency*, LNCS 4024, pages 181–200. Springer.

Gilbert, D., Heiner, M., Liu, F., and Saunders, N. (2013). Colouring Space - A Coloured Framework for Spatial Modelling in Systems Biology. In *Proc. PETRI NETS 2013*, LNCS, page to appear. Springer.

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

Heath, A. P. and Kavraki, L. E. (2009). Computational challenges in systems biology. *Computer Science Review*, 3(1):1–17.

Heiner, M. (2009). *Understanding Network Behaviour by Structured Representations of Transition Invariants – A Petri Net Perspective on Systems and Synthetic Biology*, pages 367–389. Natural Computing Series. Springer.

Heiner, M., Donaldson, R., and Gilbert, D. (2010). *Petri Nets for Systems Biology*, chapter 3, pages 61–97. Jones & Bartlett Learning, LCC.

Heiner, M. and Gilbert, D. (2011). How Might Petri Nets Enhance Your Systems Biology Toolkit. In *Proc. PETRI NETS 2011*, pages 17–37. LNCS 6709, Springer.

Heiner, M. and Gilbert, D. (2012). Biomodel engineering for multiscale systems biology. *Progress in Biophysics and Molecular Biology*.

Heiner, M., Gilbert, D., and Donaldson, R. (2008). Petri nets for systems and synthetic biology. In *Proc. of the 8th international conference on Formal methods for computational systems biology (CMSB 2008)*, LNCS 5016, pages 215–264. Springer.

Heiner, M., Herajy, M., Liu, F., Rohr, C., and Schwarick, M. (2012). Snoopy - a unifying Petri net tool. In *Proc. PETRI NETS 2012*, LNCS 7347, pages 398–407. Springer.

Heiner, M. and Koch, I. (2004). Petri net based system validation in systems biology. In *Proc. ICATPN 2004, Bologna, June*, volume 3099 of *LNCS*, pages 216–237. Springer.

Heiner, M., Lehrack, S., Gilbert, D., and Marwan, W. (2009). Extended stochastic Petri nets for model-based design of wetlab experiments. *Transaction on Computational Systems Biology XI*, LNBI 5750:138–163.

Heiner, M., Rohr, C., and Schwarick, M. (2013). MARCIE - Model checking And Reachability analysis done effiCIEntly. In *Proc. PETRI NETS 2013*, LNCS, page to appear. Springer.

Heiner, M. and Sriram, K. (2010). Structural analysis to determine the core of hypoxia response network. *PLoS ONE*, 5(1):e8600.

Herajy, M. (2013). *Distributed Collaborative and Interactive Simulation of Large Scale Biochemical Networks*. PhD thesis, Brandenburg University of Technology Cottbus.

Herajy, M. and Heiner, M. (2012). Hybrid representation and simulation of stiff biochemical networks. *J. Nonlinear Analysis: Hybrid Systems*, 6(4):942–59.

Hindmarsh, A., Brown, P., Grant, K., Lee, S., Serban, R., Shumaker, D., and Woodward, C. (2005). Sundials: suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Software*, (35):363–396.

Hofestädt, R. (1994). A Petri Net Application of Metabolic Processes. *Journal of System Analysis, Modeling and Simulation*, 16:113–122.

Jensen, K. (1981). Coloured Petri nets and the invariant-method. *Theoretical Computer Science*, 14(3):317–336.

Liu, F. (2012). *Colored Petri Nets for Systems Biology*. PhD thesis, Brandenburg University of Technology Cottbus.

Liu, F., Heiner, M., and Rohr, C. (2012). Manual for colored Petri nets in Snoopy. http://www-dssz.informatik.tu-cottbus.de/publications/btu-reports/Manual_for_colored_Petri_nets_2012_03.pdf.

Lund, E. W. (1965). Guldberg and Waage and the Law of Mass Action. *Journal of Chemical Education*, 42(10):548.

Marsan, M. A., Balbo, G., Conte, G., Donatelli, S., and Franceschinis, G. (1995). *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons.

Marwan, W., Rohr, C., and Heiner, M. (2012). *Petri nets in Snoopy: A unifying framework for the graphical display, computational modelling, and simulation of bacterial regulatory networks*, volume 804 of *Methods in Molecular Biology*, chapter 21, pages 409–437. Humana Press.

Matlab (2013). http://www.mathworks.cn.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

Parker, D. (2002). *Implementation of symbolic model checking for probabilistic systems*. PhD thesis, University of Birmingham.

Petri, C. A. (1962). *Kommunikation mit Automaten*. PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2.

Pnueli, A. (1981). The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13(1):45–60.

Reddy, V. N., Mavrovouniotis, M. L., and Liebman, M. N. (1993). Petri net representations in metabolic pathways. In *Proc. of the 1st International Conference on Intelligent Systems for Molecular Biology*, pages 328–336. AAAI Press.

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

Schwarick, M., Rohr, C., and Heiner, M. (2011). Marcie - model checking and reachability analysis done efficiently. In *Proc. of the 8th International Conference on Quantitative Evaluation of SysTems*, pages 91–100. IEEE.

Schwarick, M. and Tovchigrechko, A. (2010). IDD-based model validation of biochemical networks. *Theoretical Computer Science*.

Segel, L. A. (1980). *Mathematical Models in Molecular Cellular Biology*. Cambridge University Press.

Soliman, S. and Heiner, M. (2010). A unique transformation from ordinary differential equations to reaction networks. *PLoS ONE*, 5(12):e14284.

Stewart, W. (1994). *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press.

Tsang, E. P. K. (1993). *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego.

Wegener, J., Schwarick, M., and Heiner, M. (2011). A Plugin System for Charlie . In *Proc. International Workshop on Concurrency, Specification, and Programming (CS&P 2011)*, ISBN: 978-83-62582-06-8, pages 531–554. Białystok University of Technology.