Parameter Identification and Model Ranking of Thomas Networks[⋆]

Hannes Klarner[1], Adam Streck[2], David Šafránek[2], Juraj Kolčák[2], and Heike Siebert[1]

[1] Freie Universität Berlin, Berlin, Germany
Hannes.Klarner@fu-berlin.de
[2] Masaryk University, Brno, Czech Republic
safranek@fi.muni.cz

Abstract. We propose a new methodology for identification and analysis of discrete gene networks as defined by René Thomas, supported by a tool chain: (*i*) given a Thomas network with partially known kinetic parameters, we reduce the number of acceptable parametrizations to those that fit time-series measurements and reflect other known constraints by an improved technique of coloured LTL model checking performing efficiently on Thomas networks in distributed environment; (*ii*) we introduce classification of acceptable parametrizations to identify most optimal ones; (*iii*) we propose two ways of visualising parametrizations dynamics wrt time-series data. Finally, computational efficiency is evaluated and the methodology is validated on bacteriophage λ case study.

Keywords: Thomas network, parameter identification, model checking.

1 Introduction

Discrete modeling frameworks are commonly used in systems biology as a tool that assists in revealing regulatory mechanisms found in biological networks [14,11,20]. A widely used formalism for gene regulatory networks is that of R. Thomas et al. [21] (see [9] for review). The formalism treats changes in gene expression asynchronously, thus bringing a sort of conservatism into the discrete abstraction at the price of large state spaces with many transitions. However, the asynchronous semantics is a natural approach to formalization of concurrent systems in computer science. This enables application of well-established formal methods to Thomas networks [5,16,4,19].

Although discrete regulatory models are very abstract, parameters determining the behavior of regulated components are often unknown. An important problem is therefore inference of these parameters from biological hypotheses and wet-lab measurements e.g. time series data. There is no reliable technique to reveal the regulatory logic, and existing reverse engineering approaches are mostly based on measurement clustering or information theory (see [15] for review).

[⋆] This work has been supported by the Czech Grant Agency grant No. GAP202/11/0312.

Formal methods have been employed to assist in identifying parameters for Thomas networks, utilizing not only time series data but also arbitrary hypotheses formalized in terms of a temporal logic. Naive (bottom-up) approaches [4,12] repeat a procedure deciding for each parametrization whether it satisfies the given temporal constraints or not. That way *acceptable parametrizations* are found. Since the number of possible parametrizations increases exponentially with the number of unknown parameters, such a procedure is intractable in many real cases.

Barnat et al. [2] introduced technique of *colored LTL model checking* (CMC) based on a heuristics reducing the computation effort by means of operating on the parametrization space in a top-down manner. In particular, maximal parametrization sets sharing a required behavior are inferred instead of analyzing each possible parametrization individually. The technique was defined for multi-affine abstractions of continuous models and was based on symbolic representation of parametrization sets thus allowing effective realization of required operations. When employed on Thomas networks, an ideal symbolic representation which would allow effective realization of all required set operations was not found. Therefore the results obtained for Thomas networks were not optimal.

In [12], Klarner et al. developed a workflow for parameter identification of Thomas networks exploiting time series data. Especially notions of *edge constraints* and expression *monotonicity* in between measurements were defined to initially restrict acceptable parametrizations by preliminary known facts about network dynamics.

In this paper, authors of both groups combine their approaches to obtain efficient methods for parameter identification using colored model checking. The result of this collaboration is a comprehensive methodology that further extends the workflow of [12] introducing a *classification of acceptable parametrizations* based on optimal satisfaction of selected criteria. Our methodology guides users towards selection of parametrizations complying with given hypotheses and time series data, and proposes further filtering of obtained parameters based on criteria such as low complexity. Moreover, visualization procedures are proposed that allow a quick and intuitive understanding of the behavior generated by different parametrizations allowing for easy identification of e.g. potential ranges of poor measurement sampling. The workflow is depicted in Fig. 1.

To the best of our knowledge, the only work which attempts to employ some criteria to select most plausible parametrizations in the context of Thomas networks is mentioned in [7]. The approach is a work in progress based entirely on constraint programming. As there are no concrete criteria defined, we currently cannot compare the methodological side.

On the computational side, our approach is supported by a prototype tool chain consisting of three modules: static analyzer, model checker, and parametrization filter. The static analyzer module solves constraints related to the network structure and is implemented on the top of the model checker module. The model checker module implements CMC including computation of compliant behaviors (in model checking terms: generation of all counterexamples for a given time series formula) and parameter ranking. The parametrization filter allows browsing the parameters and filtering them wrt several criteria. Moreover, the filter module gives graphical feedback to the user.

Computational efficiency is obtained by direct distribution and shared enumeration of parametrization sets. To the best of our knowledge, there is only one other efficient approach [3] targeting discrete gene dynamics. It employs a more detailed model – the piece-wise affine framework. The representation of parameter space is specific for the level of abstraction employed. Efficiency is obtained by considering symbolic representation of parametrizations.

The paper, after introducing the basic notions in the next section, is structured according to the workflow mentioned above and depicted in Fig. 1. To illustrate the approach a case study of the bacteriophage λ is considered in Sect. 5. Further information on implementation and performance as well as final remarks conclude the paper.

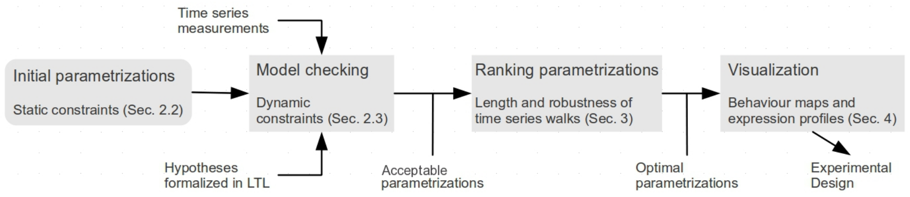

Time series measurements
Initial parametrizations
Static constraints (Sec. 2.2)
Model checking
Dynamic constraints (Sec. 2.3)
Ranking parametrizations
Length and robustness of time series walks (Sec. 3)
Visualization
Behaviour maps and expression profiles (Sec. 4)
Hypotheses formalized in LTL
Acceptable parametrizations
Optimal parametrizations
Experimental Design

Fig. 1. Parameter identification workflow

2 Background

2.1 Thomas Networks

In the following we recall the logical modeling framework introduced by C. Chaouiya et al. in [5, Section 2], which is a generalization of the formalism of R. Thomas [21].

