Chapter 6
A Petri-Net-Based Framework for Biomodel Engineering

Mary Ann Blätke, Christian Rohr, Monika Heiner, and Wolfgang Marwan

Abstract Petri nets provide a unifying and versatile framework for the synthesis and engineering of computational models of biochemical reaction networks and of gene regulatory networks. Starting with the basic definitions, we provide an introduction into the different classes of Petri nets that reinterpret a Petri net graph as a qualitative, stochastic, continuous, or hybrid model. Static and dynamic analysis in addition to simulative model checking provide a rich choice of methods for the analysis of the structure and dynamic behavior of Petri net models. Coloring of Petri nets of all classes is powerful for multiscale modeling and for the representation of location and space in reaction networks since it combines the concept of Petri nets with the computational mightiness of a programming language. In the context of the Petri net framework, we provide two most recently developed approaches to biomodel engineering, the database-assisted automatic composition and modification of Petri nets with the help of reusable, metadata-containing modules, and the automatic reconstruction of networks based on time series data sets. With all these features the framework provides multiple options for biomodel engineering in the context of systems and synthetic biology.

Keywords Automatic network reconstruction · Biomodel engineering · Dynamic systems modelling · Modular modelling · Petri nets · Molecular regulatory networks · Reverse engineering

M.A. Blätke · W. Marwan (✉)
Otto-von-Guericke-Universität Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany
e-mail: wolfgang.marwan@ovgu.de

M.A. Blätke
e-mail: mary-ann.blaetke@ovgu.de

C. Rohr · M. Heiner
Brandenburg University of Technology, Platz der Deutschen Einheit 1, 03046 Cottbus, Germany

C. Rohr
e-mail: christian.rohr@b-tu.de

M. Heiner
e-mail: monika.heiner@b-tu.de

P. Benner et al. (eds.), *Large-Scale Networks in Engineering and Life Sciences*,
Modeling and Simulation in Science, Engineering and Technology,
DOI 10.1007/978-3-319-08437-4_6

6.1 Introduction

Petri nets are mathematical structures that form the core of a versatile framework for the modeling, analysis, and simulation of (bio-)chemical networks and for the engineering of biomodels. In this chapter, we will provide a comprehensive overview of the different classes of Petri nets and review techniques for their static and dynamic analysis. We will then explain some advanced Petri-net-based techniques for modeling and engineering of biomodels: colored Petri net modeling, modular modeling, and automatic network reconstruction. As an introduction to this chapter, we will now provide a brief contextual overview on these topics and show how the different components contribute to an integrative framework for biomodel engineering. At the end of this chapter, we briefly introduce the widely used Petri net tools Snoopy, Charlie, and MARCIE, which were used for all applications mentioned in this chapter.

The basic idea of Petri nets has been introduced in the 1960s by Carl Adam Petri [68]. They are directed bipartite multigraphs that have been widely studied. Directed bipartite graphs consist of two disjoint sets of nodes U and V, where a directed edge connects a node in U to one in V, or vice versa. Directed bipartite multigraphs like Petri nets allow two nodes to be connected by multiple arcs. Upon appropriate interpretation of the two types of nodes, places, and transitions, Petri nets work as a formal modeling language for causally coupled processes that may proceed concurrently as it is typically the case in (bio-)chemical reaction networks.

The first application of Petri nets to biological processes was published 1993 by Reddy and coworkers [71]. Up to now there are numerous publications illustrating the versatility of Petri nets and their use for metabolic networks [52, 53, 83], gene regulatory networks [14, 15], and signaling networks [11, 16, 37, 73], as well as for the integration of different types of biological networks [76]. In addition, there are some review papers about the use of Petri nets in systems biology like [69].

The semantics of Petri nets supports the direct and natural representation of the kinetics of chemical reactions and even of complex mechanisms of molecular interactions as they occur within a living cell. In quantitative Petri nets, the kinetics are implemented via the firing rate equations of each transition. They can be defined as the mass action law of chemical reactions or may follow more complex kinetic laws like, for example, the Michaelis–Menten kinetics for enzymatic reactions [64] or the Hill kinetics to represent cooperativity [48] in continuous, stochastic, or hybrid scenarios. In describing complex molecular mechanisms, the operational semantics of Petri nets is particularly useful and easy to be used for obtaining realistic models and accordingly realistic simulations. Operational semantics means that the Petri net, here describing molecular mechanisms, is equivalent to a protocol, which is immediately executable on an abstract machine or on a real computer [27]. In this sense, molecular mechanisms encoded as a Petri net can be directly executed on a computer, and all possibly emerging combinatoric or nonlinear effects will be revealed accordingly.

Structural analysis of a Petri allows one to explore the behavior of the model by employing appropriate tools. Structural analysis provides important options in addition to model checking. It is also a basis of certain advanced biomodel engineering techniques like algorithmic mutation of models [10].

The graphical display of Petri nets is intuitive and similar to the way biochemists usually draw their molecular reaction schemes. By using an appropriate tool like Snoopy [44, 63], a given Petri net graph can be interpreted automatically as qualitative, continuous, stochastic, or hybrid model and directly run by one of the built-in simulators. Accordingly, a Petri net is an executable graphical representation of a computational model. The WYSIWYG representation is of considerable benefit since it enables mathematically less trained experimentalists to assess the correctness and validity of a model. The Petri net editor Snoopy supports moreover hierarchy and logical nodes, technical add-ons to the core concept of Petri nets that facilitate the modeling and visualization of large networks, as we will show.

The colored extension of low-level Petri nets, also supported by Snoopy [54], combines the strengths of Petri nets with the expressive power and mightiness of programming languages. Colored Petri nets are especially useful for the generation of multiscale models, but also for other scenarios where large populations of molecules or populations of cells are considered in time and space. Unfolding algorithms translate colored Petri nets into low-level Petri nets. Thus, colored Petri nets can enjoy the low-level Petri net analysis techniques as well.

Petri nets provide an ideal framework for the engineering of biomodels; see Fig. 1. There are two fundamental concepts of creating a biomodel: the forward and the reverse engineering approach. Forward engineering, also called bottom-up modeling, starts with biological knowledge about molecules and molecular interaction mechanisms, which is translated into a biomodel [51], a Petri net in our case. Most common are coherent, monolithic models. Alternatively, forward engineering may be performed by designing small Petri nets in the form of modules that allow the automatic composition for obtaining functional, executable Petri nets [9]. These modules are more than just Petri nets. They may contain metadata documenting knowledge and encoding functionally relevant biological information of the Petri net nodes [9]. Based on the modular organization and on the metadata, these modules can be mutated through appropriate algorithms to mimic genetic mutational analysis [10], which is quite common in wet biological research.

The alternative way is reverse engineering, also called top-down modeling [51]. Here, experimental data sets are used to directly infer structure and dynamic behavior of a biomodel. Reverse engineered models may contain nodes that represent experimentally evident comprehensive states of the system without necessarily resolving the molecular details as it is usually the case in forward engineered models. One reverse engineering approach to be highlighted in this chapter is automatic net-

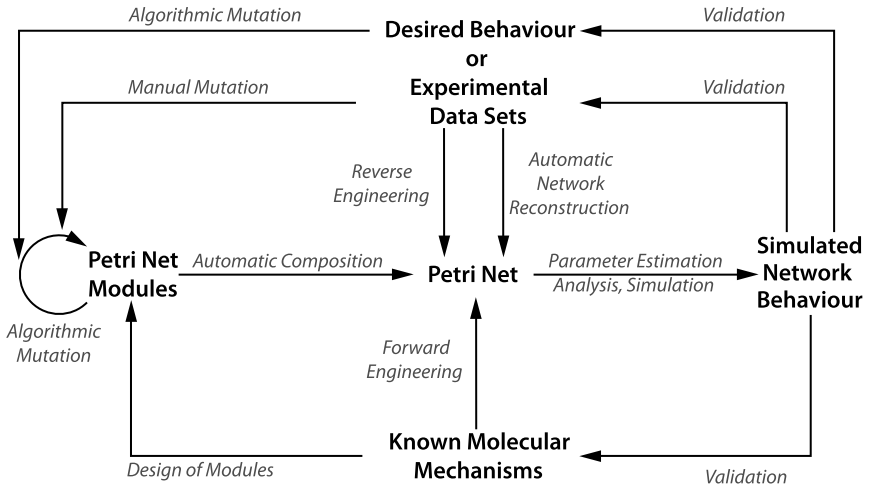


Fig. 1 Integrative framework for biomodel engineering based on Petri nets. Petri net models can be generated in various ways, manually or automatically, based on known molecular mechanisms (forward engineering; bottom-up) or solely on data sets (reverse engineering; top-down) to reflect the true or the desired behavior of a system of interest. Typically modeling is an iterative process that includes the validation of the network against a given or pre-defined behavior and its modification to meet the requirements. Petri net modules may be used as building blocks with validated properties. Automatic network reconstruction is the name of a method for the reverse engineering of Petri nets based on discrete optimization [62], which is described in this chapter. Note that there are numerous other methods for the reverse engineering of molecular or gene regulatory networks (for reviews, see [38, 60, 67, 78, 81, 82])

work reconstruction. This method converts a time series data set into a complete set of Petri nets that all are able to reproduce this data set, eliminating any bias introduced by the user. Experimental data sets can be enriched or replaced by the description of how the system is wanted to behave. Networks modified or reengineered to meet certain demands can be obtained through reverse engineering or mutation algorithms [10]; see Fig. 1.

No matter how a biomodel was generated (forward or reverse) or modified, its behavior should be explored or validated by simulation or model checking. Accordingly, biomodel engineering typically is an iterative approach, see Fig. 1.

Before explaining in detail how Petri nets support the various options of creating and simulating biomodels, we will give a brief overview of the different ways of how biomodels in terms of (bio-)chemical reaction networks are usually represented. As a simple, yet non-trivial small network, let us consider a simplified version of the repressilator, here called *simplified repressilator*, which will be used as running example throughout the chapter; see Fig. 2.

Simplified Repressilator In general, the repressilator is a cyclic negative feedback loop composed of three repressor genes and of their corresponding promotors [25]. Each of the three interconnected transcriptional repressor systems (TRSs)

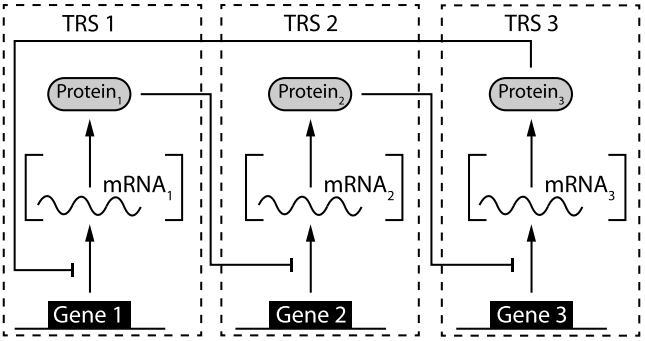


Fig. 2 Schematic plot of the simplified repressilator. The repressilator is a cyclic negative-feedback loop composed of three repressor proteins and their corresponding genes. Each repressor protein inhibits the transcription of its target gene [25] by reversibly binding to its specific binding site on the DNA. The simplified repressilator shown in this figure is not more than a toy model inspired by the repressilator originally implemented in *E. coli* as a synthetic circuit [25]. The simplified repressilator neglects the formation of the mRNA and allows the degradation of a repressor protein while it is bound to its DNA target to inhibit the transcription of the downstream gene. The model is used as a running example throughout this chapter. Abbreviation: TRS—transcriptional repressor system

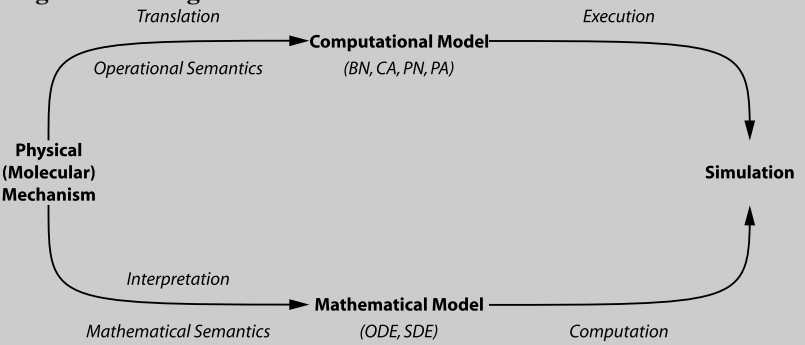
consists of the gene encoding the mRNA from which the respective repressor protein is translated (synthesized). Each repressor protein reversibly binds to its specific repressor binding site. The bound repressor protein prevents the transcription of the gene it controls. For the simplified repressilator, we assume that each gene directly catalyzes the synthesis of the repressor protein it encodes. We neglect the mRNA intermediates and the explicit processes of transcription and translation; see Fig. 2. However, we explicitly consider the binding and unbinding of the repressor proteins to their target promotor sites and the degradation of the repressor proteins in the free and bound forms [59]. Note that the simplified repressilator is just a toy network to be used for demonstration purposes and not meant as a computational model of the original repressilator that has been implemented in *E. coli* [25].

There are different standard ways of representing reaction or signaling networks like the simplified repressilator shown in Fig. 2:

- List (set of stoichiometric reactions in a reaction/species centric form).
- Hypergraph (graph where arcs connect to any number of nodes).
- Bipartite graph (graph consisting of arcs and two types of nodes, where nodes of the same type cannot be connected, e.g. Petri nets).
- Incidence matrix (equivalent to stoichiometric matrix).
- ODE (ordinary differential equation).

In Fig. 3, we illustrate these representation styles by taking one TRS of the simplified repressilator as example.

Box 1: Petri nets in the context of major alternative formalisms used in biological modeling and simulation



Determinism of modeling languages

Model type		BN	(col)PN	ODE	SDE
Deterministic	$0/1$	+	+	−	−
	$\mathbb{N}_0$	(+)	+	−	−
	$\mathbb{R}_0^+$	−	+	+	−
Stochastic	$0/1$	+	+	−	−
	$\mathbb{N}_0$	(+)	+	−	−
	$\mathbb{R}_0^+$	−	−	−	+
Hybrid	$0/1$	−	+	−	−
	$\mathbb{N}_0$	−	+	−	−
	$\mathbb{R}_0^+$	−	+	−	−

Areas of application

	BN	PN	ODE	SDE
Metabolism	−	+	+	−
Signaling	+	+	+	+
Gene Regulation	+	+	+	+
Populations	−	+	+	(+)

To compare common frameworks for modeling and simulation of molecular regulatory networks, one may distinguish between computational and mathematical models [27]. Starting from a network defined by the causal (molecular) interactions of its components, computational and mathematical models are obtained in alternative ways. Mathematical models describe with the help of equations how the network and its components are expected to quantitatively behave, usually as functions of time. The mechanisms per se are not necessarily captured by the mathematical semantics, and the mathematical model of the molecular mechanisms is in praxi often based on certain assumptions or simplifications as the result of a considerable degree of abstrac-

tion. Simulation results are then usually obtained by numerically solving the system of ordinary or stochastic differential equations (ODEs, SDEs). In contrast, computational models like Boolean networks (BN), cellular automata (CA), Petri nets (PN), or process algebras (PA) are obtained by translating the interaction mechanisms with the help of an operational semantics. Computational models can then be directly executed on an abstract machine or on a real computer in order to perform a simulation. Alternatively, computational models may be translated into differential equations which are then solved numerically. In other words, mathematical models are primarily obtained by interpretation and computational models primarily by translation of the mechanisms of causal interaction of the physical (molecular) components of the network. It depends on the class of computational model how direct this translation can be. Petri nets and process algebras allow the most direct representation of simple and complex molecular mechanisms, whereas translation into Boolean networks or cellular automata involves simplifications and abstractions. The way of how to obtain a model matters when nonlinear effects determine the dynamic behavior of a network. Nonlinear effects are caused by complex kinetic interactions of network components, which are prevalent in molecular biology. In this case, translation of the molecular mechanisms is straightforward in predicting the dynamic behavior and in implicitly representing functionally relevant combinatoric effects that may occur e.g. in clusters of interacting molecules. For obtaining deterministic, stochastic, and hybrid models, Petri nets are the most versatile framework in terms of allowing discrete and continuous approaches. In contrast to the other frameworks, Petri nets allow one to avoid abstractions as much as possible. The graphical representation of a Petri net representing a mechanism of interest remains the same no matter whether the Petri net is executed as deterministic, stochastic, hybrid, discrete, or continuous model.

6.2 Petri Net Framework

The Petri net framework consists of four Petri net classes according to the four modeling paradigms (qualitative, continuous, stochastic, and hybrid; see Fig. 4), which we will now explain in more detail. For formal definitions of the different classes of Petri nets and standard Petri net notation, see [4, 41] and references therein.

6.2.1 Qualitative Paradigm

Qualitative Petri nets $\mathcal{QPN}$ provide the basis for the definition of all other classes of Petri nets. With $\mathcal{QPN}$ describing the qualitative structure of a reaction network or gene regulatory network, one can apply different modeling paradigms (continuous,

(a) List of Reactions

$t_{i,1}$: $e_i \rightarrow^{g} e_i + X_i$
$t_{i,2}$: $X_i \rightarrow^{d} \emptyset$
$t_{i,3}$: $e_i + X_j \rightarrow^{\alpha_0} r_i$
$t_{i,4}$: $r_i \rightarrow^{\alpha_1} e_i + X_j$
$t_{i,5}$: $r_i \rightarrow^{d_r} e_i$

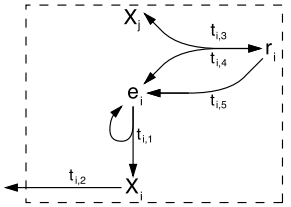
(b) Incidence Matrix

	$t_{i,1}$	$t_{i,2}$	$t_{i,3}$	$t_{i,4}$	$t_{i,5}$
e_i	0	0	−1	+1	+1
r_i	0	0	+1	−1	−1
X_i	+1	−1	0	0	0
X_j	0	0	−1	+1	0

(c) ODEs

$$\frac{dX_j}{dt} = g[e_j] - d[X_j] - \alpha_0[X_j][e_i] + \alpha_1[r_i]$$
$$\frac{dr_i}{dt} = \alpha_0[X_j][e_i] + \alpha_1[r_i] - d_r[r_i]$$
$$[r_i] + [e_i] = 1$$

(e) Hypergraph



(f) Bipartite Graph (Petri Net)

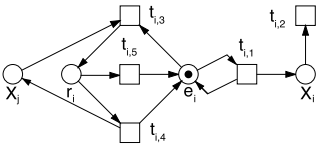


Fig. 3 Different representation styles of the simplified repressilator. Here, we show one TRS of the simplified repressilator with $(i, j) = \{(1, 3), (2, 1), (3, 2)\}$. The simplified repressilator is represented as (**a**) list of reactions, (**b**) incidence matrix, (**c**) ODEs, (**d**) hypergraph, and (**f**) bipartite graph (e.g. Petri net). As in [59], we apply mass action and assume that the kinetic constants are the same for each TRS. For simulation purposes, we set the parameters to $g = 0.05\ s^{-1}$, $d = d_r = 0.003\ s^{-1}$, $\alpha_0 = 0.5\ s^{-1}$ and $\alpha_1 = 0.01\ s^{-1}$ [59]. Abbreviations: e_i—free repressor binding site, r_i—bound repressor binding site, X_i – free repressor protein

stochastic, hybrid) by switching the Petri net class. $\mathcal{QPN}$ are referred to as "Petri nets" throughout this section.

Definition 1 (Petri net) A Petri net is a quadruple $N = (P, T, f, m_0)$, where:

- P, T are finite, non-empty, disjoint sets. P is the set of places, and T is the set of transitions.
- $f : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}_0$ defines the set of directed arcs, weighted by non-negative integer values.
- $m_0 : P \rightarrow \mathbb{N}_0$ gives the initial marking.

6.2.1.1 Elements

A Petri net is a finite bipartite directed multigraph consisting of two types of nodes, places (drawn as circles) and transitions (drawn as rectangles), that are interconnected by weighted directed arcs. Places are exclusively connected to transitions and vice versa. Depending on the definable properties of a Petri net, a place can be empty or marked by one or more tokens. Upon firing, tokens move from transition's pre-places to its post-places [66]; see Fig. 5.

Places (= circles) refer to conditions or entities. In a biological context, places may represent populations, species, organisms, multicellular complexes, single cells, proteins (enzymes, receptors, transporters, etc.), other molecules, or ions. But

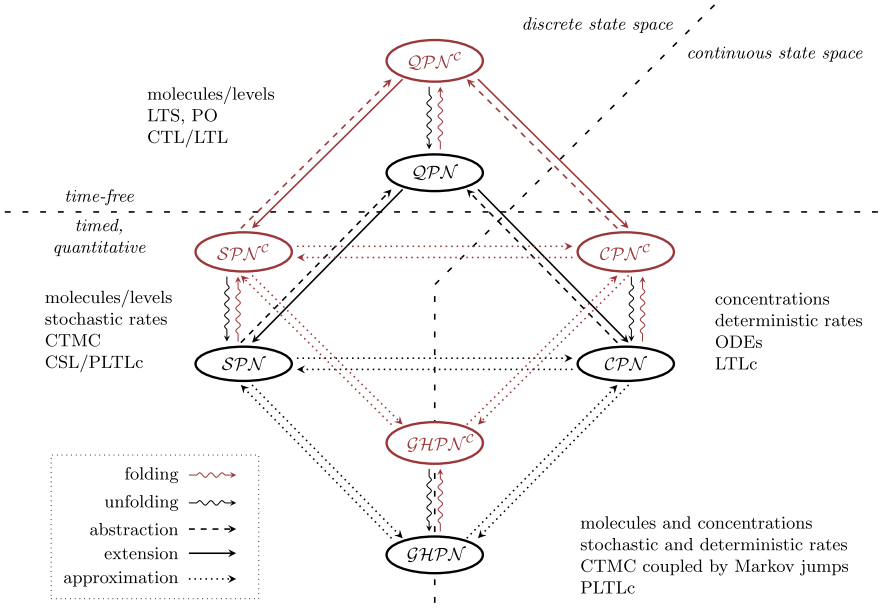


Fig. 4 Conceptual framework. The standard low-level Petri net formalism offers four classes, qualitative Petri nets ($\mathcal{QPN}$), stochastic Petri nets ($\mathcal{SPN}$), continuous Petri nets ($\mathcal{CPN}$), and generalized hybrid Petri nets ($\mathcal{GHPN}$) that differ in their type of state space and their relation with respect to time. Each Petri net class can be derived from one of the others by abstraction, extension, or approximation. All Petri net classes can be projected to the high-level colored Petri net framework. Colored Petri nets can be obtained by folding of the corresponding low-level Petri net, and low-level Petri nets can be obtained through unfolding colored Petri nets. Taken from [58]

places can also represent physical variables like temperature, pH-value, or membrane potential. Only places carry tokens; see Fig. 5(a), (b).

Transitions (= squares) describe state shifts, system events, or activities in a network. In a biological context, transitions refer to (bio-)chemical reactions, molecular interactions, or conformational changes. Places giving input to (getting output from) a transition are called pre-places ${}^\bullet t$ (post-places $t^\bullet$). Pre-transitions ${}^\bullet p$ and post-transitions $p^\bullet$ of a place are accordingly defined. Transitions consume tokens from their pre-places and produce tokens on their post-places according to the arc weights; see Fig. 5(a), (d).

Directed arcs (= arrows) specify the causal relationships between transitions and places. Thus, they indicate the effect of firing a transitions on the local token distribution. Arcs define the direction in which (bio-)chemical reactions take place. Arcs connect only nodes of different types; see Fig. 5(c). Each arc has an integer arc weight greater than zero. The arc weight sets the number of tokens that are consumed or produced upon firing of a transition and represents the stoichiometry of a (bio-)chemical reaction.

Tokens (= dots or numbers within a place) are variable elements of a Petri net and represent the discrete value of a condition or an entity. Tokens are consumed

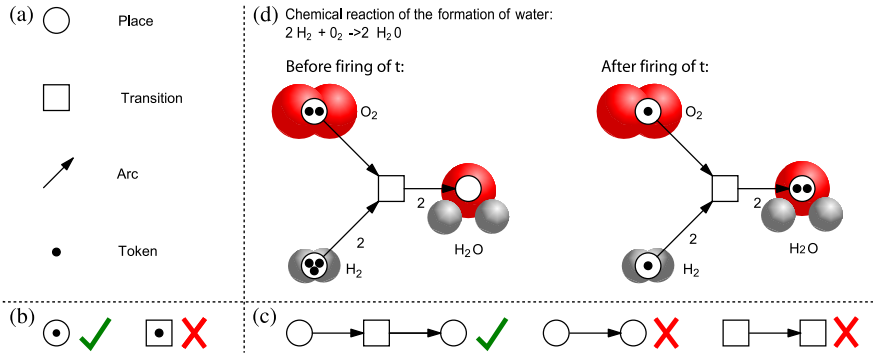


Fig. 5 Petri net formalism. (**a**) Petri nets consist of places, transitions, arcs, and tokens. (**b**) Just places are allowed to carry tokens. (**c**) Two nodes of the same type cannot be connected with each other. (**d**) The Petri net shown here represents the chemical reaction of the formation of water. Oxygen atoms (molecules) are shown in *red* and hydrogen atoms (molecules) are shown in *grey*. The arc weights indicate the stoichiometry of the reaction. A transition is enabled and may fire if its pre-places are sufficiently marked by tokens

and produced by firing transitions; see Fig. 5(a), (d). In (bio-)chemical reaction networks, tokens may refer to a concentration level or to a discrete number of individuals of a species, for example, proteins, ions, organic, and inorganic molecules. Tokens may also represent the value of physical variables like temperature, pH value, or membrane voltage according to the definition of places they mark. A particular arrangement of tokens over a net is called the marking m. For a given marking m of the Petri net, $m(p)$ refers to the number of tokens in a given place p.

6.2.1.2 Semantics

The *Petri net semantics* describes the behavior of the net, which is defined by the firing rule consisting of a precondition and the firing itself; see also Definition 2 for a formal description. The firing of a transition depends on the marking of its pre-places. A transition is enabled and may fire if all pre-places are sufficiently marked; see also Fig. 5(b). If a transition has no pre-places, it is always enabled to fire. The firing of a transition moves tokens from its pre-places to post-places and accordingly changes the number of tokens in these places. As a result, some transitions may not be enabled any more, whereas others get enabled. In the case that more than one transition is enabled in a given marking, only one of the enabled transitions is allowed to fire. Compared to boolean networks, transitions in Petri net fire asynchronously.

Definition 2 (Firing rule) Let $N = (P, T, f, m_0)$ be a Petri net:

- A transition is enabled in marking m, written as $m[t\rangle$, if $\forall p \in {}^\bullet t : m(p) \geq f(p,t)$, else disabled.

- A transition t, which is enabled in m, may fire.
- When t in m fires, a new marking m' is reached, written as $m[t\rangle m'$, with $\forall p \in P : m'(p) = m(p) - f(p,t) + f(t,p)$.
- The firing happens atomically and does not consume any time.

6.2.1.3 State Space

The behavior of a net emerges from the repeated firing of transitions. All ordered firing sequences define the behavior of the Petri net model. The set of all markings of the Petri net reachable from the initial marking m_0 defines the state space; see Fig. 4. The sequential individual firing of enabled transitions generates a possible path through the discrete state space. The two most common representations of the discrete state space and its transition relation are the labeled transitions system (LTS), also known as reachability graph, and the finite prefix of maximal branching process (PO prefix for short). The LTS describes the behavior of the Petri net by all (totally ordered) interleaving sequences, whereas the PO prefix describes the network behavior through all partially ordered sequences of transition firing events. Both kinds of representations of the discrete state space can be used for analysis purposes, for example, model checking, see Sect. 6.3.2.

Figure 4 shows that $\mathcal{QPN}$ are characterized to be time-free, meaning there is no time associated with transitions or sojourn time of tokens. Thus, the discrete state space represents all possible markings of a net that can sequentially occur independently of the time.

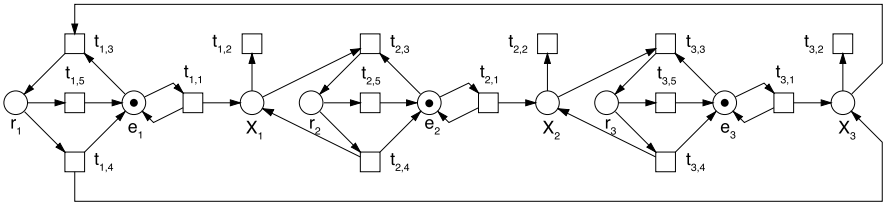
Simplified Repressilator The complete model of the simplified repressilator with degradation of the bound repressor protein (the repressor protein bound to its specific regulatory binding site on the DNA), which we are using throughout this chapter, is given in Fig. 6(a), as well as the kinetic rate functions and constants that we use further to obtain the quantitative behavior through simulations.

Every $\mathcal{QPN}$ model, for example, the model of the simplified repressilator in Fig. 6(a), can be extended to a quantitative model, stochastic, continuous, or hybrid by adding kinetic rates to the transitions. Adding kinetic rates does not induce any changes in the qualitative network structure. Since the qualitative network structure is maintained in all modeling paradigms, the same powerful analysis techniques can be applied to all Petri net classes; see Sect. 6.3. In the following sections, we explain the realization of the quantitative modeling paradigms in Petri nets.

6.2.2 Continuous Paradigm

A widely used approach in the modeling and simulation of (bio-)chemical reaction networks is to represent a system and its behavior as a continuous model in the form of a set of ODEs. Figure 4 shows that a time-dependent continuous Petri net ($\mathcal{CPN}$)

(a) Petri Net



(b) Rate Functions and Parameters

$t_{i,1} := g[e_i]$ $t_{i,4} := \alpha_1[r_i]$ $g = 0.05s^{-1}$ $\alpha_0 = 0.5s^{-1}$
$t_{i,2} := d[X_i]$ $t_{i,5} := d_r[r_i]$ $d = d_r = 0.003s^{-1}$ $\alpha_1 = 0.01s^{-1}$
$t_{i,3} := \alpha_0[X_j][e_i]$

Fig. 6 Petri net of the simplified repressilator. (**a**) depicts the complete Petri net model of the simplified repressilator consisting of three TRSs (compare also Figs. 2 and 3) with rate functions and parameters given in (**b**)

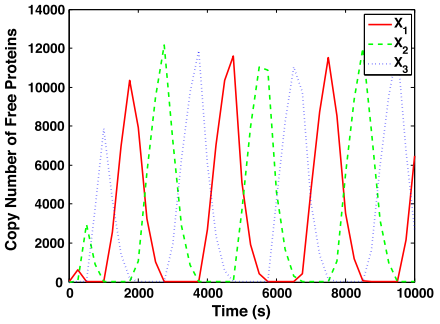


Fig. 7 Continuous simulation of the simplified repressilator. The diagram illustrates the results of the continuous simulation and shows copy numbers of the free repressor proteins X_i (repressor protein molecules currently not bound to the DNA) over time. The continuous simulation was performed with one copy of each of the three genes

can be derived from the time-free $\mathcal{QPN}$ by adding deterministic firing rates; see [41] for a formal definition. The marking of a place is now represented by continuous values, rather than by the integer number of tokens as in the case of $\mathcal{QPN}$. The semantics of $\mathcal{CPN}$ is described through the corresponding set of ODEs, which is encoded by the network structure and the added deterministic firing rates. Thus, the firing of the transitions is continuous itself.

Since a $\mathcal{CPN}$ is a continuous and deterministic model, each simulation run gives the same result for a given $\mathcal{CPN}$.

Simplified Repressilator The continuous behavior of the simplified repressilator given in Fig. 6 yields a sustained oscillation of the three repressor proteins with alternating peaks; see Fig. 7.

Further Reading $\mathcal{CPN}$ directly represent the molecular kinetic mechanisms within a biochemical reaction network in the form of an operational semantics and at the same time uniquely specify an ODE system that mathematically describes the dynamic behavior of the system [77]. Vice versa, the extraction of the reaction network underlying a given ODE system is unique only under certain conditions, see [77].

6.2.3 Stochastic Paradigm

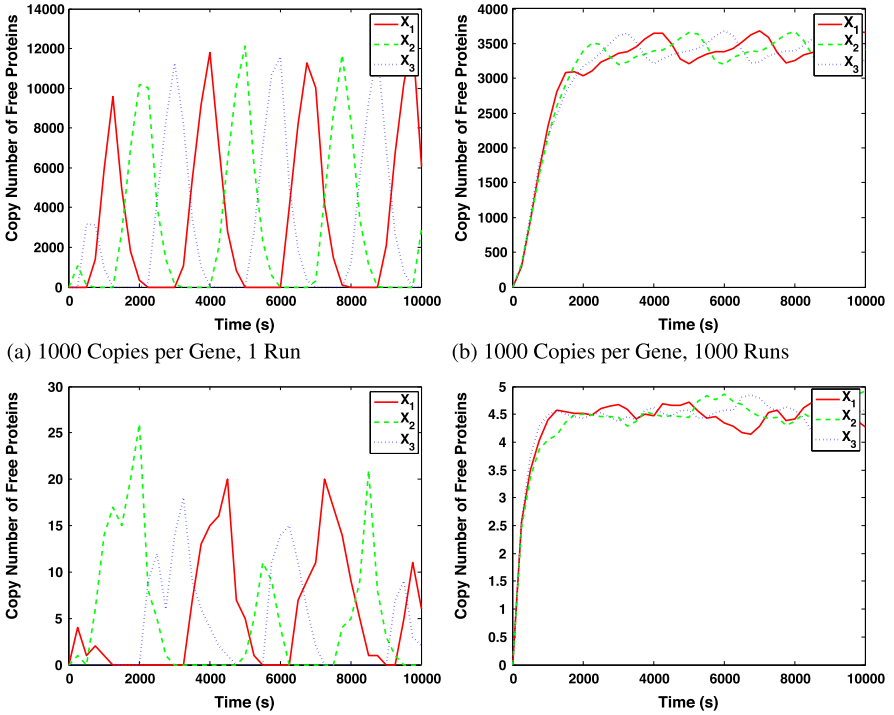
Since (bio-)chemical reactions are inherently stochastic at the molecular level, the application of the stochastic paradigm is most natural. The network structure and the discrete marking of $\mathcal{QPN}$ and thus the discrete state space are maintained in a quantitative time-depended stochastic Petri net ($\mathcal{SPN}$); see Fig. 4 and [41] for a formal definition. In $\mathcal{SPN}$, transitions become enabled if their pre-places are sufficiently marked. The time dependency is added by assigning exponentially distributed firing rates (resulting in waiting times) to the transitions. An enabled transition will only fire if its current specific waiting time has elapsed. The firing event as such does not consume any time. Thus, all reactions defined in the network structure of an $\mathcal{SPN}$ occur with a likelihood, depending on the probability distribution for each given transition. Continuous-time Markov chains (CTMCs) describe the semantics of an $\mathcal{SPN}$. Each simulation run yields one out of many possible traces through the CTMC. The stochastic simulation of the token flow can be computed by, for example, Gillespie's direct method [33].

Simplified Repressilator In Fig. 8, we show the results of the stochastic simulation of the simplified repressilator given in Fig. 6 for different initial settings concerning the simulation runs and number of copies per gene. The continuous simulation in Fig. 7 can be approximated by using a high number of copies per gene in the $\mathcal{SPN}$. (Many copies of a gene within a bacterial cell can be obtained by transforming the cell with a multi-copy plasmid [35].) Performing the simulation with only a single copy of each gene still results in an oscillation superimposed by random fluctuations. Averaging the results over several simulation runs reduces the amplitude of the oscillation as random fluctuations superimpose.