Regulatory Graphs. The structure of a system, i.e. the components (or species) involved and the dependencies between them, can be captured in a graph. We define an *interaction graph* (V, E) to be a directed graph consisting of $n \in \mathbb{N}_1$ vertices $V = \{v_1, \dots, v_n\}$ called *components* and a set $E \subseteq V \times V$ of ordered pairs of vertices called *interactions*. We use the notation $uv \in E$ for interactions and call u the *regulator of* uv and v the *target of* uv. The in-neighbors $N_E^-(v) := \{u \in V \mid uv \in E\}$ of v are called *regulators of* v and the out-neighbors $N_E^+(v)$ are called *targets of* v.

Since we are not only interested in the structure of the network but also in the dynamics, we interpret the vertices as integer variables whose values signify e.g. the level of concentration of the corresponding substance. Naturally, the impact a regulator has on its target depends on the value of the corresponding variable. This information about the interactions, i.e. the edges in the interaction graph, is also needed to specify the dynamical behavior of the system. This leads to the following definition.

A *regulatory graph* $\mathcal{R} = (V, E, \rho, \theta)$ consists of an interaction graph (V, E) and two functions ρ and θ. The function $\rho : V \to \mathbb{N}_1$ assigns a non-zero natural number $\rho(v)$, called *maximal activity level* of v, to each component. For an *integer interval* $\{k \in \mathbb{N} \mid a \leq k \leq b\}$ with boundaries $a \leq b \in \mathbb{N}$ we use the notation $[a, b]$. The interval $[0, \rho(v)]$ is called *activity interval of component* v and an element of the activity interval is called *activity level* of v.

To a regulatory graph $\mathcal{R}$ we thus associate the *state space* $X := \prod_{i=1}^{n} [0, \rho(v_i)]$. An element $x \in X$ is called a *state* of the regulatory graph and we use the subscript notation x_v to denote the activity of $v \in V$ in state x.

The other function, θ, assigns interaction thresholds $\theta(uv) = (t_1, \dots, t_k)$ to each interaction $uv \in E$. Each interaction may have a different number $1 \leq k$ of thresholds. The thresholds must be ordered: $t_1 < \dots < t_k$ and within the non-zero activities of the regulator: $1 \leq t_1$ and $t_k \leq \rho(u)$.

The interaction thresholds $\theta(uv) = (t_1, \dots, t_k)$ of an interaction uv divide the activities of u into $k + 1$ intervals $[0, t_1 - 1], [t_1, t_2 - 1], \dots, [t_k, \rho(u)]$ of different *regulation intensity*. Activities of u that belong to the same interval are characterized by being above the same number of thresholds of $\theta(uv)$. We denote the j^{th} interval by I_j^{uv}. The different regulation intervals allow us to distinguish between different effects an interaction between two components can have depending on the activity of the regulator.

Parametrizations. In this subsection we discuss how to parametrize a regulatory graph. Basically, we need to provide all the information necessary to determine effects of any regulators on its target in every state. The effect will not necessarily depend on the exact state, but only on the regulation intervals to which this state belongs. We formalize this idea in the following definitions.

A *regulatory context* ω of a component v assigns an intensity to every interaction $uv \in E$ targeting v. For every regulator $u \in N^-(v)$, there is a regulation intensity I_j^{uv}, such that $\omega(u) = I_j^{uv}$. The set of all combinatorially possible regulatory contexts of v is denoted by C_v.

A *parametrization* P assigns a *target activity value* P_v^ω to every context $\omega \in C_v$ of every component $v \in V$. A priori, the only condition on P is that $P_v^\omega \in [0, \rho(v)]$ is a valid activity of v. The set of all feasible parametrizations is denoted by $\mathcal{P}$.

A parametrized regulatory graph $(\mathcal{R}, P)$ is called *Thomas network* or *model.* Finally, a remark about the scope of the workflow we are going to propose: In Sec. 2.3, we suggest colored model checking to solve the problem of identifying feasible parametrizations. For computational reasons we will consider the values of ρ and θ fixed in a particular problem.

Asynchronous Dynamics. The dynamics of a Thomas model $(\mathcal{R}, P)$ can be captured in a so-called state transition graph, where the finite state space X constitutes the vertex set and edges between states represent state transitions as determined from the logical parameters in the following way.

For every state x and every component v, there is a unique regulatory context $\omega \in C_v$, such that $\forall u \in N^-(v): \; x_u \in \omega(u)$. To see this, recall that $\omega(u)$ is a regulatory interval, and that these intervals form a partition of the activities of u.

The parametrization P therefore defines a function F on the state space:

$$F : X \to X, \;\; x \mapsto (P_{v_1}^{\omega_1}, \dots, P_{v_n}^{\omega_n}),$$

where ω_i is the unique regulatory context of component v_i in state x.

The function F can be interpreted as a finite dynamical system, i.e., the dynamics can be derived by iterating an initial state using F. In the resulting state transition graph, each state x has exactly one outgoing edge leading to $F(x)$. Clearly, the synchronicity of the involved processes is a strong idealization, which we want to avoid here.

Instead, the representation should reflect that the time delays associated with the different biological processes corresponding to the updates may vary greatly depending on the corresponding network components. However, the experimental information to determine these time delays is often lacking. This leads to the definition of a non-deterministic transition graph where each outgoing edge from a state corresponds to one of the indicated updates.

The transitions T_P of the *asynchronous and unitary state transition graph* (X, T_P) of a model $(\mathcal{R}, P)$ are derived from F by two rules. A loop $xx \in T_P$ exists, iff $F(x) = x$. An edge $xy \in T_P, x \neq y$ exists, if there is a component v, such that xy is *asynchronous*: $\forall u \neq v : x_u = y_u$ and *unitary*: $y_v - x_v = \text{sign}(F(x)_v - x_v)$. Here *sign* denotes the sign function.

The state transition graph (X, T_P) corresponds naturally to a Kripke structure (KS) $\mathcal{S}(\mathcal{R}, P) := (P, X, X^0, T_P, L)$, which is of interest for formal verification of temporal logical properties. Here, $\mathcal{S}$ consists of states X, initial states X^0, the transition relation T_P and a labeling function L over the atomic propositions AP expressing inequalities $\doteq \in \{=, \leq, \geq, <, >\}$ with

$$\text{AP} := \{v \doteq k \mid v \in V, k \in [0, \rho(v)]\}.$$

If not otherwise noted, all states are considered as initial states, i.e., $X^0 := X$. The labeling function is defined as $L(x) := \{v \doteq k \mid v \in V, k \in [1, \rho(v)], x_v \doteq k\}$.

Finally, the Kripke structure can be generalized to incorporate all possible parametrizations $\mathcal{P}$. For a given regulatory graph $\mathcal{R}$ we consider a *parametrized Kripke structure* (PKS) to be a tuple $\mathcal{S}(\mathcal{R}) := (\mathcal{P}, X, X^0, T_{\mathcal{P}}, L)$ where $T_{\mathcal{P}} := \bigcup_{P \in \mathcal{P}} T_P$ and all other elements are defined as above. The PKS $\mathcal{S}(\mathcal{R})$ thus represents all possible behaviors that can be generated by $\mathcal{R}$.