Further Reading The modeling of (bio-)chemical networks by $\mathcal{SPN}$ was first proposed in [34], where the authors applied $\mathcal{SPN}$ to a gene regulatory network. In the following years, $\mathcal{SPN}$ have been applied to several biological case studies; see, for example, [18, 61, 74, 75, 80].

6.2.4 Hybrid Paradigm

In (bio-)chemical reaction networks, especially in signaling or genetic networks, reacting molecules may be of highly different copy numbers and react on highly



(a) 1000 Copies per Gene, 1 Run

(b) 1000 Copies per Gene, 1000 Runs

(c) 1 Copy per Gene, 1 Run

(d) 1 Copy per Gene, 1000 Runs

Fig. 8 Stochastic simulation of the simplified repressilator. The diagrams illustrate the results of stochastic simulations and show copy numbers of the free repressor proteins X_i over time. The number of gene copies and the number of simulation runs were varied. In (**a**) and (**b**), we used 1000 copies of each gene to approximate the continuous behavior. The stochastic simulation in (**c**) and (**d**) is performed with only one token according to situations where a single cell would carry only one copy of the gene. In (**b**) and (**d**), we averaged the stochastic simulation results over 1000 runs

different time scales, which ultimately results in a stiff system [50]. Simulating these networks stochastically would provide exact results, but the high copy number of components makes the simulation computationally expensive.

$\mathcal{SPN}$ are well suited to capture the naturally occurring fluctuations and the discreteness of molecular event, when only a few number of molecules are turned over per time interval. $\mathcal{CPN}$ are poor in modeling fluctuations and discreteness, but deterministic ODE solvers are computationally efficient in simulating reactions that involve a high number of molecules with molecule numbers encoded in the form of continuous concentration values. Whereas stochastic simulation is more accurate, continuous simulation is much faster. Certainly, both modeling paradigms complement each other.

Generalized hybrid Petri nets ($\mathcal{GHPN}$) integrate the formalism and semantics of $\mathcal{SPN}$ and $\mathcal{CPN}$. Thus, $\mathcal{GHPN}$ are tailored to model and simulate systems, where species of highly different copy numbers react with each other. A $\mathcal{GHPN}$

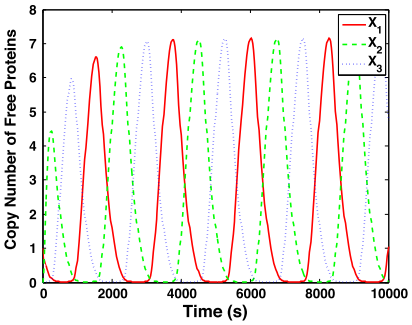


Fig. 9 Hybrid simulation of the simplified repressilator. The diagram illustrates the results of the hybrid simulation with dynamic partitioning and shows the copy numbers of the free repressor protein X_i versus time. The hybrid simulation was performed with one copy of each gene

may contain stochastic and continuous places, as well as stochastic and continuous transitions. Stochastic places contain a discrete number of tokens, whereas the marking of continuous places is given by a real number. Arcs indicating mass flow link stochastic transitions to stochastic or continuous places. However, continuous transitions are exclusively linked to continuous places by standard arcs. Continuous transitions can also depend on discrete places by special arcs, which however are introduced later in this chapter. The state space of a $\mathcal{GHPN}$ is the combination of both the discrete and continuous state spaces; see Fig. 4. A formal definition of $\mathcal{GHPN}$ and their semantics can be found in references [46, 47].

In $\mathcal{GHPN}$, the so-called partitioning of the net in stochastic and continuous parts may be static as set by user. Since the numerical values of the marking of places may drastically change during the simulation, static partitioning may not be always appropriate and efficient. Dynamic partitioning accounts for the drastic variation in marking and firing rates during a $\mathcal{GHPN}$ simulation. Here, an algorithm determines after certain time periods if a transition has to be considered as continuous or stochastic depending on a lower and upper threshold for the firing rate. If one transition violates the partitioning criteria, repartitioning of the net takes place. With the help of dynamic partitioning, it is possible to increase the accuracy and speed of a hybrid simulation [46].

Simplified Repressilator For completeness, we show in Fig. 9 the hybrid simulation of the simplified repressilator given in Fig. 6 with dynamic partitioning. The oscillation can still be obtained with dynamic partitioning.

Further Reading Case studies exemplifying the application of $\mathcal{GHPN}$ to biological system, for example, T7-phage, eukaryotic cell cycle, and circadian clock, as well as further references on hybrid modeling can be found in [46].

6.2.5 Extensions and Useful Modeling Features

For the clear graphical structuring and neat arrangement of a Petri net, logical nodes and coarse nodes are especially useful for the modeling of larger networks. Both

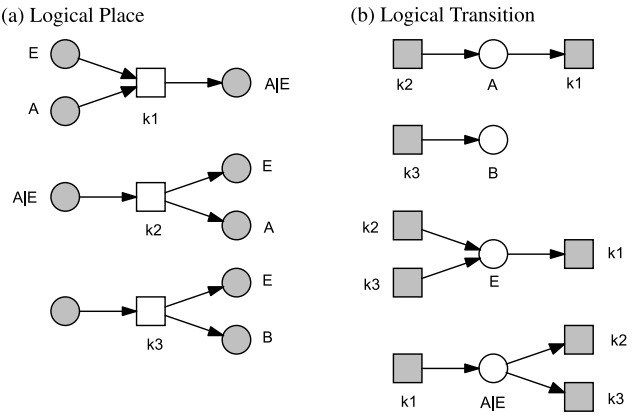


Fig. 10 Logical nodes. The figure shows the enzymatic reaction $A + E \leftrightarrow AE \rightarrow E + B$, where E is the enzyme, and A and B are substrate and product, respectively. With the help of logical nodes, the coherent Petri net model of this enzymatic reaction is displayed to show the individual reactions that link the components (**a**) or the individual components that link the reactions (**b**). Due to the declarations of the nodes shaded in grey as logical nodes, (**a**) and (**b**) show the same coherent Petri net model of the reaction sequence $A + E \leftrightarrow AE \rightarrow E + B$. Execution in Snoopy gives the same results for (**a**) and (**b**). The figure was redrawn from [63]

types of nodes do not change the expressiveness of a Petri net. Although the graphical appearance of a model will be different when logical or coarse nodes are used, the network topology in the different representations is the same.

- *Logical nodes* (= grey-shaded nodes) can be used to replace a single node with a large number of connections to other nodes by multiple graphical copies. Logical nodes are useful when the model structure due to a high number of crossing arcs becomes confusing. This can occur, when a component, for example, ATP, is involved in many different reactions. The ordinary process centered view of a Petri net graph can be changed to a reaction centered view using logical places or a component centered view using logical transitions; see Fig. 10.
- *Coarse nodes* (= boxed nodes) allow one to hierarchically structure a network. Each coarse node in a network induces a new panel containing a subnet. Coarse nodes can be arbitrarily nested. Composing a Petri net by using coarse places and coarse transitions helps to structure the network into subnets according to its functional subsystems or to represent natural hierarchical organization of a biological system. Coarse places are bordered by places and coarse transitions are bordered by transitions, see Fig. 11. Coarse nodes may also exist in isolation, but two coarse nodes cannot be directly linked by arcs.

Furthermore, advanced arc types have been introduced. Read arcs and inhibitory arcs, for example, can be used to connect places with transitions, but not vice versa.

- *Read arc* (= edge with filled dot). If a place p is connected with a transition t via a read arc, the transition t is enabled if place p and all other pre-places of transition t are sufficiently marked. By firing transition t, the amount of tokens

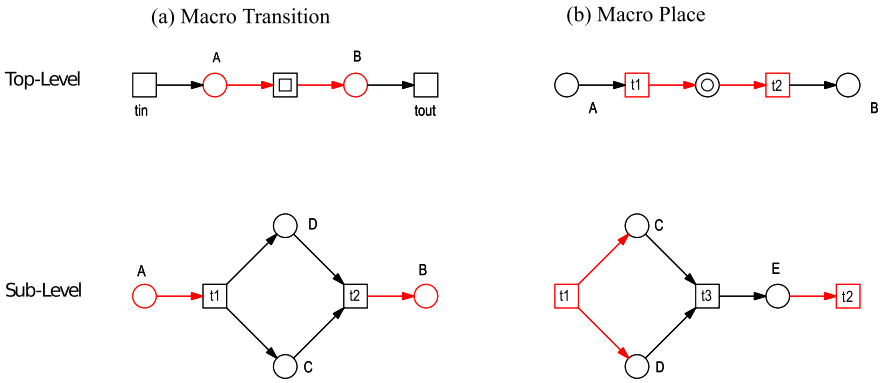


Fig. 11 Coarse nodes. Coarse nodes allow the refinement of (**a**) transitions or (**b**) places by a detailed subnets on a deeper hierarchical level. The introduced subnets may be of arbitrary complexity

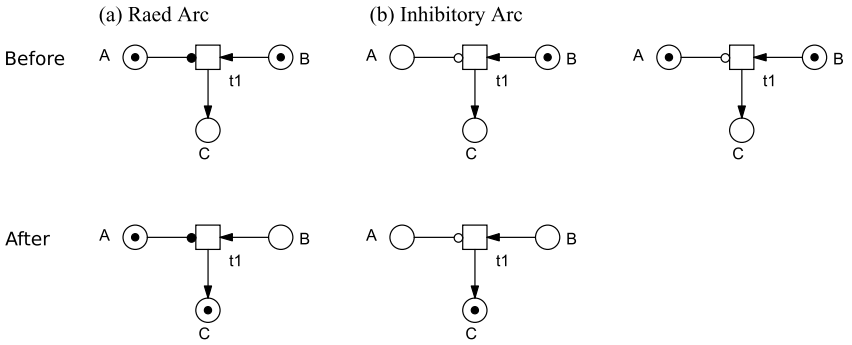


Fig. 12 Read arc and inhibitory arc. (**a**) Read arc: Transition t_1 is enabled if places A and B are sufficiently marked. After firing, tokens are deleted from place B, but not from place A, which is connected with transition t_1 by a read arc. (**b**) Inhibitory arc: Transition t_1 is enabled if place B is sufficiently marked and place A, which is connected with transition t_1 by an inhibitory arc, is not sufficiently marked. After firing tokens are deleted from place B, but not from A

on place p is not changed; see Fig. 12(a). Read arcs are equivalent to two opposed standard arcs.

- *Inhibitory arc* (= edge with empty dot). If a place p is connected with a transition by an inhibitory arc, the transition t is enabled if place p is *not* sufficiently marked, meaning that the amount of tokens must be less than the respective arc weight, and if all other pre-places of transition t are sufficiently marked; see Fig. 12(b). Tokens are not deleted from the place p if the transition t fires. Inhibitory arcs enhance the expressiveness of a Petri net and turn Petri nets into a Turing complete (computationally universal) language.

More extensions of Petri nets can be found in references [6] and [63].

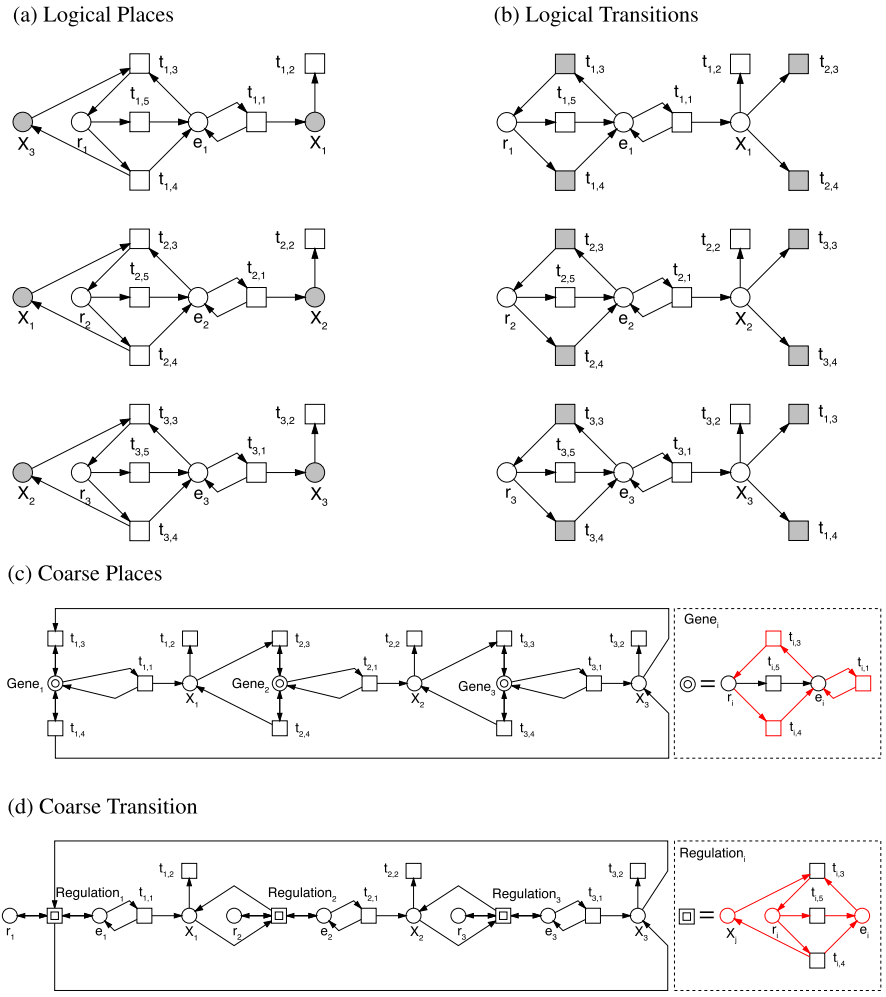


Fig. 13 Alternative representations of the simplified repressilator. In (**a**) logical places and in (**b**) logical transitions are used to split the Petri net model of the simplified repressilator as shown in Fig. 6 into subnets. Each subnet corresponds to one of three TRS (TRS1 to TRS3, see Fig. 2). While the subnets are graphically separated, they are still connected through logical nodes shown in *grey*. (**c**) shows how to encapsulate the two states of each repressor binding site of the genes into a coarse place. In (**d**) all reactions that are responsible for the regulation of each gene are given encapsulated by a coarse transition

Simplified Repressilator By using logical nodes and coarse nodes, the visualization of the simplified repressilator model can be changed without changing the structure of the underlying Petri net; see Fig. 13. The double arcs in Fig. 6 can be replaced by a read arc. Inhibitory arcs are not specifically useful for the simplified repressilator model, without drastically changing its structure.

Table 1 Analysis techniques

Static analysis (no state space construction)	Dynamic analysis (state space construction)
Methods	
• graph theory • linear algebra • linear programming • combinatorics • etc.	• analytical state space generation • simulative state space generation • model checking (temporal logics) • search algorithms • etc.
Properties	
• graph properties • structural features (T-/P-invariants, traps, siphons) • general behavioral properties	• general behavioral properties • user-defined behavioral properties • paths
primary consistency checks	customized in-depth analysis

6.3 Analysis Techniques

The Petri net community offers a rich body of powerful techniques and tools for analysis purposes, which apply standard and well-established mathematical approaches like graph theory, linear algebra, combinatorics, state space construction, model checking based on temporal logic, etc. (see Table 1). Some of those analysis techniques, so-called static analysis techniques consider the qualitative graph structure. Since the structure of a Petri net is maintained in all Petri net classes, the analysis results are valid for $\mathcal{QPN}$, as well as for $\mathcal{SPN}$, $\mathcal{CPN}$, and $\mathcal{HPN}$. The dynamic analysis techniques are based on the discrete state space, which can be constructed analytically. Results of the dynamic analysis are only valid for $\mathcal{QPN}$ and $\mathcal{SPN}$, but not for $\mathcal{CPN}$ and $\mathcal{HPN}$, because $\mathcal{CPN}$ and $\mathcal{HPN}$ do not have a discrete state space; see Fig. 4. Model checking can be applied to all Petri net classes; the temporal logic used for the respective Petri net class depends on the approach used to construct the state space, either analytically or by simulation, and on the chosen modeling paradigm. Please note that the static analysis techniques do only consider standard arcs and read arcs, they are not defined for the use of inhibitory arcs.

The techniques listed in Table 1 can be used for (adapted from [12]):

- **Model analysis** to examine general properties and the behavior of a model.
- **Model verification** to check if a model has been correctly implemented.
- **Model validation** to check if a model exhibits the expected behavior.
- **Model characterization** to assign specified properties to a model, for example, in a database of alternative models.
- **Model comparison** to determine similarities among models.
- **Model modification** to alter the model (kinetic parameters, initial conditions, structure) in order to obtain a desired behavior.

We will now briefly motivate the potential of static and dynamic analysis techniques applied to Petri net models.

6.3.1 Static Analysis

Static analysis techniques pay no attention to the state space and thus neglect any aspects of time. Even if kinetic data are missing, static analysis sheds light on fundamental structural and behavioral properties of a Petri net model. This information can be used for some basic characterization, consistency checks, and to verify the model structure in order to exclude implementation errors. The static analysis allows one to compute (i) graph properties and (ii) structural features of the Petri net model and also to decide on (iii) general behavioral properties.

(i) Graph Properties

Graph properties are elementary properties of the Petri net topology and are thus independent of the marking. Some of those properties are listed below; see reference [41] for formal definitions.

- *Pure*, there exists no pair of nodes connected in both directions.
- *Ordinary*, all arc weights are equal to 1.
- *Homogeneous*, all outgoing arcs of a place have the same arc weight.
- *Connected (Strongly Connected)*, there exists an undirected (directed) path between each pair of nodes.
- *Non-blocking Multiplicities*, the minimal arc weight of all ingoing arcs of a place is not less than the arc weight of its outgoing arcs.
- *Conservative*, each transition adds exactly as many tokens to its post-places as it subtracts from its pre-places.
- *Static Conflict Free*, there exists no pair of transitions sharing the same pre-place.
- *Boundary Nodes*, there exist places (transitions) with either no pre-transitions (pre-places) or no post-transitions (post-places).

Simplified Repressilator The model of the simplified repressilator given in Fig. 6 is strongly connected and has no boundary nodes. Since the arc weights of all arcs are equal to 1, the net is ordinary, homogeneous, and has no blocking multiplicities. The double arc in the synthesis step of repressor proteins X_i by transitions $t_{i,1}$ is in contrast to the pureness of the net. Static conflicts are given by $\{t_{i,1}, t_{i,3}\}$, $\{t_{i,2}, t_{i,3}\}$, and $\{t_{i,4}, t_{i,5}\}$, and the transitions of each set share pre-places. Only transitions $t_{i,5}$ are conservative since all other transitions differ from this rule by adding more tokens to their post-places than subtracting from their pre-places, or vice versa.

(ii) Structural Features

Structural features refer to sets of nodes forming subnets of a Petri net, which have special properties. Those structural features constrain the general behavior of the net. The four most important structural features in the Petri net context are defined as follows:

- A P-*invariant* is a set of places over which the weighted sum of tokens is constant and independent of the firing of any transition in the net; see Fig. 14(a). In the biological context, P-invariants ensure mass conservation and/or describe sets of molecular states that are interconverted. A minimal P-invariant is basically a P-invariant which does not contain another P-invariant.
- A T-*invariant* is a multiset of transitions, which reproduce, by their partially ordered (sequential) firing, a given marking of the induced subnet; see Fig. 14(b). In the biological context, T-invariants correspond to subnets that are capable of reinitialization. Another interpretation leads to the steady-state behavior: the relative transition rates follow the multiplicities prescribed by the transition multiset. A minimal T-invariant is basically a T-invariant that does not contain another T-invariant.
- A *trap* is a set of places inducing a subnet that always contains at least one token as soon as it becomes marked by a token, irrespective of whether or not the subnet is alive; see below and Fig. 14(c). In the biological context, traps are subsystems where at least one component of the subsystem always remains available after being introduced. A minimal trap is a trap that does not contain another trap.
- A *siphon* is a set of places inducing a subnet that may release all of its tokens and then can never be marked again; see Fig. 14(c). In the biological context, places of a siphon may represent finite sources of molecules or energy that become exhausted. A minimal siphon is a siphon that does not contain another siphon.

Formal definitions of those structural features can be found in [41].

In the context of metabolic networks, a P-invariant is also known as a conservation law, and a T-invariant as an elementary mode or stationary flux distribution. All analysis methods that are based on those terms can be adapted to Petri nets as well.

The existence of P-invariants, T-invariants, siphons, and traps in a Petri net decides on four more properties (formal definitions are given in reference [41]):

- *Siphon-trap property*, every siphon (a set of places that cannot switch from unmarked to marked) includes an initially marked trap (a subnet that cannot switch from marked to unmarked). The property can be used to decide about dead state freedom and liveness for specific graph structures of Petri nets [19, 36].
- *Covered with P-invariants*, every place is part of a P-invariant.
- *Covered with T-invariants*, every transition is part of a T-invariant.
- *Strongly covered with T-invariants*, the net is covered with T-invariants, where each T-invariant consists of more than two transitions.

Simplified Repressilator Each TRS of the simplified repressilator consists of one minimal P-invariant:

- $PINV1 = \{e_i, r_i\}$—the repressor binding site of the gene is free or occupied by its repressor protein.

There are three minimal T-invariants

- $TINV1 = \{t_{i,1}, t_{i,2}\}$—synthesis and degradation of the free protein X_i,
- $TINV2 = \{t_{i,3}, t_{i,5}, t_{i-1,1}\}$—synthesis, binding, and degradation of the bound repressor protein X_i,

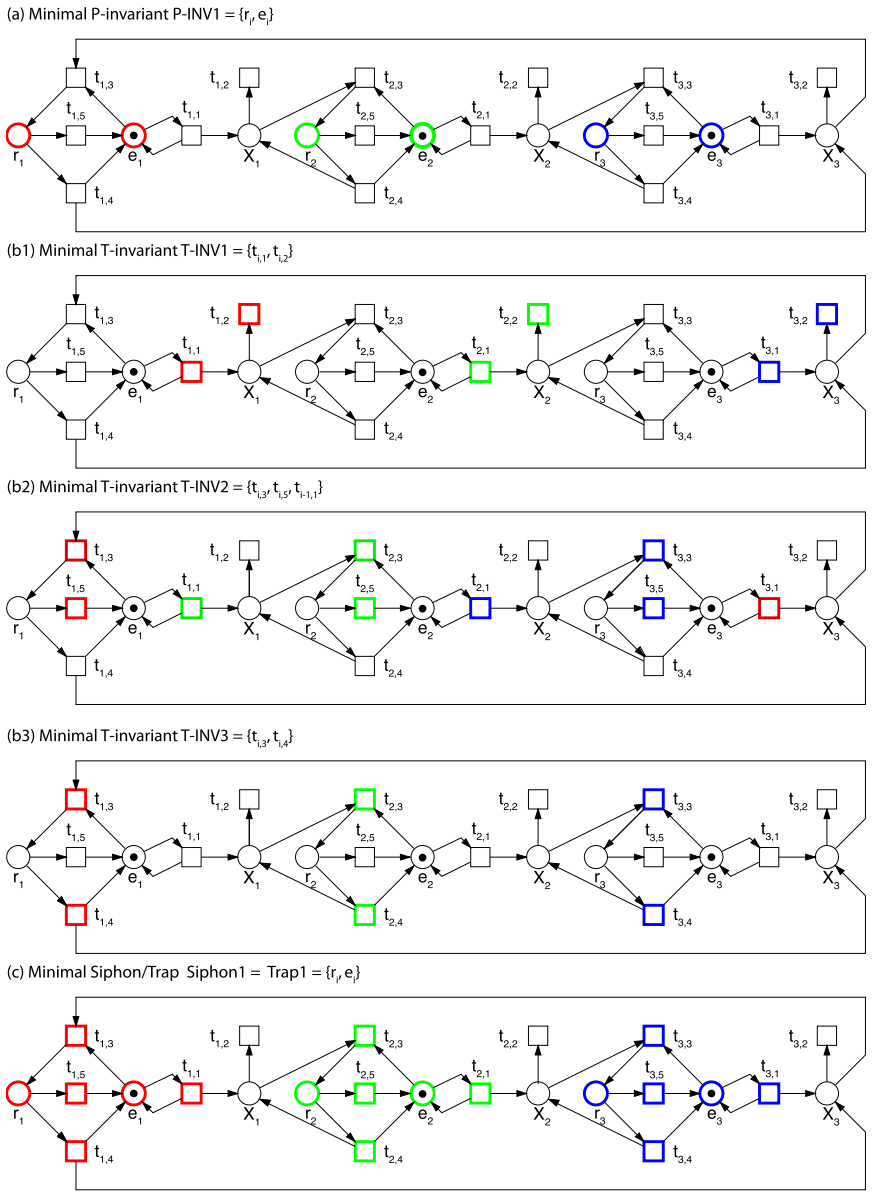


Fig. 14 Structural features of the simplified repressilator. (**a**) The free and the repressed binding site of each gene form a minimal P-invariant $PINV1 = \{e_i, r_i\}$. (**b**) Each component of the simplified repressilator consists of three minimal T-invariants: synthesis and degradation of the free repressor protein X_i, $TINV1 = \{t_{i,1}, t_{i,2}\}$, synthesis, binding and degradation of the bound repressor protein X_i, $TINV2 = \{t_{i,3}, t_{i,5}, t_{i-1,1}\}$, binding and dissociation of the repressor protein X_i from the repressor binding site of the corresponding gene, $TINV3 = \{t_{i,3}, t_{i,4}\}$. (**c**) The P-invariant subnet $PINV1 = \{e_i, r_i\}$ with transitions $\{t_{i,1}, t_{i,3}, t_{i,4}, t_{i,5}\}$ constitutes a minimal siphon and a minimal trap

- $TINV3 = \{t_{i,3}, t_{i,4}\}$—binding and dissociation of the repressor protein X_i from the repressor binding site of the corresponding gene binding site e_j.

Figure 14 illustrates those invariants. The places of the repressor proteins X_i are not part of any P-invariant, whereas all transitions are part of T-invariants. Therefore, the net is not covered with P-invariants, but with T-invariants. Since $TINV1$ and $TINV3$ comprise only two transitions, the net is not strongly covered with T-invariants. The minimal P-invariant $PINV1 = \{e_i, r_i\}$ is a minimal siphon and a minimal trap as well. The token marking the place of the repressor binding site is always contained in the P-invariant $PINV1 = \{e_i, r_i\}$. Thus, the siphon includes an initially marked trap. The production of protein X_i through transition $t_{i,1}$ will always continue. Nevertheless, place X_i can be emptied through $t_{i,2}$.

(iii) General Behavioral Properties

Based on the structure of a Petri net and previously explained properties, it is possible to decide on some so-called general behavioral properties as well: boundedness, liveness, and reversibility. These properties might be independent of the special functionality of the network; see reference [41] for formal definitions:

- *Boundedness*—For every place it holds that: Whatever happens, the maximum number of tokens on this place is bounded by a constant. Overflow by unlimited increase of tokens does not occur.
- *Liveness*—For every transition, it holds that: Whatever happens, it is always possible to reach a state where this transition gets enabled. In a live net all transitions are able to contribute to the net behavior forever. Dead states, that is, states where none of the transitions is enabled do not occur.
- *Reversibility*—For every state, it holds that: Whatever happens, the net is always able to reach this state again. Thus—since this includes the initial state—the net has the capability of self-reinitialization.

Simplified Repressilator Due to the unlimited synthesis of each repressor protein X_i by $t_{i,1}$, which is permitted by the network structure, the number of proteins can infinitely increase, and thus, the model of the simplified repressilator is not bounded. However, the repressor proteins are degraded independently of whether they are bound to the repressor binding site of the gene or free. Furthermore, the repressor binding site of the gene permanently switches between free and occupied rendering the gene active or inactive, respectively. Obviously, there is the chance that each state can be reached again, that is, there is no transition in the model of the simplified repressilator that will become finally inactive. Thus, the net is also alive.

Further Reading Reference [41] gives a more comprehensive overview about analysis techniques of the Petri net theory. Case studies demonstrating the strength of the static analysis techniques can be found in [41] (signaling cascades), [31] (biosensor gene regulation), and [43] (signal transduction network). More specific examples of applications of static analysis techniques and their usefulness are listed in [39].

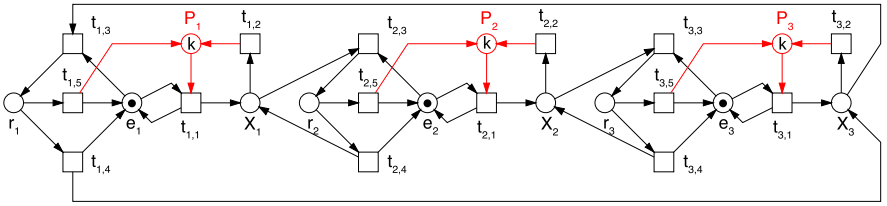


Fig. 15 Bounded Petri net model of the simplified repressilator. To limit the synthesis of the repressor proteins X_i, we introduce a precursor place P_i with the marking k. The constant k determines the upper bound for each repressor protein on place P_i

6.3.2 Dynamic Analysis

As it has been mentioned before, dynamic analysis techniques require the construction of the (partial) state space. The state space can either be constructed analytically (see Sect. 6.2.1) or by simulation (see Sects. 6.2.2–6.2.4).

The analytical exhaustive state space construction is limited to bounded Petri nets and gets computationally expensive with increasing complexity of the model. The state space explosion in complex models occurs for two main reasons: (a) concurrency is resolved by all interleaving sequences, and (b) many tokens contained in a P-invariant can redistribute themselves in multiple ways. When analytical approaches fail, the state space can be approximated by simulation. Simulative state space construction can be applied to either bounded or unbounded nets. But simulative approaches can only be used to partially construct the state space.

6.3.2.1 Behavioral Properties

The general behavioral properties, which sometimes can be determined by static analysis (see Sect. 6.3.1), can also be computed by dynamic analysis. Determining the general behavioral properties by dynamic analysis is only possible if the net is bounded and if the state space can be constructed completely. Constructing the state space by simulation is not sufficient. Based on the complete state space of bounded nets, there are additional behavioral properties that can be checked; see reference [41] for formal definitions:

- *Dynamically Conflict Free*, there exists no state, in which more than one transition is enabled and where firing of one of those transitions creates a new state in which the other transitions are not enabled any more.
- *Dead States*, no transition can fire any more.
- *Dead Transitions*, a transition that is enabled in none of the states that are reachable from the initial marking.

Simplified Repressilator Since the model of the simplified repressilator is not bounded and thus the state space is infinite, we cannot decide on the above mentioned properties. Restricting the number of protein copies for the repressor protein

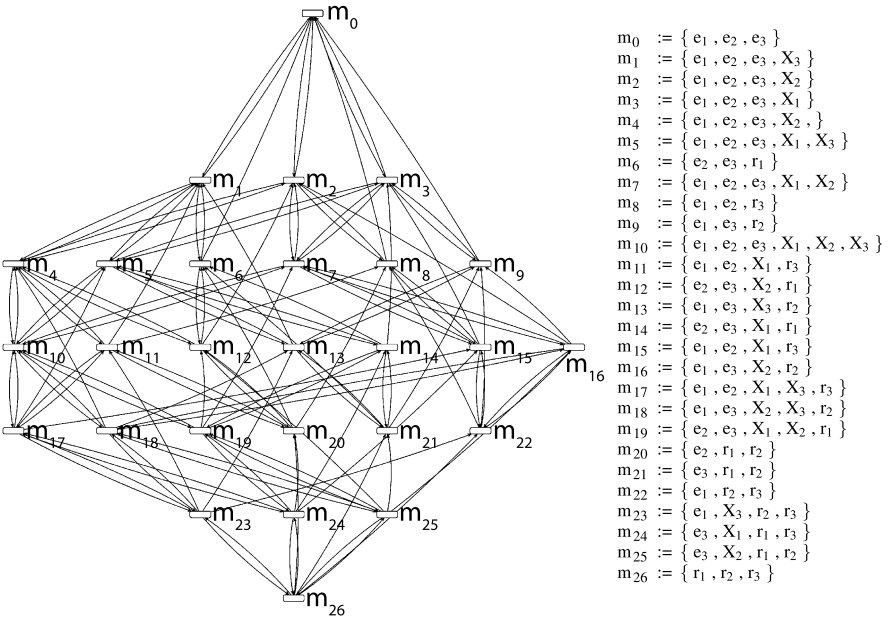


Fig. 16 Reachability graph of the bounded simplified repressilator. The structure of the simplified repressilator allows in principle an infinite increase for each repressor protein resulting into an unbounded net. For simplicity reasons, we convert the model of the simplified repressilator into a bounded model by restricting the number of proteins for each gene to one (Fig. 15). Each node in the reachability graph refers to a specific marking m_i, arcs connecting two nodes represent the firing of a specific transition. The markings are given on the right by the sets of marked places

Table 2 State space of the simplified Repressilator model for different values of k computed with MARCIE [45]

k	States
1	27
10	9261
20	68,921
50	1,030,301
100	8,120,601
1000	8,012,006,001

X_i to k results in a bounded model. A bounded Petri net could be obtained by adding place P_i representing a virtual precursor of the repressor protein X_i. The sum of tokens in P_i and X_i is equal to k (Fig. 15). Now, transition $t_{i,1}$ transforms the precursor of X_i into the actual repressor protein X_i. The degradation of X_i by $t_{i,2}$ and $t_{i,5}$ restores the precursor. The complete state space for $k = 1$ in the form of a reachability graph is given in Fig. 16. The reachability graph has no dead transitions, no dead states, and is free of dynamic conflicts. Table 2 gives the size of the reachability graph for different values of k to illustrate the state space explosion.

Table 3 State space construction and corresponding temporal logics

State space construction	Temporal logic	QPN	SPN	CPN	HPN
Analytical	Computational Tree Logic (CTL)	+	+		
	Linear-time Temporal Logic (LTL)	+	+		
Analytical/ simulative	Continuous Stochastic Logic (CSL)		+		
Simulative	Probabilistic Linear-time Temporal Logic with Constraints (PLTLc)		+		
	Linear-time Temporal Logic with Constraints (LTLc)			+	+

6.3.3 Model Checking

Powerful model checking approaches that are well established in computer science are also useful for systems and synthetic biology applications. In general, model checking is an automatic, model-based approach for the verification of properties defined by the user and revealed by applying the unambiguous expressiveness of temporal logics. In the biological context, model checking can be specifically applied to verify properties in terms of transient behavior, which reflects the intended functionality of the modeled system.

Model checking is possible in all modeling paradigms. Thus, it can be applied to the analytically constructed state space (analytical model checking) and to the state space constructed by simulation (simulative model checking). The type of temporal logic used for each Petri net class depends on the approach used to construct the state space and the modeling paradigm; see Table 3.

The general elements of temporal logics are:

- Atomic propositions:
  Atomic propositions consist of statements describing the current token situation in a given place. Discrete places are read as Boolean variables (integer variables) for 1-bounded (k-bounded or unbounded) Petri nets, and continuous places as (non-negative) real-valued variables. Each atomic proposition $\phi_1, \phi_2, \ldots, \phi_n \in \Phi$ is a temporal logics formula.
- Standard logical operators:
  Atomic propositions can be combined by logical operators to build more complex propositions. $\neg\phi_1$ (negation), $\phi_1 \wedge \phi_2$ (conjunction), $\phi_1 \vee \phi_2$ (disjunction), $\phi_1 \rightarrow \phi_2$ (implication) are temporal logics formulas.
- Temporal operators:

  $\mathrm{X}\,\phi$ (Ne**X**t): The proposition ϕ is valid in the next, directly following state.
  $\mathrm{F}\,\phi$ (**F**inally): The proposition ϕ is eventually valid at some time in the future.
  $\mathrm{G}\,\phi$ (**G**lobally): The proposition ϕ is always globally valid forever.
  $\phi_1\,\mathrm{U}\,\phi_2$ (**U**ntil): The proposition ϕ_1 continually holds until ϕ_2 becomes valid. At this position, ϕ_1 does not have to be valid any more.