2.2 Constraints

In the following we introduce several notions that allow us to restrict the parameter space to the parametrizations in agreement with all the information we have on the system. We distinguish between static and dynamic constraints as already indicated in Fig. 1. Static constraints refer to information related to the regulatory graph, e.g. existence and character of interactions. In contrast, dynamic constraints capture properties of state transition graphs such as reachability requirements.

Static Constraints. Here we focus on edge labels, which are used to characterize the impact that a regulator has on its target. If there is an effect observable at all, it can be either *activating*, i.e., causing an increase, or *inhibiting*, i.e., causing a decrease in the activity of the target. Formally, several semantics result from combinations of these effects (see [12, Def. 2.9]). Certain edge labels have already been used successfully in case studies of D. Thieffry (see e.g. [18],[10]) and also implemented in analysis tools [17, p. 6].

Since we are dealing with regulatory graphs, whose interactions may have more than one threshold, the concept of edge label must be adjusted accordingly. An edge label is therefore not assigned to a single edge uv, but to a tuple (uv, t_j) where $uv \in E$ and $t_j \in \theta(uv)$. In this paper, we restrict ourselves to unlabeled edges and labels chosen from the set $\{+, -, \text{mon}+, \text{mon}-\}$, where the different notions are defined as follows.

Assume a tuple (uv, t_j) is labeled with mon+. A parametrization P satisfies this label, if for all regulatory contexts $\omega \in C_v$, such that $\omega(u) = I_j^{uv}$ and $\omega' \in C_v$ such that

$$\omega'(w) := \begin{cases} I_{j-1}^{uv} & \text{if } w = u \\ \omega(w) & \text{else} \end{cases}$$

the target value inequality $P_v^{\omega'} \leq P_v^{\omega}$ holds. If instead the label is mon−, then P satisfies this label if for all $\omega, \omega' \in C_v$ as defined above $P_v^{\omega'} \geq P_v^{\omega}$ is true.

The labels + and − correspond to mon+ and mon−, but require observability in addition. *A parametrization P satisfies the observability of* (uv, t_j), if contexts $\omega, \omega' \in C_v$ as defined above, exist, such that the target value inequality $P_v^{\omega'} \neq P_v^{\omega}$ holds.

Dynamic Constraints. In this paper we focus on identifying parametrizations that are in agreement with time series data, which can be interpreted as conditions constraining the dynamical behavior of a system. A *measurement* is a rectangular subset of the state space X. That is, we describe a measurement m by assigning to each component v a *measurement interval* $m_v = [a_v, b_v] \subseteq [0, \rho_v]$. We then identify this description m with the set of all states $x \in X$, such that $\forall v \in V, x_v \in m_v$.

A *time series* is a sequence of measurements $(m^1, \ldots, m^k)$. Notice that measurements may intersect, i.e., there may be states $x \in m^i \cap m^j$ for $i \neq j$.

A state transition graph $S = (X, T)$ *reproduces a time series* $(m^1, \ldots, m^k)$, if it contains a finite walk $(x^i)_{1 \leq i \leq r}, r \in \mathbb{N}_1$, such that there is a mapping $M : [1, k] \to [1, r]$ that is *ordered*: $i < j \implies M(i) \leq M(j)$ and *correct*: $x^{M(i)} \in m^i$.

We call such walk *time series walk.* Notice that we allow $M(i) = M(j)$. The walk can be thought of as a discrete simulation, and the mapping M as describing at which simulation steps the measurements were recorded. We say that a parametrization reproduces a time series, if its transition graph does.

There may of course be multiple walks satisfying these properties. We will discuss this in Section 3, where we introduce a ranking to capture *how well* a model reproduces a time series.

The existence of a time series walk is determined by LTL model checking over the Kripke structure (X, X^0, T, L) associated with the state transition graph (X, T) (see [1] for an introduction). The initial states are chosen in correspondence with a time series $(m^1, \dots, m^k)$ by $X^0 := m^1$.

A measurement m is translated into the LTL specification

$$\sigma(m) := \bigwedge_{v \in V} \bigvee_{k \in m_v} v \doteq k.$$

A state transition graph reproduces a time series $(m^1, \dots, m^k)$ if and only if there is a state $x \in X^0$, such that the LTL specification

$$\mathbf{F}(\sigma(m^2) \wedge \mathbf{F}(\sigma(m^3) \wedge \dots \mathbf{F}(\sigma(m^k)) \dots) \tag{1}$$

is satisfied in x.

Time series formulae of the form (1) constitute a specific class of properties enabling our analysis method as developed in Section 3. More general LTL formulae are used to specify, e.g., monotonicity of gene expression between two adjacent measurements m^i, m^{i+1} [12] or steady gene activity expected after the last measurement.

2.3 Parameter Identification by LTL Model Checking

In this section we describe the technology of colored model checking used for computing parametrizations satisfying constraints encoded in LTL. This technology is employed in the next sections as a cornerstone for identifying optimal parametrizations. The central notion is the construction of a map (coloring) relating each state x of a regulatory graph to the set of all those parametrizations from $\mathcal{P}$ under which x is reachable.

For a parametrization $P \in \mathcal{P}$ and its corresponding Kripke structure $\mathcal{S}(\mathcal{R}, P) \equiv (P, X_S, X_S^0, T_P, L)$, we define a run, denoted π, as an infinite path in $\mathcal{S}(\mathcal{R}, P)$. The notation π^0 is used to denote a run whose first node is in X_S^0. Since we aim to explore parametrizations which are realizable, i.e. there exists at least one behavior that satisfies given LTL constraints, we consider existential interpretation of LTL. We say that $\mathcal{S}(\mathcal{R}, P)$ satisfies φ, written $\mathcal{S}(\mathcal{R}, P) \models \varphi$, if there exists a run π^0 in $\mathcal{S}(\mathcal{R}, P)$ satisfying φ.

For a given regulatory graph $\mathcal{R}$ and an LTL formula φ, automata-based model checking is employed on $\mathcal{S}(\mathcal{R})$ to identify all parametrizations satisfying φ. As a prerequisite, we assume an alphabet $\Sigma = 2^{AP}$. Then φ is represented by means of a Büchi automaton over Σ, denoted $BA(\varphi)$, and defined $BA(\varphi) := (\Sigma, X_A, X_A^0, \delta_A, F_A)$, where X_A is a set of states, $X_A^0 \subseteq X_A$ is a set of initial states, $\delta_A \subseteq X_A \times \Sigma \times X_A$ is a transition relation, and $F_A \subseteq X_A$ is a set of accepting states. See [1] for techniques of translating φ into $BA(\varphi)$.