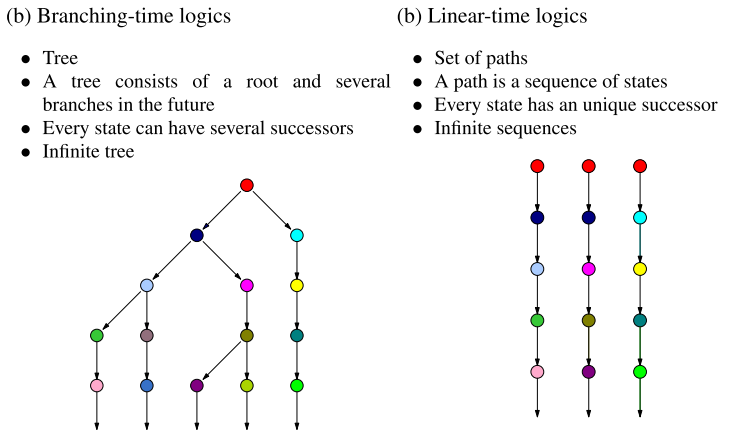


Fig. 17 Linear-time and branching-time logics. Temporal logics are used to specify properties of a model. They can be categorized into linear-time logics (*left*) and branching-time logics (*right*) with distinct properties

Analytical model checking of bounded models, can be performed with either the computational tree logic (CTL) [17], the linear-time temporal logic (LTL) [70], or the continuous stochastic logic (CSL) [1, 2], which is the stochastic counterpart of CTL. Since only $\mathcal{QPN}$ and $\mathcal{SPN}$ allow for the analytical construction of the state space, CTL and LTL can be applied to both net classes, whereas CSL can only be applied to $\mathcal{SPN}$.

Both, CTL and CSL are branching-time logics; see Fig. 17(a). In addition to the standard elements of temporal logics, CTL uses two path quantifiers:

- $\mathrm{E}\,\phi$ (**E**xistence): The proposition ϕ is valid for at least one path.
- $\mathrm{A}\,\phi$ (**A**ll): The proposition ϕ is valid for all computed paths.

The combination of temporal operators and path quantifiers creates eight operators, which can be used to specify temporal properties of a model. Let $\phi_{[1,2]}$ be an arbitrary temporal-logic formula. Then, the following formulas are valid in state m:

- $\mathrm{EX}\,\phi$: if there is a state reachable by one step where ϕ holds.
- $\mathrm{EF}\,\phi$: if there is a path where ϕ holds finally, that is, in some state of this path.
- $\mathrm{EG}\,\phi$: if there is a path where ϕ holds globally, that is, in all states of this path.
- $\mathrm{E}(\phi_1\,\mathrm{U}\,\phi_2)$: if there is a path where ϕ_1 holds until ϕ_2 holds.

The other operators can be obtained by replacing the **E**xistence operator by the **A**ll operator. In this case, the explanations start with "for all paths" instead of "there is a path".

CSL replaces the path quantifiers (**E**, **A**) in CTL by the probability operators $\mathbf{P}_{\triangleright\triangleleft p}$ (transient analysis) and $\mathbf{S}_{\triangleright\triangleleft p}$ (steady-state analysis) whereby $\triangleright\triangleleft p$ specifies the probability of the given formula (the comparison operator $\triangleright\triangleleft$ can be replaced by $<, \leq, =, \neq, >, \geq$). The operator $\mathbf{P}_{=?}$ is used to return the probability (rather than compare probabilities).

As the name suggests, LTL is a linear-time logic; see Fig. 17(b). Linear-time logics do not require path quantifiers, because they operate implicitly over all paths. CTL and LTL are both subsets of CTL* [26], but are not equivalent to each other. The CTL formula $\mathrm{EF}\,\phi$ can not be expressed in LTL, neither can the LTL formula $\mathrm{FG}\,\phi$ be written as CTL.

In addition, LTL with constraints (LTLc) [13] can be applied to the continuous state space; thus, it is used for $\mathcal{CPN}$ and $\mathcal{HPN}$. A probabilistic extension of LTLc is called PLTLc [20]. PLTLc can be used for model checking with $\mathcal{SPN}$.

It adds the probability operators **P** (transient analysis) [20] and **S** (steady-state analysis) [72]. Both operators appear only once in a formula at the top level and may not be nested as in CSL.

Since the paths generated with linear-time logics refer to sequences of states, Linear-time logics might be more convenient for reasoning about time-series behavior in biology [39].

Simplified Repressilator For the analytical model checking, we rely again on the bounded model of the simplified repressilator with a restricted copy number k of each repressor proteins X_i; see Fig. 15. Furthermore, we assume that each TRS consists of only gene, $e_i + r_i = 1$. We applied CTL to formalize some basic properties of the simplified repressilator.

- The repressor binding site of each TRS in the simplified repressilator model is either free e_i or repressed r_i:

$$\mathrm{AG}\big[(e_i = 1 \wedge r_i = 0) \vee (e_i = 0 \wedge r_i = 1)\big]$$

- Each protein X_i is intended to oscillate, that is, it fluctuates around a value $X_i = c$. Furthermore, we have to take some noise n into account because of the stochastic nature of the model. A noise filtered oscillation [3] of protein X_i can be characterized in CTL by the formula

$$\begin{aligned}\mathrm{AG}\big[\big((X_i = c) \rightarrow \mathrm{EF}\big[(X_i > c + n) \vee (X_i < c + n)\big]\big)\\ \wedge \big(\big((X_i > c + n) \vee (X_i < c + n)\big) \rightarrow \mathrm{EF}[X_i = c]\big)\big]\end{aligned}$$

  The domain of c is $(0, k)$, and a typical value for checking the oscillation is $k/2$. The noise n is a fraction of c, so the domain of n is $(0, c)$, for example, $c/10$ or $c/20$. The above CTL formula is read as follows: at any time point in the future, if the number of copies of the repressor protein gets $X_i = c$ $(c \leq k)$, then it has to be possible to reach a state where $X_i < c + n$ or $X_i > c + n$, and vice versa.
- Sequential oscillation [3] of proteins X_1, X_2, and X_3:

$$\begin{aligned}\mathrm{AG}\big[\big(\big((X_1 = c) \wedge (X_2 \neq c) \wedge (X_3 \neq c)\big)\\ \rightarrow \mathrm{EF}\big[\big((X_1 \neq c) \wedge (X_2 = c) \wedge (X_3 \neq c)\big)\big]\big)\\ \wedge \big(\big((X_1 \neq c) \wedge (X_2 = c) \wedge (X_3 \neq c)\big)\end{aligned}$$

$$\rightarrow \mathrm{EF}\big[\big((X_1 \neq c) \wedge (X_2 \neq c) \wedge (X_3 = c)\big)\big]\big)$$
$$\wedge \Big(\big((X_1 \neq c) \wedge (X_2 \neq c) \wedge (X_3 = c)\big)$$
$$\rightarrow \mathrm{EF}\big[\big((X_1 = c) \wedge (X_2 \neq c) \wedge (X_3 \neq c)\big)\big]\big)\big]$$

At any time point in the future, if the number of copies of the repressor protein $X_i = c$ and X_{i+1}, X_{i+2} are unequal to c, it has to be possible to reach a state where $X_{i+1} = c$ and X_i, X_{i+2} are unequal to c.

The given CTL formulas can be translated to CSL, by replacing the path quantifiers with the probability operator **P**, and thus compute how likely the oscillation is. A transformation into LTL is not possible because of the path quantifier **E**. But this formula can be transformed into a PLTLc formula by removing the path quantifier **E** and enclosing the whole formula with the probability operator **P**. Now we can compute how unlikely (or likely) the oscillation is, even for the unbounded simplified repressilator model via simulative model checking.

Using model checking of quantitative models, properties can be expressed by distinct descriptive approaches, with increasing specificity: qualitative, semi-qualitative, semi-quantitative, and quantitative [20].

The basic qualitative formula consists of derivatives of biochemical species concentrations or mass, given by the function $d(\cdot)$. Together with the temporal operators, we can now express the general trend of the behavior. The semi-qualitative extension adds to the qualitative formula the relative concentration by applying functions like, for example, $\max(\cdot)$, $\min(\cdot)$, $\mathrm{average}(\cdot)$ to the formulae. Semi-quantitative approaches consider in addition to semi-qualitative formulas absolute time values by referring to the predefined systems variable time. Moreover, a quantitative description extends the semi-quantitative formula by expressing absolute concentration values as well.

Simplified Repressilator We exemplify the four distinct descriptive approaches mentioned above by applying them to the repressor protein X_i of the simplified repressilator model (formulas adapted from [20]):

- *Qualitative.* The repressor protein X_i raises, then falls:

$$\mathbf{P}_{=?}\big[d(X_i) > 0\ U\ \big(G\big(d(X_i) < 0\big)\big)\big]$$

- *Semi-qualitative.* The repressor protein X_i raises, then falls to less than 50 % of its peak concentration:

$$\mathbf{P}_{=?}\big[d(X_i) > 0\ U\ \big(G\big(d(X_i) < 0\big)$$
$$\wedge F\big(X_i < 0.5 \cdot \max(X_i)\big)\big)\big]$$

- *Semi-quantitative.* The repressor protein X_i raises then falls to less than 50 % of its peak concentration at 5000 s:

$$\mathbf{P}_{=?}\big[d(X_i) > 0\ U\ \big(G\big(d(X_i) < 0\big) \\ \wedge F\big(\text{time} = 5000 \wedge \text{Protein} < 0.5 \cdot \max(X_i)\big)\big)\big]$$

- *Quantitative.* The repressor protein X_i raises then falls to less than 10 *Molecules* at 5000 s:

$$\mathbf{P}_{=?}\big[d(X_i) > 0\ U\ \big(G\big(d(X_i) < 0\big) \\ \wedge F(\text{time} = 5000 \wedge X_i < 10)\big)\big]$$

Compare properties with Fig. 8(c).

Further Reading We recommend reference [6] for a general gentle introduction into model checking and the different temporal logics. In [41–43], model checking has been applied in several advanced case studies for all three modeling paradigms. In [57], another repressilator version serves as a running case study demonstrating various analysis techniques and, among them, model checking in the different paradigms.

6.4 Multiscale Modeling with Colored Petri Nets

Computational modeling of multicellular systems at different levels of molecular and cellular organization requires powerful computational multiscale modeling frameworks. In general, biological systems consist of similar components and structures, which are hierarchically organized into subsystems. Modeling of such subsystems introduces various challenges [40]:

- *Repetition of components*; multiple components with the same definition, for example, cells of the same type.
- *Variation of components*; multiple components with defined variability in their definition, for example, wild-type cells versus mutated cells.
- *Organization of components*; one-, two-, or three-dimensional organization of components of a specific shape, for example, organization of cells of a certain shape in a tissue.
- *Hierarchical organization of components*; components containing sub-components, for example, cells consisting of defined compartments.
- *Pattern formation by components*; (self-)organization of components within appropriate one-, two-, or three-dimensional structures in time and space, for example, chemotaxis involved in developmental phenomena.
- *Irregular/semi irregular organization of components*; deviating organization or interrupted patterns of components, for example, mutated epidermal cells.

- *Communication between components*; defined exchange of information between components restricted by their spatial relation and position in a spatial network, for example, signal transduction between neurons.
- *Mobility/Motility of components*; active or passive transport of components within a spatial network, for example, motile cells in a tissue or transport of molecules via microtubules.
- *Replication of components*; formation of new components in a system, for example, cell division.
- *Deletion of components*; removing components form a system, for example, cell death.
- *Differentiation of components*; components gaining (or losing) functionality, for example, stem cells differentiate into immune cells.
- *Dynamic grid size*; variable dimension and composition of components/systems, for example, grid changing in size and/or structure (required to remove and insert items).

In multiscale modeling of biological systems, components can be either molecules, organelles, cells, tissues, organs, organisms, populations, or eco-systems.

Multiscale systems can certainly be modeled using the standard approaches, but the models become unhandy and impractical with increasing complexity. Reflecting on the structure and organization of complex components in a conceptual way is difficult, if not impossible, with the standard approaches, but it might be necessary to understand a system based on the interaction of its components.

6.4.1 Colored Petri Nets

Colored Petri nets turn low-level Petri nets (which we considered so far, see Sect. 6.2) into a high-level modeling framework, see also Fig. 4. Each modeling paradigm in low-level Petri nets (qualitative, continuous, stochastic, hybrid) has its colored counterpart. In colored Petri nets, the formalism and semantics of low-level Petri nets are combined with the capability and flexibility of a programming language to express various data types and operations. With the defined data types, groups of similar subnets can be implemented as one subnet and distinguished by the color of the tokens that move through the net. Colored Petri nets can be constructed from low-level Petri nets for a given partitioning of places and transitions. Vice versa, colored Petri nets can be unfolded to low-level Petri nets. Thus, colored Petri nets provide a parameterized and compact representation of complex low-level Petri nets while sustaining the analysis capabilities of low-level Petri nets (Sect. 6.3). A formal definition of colored Petri nets can be found in [54].

A convenient way to construct a colored Petri net is to first start with the low-level representation of a single subnet; see Fig. 18; for application examples, see below. The next step is to define a suitable color set by setting its data type, for example, integer, boolean, enumeration, string, etc., and its values (colors). The number of values in a color set may be defined by suitable constants. Subsequently, the defined

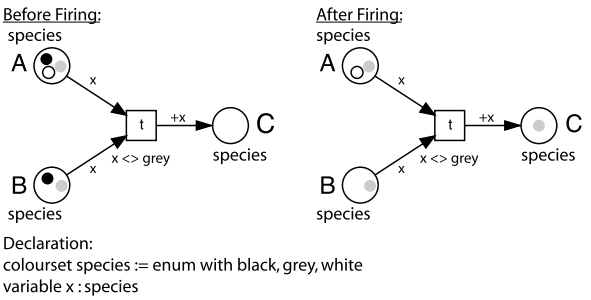


Fig. 18 Colored Petri net example. The color set *species* of the places A, B, and C is of the type *enumerate* (*enum*) with the colors *black*, *grey*, *white*. Thus, the places can carry *black*, *grey*, or *white* tokens. Transition t can only fire if there are tokens of the same color at place A and B, except for grey according to the guard $x <> grey$ of transition t. The variable x of the arc expressions must be bound to either *black* or *white*. Since a *white* token is missing at place B, transition t is only enabled and can fire if x is bound to *black*. After firing of transition t, the black tokens are deleted at place A and B as usual, and a new token of the successor color is produced at place C, which is defined by the arc expression $(+x)$. Here, a *grey* token is produced at place C

color set is assigned to the places of the subnet. Each color set needs at least one variable. The variable is used in the arc expressions to carry the token of a specific color to the transition, or vice versa.

Boolean expressions can be used along with places, transitions, and arcs to express the variability between subnets or their interactions. Using boolean expressions to define the marking of places allows one to set how many tokens of a color are initially available, for example, resources of a component. Arc expressions might use boolean expressions to define which tokens of a color of a color set can move via an arc. Boolean expressions can also be used to distinguish varying firing rates for different colors of a transition, for example, a reaction might be slower or faster depending on the component. In addition, it is also possible to set guards for a transition with the help of boolean expressions to define constraints on the token colors that eventually can enable the respective transition.

Not all places have to be of the same color set and places with different color sets can interact via common transitions. An example are subnets of a Petri net that represent components of the system of different copy number, for example, three cells of type A communicating with five cells of type B within the tissue of an organism. Even more, color sets can be combined in a compound color set via their union or product. By combing color sets one-, two-, and three-dimensional grids can be easily implemented to consider spatial aspects, for example, spatial organization of molecules, specific shapes of cells, pattern formation, mobility/motility, etc. [40]. The hierarchical design of color sets can reflect the inherent hierarchy in a system and thus allows the abstraction over network motifs and the hierarchical representation of locality.

The flexibility of compactly representing a Petri net in the form of a colored Petri net allows one to arbitrarily scale a model by creating multiple copies of its

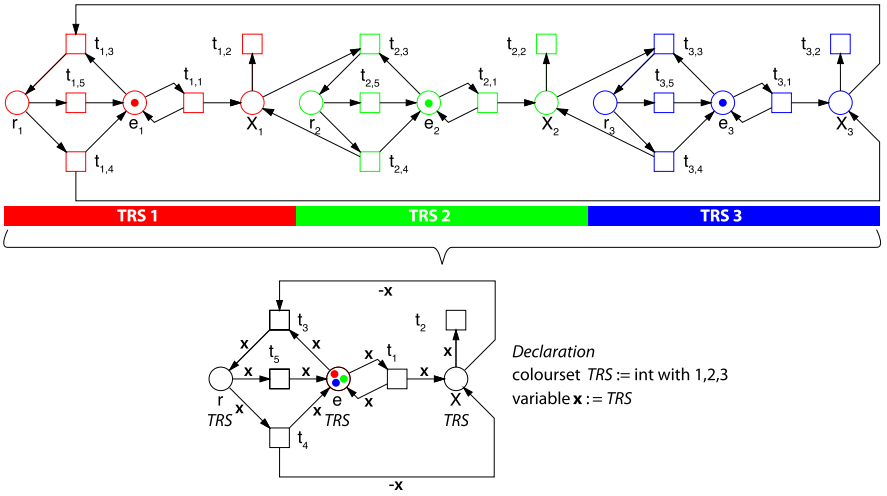


Fig. 19 Colored version of the simplified repressilator. It is obvious from the graph structure that the model of the simplified repressilator consists of three similar subnets marked by *red*, *green*, and *blue outlines*. These subnets can be folded into a single one using color. Therefore, we take the structure of one TRS and define a color set *component* of the type *integer* with the colors $1, 2, 3$ (equivalent to *red*, *green*, *blue*). Variable x is used for the color set *TRS*. The color set *component* is assigned to the places X, e, r. The arc expression $-x$ denotes always the (modulo) predecessor of the current color bound to x

subsystems. Colored Petri nets preserve the advantages of low-level Petri nets and thus enjoy the rich choice of analysis approaches.