We utilize the approach of *colored model checking* (CMC) as introduced in [2]. CMC takes a PKS $\mathcal{S}(\mathcal{R})$, a parametrization space $\mathcal{P}$, and a Büchi automaton $BA(\varphi)$. It returns a *set of all acceptable parametrizations* $\mathcal{P}_\varphi := \{P \in \mathcal{P} \mid \mathcal{S}(\mathcal{R}, P) \models \varphi\}$. The procedure takes the following steps:

– constructing product automaton $BA(\mathcal{R}, \varphi) := \mathcal{S}(\mathcal{R}) \cap BA(\varphi)$
– computing $\mathcal{P}_\varphi$ by executing colored model checking on $BA(\mathcal{R}, \varphi)$

Product Automaton. $BA(\mathcal{R}, \varphi)$ is computed in the standard way [1] as a product of a PKS $\mathcal{S}(\mathcal{R}) \equiv (\mathcal{P}, X_S, X_S^0, T_\mathcal{P}, L)$ and $BA(\varphi) \equiv (\Sigma, X_A, X_A^0, \delta_A, F_A)$: $BA(\mathcal{R}, \varphi) := (\mathcal{P} \times \Sigma, X, X^0, \delta, F)$ where

$$X := X_S \times X_B, X^0 := X_S^0 \times X_A^0, F := X_S \times F_A \text{ and}$$

$$((x_s, x_a), (P, \alpha), (x_s', x_a')) \in \delta \text{ iff } x_s x_s' \in T_P \wedge (x_a, \alpha, x_a') \in \delta_A \wedge \alpha \in L(x).$$

If there exists $\alpha \in L(x)$ such that $(x, (P, \alpha), x') \in \delta$, we use the simplifying notation $x \xrightarrow{P} x'$. Transitive and reflexive closure of the relation $\rightarrow$ is denoted $\rightarrow^*$.

$BA(\mathcal{R}, \varphi)$ accepts π^0 - an infinite run through this product automaton - if and only if there is an $x \in F$ that occurs infinitely often on π^0 (projection of π^0 to the second component is an accepting run in $BA(\varphi)$). Hence $BA(\mathcal{R}, \varphi)$ accepts exactly the paths satisfying φ, and the acceptance is always caused by a cycle in $BA(\mathcal{R}, \varphi)$ containing some state in F – therefore we are interested in *accepting cycles* and their *reachability* from initial states.

Our interest is in paths that are realizable in a certain parametrization $P \in \mathcal{P}$. We denote by $BA(\mathcal{R}, \varphi)_P$ the product automaton $BA(\mathcal{R}, \varphi)$ with the alphabet $\{P\} \times \Sigma$ (restricted to the parametrization P). A run in $BA(\mathcal{R}, \varphi)_P$ is denoted π_P. We can conclude that $\mathcal{S}(\mathcal{R}, P)$ satisfies φ iff there exists a run π_P^0 in $BA(\mathcal{R}, \varphi)_P$ that is accepted.

Colored Model Checking. Naive (bottom-up) computation of P_φ by checking each parametrization $P \in \mathcal{P}$ individually suffers from the exponential explosion of $|\mathcal{P}|$ wrt number of unknown parameters. CMC [2] is a heuristic method based on the idea that transitions within PKS are shared by many parametrizations, therefore utilizing a single PKS for a check (top-down) is significantly faster than doing a check on every single KS $\mathcal{S}(\mathcal{R}, P)$.

An important notion is mapping $cl_{\hat{X}}^{\hat{\mathcal{P}}} : X \rightarrow 2^\mathcal{P}, \hat{X} \subseteq X, \hat{\mathcal{P}} \subseteq \mathcal{P}$, called *coloring*, in which each state $x \in X$ is assigned a set of parametrizations for which x is reachable from some state in $\hat{X}$, defined and denoted $cl_{\hat{X}}^{\hat{\mathcal{P}}}(x) := \{P \in \hat{\mathcal{P}} \mid \exists \hat{x} \in \hat{X} : \hat{x} \xrightarrow{P}^* x\}$. Using this mapping, the CMC procedure can be described as follows:

For each $x \in F$:
(1) Compute coloring $reach_x \equiv cl_{X^0}^{\mathcal{P}}(x)$ reaching accepting state x.
(2) Compute coloring $cycle_x \equiv cl_{\{x\}}^{reach_x}(x)$ enabling (accepting) cycles on x.

These two steps correspond to traditional LTL model checking [1], where we ask if there exists (1) a path from an initial to a final state and (2) a cycle containing this state, which implies existence of an accepting run. In our case, we do not ask for an existence of a single accepting run for each KS, but directly build a set of parametrizations that have an accepting run in PKS.

To obtain such a set, one has to perform a graph search, which can be done in numerous ways - in Section 6 we explain how to do those steps efficiently. Performance of the algorithm can be also greatly increased by omitting step (2) when using time series formula. This property is within a set of so-called *reachability* properties that can be computed without cycle detection [1].

3 Optimal Parametrizations

In the classical enumerative model checking approach to reverse engineering of Thomas networks, that was introduced by G. Bernot et al. in [4], a given set of parametrizations is divided into acceptable and unacceptable parametrizations depending on whether the transition graph associated to a parametrization satisfies the temporal logic specification or not.

From the perspective of the temporal specification, all acceptable parametrizations are equally suitable and the parameter model checking process ends here.

For the particular class of LTL specifications that we are interested in – the time series constraints as defined in Section 2.2, we introduce a method for ranking acceptable parametrizations.

3.1 The Length Cost

This section starts with a regulatory graph $\mathcal{R}$, a time series $(m^1, \ldots, m^k)$ and a non-empty set of parametrizations $\mathcal{P}' \subseteq \mathcal{P}$ that all reproduce the time series.

Denote by W_P the set of all time series walks of $(m^1, \ldots, m^k)$ in the state transition graph of a *single* parametrization $P \in \mathcal{P}'$. W_P may in general be an infinite set, but most of its walks are not relevant for our purposes. To impose a ranking on the set of time series walks, and through that a ranking on the set of parametrizations, we impose a preference for short walks. Since the walk length can be seen as a measure for the complexity of the behavior in terms of the number of processes that have to be executed to produce the desired result, this approach favors models that provide simple explanations for the observed behavior. In other words, we try to penalize unnecessarily complex realizations of time series data in a model which might also be related to a higher energy cost for the system.

We define the *length cost of a parametrization $P \in \mathcal{P}'$ with respect to the time series* as $\mathrm{Cost}(P) := \min\{r \in \mathbb{N} \mid \exists (x^i)_{1 \leq i \leq r} \in W_P\}$, and denote by

$$\mathrm{SW}_P = \{(x^i)_{1 \leq i \leq r} \in W_P \mid r = \mathrm{Cost}(P)\} \subseteq W_P$$

the set of *shortest walks of P*.

The length cost partitions $\mathcal{P}'$ into classes of equal cost, and we are particularly interested in parametrizations with the minimum cost, denoted by $\min_{\mathrm{Cost}}(\mathcal{P}') \subseteq \mathcal{P}'$.

3.2 Robustness

Since the dynamics in the Thomas formalism are non-deterministic, several paths may lead from one state to another and the path corresponding to the actual behavior of the system depends on the time delays associated with the different update processes. If these time delays change, maybe due to environmental influences, the system may follow a different trajectory even when considering the same initial state. However, in some cases, namely when there is only one path between two states in the state transition graph, the behavior of the system is independent of the actual values of the time delays. This can be interpreted as robustness of the system wrt perturbations of the time delays. In the following we will formalize this idea as a property of a given parametrization. Since we are interested in the realization of time series, we will focus our notion of robustness on the time series walks.

We use the standard notion of probability for a finite walk, as defined in [1], where each successor of a node is chosen with equal probability. Then, we say that the probability of a finite time series walk w of length l is

$$\text{Prob}(w) := \prod_{i=1}^{l-1} \frac{1}{deg^+(x^i)},$$

where $deg^+(x^i)$ is the out-degree of the state x^i of the walk.

We now define *robustness of a parametrization wrt time series* as the sum of probabilities of all distinctive time series walks. For set SW_P and set m^1 of all states that fit the first measurement, we set

$$\text{Robustness}(P) := \frac{\sum_{w \in \text{SW}_P}(\text{Prob}(w))}{|m^1|}.$$

For example, if the time series and the parametrization only allows for a single shortest time series walk, the robustness will be high if the states of the walk have low out-degree. In case the initial measurement has some unknown values, we take the average robustness of walks from all states that fit the measurement.

This notion of robustness is a good starting point for analysis since it distinguishes parametrizations that reproduce the time series with low ambiguity. In addition, it is easy to formalize and compute. A more involved definition should be based not only on the out-degree of a state of a time series walk, but differentiate and weight whether the different successors of the state are themselves states of a time series walk. This would extend the notion of robustness, taking into account not only perturbations of the time delays but also of the states, and will be investigated in future work.

3.3 Computing Optimal Parametrizations

The set of optimal parametrizations is obtained in the following manner:

1. Describe the set $\mathcal{P}$ of all possible parametrizations.
2. Remove parametrizations that do not satisfy edge constraints.

3. Compute the set of acceptable parametrizations using model checking.
4. Take the subset of those that have minimal Cost.
5. Finally, select parametrizations with maximal Robustness.

This way we obtain only parametrizations we have identified to be *optimal*, whose number is usually significantly smaller then the size of $\mathcal{P}$.

Such a procedure can be done automatically. Interpretation and further analysis of the results is left to the user. To support this step, in the following section we suggest two methods for visualization of results.

4 Visualization

In this section we present methods to visualize differences and similarities of parametrizations. To our knowledge, two automated lines of analysis of a set of parametrizations exist. In [8, Sec. 3.2], consensus target value inequalities are derived, while in [12, Sec. 5.1] the focus is on deriving consensus edge labels.

Here we present a novel approach that visualizes the transitions of *a set of acceptable parametrizations* in between measurements, highlighting agreement between parametrizations. We propose, firstly, *behavior maps* that represent state transitions according to the considered parametrizations, and, secondly, *expression profiles* that focus on the activity of a single component.

4.1 Behavior Maps

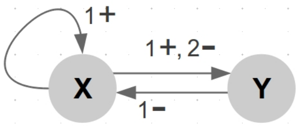
There is no reason to expect that the time series walks of different parametrizations coincide. However, the information whether certain state transitions are shared by the walks can be immediately exploited for experimental design. For example, new measurements would be most useful if placed between two original measurements that generated many different walks leading from one to the other across the valid parametrizations, since the additional information would then enable us to distinguish between them. The plots proposed in this section aim at making this information about the distribution of state transitions of time series walks easy to assess.

Let W be any finite set of time series walks of $(m^1, \dots, m^k)$. In each walk we mark the measurements $1, \dots, k$. We lay out all walks horizontally and align for every $1 \leq i \leq k$, the states marked as the i^{th} measurement vertically. If a measurement is realized by several states in the walk, we choose the state with the smallest index within the walk to represent this measurement. This way we can interpret the horizontal axis as a discrete time axis, progressing from earlier (left) to later (right).

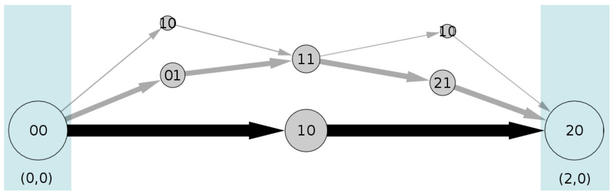
Notice that a state may of course appear in more than one walk, but also multiple times within a single walk. Behavior maps are an attempt to find a compact representation, by removing some of these duplicates, while keeping the acyclic progression from earlier to later states. Therefore we are not interested in the graph defined as the union of walks in W, because it may destroy this progression (by creating cycles).

We treat each pair of successive measurements m^i, m^{i+1} independently and partition the walks into classes of equal length in between m^i and m^{i+1}. Two states are identified as one if they represent the same state of the KS and appear in equal number of steps since last measurement.

We scale the size of a node and width of a transition by the number of parametrizations that produce a walk passing through it. The gray scale shading of a transition is determined by the class it belongs to. Black transitions belong to the overall shortest walks, light gray transitions to the longest walks. E.g., consider following regulatory graph:



The behaviour map of this graph wrt time series $m^1 = \{(0,0)\}, m^2 = \{(2,0)\}$ is:

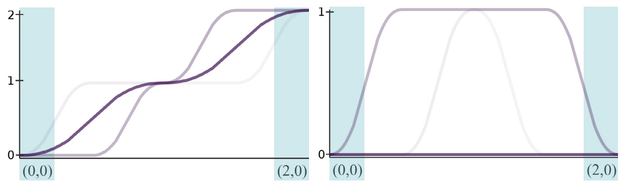


The blue vertical boxes represent measurements. Black transitions belong to length class 3, gray transitions to length class 5. The wider the stroke of a transition or the larger a node, the more walks in W pass through them. Note that the state '10' appears three times. Once in length class 3, and twice in the class with length 5, because it can be visited either early or late in discrete time. State '11' has only one node because it always appears after two transitions.