Simplified Repressilator Since the model of the simplified repressilator consists of three similar subnets, it is an ideal example for folding a low-level Petri net into a colored Petri net; see Fig. 19. In the colored version, only the structure of one TRS is needed, whereas the number of TRS is defined through the color set. Therefore, we use the color set *TRS* of type integer with colors $1, 2, 3$. The variable x is of type *TRS*, and $-x$ refers to the (modulo) predecessor in *TRS*. It is easy to increase the number of TRS in the colored model of the simplified repressilator model by changing the number of colors in the color set *TRS*, for example, to 20; see Fig. 20.

A complex biological phenomenon that could also be implemented as part of the simplified repressilator model is protein biosynthesis through explicitly considering transcription and translation. In bacterial cells, the two processes are coupled in the sense that the translation of nascent transcripts starts before transcription of the gene is finished; see Fig. 21. The polymerase slides along the DNA and multiple ribosomes are engaged with each nascent mRNA molecule, forming a polysome (polyribosomes). Both processes, transcription and translation, are one-dimensional, directed walks. Before the polymerase can slide along the DNA, it has to bind to the promotor region of the gene to initialize transcription. Translation starts when the

(a) Simplified Repressilator with 20 TRS (b) Stochastic Simulation

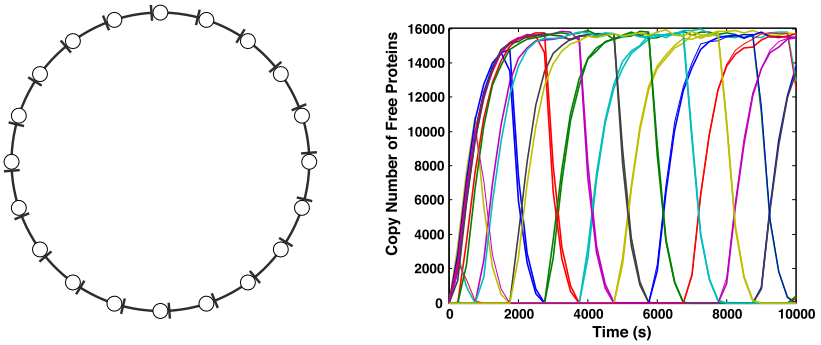


Fig. 20 Stochastic simulation of the simplified multi-gene-repressilator. (**a**) We implement 20 TRS (see Fig. 2) arranged in a negative-feedback loop of the colored simplified repressilator model shown in Fig. 19 by accordingly increasing the number of colors in a color set *TRS* to 20. (**b**) For stochastic simulation, we used 1000 copies of each gene and performed one simulation run. The diagram shows the copy numbers of free repressor proteins versus time

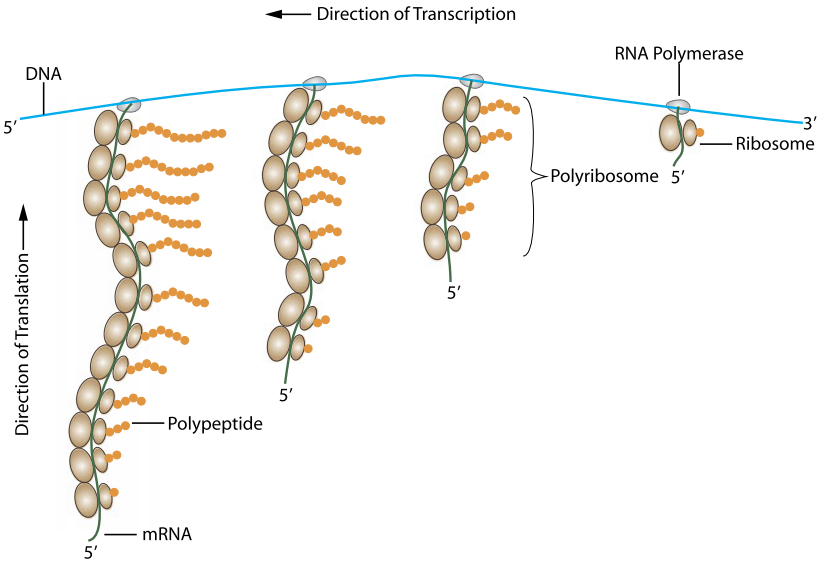
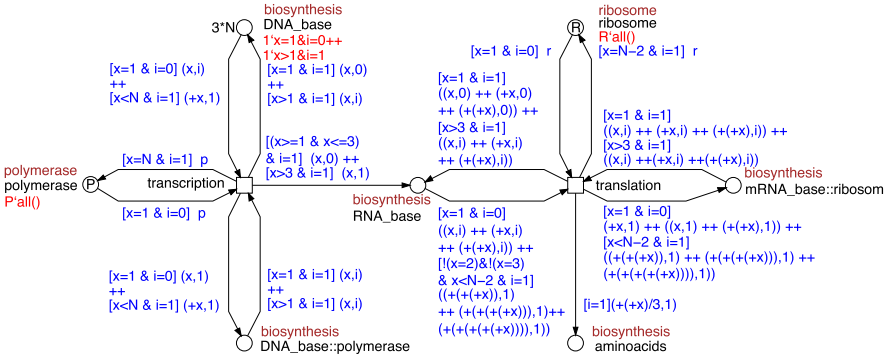


Fig. 21 Simultaneous transcription of a gene and translation of the nascent mRNAs in a bacterial cell. The cartoon was redrawn from [65]

assembled ribosome has reached the start codon of the mRNA. Colored Petri nets can easily express the processes of transcription and translation as polymerization reactions; see Fig. 22. The polymerization and depolymerization of cytoskeletal proteins could be modeled in a similar way.



- simple colorsets:
  - *sequence*: *int* with $\{1-3*N\}$
  - *init*: *int* with $\{0,1\}$
  - *polymerase*: *int* with $\{1\}$
  - *ribosome*: *int* with $\{1\}$
- compound colorsets:
  - *biosynthesis*: *product* with $\{sequence, init\}$
- variables:
  - x: *sequence*
  - i: *init*
  - p: *polymerase*
  - r: *ribosome*
- constants:
  - N: *int* (number of triple nucleotide codons)
  - P: *int* (number of polymerase molecules)
  - R: *int* (number of ribosomes)

Fig. 22 Colored Petri net model of simultaneous transcription and translation. The compound color set *biosynthesis* is the product of two simple color sets *sequence* and *init*, which are both of type *int*. The number of entities in the color set *sequence* is defined by the sequence length N of the polypeptide chain $\{1,\ldots,3\cdot N\}$ with the variable x. The color set *init* has only two values $\{0,1\}$ and is used for the variable i. If $i=0$, then initialization is needed to start transcription by the polymerase at the first DNA base of the start codon or to start translation by the ribosome at the first three RNA bases (corresponding to the start codon of the coding sequence). There are two other color sets used here, *polymerase* and *ribosome*, both are again of type *int* and have only one color. The variables used are p for *polymerase* and r for *ribosome*. The color set *polymerase* is assigned to the respective place *polymerase* and *ribosome* to the place *ribosome*. All other places use the compound color set *biosynthesis*. To start the transcription, the polymerase needs to bind to the first base (the initialization step), which is notated by $(x=1,\ i=0)$. The polymerase then can move to the next DNA base $(+x,\ i=1)$ while transcribing the first one and so on. Moving to the next base means to increment the color value by $+x$. The process ends when the last base is reached $(x=3\cdot N,\ i=1)$. As soon as the first base is no longer occupied by the recent polymerase, a new one can bind. Once the first three mRNA bases (start codon) have been produced, a ribosome can bind to the nascent mRNA molecule and translate the mRNA into a polypeptide while transcription is still proceeding. The process of translation is represented by a similar model of polymerization as transcription; the only difference is that three sequential mRNA bases yield one amino acid of the polypeptide chain. Thus, the color is now incremented by 3, which makes the arc expressions more complex. If the start codon is free, then a new ribosome can bind to the mRNA. After the translation is finished, the ribosome is available for the next round. Please note that this example just illustrates how colored Petri nets can be used to implement highly complex processes such as coupled polymerization reactions in the form of a very simple Petri net. To understand the meaning of the blue arc expressions, one needs to be familiar with the standard formalism of colored Petri nets as it is used in Snoopy [54]

In our model, transcription starts when the DNA-polymerase binds to the first nucleotide of the coding sequence of the DNA strand and proceeds until the third base of the stop codon has been incorporated. The next DNA-polymerase molecule can start transcription as soon as the preceding polymerase has released the first base of the start codon (note that this is a simplification as the initiation and the termination of transcription are complex processes). While a polymerase molecule synthesizes an mRNA molecule, it slides along the coding sequence of the gene. The progressing polymerization of the mRNA molecule is modeled by delivering tokens of incremental color into the mRNA place. If, for example, the first 25 bases of an mRNA molecule have been synthesized, the mRNA place contains 25 tokens of sequential colors. The total number of colors in the color set represents the number of bases in the coding sequence. Once the mRNA is synthesized, the mRNA place contains a token of each color of the color set. Multiple mRNA molecules give multiple tokens of the same color. The same principle is used for polymerization of the proteins, only that three tokens of successive colors are consumed from and restored to the mRNA place for each incorporated amino acid. For further details, see Fig. 22. Note that the amino acid sequence of a synthesized protein could be easily encoded by tokens by creating a two-dimensional color set ($P \times I$), where the color set P defines the position of the amino acid with respect to the N-terminus of the protein, and colorset I encodes the chemical identity of the incorporated amino acid in terms of an ordinal number.

Further Readings A more comprehensive review on biomodel engineering for multiscale modeling in systems biology is given in [40]. In [32], it is shown how spatial attributes of dynamic systems can be encoded by the use of colored Petri nets. Some examples of case studies demonstrating the power of colored Petri nets for multiscale modeling are: (1) phase variation in bacterial colony growth [30], (2) planar cell polarity in Drosophila wing [29], (3) membrane systems [55], and (4) coupled calcium channels [56].

6.5 Composing Models from Molecule-Centered Modules

In biomodel engineering, molecular networks of biological processes are most frequently designed as monolithic models in the form of ODEs. Since the amount of data produced by technically advanced high-throughput (*omics*) approaches is increasing, the integration of those data into coherent models is a considerable challenge. In this context, we propose an approach, which is successfully used in engineering, namely the modular construction of a system. In its general form, modules—as we use them—are molecule-centered Petri nets with a standardized interface [5, 7–9]. The advantage of producing one module for each type of molecule is that one can arbitrarily compose these Petri nets into complex models without rebuilding the models from scratch. Specifically, recombining modules allows one to easily, quickly, and safely generate different versions of a model. This may include

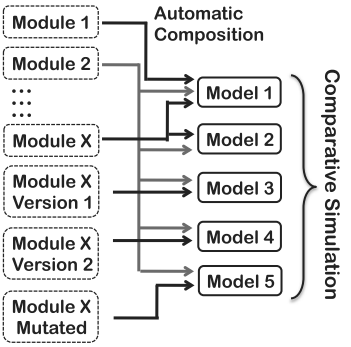


Fig. 23 Alternative models. Modules can be reused and recombined in various combinations. The obtained models can be used to test for the effect of alternative or modified reaction mechanisms

the exchange of different versions of a module within a model for comparative simulation; see Fig. 23. The management and composition of modules are supported by the BioModelKit database (see below). The database helps to:

- maintain and update modules easily,
- compose models arbitrarily from modules to generate alternative models,
- handle arbitrary levels of abstraction, and
- integrate top-down and bottom-up models.

Using molecule-centered modules provides a variety of options for the advanced engineering of biomodels with the help of appropriate algorithms. Algorithms for modification of modules and the composition of models from modules also in combination with the database allows one to [8, 10]:

- modify, mutate, or redesign modules and thus models,
- automatically compose large-scale models to simulate *omics* data sets, and
- reverse engineer models from *omics* data sets.

Proteins as compared to nucleic acids (RNAs, DNAs) display a high variety in their (bio-)chemical and kinetic reaction mechanisms. Although the reaction mechanisms for members of a given class of proteins (e.g. heterotrimeric G-proteins) are similar and will show up in the modules representing these proteins, the modules for each individual protein have to be designed at the very end by hand according to the specific knowledge that is available for this protein. In contrast, biosynthesis and degradation processes, which may be very similar for nucleic acids or proteins from the kinetic point of view, can be simply modeled by cloning appropriate modules in the form of module prototypes. For this reason, it is advisable to implement special module types for proteins, mRNAs, and genes and for the (controlled) degradation of proteins [8]; see Table 4. For maximal flexibility in the context of reverse engineering approaches, causal interaction modules and allelic influence modules were introduced. Causal interaction modules are used to represent cellular processes of ill-defined or unclear molecular mechanisms. Allelic influence modules account for differences in the network behavior, which is due to the effect of gene mutations (Table 4) [8]. These two module types are obviously not molecule-centered.

Table 4 Module types

Molecular interaction		Causal dependency
Protein module	Protein degradation module	Causal interaction modules
• binding and unbinding reactions • formation and cleavage of covalent bonds • conformational changes	• inactivation and degradation	• causal influence on molecular and cellular processes
Gene module	RNA module	Allelic influence modules
• transcriptional activity • binding and unbinding reactions • covalent modification	• transcription • processing (alternative splicing) • binding and unbinding reactions • translation • degradation	• allelic influence of genes on molecular and cellular processes

The Biomodelkit database (BMKdb, www.biomodelkit.org) is a tool with public access to organize modules. The modules are organized in BMKdb in such a way that each node (transitions, places) and their directed connections (arcs) are stored, as well as their appearance in the respective modules. This allows one to tag modules, in particular, each node or arc with specific metadata, for example, general documentation, functional descriptions, literature references, or suitable identifiers of other molecular databases. The metadata can be used to formulate queries in order to find modules of interest and connections between modules. In addition, BMKdb supports the module versioning. Thus, related modules, for example, modules with different resolution in mechanistic details or with reaction mechanisms according to competing hypotheses on molecular mechanism, can be stored and organized in BMKdb. Furthermore, purposefully designed features facilitate the automatic composition of models from an ad hoc chosen set of modules and the algorithmic generation of biological relevant mutations of those modules [10].

We successfully demonstrated the applicability of our approach using two case studies (1) JAK/STAT signaling [9] and (2) pain signaling [5], which both involve complex networks with massive crosstalk. The JAK/STAT pathway is one of the major signaling pathways in multicellular organisms controlling cell development, growth, and homeostasis by regulating the gene expression. The modular network of the JAK/STAT pathway in IL-6 signaling comprises seven protein modules (IL6, IL6-R, gp130, JAK1, STAT3, SOCS3, and SHP2). Overall, the model consists of 92 places, 102 transitions spread over 58 panels with a nesting depth of 4. The nociceptive network in pain signaling consists of several crucial signaling pathways, which are hitherto not completely revealed and understood. The latest version of the nociceptive network consists of 38 modules; among them, there are several membrane receptors, kinases, phosphatases, and ion-channels. So far, the model is made up of

713 places and 775 transitions spread over 325 panels, again with a nesting depth of 4.

In [10], we formalize our modular modeling framework for biomodel engineering and explain in detail the principles of constructing a module and how the composition of modules is performed. Composing Petri nets from modules can be easily and quickly done and is safe in obtaining the correct structure. In the case that kinetic parameters for the interaction of molecules represented by the modules have been estimated, they automatically apply to the composed model as well. Afterwards, the dynamic behavior of the composed model has to be checked for consistency. In addition, we explain the algorithmic structural modification of modules supported by BMKdb in order to generate *in silico* biological meaningful mutations. We suggested three algorithms to systematically (1) knockout genes by deleting modules, (2) mutate structural protein units by altering the module structure, or (3) affecting nodes that are specifically tagged according to their (bio-)chemically defined function.

With all these possibilities of biomodel engineering at hand, it seems straightforward to devise bioinformatic pipelines for the generation of models optimized to obey a pre-defined behavior.

Modules of the types described above can be combined with a completely different type of module that represents space in general and compartments in particular. Combination of such space modules with models composed of molecule-oriented modules allows one to model the positioning of molecular species and their diffusion or movement through space. This is important when compartmentalization of biomolecules is of functional relevance (e.g. the translocation of a transcription factor into the nucleus, which induces the transcription of a target gene). Spatial organization of molecules or even cells is also highly relevant in many developmental processes ranging from embryonic development to the generation of functional structures in populations of entire organisms. For details, see reference [9].

Simplified Repressilator Each TRS of the simplified repressilator can be decomposed into a set of modules (Fig. 24): (1) protein modules describing the binding/unbinding process of the repressor proteins to the respective genes, (2) gene modules switching on and off the genes by binding the respective repressor protein, (3) mRNA modules illustrating the biosynthesis of the repressor proteins, and (4) the protein degradation modules. Indeed, in this trivial case, protein modules and gene modules are identical since both model the same interaction. The modules when composed as shown Fig. 24 give a functional model of the simplified repressilator, which is directly executable in Snoopy, meaning that the token flow can either be animated or simulated in all Petri net classes. Our modular modeling concept might seem unnecessary complicated for the small network of the simplified repressilator, but it becomes of tremendous advantage as soon as the complexity of the involved modules increases [9].