4.2 Expression Profiles

Behavior maps visualize possible model behaviors in accordance with a time series. For a more refined analysis aiming at experimental design, it is helpful to focus on the behavior of the separate components. The plot of all the ways a component may change its level alongside all time series walks can highlight which components are responsible for differences between walks. In addition, as in the case of behavior maps, we can easily pinpoint for each component between which measurements the component behavior is the most ambiguous.

For single components we suggest to plot sigmoid expression profiles. The transparency of a curve is proportional to the number of shortest walks that share the corresponding transition. Below are the profiles of x and y of the example structure presented in Section 4.1. The most opaque line in the profile of x shows that along most walks it increases evenly from 0 to 2.



Profile of x Profile of y

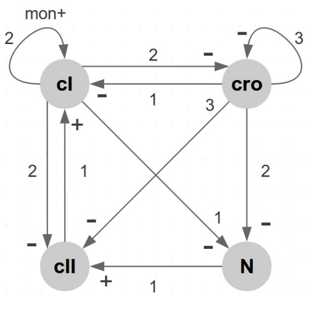
The profile of y indicates that its activity along most walks is constant at 0. In contrast to x, some ambiguity is present here - if this was a real system, experiments should focus on y.

5 Case Study

In this section we apply our workflow to the gene regulatory network of bacteriophage λ. Its discrete version was formulated by Thieffry and Thomas in [20]. The authors also discuss two time series, the lytic (Fig. 5) and lysogenic (Fig. 5) fate of bacteria infected by the bacteriophage (see [20, p.290]). Finally, for comparison, a realistic parametrization denoted R is considered (taken from [20, p.291]). We will judge our results by how close our optimal parametrizations are to R.

Parameter Set Reduction. First, we compute the initial parametrizations with the edge labels of the regulatory graph in Fig. 5. The self-activation of cI at threshold 2 is not observable in R. Since we want R to belong to the initial parametrizations, we also relax this constraint and assign the label mon+ to $(cI\, cI, 2)$. These static constraints reduce the set of feasible parametrizations from $6, 879, 707, 136$ to $82, 008$.

Now, we execute the CMC procedure for both time series. The lytic time series is reproduced by $28, 043$ parametrizations. Of these, 537 also reproduce the lysogenic time series. During this step we also compute Cost and Robustness functions.



(a)

	cI	cII	cro	N
1.	0	0	0	0
2.	2	1	0	1
3.	2	0	0	0

(b)

	cI	cII	cro	N
1.	0	0	0	0
2.	0	0	2	1
3.	0	0	2	0
4.	0	0	3	0
5.	0	0	2	0

(c)

Fig. 2. (a) Regulatory graph of bacteriophage λ with edge constraints. (b) Lysogenic time series. (c) Lytic time series. The last three measurements indicate an oscillation.

Optimal Parametrizations. To illustrate the two step model ranking by Cost and Robustness, we have to focus on one of the time series. We pick the lysogenic series. The theoretical minimum Cost, required to execute the 6 activity changes of the lysogenic series (Fig. 5), is 7. The actual minimum Cost among the 537 feasible parametrizations is 9, and 28 parametrizations contain a walk of this length.

Among those 28, the maximum Robustness is 9.72% and there are 3 parametrizations that attain it. In comparison, R also has a Cost of 9 but its Robustness is only 0.54%.

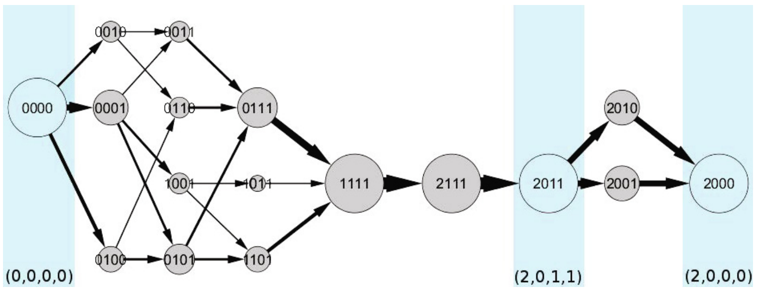
These 3 parametrizations are equal in all target values except for a single context of *cro*: If the only regulator acting above its threshold is *cro* itself, the target value may be any of 0, 1 or 2 (giving us the 3 remaining parametrizations). Since the threshold of the self-regulation is $\theta(cro\,cro) = 3$, all target values below 3 cause this inhibition to stop itself, giving us identical state transition graphs. Hence the remaining 3 parametrizations agree with R on that the target value must be below 3. Any of the 3 parametrizations is optimal wrt Cost and Robustness, and we denote all of them by O.

We compare the target values of the parametrizations R and O in the table below. Values that differ between the two are bold. The component name in the first row of each column denotes the target component. Each successive row contains a list of regulators that are above their (unique) interaction thresholds. The corresponding target values are in the columns R and O.

cI	R	O	cII	R	O	cro	R	O	N	R	O
$\emptyset$	**2**	**1**	$\emptyset$	**0**	**1**	$\emptyset$	3	3	$\emptyset$	1	1
cI	2	2	cI	0	0	cI	0	0	cI	0	0
cro	0	0	cro	0	0	cro	2	<3	cro	0	0
cII	2	2	N	1	1	cI, cro	0	0	cI, cro	0	0
cI, cII	2	2	cI, cro	0	0						
cI, cro	**0**	**2**	cI, N	0	0						
cII, cro	2	2	cro, N	**0**	**1**						
cI, cII, cro	2	2	cI, cro, N	0	0						

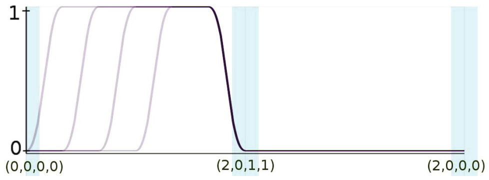
Note that each of the disagreements for cI causes the interaction $cI\,cI$, that is not observable in R, to be observable in O.

Visualization. Using a behavior map including all the shortest time series walks of every acceptable parametrization, we obtain the following graph:



As before, blue boxes mark measurements. The map indicates many possibilities in ordering of activations which always lead from state $(0,0,0,0)$ to state $(1,1,1,1)$, suggesting that it would be reasonable to measure activity levels between these two states.

Number of steps between measurements m^2 and m^3 corresponds to their distance, but for walk from m^1 to m^2 it is not the case.



In the cII expression profile above we can see that alongside every path cII is activated and inhibited between measurements m^1 and m^2 – this also explains why in O this component can be activated more often, as this step is necessary for the model to be able to reproduce the time series.

6 Implementation and Evaluation

In this section we briefly describe methodology of synthesis and analysis together with tools deployed for these tasks. Further we focus on description of a time and space-efficient computation of acceptable parametrizations and evaluate it using two different models.

6.1 Usage Description

Our current workflow of analysis is divided into following steps:

1. Creation of a model - regulatory network is described in a single XML file using our own syntax designed for this purpose. In a future work we expect to implement an option to import models from standard formats.
2. Specification of the property - the property (most usually a time series) is currently specified within a model file in the form of Büchi automaton, also using an XML-based syntax.
3. Synthesis - the model is analyzed using the colored model checker *Parsybone* [1], implemented in C++. The tool works in two steps. First, reduction of parametrization space is conducted if there are any initial constrains specified. The reduced parameter space than undergoes the process of parameter synthesis. By default, this step produces only enumeration of acceptable parametrizations. However, for each of the parametrizations we can optionally compute and output its shortest paths or the robustness value.

[1] *Parsybone* – `http://github.com/sybila/Parsybone`

4. Filtering - the amount of data produced by synthesis is vast in most cases, therefore we usually employ a second tool, *ParameterFilter* [2], implemented in C$\sharp$. It is a GUI elaborating on the output from the synthesis step allowing to select and compare parametrizations based on their ranking.
5. Plotting - finally, for parametrizations chosen during the previous step we plot their expression profiles (using *ParameterFilter*) or their behavior map which is produced from the synthesis output using our converter, implemented in Java. The converter creates a behaviour map for a given parametrization and allows its visualization using Cytoscape [6].

6.2 CMC Procedure Implementation

Algorithm for colored model checking as presented in [2] does not specify, how distinct parametrizations should be stored and manipulated. For continuous models, we have used bounded intervals of values for each component, creating a parametrization space as a Cartesian product of those. We have later employed this approach for discrete models as well, but it turned out that in this case it suffers from high complexity of often performed operations like set intersection (for more information about the algorithm, see [13]). To tackle this problem, we have moved to explicit representation where all parametrizations are enumerated. We will show that this approach provides numerous advantages and allows for analysis of large parametrization sets.

Encoding. Our approach is based on a computationally efficient encoding of parametrization space. We encode each parametrization set $\mathcal{P}' \subseteq \mathcal{P}$ as a word of length $|\mathcal{P}|$ over alphabet $\Sigma = \{0, 1\}$. Such a word naturally corresponds to a bit vector of the same length and allows fast computation using bitwise operations.

We consider lexicographical ordering of the set $\mathcal{P}$. We denote $P^i \in \mathcal{P}$ an i-th parametrization in $\mathcal{P}$. Now to encode an ordered set $\mathcal{P}' \subseteq \mathcal{P}$, we use the *encoding function* $Code : 2^{\mathcal{P}} \to \{0,1\}^{|\mathcal{P}|}$ where $Code(\mathcal{P}') = b_1 b_2 ... b_{|\mathcal{P}|}, \forall i (b_i = 1 \Leftrightarrow P^i \in \mathcal{P}')$. This way we encode a coloring of every state as a single word of length $|\mathcal{P}|$.

The encoding function is of a crucial importance, because the idea of the CMC and its main performance improvement lies in the option to create only a single PKS for the whole parametrization space. To create such a structure, we need to be able to label edges of the PKS with transitive parametrizations. This can be done using the encoding function by which we label every transition $x \to x'$ with a word $Code(\{P | x \xrightarrow{P} x'\})$.

In general, by using such an encoding we reduce the CMC problem to a sequence of bitwise operations.

Splitting. Our coloring algorithm is based on an iterative computation of a fixed point. Complexity of this computation can be improved using multiple heuristics, for complete information we refer to [13]. The most important is the procedure of splitting.

[2] *ParameterFilter* – `http://github.com/sybila/ParameterFilter`

Our idea is based on the assumption that similar parametrizations generate similar KSs [2]. When computing a coloring of a PKS we split its parametrization space to multiple neighbouring regions and work only with a single region at a time. Most of parametrizations within a single region are likely to be either all accepted or all rejected, allowing us to quickly reach the fixed point.

Due to lexicographical ordering of possible parametrizations within a bit vector, we already have similar parametrizations in the neighbouring positions. During the computation we then split the parametrization space by working always with next m bits of the bit vector. Each region is stored within a single *integer* variable, therefore m is equal to size of an integer in bits on a target platform. Note that usage of integers also ensures quick computation of bitwise operations. With this region, we go through the whole process of analysis, output the data, free the memory and continue with another round (ensuring low memory requirements).

Distribution. When using the split parameter space (which we can do only when using explicit data representation), we can easily distribute the computation. This is because every parametrization is completely independent on all others, giving us great potential for a data-parallel distribution. Therefore, we distribute regions of parametrization space between non-communicating processes differing only in their ID.

Each independent worker does its own parsing and pre-computation and then goes through the procedure of parameter identification with a subset of parametrization space that is disjunctive with subsets of other workers.

To achieve as optimal load balance as possible, distribution of regions is interlaced, meaning that in computation of n processes, process with ID $i, 1 \leq i \leq n$ is assigned only regions $i + k \cdot n, k \in \mathbb{N}$. This method is again based on the assumption that similar parametrizations generate similar behaviour, causing acceptable parametrizations to cluster. This way we ensure that such clusters are distributed evenly between processes.

6.3 Evaluation

Mammalian Cell Cycle. To test capabilities of our algorithm, we had it analyze a model of mammalian cell cycle [10] with 9 components. For this model we have defined partial specification, reducing size of parametrization space to final number of $675,584,064$ parametrizations. As a guide for the analysis we have used time series with 8 measurements. More detailed information are presented in Technical report [13].

Parametrization space was evenly distributed between 8 independent process, each one of them having initial set of size $84,448,008$. Computation was run on a Linux server using two processors with four 2.27 GHz cores and took roughly a day with $308,180,639$ acceptable parametrizations computed. During computation each of the processes used less than 15 MB of RAM. Exact results for each process are presented in the Figure 6.3. As can be seen, parametrizations space has been partitioned to sets with almost identical numbers of acceptable parametrizations.

Process ID	Runtime	Result set size	Process ID	Runtime	Result set size
1	29.07 h	38,522,403	5	29.70 h	38,523,691
2	31.08 h	38,521,943	6	28.81 h	38,523,255
3	27.22 h	38,521,656	7	29.55 h	38,522,328
4	32.32 h	38,522,343	8	28.83 h	38,523,020

Fig. 3. Results of distributed analysis of Mammalian cell cycle

Bacteriophage. As a main benchmark we have employed the bacteriophage λ network. Minor utilization was necessary mainly because our old tool (which is employed for comparison) is not able to work with edge labels. We have therefore created unlabelled and partially specified version of the model with $|\mathcal{P}| = 589,824$ out of which $90,112$ parametrizations are acceptable. We ran the analysis five times using each tool. Analysis using the old version took on the average 967 seconds and used at max 50 MB of RAM, analysis using the new version took always less than 6 seconds and did not use more than 3 MB RAM.