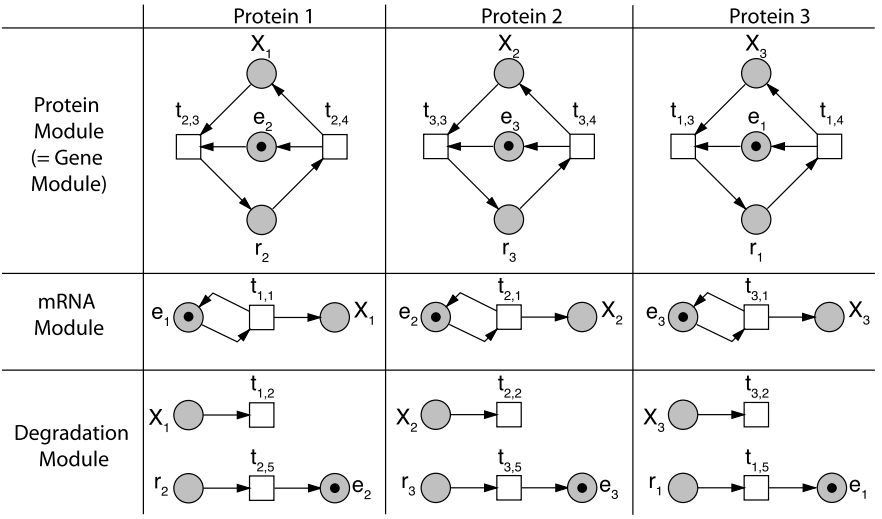


Fig. 24 Modular composition of the simplified repressilator. The model of the simplified repressilator can be composed through a gene, protein, mRNA, and protein degradation module for each of the three components. The connection is established through identical subnets (places and transitions), called interface subnetworks (logical nodes indicated in *grey*). In this trivial example, the modules in each row seem at the very first glance to look like instances of one and the same module, but each repressor protein has its own individual protein, mRNA, gene, and degradation module. One could easily extend the modules individually to represent the original repressor proteins (*lacI*, *tetR*, *cI*) [25], their interactions, biosynthesis, and degradation in more detail. Note that the model as depicted here is directly executable in Snoopy, meaning that the token flow can either be animated or simulated in all Petri net classes

6.6 Automatic Network Reconstruction

By simulation one can determine the time-dependent dynamic behavior of a Petri net. However, it remains unclear whether or not Petri nets of alternative structure would display a similar or even almost identical behavior.

When simulation results cannot be fitted to experimental data no matter which parameter sets are used, it can be concluded that the model is invalid in a sense that the model does not provide a sufficiently good abstraction of the reality. With other words, simulations can demonstrate that the underlying assumptions were wrong.

On the other hand, when simulation results obtained with a Petri net model fit a set of experimental data, this unfortunately does not mean that the model correctly reflects the real mechanisms. It only means that the given model is able to reproduce the experimental data. This is true in systems biology, but it is also a basic fact in chemical kinetics. Potentially, there might be thousands of models that could behave in a very similar way. From the scientific point of view, the first case, disagreement, is more helpful for the experimental researcher. Disproving a model definitely justifies further research while being in agreement may motivate to not design new experiments, although this would in principle be necessary. In this respect, mod-

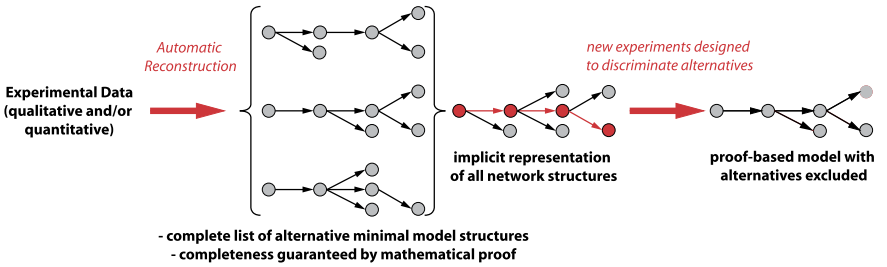


Fig. 25 Steps on the way from time series data to a proof-based dynamic Petri net model. The alternative network structures as determined by the ANR algorithm can be summarized in the form of an implicit representation telling which structural features all models have in common (nodes shown in *red*) and which nodes of the network might be wired up alternatively as displayed. By considering the nodes with alternative connections one can design new experiments that specifically discriminate between the alternatives to finally obtain a model structure based on mathematical proof

eling and simulation can even lead to very counter-productive results in retarding research.

Based on these thoughts, we wanted to go the alternative way by developing a reverse engineering approach to reconstruct Petri nets from experimental time series data sets. The approach should work in a fully automatic manner, that is, without heuristic input, as this might introduce a bias by the operator and hence give different results for different persons working on the same data set. The idea behind *automatic network reconstruction* therefore was to have an algorithm that automatically gives all possible Petri nets that comply with a set of experimental results or observations. To be trustworthy, the completeness of this list should be proven mathematically.

Since Petri nets in their plain form model discrete events, the developed method relies on discrete optimization [62]. Before we explain the basic principle, let us first consider what input data are required and what kinds of results the method delivers.

Input data used for network reconstruction are usually time series data reflecting the response of the system to perturbation. Often, experimental data obtained in the bio-lab are innately discrete, for example, like the occurrence of a certain phenotype in response to stimulation of a cell. In other cases the response to perturbation will be measured through the change in the cellular concentration of biomolecules. Such data then have to be discretized to be used for automatic network reconstruction. In doing so, one will not count the number of molecules or focus on small changes in concentrations even if they should be statistically significant. Instead, perturbations are chosen that cause a considerable, extensive response of the system. In other words, experiments are designed in a way that discretization of quantitative (continuous) time series data is uncritical for the performance of the reconstruction algorithm as long as the qualitative behavior of the system is concerned.

Starting from discrete time series data sets, the method gives a complete list of all Petri nets that are able to quantitatively reproduce the input data; see Fig. 25.

Since this list may contain many thousands of nets, the alternative network structures found may be displayed in the form of implicit representations. The implicit representations tell which structural features all models have in common and which nodes of the network might be wired up alternatively. By considering the nodes that may have alternative connections one can design new experiments that specifically discriminate between these alternatives. In the best case, a model structure can be obtained, which is finally proven mathematically through exclusion of all possible alternatives; see Fig. 25. Certainly, this is an iterative process that requires high-quality data at high density. Continuous data have to be discretized to fit the algorithm. In the simplest form, the result of discretization is boolean $(0/1)$, but discrete numbers would be possible as well. Using discrete values is entirely in agreement with the format, in which experimental results are obtained in the bio-wetlab. Often, entities are measured quantitatively, but the findings are stated in a discrete manner anyway. "If the gene *XYZ* is deleted, cells lose the ability to use mannitol as a food source" is be a typical way of how experimental findings are stated in the literature. Mechanistic models are widely based on such kind of statements. Often, perturbation and response both appear in discrete format as an experimental result. Of course, once a Petri net is established through reverse engineering, stochastic or continuous simulations can be run, and quantitative experimental data, as far as available, can be used accordingly to fit kinetic parameters.

The basic concept of *automatic network reconstruction* (ANR) is simple. Consider a time series where three components (or states of components) A, B, and C are measured as functions of time; see Fig. 26. Each component corresponds to a place. At time t_1, places A and B are both marked by a token, whereas C is not marked. At time t_2, the tokens in A and B have disappeared, but C is marked. The difference between the two time points t_1 and t_2 in the time series data set defines the difference vector $\mathbf{d} = (-1, -1, 1)^{\mathrm{T}}$. This difference vector can be realized by the corresponding reaction vector $\mathbf{r} = (-1, -1, 1)^{\mathrm{T}}$ (a column in the incidence matrix), which means that one token is removed from A and B, whereas one token appears in place C upon firing of the transition T1; see Fig. 26. For this trivial example, the principle of ANR is easy to illustrate. The mathematical challenge of ANR arises from the fact that a given difference vector $\mathbf{d}$ can be the sum of different reaction vectors $\mathbf{d} = \sum_{i=1}^{n} \mathbf{r}_i$. This results in Petri nets of different structure since the marking of a Petri net may have changed $n - 1$ times in between two experimentally measured states; see Fig. 26. These changes of the marking may escape from being measured because they are simply missed by the measurement or because they involve components (places) that are not measured at all or not even known to be involved in the overall process [62]. In other words, more than one transition may fire in between two measurements. The task of the algorithm is to find the minimal set of places P connected with a minimal set of transitions such that all observed difference vectors $\mathbf{d}_j$ can be reached in the sequential order as given by the data set [21, 62]. In order to exactly reproduce the experimental observations, we additionally use priorities among transitions to enforce an order in which the competing transitions fire [21]. These priorities reflect relative kinetic rate constants. A prerequisite for the algorithm to give correct results is that the number of time points

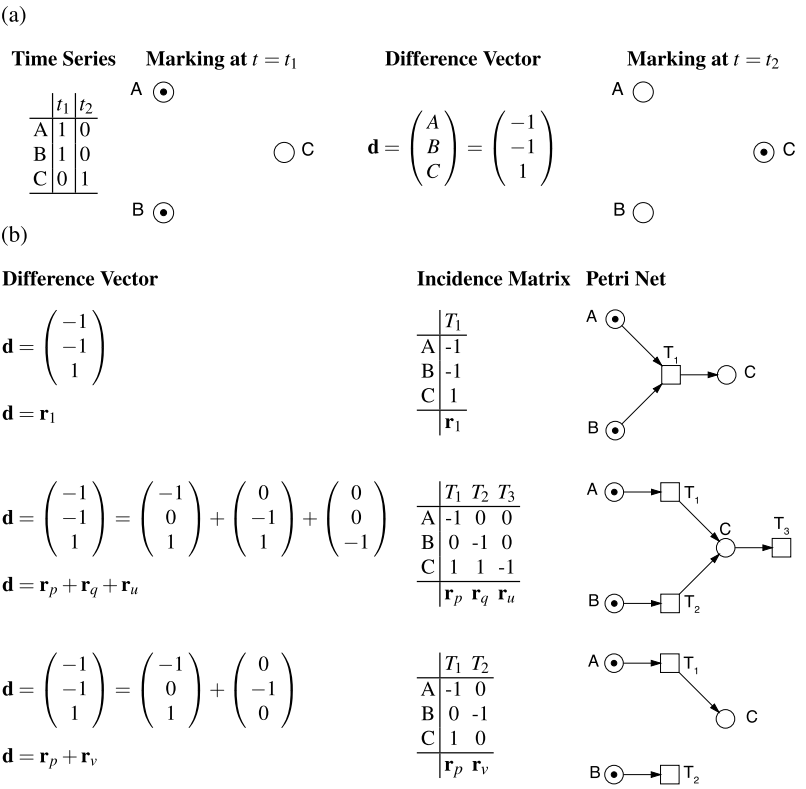


Fig. 26 Illustration of the basic principle of automatic network reconstruction. (**a**) The input for the reconstruction algorithm is a time series data set describing the time course of the components of interest (A, B, C) in the form of discrete values. At a given time point, the value for each component corresponds to the marking of the places representing each of the respective components. The difference in marking of the places between two successive time points in the time series defines a difference vector. (**b**) Relationship between difference vectors **d**, reaction vectors **r**, incidence matrix, and the corresponding graphical representations of the Petri net. In the example given, the same difference vector can be decomposed into sums of different reaction vectors. For the reconstruction of plain Petri nets, the reaction vectors of each difference vector directly correspond to columns in the incidence matrix that defines the structure of the Petri net. Note however that for extended Petri nets, this is not necessarily the case since a given reaction vector can result from different transitions, which are controlled by different places, as seen in Fig. 27. The figure is redrawn from [23] and [21]

taken for a series needs to be sufficiently high to correctly capture the time-discrete characteristics of the components that change in time.

It makes sense that the described reconstruction algorithm [21] considers only macroscopic changes, which can be observed at the time scale at which the measurements are performed. The algorithm does not consider periodically changing components if their cyclic formation and decay are so fast that these reactions cannot even be observed at the time scale of interest. This restriction prevents an ex-

plosion of solutions [21]. However, fast periodic processes like formation and decay of enzyme-substrate-complexes during enzymatic (catalytic) reactions, which are of fundamental importance in biochemical networks, are systematically excluded if the reconstructed networks are restricted to plain (simple) Petri nets [21]. This limitation is even more severe as genes in general catalyze the biosynthesis of the proteins they encode. A way to overcome this limitation is to model a catalyst (e.g. an enzyme or a gene) as place coupled by a read arc to the transition mediating the catalyzed reaction [23]. Inhibition, an essential phenomenon in regulatory networks, is represented accordingly using an inhibitory arc instead of a read arc. Hence, we are left with the task of reconstructing extended Petri nets. Extended Petri nets are Petri nets that contain read and/or inhibitory arcs [66].

An extended Petri net can be viewed as consisting of two parts. One part is composed of the places and transitions that are linked with each other by standard arcs. The complementing part is composed of places and transitions that are connected to each other with read arcs or inhibitory arcs. Accordingly, the problem of reconstructing extended Petri nets is split into two tasks: (1) reconstruct how places and transitions are linked through standard arcs, as described above, and (2) reconstruct how places do control transitions by read arcs or inhibitory arcs. Accordingly, each set of transitions that connect the same places in the same direction is encoded by a controlled reaction $\mathbf{R}_c = (\mathbf{r}, f_r)$. The reaction vector $\mathbf{r}$ indicates the change in the marking of places caused by firing of any of the transitions of the set. The control function f_r encodes the read arcs and inhibitory arcs connected to the transitions; see Fig. 27 [23]. For the control function $f_r = 1$, the transition with the corresponding reaction vector is controlled neither by a read arc nor by an inhibitory arc. Transitions that are controlled can only fire if the marking of the controlling places is according to the boolean expression of the control function; see Fig. 27. Any transition could be under the control of multiple places. Finally, a set of possible controlled reactions $(\mathbf{r}, f_r)$ is obtained for each difference vector of a given sequence of difference vectors as defined by the time series data set, which has been used for network reconstruction. A table displaying these controlled reactions is an implicit representation of all Petri nets that can simulate the data set. Any arbitrary sequence of controlled reactions composed by taking one set of controlled reactions $(\mathbf{r}, f_r)$ from each of the columns of the table displaying subsequently occurring difference vectors (see Fig. 28) gives one functional extended Petri net. The obtained extended Petri net is fully compatible with the time series data set that originally served as input [23]. Again, it is guaranteed that a complete set of Petri nets all of which comply with the input data is obtained [24]. A unique solution in terms of a single Petri net is obtained if the algorithm finds only one entry of controlled reactions for each difference vector. Recently, both answer set programming [22] and integer logic programming [79] have been employed to solve the network reconstruction problem.

Simplified Repressilator In Fig. 28(b), the model of the simplified repressilator is displayed by a set of controlled reactions. An extended Petri net is obtained by interpreting places with identical names as logical places.

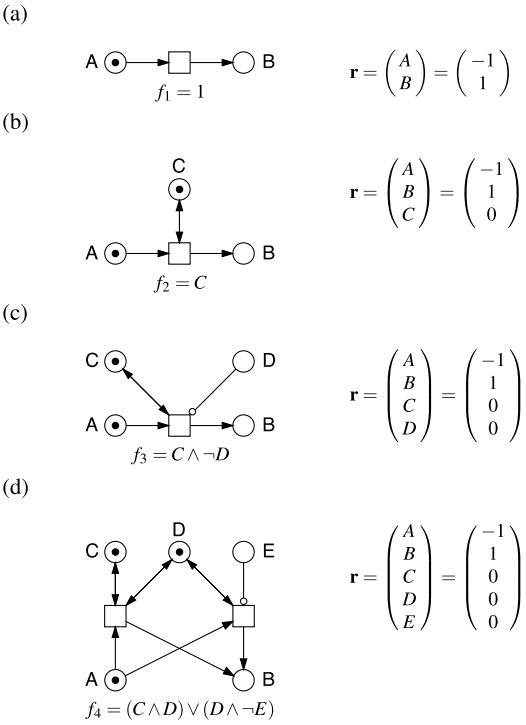


Fig. 27 Implicit representation of extended Petri nets by controlled reactions. A controlled reaction is a pair $(\mathbf{r}_i, f_i)$ composed of the reaction vector $\mathbf{r}_i$ and the associated control function f_i. The arcs of an extended Petri net can be thought as consisting of two sets. (1) The standard arcs and (2) the control arcs (read arcs/bidirected arcs and inhibitory arcs). A reaction vector describes how the marking of the connected places changes upon firing of a transition. The control function defines the conditions under which the firing of at least one among all transitions with the same reaction vector may occur. The marking of the places of all four Petri nets shown in panels (**a**) to (**d**) has been chosen such that all transitions can fire. In panel (**d**), the reaction vector $\mathbf{r}_4$ of the controlled reaction $(\mathbf{r}_i, f_i)$ represents the set of two transitions, each of which connects the places A and B in the same direction through standard arcs while the transitions are connected to different control arcs. The figure and legend are taken from [23] with slight modifications. Symbols: $\wedge$, logic AND; $\vee$, logic OR; $\neg$, logic NOT

6.7 Petri Net Tools

We used the sophisticated toolkit consisting of Snoopy, Charlie, and MARCIE, provided and publicly available at http://www-dssz.informatik.tu-cottbus.de.

Snoopy [44, 63] is a tool to model and animate/simulate hierarchically structured graphs, among them, $\mathcal{QPN}$, $\mathcal{SPN}$, $\mathcal{CPN}$, and $\mathcal{HPN}$. Furthermore, it comprises the colored counterparts of those net classes. Petri nets can be exported in systems biology markup language (SBML) code to be coherent with the systems biology community [49]. Models given in SBML can also be imported in Snoopy and represented as a Petri net.

(a)

$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$	...
$(\mathbf{r}_1, f_1)$	...	...	...	...
$(\mathbf{r}_2, f_2)$, $(\mathbf{r}_7, f_7)$	...	...	...	...
$(\mathbf{r}_7, f_7)$, $(\mathbf{r}_2, f_2)$	...	...	...	...
$(\mathbf{r}_3, f_3)$, $(\mathbf{r}_6, f_6)$	...	...	...	...
$(\mathbf{r}_6, f_6)$, $(\mathbf{r}_3, f_3)$	...	...	...	...
$(\mathbf{r}_4, f_4)$, $(\mathbf{r}_5, f_5)$	...	...	...	...
$(\mathbf{r}_5, f_5)$, $(\mathbf{r}_4, f_4)$	...	...	...	...
...	...	...	...	...

(b)

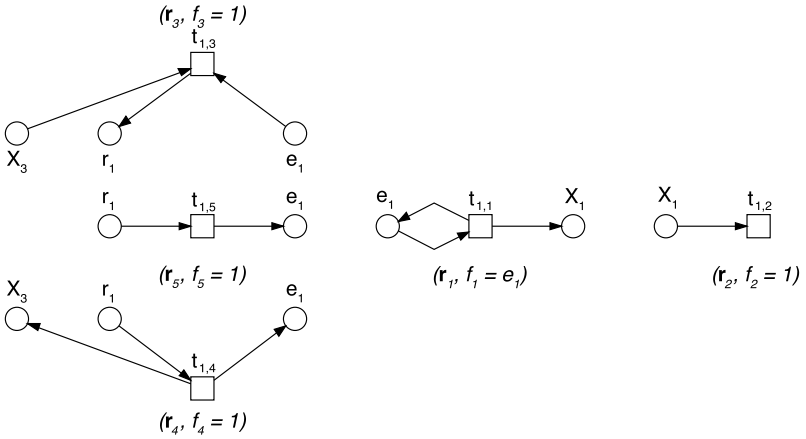


Fig. 28 A composition of Petri nets from controlled reactions. (**a**) The algorithm for reconstructing extended Petri nets provides for each difference vector $\mathbf{d}_i$ the complete set of possible controlled reactions $(\mathbf{r}_i, f_i)$, as schematically arranged in a table where all possible controlled reactions of subsequent difference vectors are listed in subsequent columns. Any arbitrary sequence of controlled reactions obtained by taking one difference vector from each of the subsequent columns gives one extended Petri net that behaves according to the time series data set that originally served as input. *Red boxes* indicate one possible trajectory for the assembly of a valid Petri net. Panel (**b**) shows the Petri net structures corresponding to six controlled reactions as part of the simplified repressilator. If the places with the same name are interpreted as logic places, then the six networks corresponding to the controlled reactions give a functional extended Petri net. (**a**) is redrawn from [23]

Charlie [28] is a multi-thread analysis tool for basic Petri net properties and techniques like structural boundedness check, invariant computation, siphon-trap property, etc. Moreover, Charlie supports the basic vocabulary of explicit CTL and LTL model checking.

MARCIE [45] is a symbolic CTL model checker for $\mathcal{QPN}$ and a multi-thread symbolic CSL model checker for generalized $\mathcal{SPN}$. Additionally, MARCIE supports simulative PLTLc model checking of extended stochastic Petri nets.

Acknowledgement We thank Mostafa Herajy, Fei Liu, and Martin Schwarick for their continuous support in developing Snoopy, Charlie, and MARCIE. Mary-Ann Blätke and Christian Rohr

were financially supported by the IMPRS Magdeburg through the Excellence Initiative of Saxony-Anhalt.

References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model checking continuous time Markov chains. ACM Trans. Comput. Log. **1**(1), 162–170 (2000)
2. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. **29**(6), 524–541 (2003)
3. Ballarini, P., Mardare, R., Mura, I.: Analysing biochemical oscillation through probabilistic model checking. Electron. Notes Theor. Comput. Sci. **229**(1), 3–19 (2009)
4. Baumgarten, B.: Petri-Netze—Grundlagen und Anwendungen. Spektrum, München (1996)
5. Blätke, M.A., Meyer, S., Stein, C., Marwan, W.: Petri net modeling via a modular and hierarchical approach applied to nociception. In: Int. Workshop on Biological Processes & Petri Nets (BioPPN), Satellite Event of Petri Nets 2010, pp. 131–145 (2010)
6. Blätke, M.A., Heiner, M., Marwan, W.: Tutorial—Petri Nets in Systems Biology. Otto von Guericke University and Magdeburg, Centre for Systems Biology (2011)
7. Blätke, M.A., Dittrich, A., Heiner, M., Schaper, F., Marwan, W.: JAK-STAT signaling as example for a database-supported modular modeling concept. In: Gilbert, D., Heiner, M. (eds.) Proceedings of the 10th Conference on Compuational Methods in Systems Biology. LNCS/LNBI, vol. 7605, pp. 362–365. Springer, Berlin (2012)
8. Blätke, M.A., Heiner, M., Marwan, W.: Predicting phenotype from genotype through automatically composed Petri nets. In: Gilbert, D., Heiner, M. (eds.) Proceedings of the 10th Conference on Compuational Methods in Systems Biology. LNCS/LNBI, vol. 7605, pp. 87–106. Springer, Berlin (2012)
9. Blätke, M.A., Dittrich, A., Rohr, C., Heiner, M., Schaper, F., Marwan, W.: JAK/STAT signaling—an executable model assembled from molecule-centered modules demonstrating a module-oriented database concept for systems and synthetic biology. Mol. BioSyst. **9**(6), 1290–1307 (2013)
10. Blätke, M.A., Heiner, M., Marwan, W.: Linking protein structure with network behavior to generate biologically meaningful mutations in computational models of regulatory networks. Unpublished work
11. Breitling, R., Gilbert, D., Heiner, M., Orton, R.: A structured approach for the engineering of biochemical network models, illustrated for signaling pathways. Brief. Bioinform. **9**(5), 404–421 (2008)
12. Breitling, R., Donaldson, R., Gilbert, D., Heiner, M.: Biomodel engineering—from structure to behavior (position paper). In: Transactions on Computational Systems Biology XII, Special Issue on Modeling Methodologies, vol. 5945, pp. 1–12 (2010)
13. Calzone, L., Chabrier-Rivier, N., Fages, F., Soliman, S.: Machine learning biochemical networks from temporal logic properties. In: Transactions on Computational Systems Biology VI, pp. 68–94 (2006)
14. Chaouiya, C., Remy, E., Ruet, P., Thieffry, D.: Qualitative modeling of genetic networks: from logical regulatory graphs to standard Petri nets. In: Applications and Theory of Petri Nets 2004, pp. 137–156. Springer, Berlin (2004)
15. Chaouiya, C., Remy, E., Thieffry, D.: Petri net modeling of biological regulatory networks. J. Discrete Algorithms **6**(2), 165–177 (2008)
16. Chen, L., Qi-Wei, G., Nakata, M., Matsuno, H., Miyano, S.: Modeling and simulation of signal transductions in an apoptosis pathway by using timed Petri nets. J. Biosci. **32**(1), 113–127 (2007)
17. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)
18. Curry, E.: Stochastic simulation of entrained circadian rhythm. Master thesis (2006)

19. Desel, J., Esparza, J.: Free Choice Petri Nets, vol. 40. Cambridge University Press, Cambridge (1995)
20. Donaldson, R., Gilbert, D.: A model checking approach to the parameter estimation of biochemical pathways. In: Computational Methods in Systems Biology. LNCS (LNBI), vol. 5307, pp. 269–287. Springer, Berlin (2008)
21. Durzinsky, M., Weismantel, R., Marwan, W.: Automatic reconstruction of molecular and genetic networks from discrete time series data. Biosystems **93**(3), 181–190 (2008)
22. Durzinsky, M., Marwan, W., Ostrowski, M., Schaub, T., Wagler, A.: Automatic network reconstruction using ASP. Theory Pract. Log. Program. **11**, 749–766 (2011)
23. Durzinsky, M., Wagler, A., Marwan, W.: Reconstruction of extended Petri nets from time series data and its application to signal transduction and to gene regulatory networks. BMC Syst. Biol. **5**(1), 113 (2011)
24. Durzinsky, M., Marwan, W., Wagler, A.: Reconstruction of extended Petri nets from time-series data by using logical control functions. J. Math. Biol. **66**, 203–223 (2013). doi:10.1007/s00285-012-0511-3
25. Elowitz, M.B., Leibler, S.: A synthetic oscillatory network of transcriptional regulators. Nature **403**(6767), 335–338 (2000)
26. Emerson, E.A., Halpern, J.Y.: Sometimes and not never revisited: on branching versus linear time temporal logic. J. ACM **33**, 151–178 (1986)
27. Fisher, J., Henzinger, T.A.: Executable cell biology. Nat. Biotechnol. **25**(11), 1239–1249 (2007)
28. Franzke, A.: Charlie 2.0—a multithreaded Petri net analyzer. Diploma thesis (2009)
29. Gao, Q., Gilbert, D., Heiner, M., Liu, F., Maccagnola, D., Tree, D.: Multiscale modeling and analysis of planar cell polarity in the Drosophila wing. IEEE/ACM Trans. Comput. Biol. Bioinform. **99**, 1 (2012)
30. Gilbert, D., Heiner, M.: Multiscale modeling for multiscale systems biology (2011). http://multiscalepn.brunel.ac.uk
31. Gilbert, D., Heiner, M., Rosser, S., Fulton, R., Gu, X., Trybiło, M.: A case study in model-driven synthetic biology. In: IFIP WCC 2008, 2nd IFIP Conference on Biologically Inspired Collaborative Computing (BICC 2008). IFIP, vol. 268, pp. 163–175. Springer, Boston (2008)
32. Gilbert, D., Heiner, M., Liu, F., Saunders, N.: Coloring space—a colored framework for spatial modeling in systems biology. In: Colom, J., Desel, J. (eds.) Proc. PETRI NETS 2013. LNCS, vol. 7927, pp. 230–249. Springer, Berlin (2013)
33. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
34. Goss, P.J., Peccoud, J.: Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. Proc. Natl. Acad. Sci. **95**(12), 6750–6755 (1998)
35. Green, M., Sambrook, J.: Molecular Cloning. A Laboratory Manual, 4th edn. Cold Spring Harbor Laboratory Press, Cold Spring Harbor (2012)
36. Hack, M.: Analysis of production schemata by Petri nets (1972)
37. Hardy, S., Robillard, P.N.: Petri net-based method for the analysis of the dynamics of signal propagation in signaling pathways. Bioinformatics **24**(2), 209–217 (2008)
38. Hecker, M., Lambeck, S., Toepfer, S., Van Someren, E., Guthke, R.: Gene regulatory network inference: data integration in dynamic models—a review. Biosystems **96**(1), 86–103 (2009)
39. Heiner, M., Gilbert, D.: How might Petri nets enhance your systems biology toolkit. In: LNCS, vol. 6709, pp. 17–37. Springer, Berlin (2011)
40. Heiner, M., Gilbert, D.: Biomodel engineering for multiscale systems biology. Prog. Biophys. Mol. Biol. **111**(2–3), 119–128 (2013)
41. Heiner, M., Gilbert, D., Donaldson, R.: Petri nets for systems and synthetic biology. In: LNCS, vol. 5016, pp. 215–264. Springer, Berlin (2008)
42. Heiner, M., Lehrack, S., Gilbert, D., Marwan, W.: Extended stochastic Petri nets for model-based design of wetlab experiments. In: Transactions on Computational Systems Biology XI. LNCS/LNBI, vol. 5750, pp. 138–163. Springer, Berlin (2009)

43. Heiner, M., Donaldson, R., Gilbert, D.: Petri Nets for Systems Biology, pp. 61–97. Jones & Bartlett Learning (2010)
44. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy—a unifying Petri net tool. In: Proc. PETRI NETS 2012. LNCS, vol. 7347, pp. 398–407. Springer, Berlin (2012)
45. Heiner, M., Rohr, C., Schwarick, M.: MARCIE—Model checking And Reachability analysis done effiCIEntly. In: Colom, J., Desel, J. (eds.) Proc. PETRI NETS 2013. LNCS, vol. 7927, pp. 389–399. Springer, Berlin (2013)
46. Herajy, M.: Computational steering of multi-scale biochemical networks. PhD thesis, BTU Cottbus, Department of Computer Science (2013)
47. Herajy, M., Heiner, M.: Hybrid representation and simulation of stiff biochemical networks. Nonlinear Anal. Hybrid Syst. **6**(4), 942–959 (2012)
48. Hill, A.V.: The combinations of haemoglobin with oxygen and with carbon monoxide. I. Biochem. J. **7**(5), 471 (1913)
49. Hucka, M., Finney, A., Sauro, H.M., Bolouri, H., Doyle, J.C., Kitano, H., Arkin, A.P., Bornstein, B.J., Bray, D., Cornish-Bowden, A., et al.: The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. Bioinformatics **19**(4), 524–531 (2003)
50. Kiehl, T.R., Mattheyses, R.M., Simmons, M.K.: Hybrid simulation of cellular behavior. Bioinformatics **20**(3), 316–322 (2004)
51. Klipp, E., Liebermeister, W., Wierling, C., Kowald, A., Lehrach, H., Herwig, R.: Systems Biology. A Textbook. Wiley-VCH, Weinheim (2009)
52. Koch, I., Junker, B.H., Heiner, M.: Application of Petri net theory for modeling and validation of the sucrose breakdown pathway in the potato tuber. Bioinformatics **21**(7), 1219–1226 (2005)
53. Küffner, R., Zimmer, R., Lengauer, T.: Pathway analysis in metabolic databases via differential metabolic display (dmd). Bioinformatics **16**(9), 825–836 (2000)
54. Liu, F.: Colored Petri nets for systems biology. PhD thesis, Brandenburg Technical University (2012)
55. Liu, F., Heiner, M.: Modeling membrane systems using colored stochastic Petri nets. Nat. Comput. (online), 1–13 (2013). doi:10.1007/s11047-013-9367-8
56. Liu, F., Heiner, M.: Multiscale modeling of coupled Ca^{2+} channels using colored stochastic Petri nets. IET Syst. Biol. **7**(4), 106–113 (2013)
57. Liu, F., Heiner, M.: Petri Nets for Modeling and Analyzing Biochemical Reaction Networks. Springer, Berlin (2014). Chap. 9
58. Liu, F., Heiner, M., Rohr, C.: The manual for colored Petri nets in Snoopy—$QPN^C/SPN^C/CPN^C/GHPN^C$. Tech. Rep. 02-12, Brandenburg University of Technology Cottbus, Department of Computer Science, Cottbus (2012)
59. Loinger, A., Biham, O.: Stochastic simulations of the repressilator circuit. Phys. Rev. E **76**(5), 051,917 (2007)
60. Marbach, D., Prill, R.J., Schaffter, T., Mattiussi, C., Floreano, D., Stolovitzky, G.: Revealing strengths and weaknesses of methods for gene network inference. Proc. Natl. Acad. Sci. USA **107**(14), 6286–6291 (2010)
61. Marwan, W., Sujatha, A., Starostzik, C.: Reconstructing the regulatory network controlling commitment and sporulation in Physarum polycephalum based on hierarchical Petri net modeling and simulation. J. Theor. Biol. **236**, 349–365 (2005)
62. Marwan, W., Wagler, A., Weismantel, R.: A mathematical approach to solve the network reconstruction problem. Math. Methods Oper. Res. **67**(1), 117–132 (2008)
63. Marwan, W., Rohr, C., Heiner, M.: Petri nets in Snoopy: a unifying framework for the graphical display, computational modeling, and simulation of bacterial regulatory networks. In: Methods in Molecular Biology, vol. 804, pp. 409–437. Humana Press, Clifton (2012). Chap. 21
64. Michaelis, L., Menten, M.L.: Die Kinetik der Invertinwirkung. Biochem. Z. **49**(333–369), 352 (1913)

65. Miller, O. Jr, Hamkalo, B.A., Thomas, C. Jr: Visualization of bacterial genes in action. Science **169**(943), 392 (1970)
66. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
67. Papin, J.A., Hunter, T., Palsson, B.O., Subramaniam, S.: Reconstruction of cellular signaling networks and analysis of their properties. Nat. Rev. Mol. Cell Biol. **6**(2), 99–111 (2005)
68. Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Technische Hochschule Darmstadt (1962)
69. Pinney, J.W., Westhead, D.R., McConkey, G.A., et al.: Petri net representations in systems biology. Biochem. Soc. Trans. **31**(6), 1513–1515 (2003)
70. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, 1977, pp. 46–57. IEEE, New York (1977)
71. Reddy, V.N., Mavrovouniotis, M.L., Liebman, M.N., et al.: Petri net representations in metabolic pathways. In: Proc. Int. Conf. Intell. Syst. Mol. Biol., vol. 1, p. 96038982 (1993)
72. Rohr, C.: Simulative model checking of steady-state and time-unbounded temporal operators. In: ToPNoC VIII. LNCS, vol. 8100, pp. 142–158 (2013)
73. Sackmann, A., Heiner, M., Koch, I.: Application of Petri net based analysis techniques to signal transduction pathways. BMC Bioinform. **7**(1), 482 (2006)
74. Schulz-Trieglaff, O.: Modeling the randomness in biological systems. Master thesis (2005)
75. Shaw, O., Steggles, J., Wipat, A.: Automatic parameterisation of stochastic Petri net models of biological networks. Electron. Notes Theor. Comput. Sci. **151**(3), 111–129 (2006)
76. Simao, E., Remy, E., Thieffry, D., Chaouiya, C.: Qualitative modeling of regulated metabolic pathways: application to the tryptophan biosynthesis in E. coli. Bioinformatics **21**(suppl 2), ii190–ii196 (2005)
77. Soliman, S., Heiner, M.: A unique transformation from ordinary differential equations to reaction networks. PLoS ONE **5**(12), e14284 (2010)
78. Sontag, E., Kiyatkin, A., Kholodenko, B.N.: Inferring dynamic architecture of cellular networks using time series of gene expression, protein and metabolite data. Bioinformatics **20**(12), 1877–1886 (2004)
79. Srinivasan, A., Bain, M.: Knowledge-guided identification of Petri net models of large biological systems. In: Inductive Logic Programming, pp. 317–331 (2012)
80. Srivastava, R., Peterson, M.S., Bentley, W.E.: Stochastic kinetic analysis of the Escherichia coli stress circuit using sigma32-targeted antisense. Biotechnol. Bioeng. **75**, 120–129 (2001)
81. Stark, J., Brewer, D., Barenco, M., Tomescu, D., Callard, R., Hubank, M.: Reconstructing gene networks: what are the limits? Biochem. Soc. Trans. **31**(Pt 6), 1519–1525 (2003)
82. Stark, J., Callard, R., Hubank, M.: From the top down: towards a predictive biology of signaling networks. Trends Biotechnol. **21**(7), 290–293 (2003)
83. Zevedei-Oancea, I., Schuster, S.: Topological analysis of metabolic networks based on Petri net theory. In Silico Biol. **3**(3), 323–345 (2003)