To demonstrate scalability we analyze the bacteriophage model using up to 8 independent processes. In Fig. 4 we show average runtime of all processes used. Resulting numbers are taken as an average of three independent experiments on the same platform as in case of mammalian cell cycle. As can be seen from the graph, scaling of our algorithm is roughly linear. This result suggests that in every case where distribution of computation seems sensible, it is possible to achieve almost a linear speedup and therefore we can extend a set of models that can be practically analyzed by using high-performance computational platforms.

Process count	Average runtime
1	5.315 s
2	2.634 s
3	1.767 s
4	1.332 s
5	1.048 s
6	0.884 s
7	0.754 s
8	0.657 s

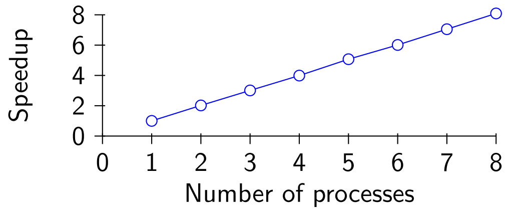


Fig. 4. Scalability of algorithm on bacteriophage λ

7 Conclusions

We have contributed to solving the parameter identification problem for Thomas networks in three aspects. First, we have proposed a new methodology based on a colored model checking approach extended with parametrization ranking procedures. Second, we have introduced a new idea of parametrization encoding that allows us to synthesize parametrizations in an efficient manner on distributed platforms. Third, we have implemented a prototype tool chain that supports all steps of our methodology including feasible visualization of obtained results.

By evaluating our algorithms on several biological models, we have demonstrated that the computation achieves good scaling, and moreover, that it copes with larger parameter spaces. Comparing these results with our previous achievements [2,12], possibilities of parameter identification solved by model checking have been significantly improved.

On the methodological side, our achievement brings new insights into applying discrete modeling frameworks to gene networks. The case study has shown that the approach is can help modelers to identify reasonable parametrizations and derive supported suggestions for experimental design.

On the computational side, improving the efficiency of the parameter filtering and visualization part of the tool chain will be a focus of future work.

References

1. Baier, C., Katoen, J.-P.: Principles of Model Checking. The MIT Press (2008)
2. Barnat, J., Brim, L., Krejci, A., Streck, A., Safranek, D., Vejnar, M., Vejpustek, T.: On Parameter Synthesis by Parallel Model Checking. IEEE/ACM Transactions on Computational Biology and Bioinformatics 9(3), 693–705 (2012)
3. Batt, G., Page, M., Cantone, I., Goessler, G., Monteiro, P., de Jong, H.: Efficient parameter search for qualitative models of regulatory networks using symbolic model checking. Bioinformatics 26(18), i603–i610 (2010)
4. Bernot, G., Comet, J.-P., Richard, A., Guespin, J.: Application of formal methods to biological regulatory networks: Extending Thomas' asynchronous logical approach with temporal logic. Journal of Theoretical Biology 229(3), 339–347 (2004)
5. Chaouiya, C., Remy, E., Mossé, B., Thieffry, D.: Qualitative Analysis of Regulatory Graphs: A Computational Tool Based on a Discrete Formal Framework. In: Benvenuti, L., De Santis, A., Farina, L. (eds.) Positive Systems. LNCIS, vol. 294, pp. 119–126. Springer, Heidelberg (2003)
6. Cline, M., et al.: Integration of biological networks and gene expression data using Cytoscape. Nat. Protocols 2(10), 2366–2382 (2007)
7. Corblin, F., Fanchon, E., Trilling, L., Chaouiya, C., Thieffry, D.: Automatic Inference of Regulatory and Dynamical Properties from Incomplete Gene Interaction and Expression Data. In: Lones, M.A., Smith, S.L., Teichmann, S., Naef, F., Walker, J.A., Trefzer, M.A. (eds.) IPCAT 2012. LNCS, vol. 7223, pp. 25–30. Springer, Heidelberg (2012)
8. Corblin, F., et al.: A declarative constraint-based method for analyzing discrete genetic regulatory networks. Biosystems 98(2), 91–104 (2009)
9. de Jong, H.: Modeling and Simulation of Genetic Regulatory Systems: A Literature Review. Journal of Computational Biology 9(1), 67–103 (2002)
10. Fauré, A., Naldi, A., Chaouiya, C., Thieffry, D.: Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. In: ISMB (Supplement of Bioinformatics) 2006, pp. 124–131 (2006)
11. Helikar, T., Konvalina, J., Heidel, J., Rogers, J.A.: Emergent decision-making in biological signal transduction networks. Proceedings of the National Academy of Sciences 105(6), 1913–1918 (2008)
12. Klarner, H., Siebert, H., Bockmayr, A.: Time series dependent analysis of unparametrized thomas networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics 99(PrePrints) (2012)

13. Klarner, H., Streck, A., Safranek, D., Kolcak, J., Siebert, H.: Parameter identification and model ranking of Thomas networks. Technical Report FIMU-RS-2012-03, Masaryk University (2012)
14. Laubenbacher, R., Mendes, P.: A discrete approach to top-down modeling of biochemical networks. In: Kriete, A., Eils, R. (eds.) Computational Systems Biology, pp. 229–247. Elsevier Academic Press (2005)
15. Lee, W.-P., Tzou, W.-S.: Computational methods for discovering gene networks from expression data. Briefings in Bioinformatics 10(4), 408–423 (2009)
16. Naldi, A., Remy, E., Thieffry, D., Chaouiya, C.: Dynamically consistent reduction of logical regulatory graphs. Theor. Comput. Sci. 412(21), 2207–2218 (2011)
17. Richard, A.: SMBioNet-1.4 User manual (2005)
18. Sánchez, L., Thieffry, D.: A logical analysis of the drosophila gap-gene system. Journal of Theoretical Biology 211(2), 115–141 (2001)
19. Siebert, H., Bockmayr, A.: Incorporating Time Delays into the Logical Analysis of Gene Regulatory Networks. In: Priami, C. (ed.) CMSB 2006. LNCS (LNBI), vol. 4210, pp. 169–183. Springer, Heidelberg (2006)
20. Thieffry, D., Thomas, R.: Dynamical behaviour of biological regulatory networks II. Immunity control in bacteriophage lambda. Bulletin of Mathematical Biology 57, 277–297 (1995)
21. Thomas, R.: Regulatory networks seen as asynchronous automata: A logical description. Journal of Theoretical Biology 153(1), 1–23 (1991)